

仿真机器人足球：设计与实现

（第一部分：1-5 章，介绍仿真机器人足球基本原理和简单实现。约 100-110 页）

第一章、导论

1.1 机器人足球概述

什么是 RoboCup（组织、内容、比较 FIRA、梦想、挑战计划）

RoboCup 历史和现状（国际、国内、教育发展）

RoboCup 应用技术与相关发展介绍（面向社会生产、教育、娱乐）

1.2 仿真机器人足球

内容、特点

研究重点（人工智能新的标准问题、合成智能体挑战）

1.3 本书安排

章节安排

学习指导

第二章、2D 比赛平台

2.1 平台结构特点

Server/Client 结构及各部分介绍（Server、Client、Monitor、Logplayer）

2.2 比赛规则

自动裁判、人为干预

2.3 比赛平台安装使用

server 源码包结构说明、安装、使用

2.4 比赛流程

第三章、快速入门

3.1 让球员上场比赛

与 server 建立连接

3.2 球员可以发送的命令

身体控制、通讯、数据请求

3.3 球员可以获得的信息

视觉、听觉、身体感知

3.4 一般球员程序的框架结构

网络连接特性分析

球员程序框架

第四章、球员智能体建模

4.1 简单世界模型的构建

平台中的对象（结构、命名）

感知模型（视觉、听觉、身体感知）

4.2 复杂世界模型的构建

运动模型

4.3 基本行为模型

扑球、踢球、移动、加速模型和体力模型

调整方向模型（转身、扭头）

通讯模型

其他行为模型（铲球、pointto、attentionto）

4.4 其他

异质球员、裁判模型

4.5 世界模型更新

- 简单模型
- 带预测模型

4.6 更复杂的个体行为

- 如何跑到指定的位置
- 如何将球踢到指定的位置
- 带球的简单实现

第五章、设计更高级的球员智能体

5.1 决策的重要性

5.2 简单决策原理

- 进攻、防守、站位

5.3 简单团队合作实现

- 通过 `set_play` 实现合作及实例分析

5.4 更有效的获取信息

- 信息更新采集、通讯

（第二部分：6-10 章，从仿真机器人足球平台中抽取典型问题，结合典型球队的设计思想，分析各自特点、实现。约 75-85 页）

第六章、球员智能体体系结构

6.1 CMU 结构分析

6.2 FCP 结构分析

6.3 Tsinghua 结构分析

6.4 BS 结构分析

6.5 WE 结构分析

6.6 小结

第七章、信息获取与维护

7.1 信息获取的一般方法

7.2 提高信息处理的精度

7.3 信息不确定时的预测

7.4 主动信息采集

7.5 实现一个简单的 client 程序

第八章、球员行为分析

8.1 快速踢球

根据踢球模型搜索，用爬山法优化搜索

动态规划

基于范例学习、强化学习

8.2 拦截模型

问题分析
模拟法
解析法、二分法

8.3 带球模型

问题分析与实现
强化学习

8.4 射门模型

模型分析、提高成功率

8.5 传球模型

第九章、高层决策

9.1 阵型与跑位

基于角色的阵型
阵型的一种描述 SBSP
基于阵型的全局跑位

9.2 行为选择

基于规则的选择
基于目标的选择
基于效用的选择

9.3 防守策略

防守体系的一般想法
防守的基本行为
防守行为评估、选择

9.4 团队合作

战术表示、实现

(第三部分：10-12 章 高级专题。约 85-95 页)

第十章、机器人足球中的学习

10.1 决策树学习

10.2 神经网络学习

10.3 强化学习

10.4 机器人足球中的应用

第十一章、教练智能体

11.1 教练简介

离线、在线

11.2 教练语法

教练语言的语法简介

11.3 教练程序结构

给出一个教练的基本框架，以及实现的原理

11.4 教练功能

训练、调试，在线分析

第十二章 仿真机器人足球 3D 比赛

12.1 3D 比赛介绍

12.2 3D 平台结构

12.3 3D 球员程序设计

第一章、导论

1.1 机器人足球概述

机器人足球世界杯,是近年国际信息科技界新兴的一项集前沿研究与教育于一体的大型活动。在研究方面,“机器人足球”涉及计算机、机器人、通讯、自动化和机电一体化等多学科的前沿技术的开拓和综合集成;在教育方面,已被视为培养 21 世纪信息科技一流人才的重要手段(欧美一些学校已将“机器人足球”作为许多课程的核心内容)。概括地说,机器人足球赛是以体育竞赛为载体的前沿科研竞争和高科技对抗,是培养信息—自动化科技人才的重要手段,同时也是展示高科技进展的生动窗口和促进科技成果实用化和产业化的新途径。

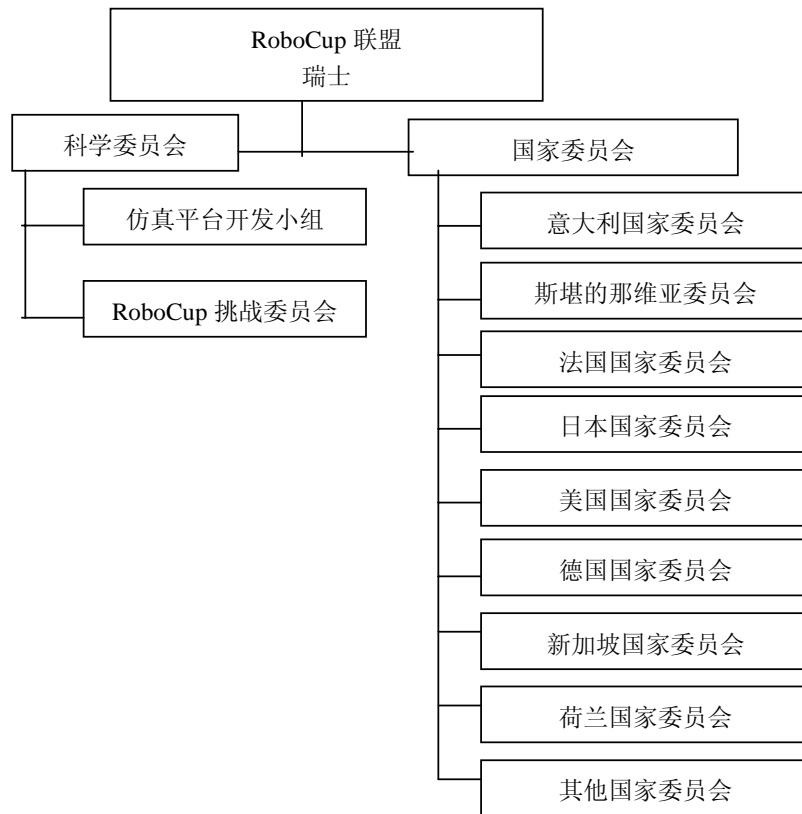
1.1.1 什么是 RoboCup

RoboCup 联盟(起初称作 Robot World Cup Initiative)是一个国际性研究和教育组织,它通过提供一个标准问题来促进人工智能和智能机器人的研究。这个领域应该可以集成并检验很大范围内的技术,同时也可被用作综合的面向工程应用的教育。

为了这个目的,RoboCup 联盟选择了足球比赛作为一个基本领域,并组织了国际上级别最高、规模最大、影响最广泛的机器人足球赛事和学术会议——机器人足球世界杯及学术会议(The Robot World Cup Soccer Games and Conferences, 简称 RoboCup)。为了能让一个机器人球队真正能够进行足球比赛,必须集成各种各样的技术,包括自治智能体的设计准则、多主体合作、策略获取、实时推理、机器人学以及感知信息融合等。对一个由许多快速运动的机器人组成的球队来说,RoboCup 是一项在动态环境下的任务。在软件方面,RoboCup 还提供了软件平台以便于研究。

1.1.1.1 组织结构

RoboCup 联盟是世界上最大的、占主导地位的机器人足球国际组织,总部设在瑞士,现有成员国近 40 个。联盟由国际著名科学家、在 IJCAI-93 大会上获得国际人工智能最高奖——“计算机与思维”大奖的北野宏明(Hiroaki Kitano)发起,他也成为 RoboCup 联盟的第一任主席。联盟负责世界范围的学术活动和竞赛,包括每年一届的世界杯赛和学术研讨会,并为相关的本科生和研究生教育提供支持(教材、教学软件等)。



• 图 1.1 RoboCup 联盟组织结构

1.1.1.2 活动内容

在足球比赛作为标准问题的同时，还会有其它各种各样的努力，比赛只是 RoboCup 各项活动的一部分。

当前 RoboCup 的活动包括：

技术研讨

机器人国际比赛和学术会议

RoboCup 挑战计划

RoboCup 教育计划

基础组织的发展

1. RoboCup 比赛

尽管有上述这么多的活动内容，机器人足球世界杯比赛还是各项活动的中心，在那儿研究工作者们可以在一起评估研究进展。

现在，RoboCup 比赛主要有三个领域：



• 图 1.2 RoboCup 比赛三个领域关系图

● RoboCup 足球赛——技术挑战

RoboCup 足球赛是由硬件或仿真机器人进行的足球赛，比赛规则与人类正规的足球赛类似。硬件机器人足球队的研发涉及计算机、自动控制、传感与感知融合、无线通讯、精密机械和仿生材料等众多学科的前沿研究与综合集成。仿真机器人足球赛在标准软件平台上进行，平台设计充分体现了控制、通讯、传感和人体机能等方面的实际限制，使仿真球队程序易于转化为硬件球队的控制软件。仿真机器人足球的研究重点是球队的高级功能，包括动态不确定环境中的多主体合作、实时推理—规划—决策、机器学习和策略获取等当前人工智能的热点问题。

目前比赛项目包括：

- 仿真组
- 小型机器人组(f-180)
- 中型机器人组(f-2000)
- 四腿机器人组(由 Sony 赞助，从 1999 年开始)
- 人形机器人组(从 2002 年开始)
- E-League（从 2004 年开始）
- RoboCup 评论员系统演示

RoboCup 救援比赛——实践应用

RoboCup 救援比赛是 RoboCup 一个新的研究领域，目标是在如日本神户大地震这种大灾难中的高效、准确地找到受难者，并实施救援行动。选择这个领域一方面是因为这是一个意义重大的社会问题，另一方面它在很多方面都和足球比赛有相似的特点，如动态环境，不完全、有噪声的信息。同时它还有些足球比赛没有的特点，如后勤学、紧急团队间的合作等。相比足球比赛，对救援问题的研究更能给社会带来直接的益处，所以同样受到广泛的关注。

目前救援比赛分仿真组和机器人组。



• 图 1.3 RoboCup 救援比赛

RoboCup 初级比赛——下一代教育

RoboCup 初级比赛是 RoboCup 教育、推广的重要组成部分，是面向中小学生的教育方式，使得他们不需要具有很高深的知识就可以轻松进入机器人学领域，通过对电子装置，硬件和软件的直接动手实践来提高自身能力，同时更有助于他们学习团队合作。这种教育方式与传统的教育相比更能激发参与者的广泛的兴趣，更容易培养出合作完成一个共同目标的能力。

去前连个部分对应的，初级组比赛主要分以下三个部分：

- 足球比赛
- 跳舞比赛
- 救援比赛

2. 挑战计划

RoboCup 吸引了这么多研究工作者的主要原因就是它需要将很大范围的技术集成到一个完整的智能体团队中，这就有别于一些基于专门任务的功能模块。RoboCup 提供了一些重大的长期挑战，将会需要几十年的时间来完成。长期的研究问题实在太广泛，难以编撰成一系列条目的列表。虽然如此，还是可以说，这些挑战包括了从物理部件的开发(如高性能电池、马达)到高智能化的实时感知和控制软件这些极其广泛的技术问题。

为了更进一步明确目标，可以分解这些长期目标得到几个子目标，这就是中短期的挑战。中期技术挑战是今后十年的目标，可以说得更具体一些，以下就是 RoboCup 中涉及到的研究领域的部分列表，主要以中期时间段为目标：(1)通用的智能体体系结构；(2)综合反应式方法和建模/规划式方法；(3)实时识别、规划和推理；(4)在动态环境中推理和行动；(5)传感器数据融合；(6)通用的多主体系统；(7)复杂任务中的行为学习；(8)策略获取；(9)通用的认知模型。

除了这些技术，提供一个带有高质量 3D 图形能力的网络足球仿真平台需要在模拟足球运动员的实时动画和基于网络的交互式多用户系统方面有一定的技术进步。这些都是今后几年基于网络的服务中的关键技术。

RoboCup 的挑战应该理解为更大、更长期的挑战，而不是一个一次性的挑战。因此，还要提供一系列的短期挑战，这将会很自然的引导中长期挑战的完成。RoboCup 挑战主要分为三类：(1)合成智能体挑战；(2)物理智能体挑战；(3)基础组织挑战。RoboCup 合成智能体挑战处理可以用软件仿真平台开发的技术；RoboCup 物理智能体挑战的意图是促进使用实际机器人的研究，因此需要更长的时间来完成每一项挑战。它们将和 RoboCup 合成智能体挑战同时进行研究，但需要更长的时间。提出基础组织挑战是为了方便研究而建立一个关于 RoboCup、人工智能和机器人学的总的基础组织。这些挑战包括教育计划、通用机器人平台和部件标准、自动评论员系统和进行 RoboCup 比赛的智能体育场系统。

3. 教育计划

RoboCup 教育计划着眼于在一些大学进行以 RoboCup 为基础的人工智能或机器人学教育，例如可以开设以下课程：

- RoboCup 中的人工智能(本科生或研究生课程)，
- RoboCup 中的机器人学导论(本科生课程)，
- RoboCup 中的人工智能程序设计(本科生课程)，
- 多主体系统 (研究生课程)，及
- 其它

已经开设的以 RoboCup 为基础的课程有：

- 人工智能程序设计, Artificial Intelligence Programming (with RoboCup) by Silvia Coradeschi and Jacek Malec, Linköping University, Sweden
- 自治系统, Autonomous Systems by Andreas Birk; Vrije Universiteit Brussel, Brussels, Belgium
- 自治多主体系统, Autonomous Multiagent Systems by Peter Stone, **Texas University, Austin, U.S.A**
- 机器人足球研讨班, 陈小平、杨斌, 中国科学技术大学, 合肥, 中国

与此同时, RoboCup 组织开展了 RoboCup 初级组活动以促进 RoboCup 在学生、家庭、娱乐方面的发展。

中国科技大学在开展 RoboCup 研究工作的同时, 已将这一活动纳入“本科生研究计划”, 是国内最早开设机器人足球课程的高校, 也是目前教学规模最大的高校。在校有关领导和教务处的关心和指导下, 现已初步形成了较完备的教学体系, 包括仿真机器人足球初级班、提高班和高级研讨班, 以及四腿机器人研讨班, 在课堂之外还举办了“仿真机器人足球教学公开赛”、“机器人比赛活动周”等多种形式的比赛。

4. 基础组织

基础组织是为了方便研究而建立的一个关于 RoboCup、人工智能和机器人学的总的组织。并制定了相应的基础组织挑战计划包括上面提到的教育计划, 另外还包括通用机器人平台和部件标准挑战计划、自动评论员系统和进行 RoboCup 比赛的智能体育场系统挑战计划。

在与 RoboCup 相关的自然语言处理方面, RoboCup 希望自然语言研究人员能作出他们的贡献, 如 RoboCup 比赛的自动评论及描述生成系统。作为一个 RoboCup 与自然语言研究结合的例子, DFKI 的 VITRA 项目展示了自然语言研究在 RoboCup 中的应用, 它可以从足球比赛的画面生成自然语言描述。

对于仿真比赛, 正在开发基于 VRML 或 Java applet 等技术的 3D 可视化系统。

1.1.1.3 RoboCup 与 FIRA

目前有两个主要的机器人足球全球性学术组织, 除 RoboCup 联盟之外, 还有其他一些国际组织。其中较大的一个是 FIRA(Federation of International Robot-soccer Association), 由韩国 KAIST 的金钟焕等人发起, 成立于 1997 年, 总部设在韩国大田, 每年举办一次国际性比赛。目前有成员国 20 多个, 我国的东北大学和哈工大加入了该组织。

FIRA 与 RoboCup 的主要区别如下:

技术标准不同 FIRA 允许采用集中控制方式, 因此一个球队中的不同队员是同一个“大脑”(控制程序)的不同执行器; 而 RoboCup 要求每个队员必须是自主的, 因而球队中的每个队员是一个独立的“主体”, 球队是一个“多主体”系统。

研究重点有区别 FIRA 重点研究“灵巧机器人”, 特别是动态环境中的运动控制; 而 RoboCup 着重研究动态环境中的多主体合作、实时规划和 Agent 体系结构设计。也可以认为, FIRA 主要面向机器人学, 而 RoboCup 主要面向人工智能。

主要参加国不同 FIRA 的主要成员来自东亚和南美等第三世界国家, 而 RoboCup 的主要成员为日美和欧洲各发达国家。

比赛规模不同 相比每年 FIRA 世界杯赛的规模要小得多

1.1.1.4 梦想

RoboCup 联盟的目标是通过提供引人瞩目但又非常困难的挑战, 将 RoboCup 作为一个

WrightEagle

工具来促进人工智能和机器人学研究。促进这种研究的一个有效途径是制定一个重大的长期目标。当这个目标完成时，将产生巨大的社会影响，这就可以称之为重大挑战计划。制造一个会踢足球的机器人本身并不能产生巨大的社会和经济影响，但是这种成功的确会被认为是这个领域的重大成果。RoboCup 在作为一个研究的标准问题的同时也是一个划时代的计划。

RoboCup 的最终目标是：

到 21 世纪中叶，一支完全自治的人形机器人足球队应该能在遵循国际足联正式规则的比赛 中，战胜最近的人类世界杯冠军队。

这个目标是人工智能与机器人学今后 50 年的一个重大挑战。从目前的技术水平看来，这个目标可能是过于雄心勃勃了。但提出这样的长期目标并为之而奋斗是非常有必要的。从莱特兄弟的第一架飞机到阿波罗计划将人类送上月球并安全返回地球只花了 50 年。同样，从数字计算机的发明到深蓝击败人类国际象棋世界冠军也只花了 50 年。可以预见到，建立人形机器人足球队也需要大致相当的时间及很大范围内研究人员的极大努力，这个目标是不能在短期内完成的。

表 1.1 划时代的计划

	阿波罗计划	计算机国际象棋	RoboCup
目标	送一个宇航员登陆月球并安全返回地球	开发出能战胜人类国际象棋世界冠军的计算机	开发出能象人类那样踢球的足球机器人
技术	系统工程, 航空学, 各种电子学等	搜索技术, 并行算法和并行计算机等	实时系统, 分布式协作智能体等
应用	遍布各处	各种软件系统、大规模并行计算机	下一代人工智能, 现实世界中的机器人和人工智能系统

一个成功的划时代计划必须完成一个非常引人注目而且能引起广泛关注的目标。其重要问题是设定一个足够高的目标, 才能取得一系列为完成这个任务而必需的技术突破, 同时这个目标也要有广泛的吸引力和兴奋点。另外, 这些完成目标所需的技术必须是可以成为下一代工业基础的技术。在 RoboCup 中, 最终目标是“开发一支能战胜人类世界冠军队的机器人足球队。”(更适当的目标是“开发一支能象人一样踢球的机器人足球队”)

不用说, 最终目标的实现, 如果不是几个世纪的话, 至少也需要几十年的努力。根据现在的技术水平, 是不可能在短期内完成这个目标的。然而, 这个目标可以很容易的创立一系列相应的子目标。对任何雄心勃勃的计划来说, 这种途径是较普遍的。在美国太空计划中, 完成载人轨道飞行的 Mercury 计划和 Gemini 计划就是阿波罗计划的前导。RoboCup 第一个要完成的子目标是“创建真实的和软件的机器人足球队, 在修改过的规则下很好的踢球”。即使完成这个目标都会毫无疑问的产生能影响很大范围工业的技术。

1.1.2 RoboCup 历史和现状

在人工智能与机器人学的历史上, 1997 年将作为一个转折点被记住。在 1997 年 5 月, IBM 的深蓝击败了人类国际象棋世界冠军, 人工智能界四十年的挑战终于取得了成功。在 1997 年 7 月 4 日, NASA 的“探路者”在火星成功登陆, 第一个自治机器人系统 Sojourner 释放在火星的表面上。与此同时, RoboCup 也朝开发能够战胜人类世界杯冠军队的机器人足球队走出了第一步。

1.1.2.1 历史发端

机器人足球的最初想法由加拿大不列颠哥伦比亚大学的艾伦·马克沃斯 (Alan Mackworth) 教授于 1992 年提出。日本学者迅速对这一想法进行了系统的调研和可行性分析。1993 年 6 月, 包括浅田埴 (Minoru Asada)、Yasuo Kuniyoshi 和北野宏明 (Hiroaki Kitano) 在内的一些研究工作者决定创办一项机器人比赛, 暂时命名为 RoboCup J 联赛 (J 联赛是刚创办的日本足球职业联赛的名字)。然而在一个月之内, 他们就接到了绝大部分是日本以外的研究工作者的反应, 要求比赛扩展成一个国际性的联合项目。由此, 他们就将这个项目改名为机器人足球世界杯赛 (Robot world cup soccer games, 简称 RoboCup)。

与此同时, 一些研究人员开始将机器人足球作为研究课题。隶属于日本政府的电子技术实验室 (ETL) 的松原仁 (Itsuki Noda) 以机器人足球为背景展开多主体系统的研究, 并已经开始开发一个专用的足球比赛模拟器。这个模拟器后来成了 RoboCup 的正式足球比赛仿真平台。日本大坂大学的浅田埴教授、美国卡内基—梅隆大学的 Veloso 教授 (RoboCup 联盟现任主席) 和她的学生 Peter Stone 等也开展了同类工作。没有这些先驱者的参与, RoboCup 就不可能产生。由此, 机器人足球迅速成长为国际人工智能和机器人学研究的一个重要主题

和方向。

1993 年 9 月, RoboCup 第一次发表公告, 并草拟了明确的规则。于是, 在很多会议和研讨会上进行了关于组织和技术问题的讨论, 包括 AAAI-94(美国人工智能联合会会议), JSAI(日本人工智能学会)研讨会以及其他机器人界的会议。同时, 松原仁(Noda)在 ETL 的小组宣布了仿真比赛平台初始版本(LISP 版本), 这是为进行多主体系统研究而开发的第一个足球领域的开放系统仿真平台, 后来又通过 Web 发布了 1.0 版本的仿真比赛平台(C++版本)。这个仿真平台的第一次公开演示是在 IJCAI-95。

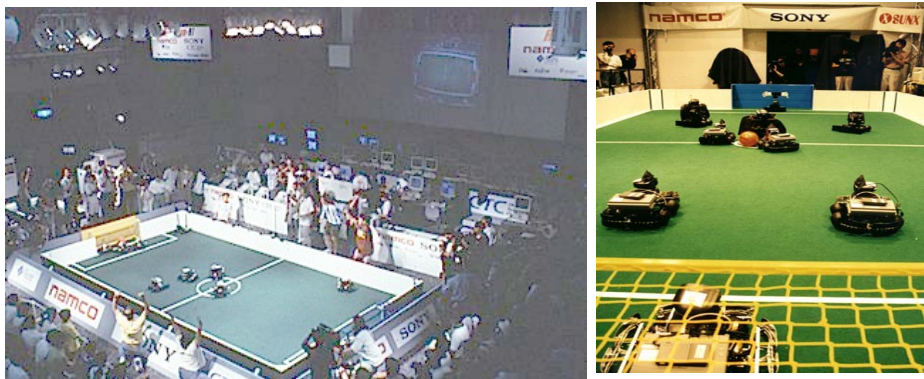
1995 年 8 月, 在加拿大蒙特利尔召开的国际人工智能会议(IJCAI-95)上发表了公告, 将在名古屋与 IJCAI-97 联合举办首届机器人世界杯足球赛及会议。同时, 为了发现与组织大型 RoboCup 比赛有关的潜在问题, 决定先举办 Pre-RoboCup-96。作出这个决定是为了留出两年的准备和开发时间, 这样研究小组就可以开始开发机器人和仿真的球队, 同时也能有时间筹集研究经费。

1996 年 11 月 4 日到 8 日, 在大坂的国际智能机器人与系统会议(IROS-96)上举行了 Pre-RoboCup-96。有 8 支球队参加了仿真组比赛, 并展示了参加中型组比赛的真正的机器人。虽然规模不大, 但这是第一次将足球赛用于促进研究与教育的比赛。

1997 年, 在国际最权威的人工智能系列学术大会——第 15 届国际人工智能联合大会(The 15th International Joint Conference on Artificial Intelligence, 简称 IJCAI-97)上, 机器人足球被正式列为人工智能的一项挑战。至此, 机器人足球成为人工智能和机器人学新的标准问题。

1.1.2.2 历届杯赛情况

1997 年 8 月 25 日, 经过数年准备和 1996 年的试验性比赛(Pre-RoboCup-96), RoboCup 机器人足球世界杯赛在日本名古屋与 IJCAI-97 联合举行。这是第一次正式的 RoboCup 比赛和会议, 获得了巨大的成功。比赛仅设了机器人组和仿真组两个组别, 来自美国、欧洲、澳大利亚、日本等 40 多支球队参加, 5000 多名观众观看了比赛。从此, RoboCup 作为机器人学和人工智能研究领域的最重要的活动之一蓬勃发展起来。



• RoboCup-97 (日本名古屋)

1998 年 7 月 4 日, 正当第 16 届世界杯足球赛渐入佳境之时, 第二届 RoboCup 世界杯赛在法国巴黎举行, 来自世界各地 20 多个国家达 60 多支球队参加了比赛。这也成为历史上最大的移动机器人活动。另外, 1998 年还举办了一系列的其它活动, 如新加坡的 RoboCup-98 环太平洋系列, 加拿大维多利亚的 RoboCup-98 IROS 系列, 德国的 VISION(欧洲最大的计算机视觉国际会议)RoboCup-98, RoboCup-98 日本公开赛, Autonomous Agent 98 上的 RoboCup 仿真组演示, AAAI-98 移动机器人比赛和展示。

随后在瑞典斯德哥尔摩（增设由 Sony 公司赞助的四腿机器人比赛项目）、澳大利亚墨尔本（开设初级组和救援比赛项目）、美国西雅图、日本福岡/韩国釜山、意大利帕多瓦，RoboCup 世界杯一年一度的在举行，同时还伴随着各种各样的地区联赛，越来越多的国家参与其中，比赛内容、规模也不断丰富、扩大。到了 2004 年在葡萄牙里斯本举行的第八届 RoboCup 世界杯赛，来自近 40 个国家超过 1600 人参与的共 346 支球队参加了本届比赛。

机器人足球正以一种高技术对抗的形式赢得学术界的认同。一些学术刊物如 Artificial Intelligence Journal、AI Magazine、Applied Artificial Intelligence、Advanced Robotics Journal、The Journal on Robotics and Autonomous System、The International Journal of Intelligent Automation and Soft Computing 等都出版了机器人足球专辑。一些有影响的国际学术会议如 IJCAI、IROS、Agent、ICMAS、AAAI 研讨会、JSAI 研讨会等也都安排了这方面的专题讨论。一年一度的 RoboCup 也同时召开学术会议，推动相关学科的进展。可以预料，有朝一日机器人足球会因为它的娱乐性、观赏性，以及高技术挑战性，不仅有助于科研与教学的进展，也会形成相当数量的需求与产业，从而长盛不衰，不断开创新的局面。

1.1.2.3 RoboCup 在中国

1999 年 5 月，由清华大学计算机系和中国科技大学计算机系共同发起，成立了 RoboCup 中国分会筹备委员会。东北大学、哈工大、国防科大和香港科大等高校也加入了该组织。根据 RoboCup 联盟的正式授权，RoboCup 中国分会负责组织中国境内的一切 RoboCup 活动，包括学术研讨、比赛和培训等等。

1999 年 10 月，RoboCup 中国分会在重庆组织了中国第一次 RoboCup 仿真机器人足球赛。随后，一年一度的全国比赛也在国内开展起来，比赛规模也不断壮大。

与此同时，各高校也开展了相关的活动以促进 RoboCup 在中国的发展。中国科技大学在开设 RoboCup 研讨班的同时还举办了许多讲座以及校内、省内比赛，受到了广泛好评。

2001 年 6 月 21 日，中国自动化学会机器人竞赛工作委员会成立大会在清华大学隆重召开，该委员会将负责统一协调、组织全国的机器人竞赛活动，863 计划还提供了专项基金予以资助，标志着我国机器人竞赛事业进入了一个崭新阶段。

该委员会为中国科学技术协会和中国自动化学会的组成部分，其宗旨是通过机器人比赛（主要包括 RoboCup 和 FIRA 两类机器人足球比赛），让更多的群众尤其是青少年朋友了解机器人、喜爱机器人，普及现代科学知识，为我国的机器人事业培养更多的优秀人才，推动自动化与机器人技术的发展与创新，为我国的快速持续发展贡献力量。

1.1.3 RoboCup 应用技术与相关发展介绍

机器人足球是一项科技、教育和产业化三位一体的事业，具有极其重大、极其深远的社会和产业意义。从技术的角度看，机器人足球的直接应用领域是各种智能机器人，包括家用机器人、工业机器人和服务业机器人等等；最大的应用领域是网络信息处理，包括电子商务中的一些关键技术。简单地说，只要机器人能“踢”足球，就能做很多其他事情。随着研究的深入，多主体系统和机器人足球的应用成果将不断地涌现出来。

由于 RoboCup 中涉及到的许多研究领域都是目前研究与应用中遇到的关键问题，因此可以很容易的将 RoboCup 的一些研究成果转化到实际的应用中。例如，最典型的应用有以下这些：

- 搜索与救援

搜索与救援领域有以下一些特点：如在执行搜索与救援任务时，一般是分成几个小分队，而每个小分队往往只能得到部分信息，而且有时还是错误的信息；环境是动态改变的，

往往很难做出准确的判断；有时任务是在敌对环境中执行，随时都有可能会有敌人；几个小分队之间需要有很好的协作；在不同的情况下，有时需要改变任务的优先级，随时调整策略；需要满足一些约束条件，如将被救者拉出来，同时又不能伤害他们。这些特点与 RoboCup 有一定的相似，因此，在 RoboCup 中的研究成果就可以用于这个领域。事实上，RoboCup-Rescue 就是专门负责这方面的问题。

- 太空探险

在太空探险时，一般都需要有自治的系统，能够根据环境的变化做出自己的判断，而不需要研究人员直接控制。在探险过程中，可能会有一些运动的阻碍物，必须要能够主动的进行躲避。另外，在遇到某些特定情形时，也会要求改变任务的优先级，调整策略以获得最佳效果。

- 办公室机器人系统

用于在办公室中完成一些日常事务的机器人或机器人小组，这些日常事务一般包括收集废弃物，清理办公室，传递某些文件或物件物品等。由于办公室的环境具有一定的复杂性，而且由于经常有人员走动，或者是办公室重新布置了，使这个环境也具有动态性。另外，由于每个机器人都只能有办公室的部分信息，为了更好的完成任务，他们必须进行有效的协作。从这些可以看出，这又是一个类似 RoboCup 的领域，当然可以采用一些 RoboCup 中的技术。事实上，在前几年有许多研究人员从事办公室机器人系统的研究，而最近改为进行 RoboCup 方面的研究就是因为这一点。

- 其它多主体系统

这是一个比较大的类别，RoboCup 中的一个球队可以认为就是一个多主体系统，而且是一个比较典型的多主体系统。它具备了多主体系统的许多特点，因此在 RoboCup 中的研究成果可以应用到许多多主体系统中。其实，上面的三个例子就是三个不同方面的多主体系统，从中可以看出 RoboCup 技术的普遍性，除了这些例子外，还有许多可以应用 RoboCup 技术的领域，如空战模拟、信息代理、虚拟现实、虚拟企业等等。

- 面向中小学生的普及型教学软件和机器人产品。

在 2000 年的世界杯赛上，已经设立了初级组比赛，来自世界各国的几十支由中小學生组成的球队参加了“舞蹈组”、“一对一”和“二对二”三种类型的比赛。在 2001 年的世界杯赛上，将举行更大规模的初级组比赛。另外，美国、日本、澳大利亚和加拿大等国已经举办或即将举办全国性的初级组比赛。相应的各种产品已经开发出来，在国外市场上已经很流行，中小學生可以自由组装自己的机器人，而且其价位即使对国内消费者也是可接受的。初级组比赛对于培养中小學生的科技素养、创新能力和实践能力具有十分重要的作用。对于改变我国中小学教育的落后状况，具有重大的现实意义。我国也举办了很多这类比赛，但目前还是面向大学，目前我国有不少中小学校也积极组织学生参与。相信这类比赛及相关的教学必将得到社会各界的普遍欢迎。可以预计，这将是一个非常重大的市场，目前国内已经有相关产品研发出来。

- 面向娱乐的游戏、电子宠物产品

电子竞技已经在我国正式被列为体育项目，随之游戏产业也必将越来越受到重视。RoboCup 基础组织挑战中就包括提供一个带有高质量 3D 图形能力的网络足球仿真平台以带来更好的视觉效果和更丰富的功能，这本身就是一个很有吸引力的游戏；另一方面还有一些研究机构将 RoboCup 的思想运用到对战游戏中，制作过一个机器人对战游戏 gamebots，并举办过几届比赛。可以看到 RoboCup 研究的相关技术必将对游戏中的智能设计提供更多的技术支持，也必将带动游戏向智能化发展。

电子宠物是另外一种研究成果转化为产品的方向，有腿足球机器人既是通向人形机器人的必要中间环节，又是机器宠物的原型研究。据报道，SONY 公司在其有腿足球机器人基础

上开发的机器狗已售出近 4 万台。可以预计，随着有腿足球机器人研究的不断进展，各种新型的机器宠物将不断开发出来并获得越来越大的效益。类似地，随着其他类型的足球机器人及球队在技术方面的逐步成熟，相关的产品化和经济效益亦将水到渠成。

1.2 仿真机器人足球

1.2.1 人类足球比赛的逼真模拟

仿真机器人足球赛利用计算机模拟机器人进行足球比赛，由 RoboCup 仿真平台开发小组提供一个标准比赛软件平台，平台设计充分体现了控制、通讯、传感和人体机能等方面的实际限制，使仿真球队程序易于转化为硬件球队的控制软件。由于避免了现实物理环境和当前机器人制造技术的限制，仿真机器人足球主要把研究重点放在球队的高级功能的研究上，包括动态不确定环境中的多主体合作、实时推理—规划—决策、机器学习和策略获取等当前人工智能的热点问题。也正是由于摆脱了硬件限制，仿真组比赛比较容易实现，成为 RoboCup 比赛中历史最老、参赛队最多的一个项目，研究步伐也快于其他项目。

仿真组比赛平台模拟的是一个二维场地，一个动态、实时、不确定的环境，双方各 11 个球员，每个球员由一个独立的程序控制，平台提供给队员的只是一些很简单的动作，如踢球、加速、转身等，程序必需合理地组合这些动作，形成更“智能”的行为。比赛上下半场各 5 分钟，每个模拟周期只有 100ms，要求程序的实时性非常高，也使得过去的大部分研究成果都很难直接运用。

经过这些年的努力，在一些关键问题上已取得一些成果，为了更好的促进研究，仿真平台又引入了教练智能体，可以从全局把握比赛，更好的实现在线适应、对手建模等人工智能领域关键问题，比赛也增加了教练比赛，让各队的教练智能体按照标准的教练语言控制各种不同类型的球队进行比赛。教练可以从过去球队比赛的数据以及比赛中获得的数据进行分析，选择适合该球队的战术方针。

为了使仿真研究成果最终能够比较好的运用的实物机器人中，今年仿真组开发了一个新的三维比赛平台，并逐步将比赛从二维向三维环境过渡。目前三维比赛平台还只是在试验阶段，模型还有许多不完善的地方，球员只是被模拟为一个弹性小球，可以获得全场信息，但有噪声。但今年第一次开设此比赛项目就有 15 支球队参加了正式比赛。

另外，由于比赛没有实际的机器人比赛现场，观众要通过一个显示界面程序在屏幕上观看比赛情况，看起来有点像电脑游戏。为了提高比赛的观赏性，这就需要有一个更好的可视化界面展现比赛的魅力，因此又增设了 3D 可视化界面比赛来促进这套程序的开发，比赛从界面的多媒体表现（三维生动显示、实时语音解说等）以及界面程序对比赛的综合分析能力两个方面进行评价。

1.2.2 人工智能新的标准问题

在第 15 届国际人工智能联合大会上，由 Kitano, Veloso 和 Tambe 等来自美、日、瑞典的 9 位国际著名或知名学者联合发表重要论文“The RoboCup synthetic agent challenge 97”，系统阐述了机器人足球的研究意义、目标、阶段设想、近期主要内容和评价原则。概括的说，过去 50 年中人工智能研究的主要问题是“单主体静态可预测环境中的问题求解”，其标准

问题是国际象棋人—机对抗赛；未来 50 年中，人工智能的主要问题是“多主体动态不可预测环境中的问题求解”，其标准问题是足球的机—机对抗赛和人—机对抗赛。从科学研究的观点看，无论是现实世界中的智能机器人或机器人团队（如家用机器人和军用机器人团队），还是网络空间中的软件智能体（如用于网络计算和电子商务的各种自主软件以及它们组成的“联盟”），都可以抽象为具有自主性、社会性、反应性和能动性的“智能体”（agents）。由这些智能体以及相关的人构成的多主体系统（multi-agent systems），是未来物理和信息世界的一个缩影。其基本问题是主体（包括人）之间的协调，可细分为智能体设计、多主体体系结构、合作和通讯、自动推理、规划、机器学习与知识获取、认识建模、系统生态和进化等一系列专题。这些专题有的是新提出的（如“合作”），有的是过去未能彻底解决并在新的条件下更加复杂化的（如机器学习）。这些问题不解决，未来社会所需的一些关键性技术就无法得到。值得注意的是，上述一系列问题中的大多数都在机器人足球中得到了集中的体现。在这个意义下，将机器人足球作为未来人工智能和机器人学的标准问题是十分恰当的；而这一研究意义之深远重大，也是不言而喻的。

RoboCup 被看作是一个标准问题，可以用来评价各种不同的理论、算法和体系结构。计算机国际象棋是一个典型的标准问题。各种搜索算法可以在这个领域中评价和发展。计算机国际象棋作为一个标准问题的成功，主要原因之一是清楚的定义了对研究进展的评价，即可以用系统的棋力来评价。同样，RoboCup 也是可以通过实际的比赛对研究进展作出客观清楚的评价。随着深蓝的成功，按照正式规则击败人类的国际象棋特级大师卡斯帕罗夫（Garry Kasparov），计算机国际象棋的挑战已经画上了圆满的句号。然而国际象棋只是完成了一个初级的目标，人工智能的发展需要一个新的挑战，一个必须能够产生一系列发展下一代工业所需技术的挑战。众多研究工作者把搜寻的目光锁定在 RoboCup。机器人足球赛与国际象棋人机对抗赛所体现出的研究背景区别主要有以下几个方面：

(1)静态环境和动态环境。国际象棋赛中，“深蓝”所处的环境是静态的——当轮到它走而它还未走时，即在它“思考”下一步棋的过程中，环境不发生变化。相反，足球赛中每一时刻场上形势都在变化。

(2)行动效果和环境的可预测与不可预测。国际象棋赛中，从一个给定的棋局出发可以达到的棋局是确定的和有限的，理论上可以穷举一切可能结果并从中选择最佳方案。而对足球赛来说，理论上不可穷举；通过离散化实现近似的穷举也存在理论上和技术上难以克服的困难。

(3)个体与团体。“深蓝”至多只相当于一个人。而机器人足球队中的每一个队员相当于一个人，它们之间必须合作和协调，因而出现个人（队员）和团体（球队）在目标、知识、计划和行动等方面既有区别又密切相关的复杂情形。这种复杂性是深蓝没有的。

以上三方面的差别使得每个足球机器人必须是一个能自主行动且与队友合作的智能体，而一支球队则是一个内部协调的多主体系统。相反，“深蓝”本质上只是一个能在几分钟内分析 600 亿个棋局的搜索程序。

表 1.2 列出了计算机国际象棋和 RoboCup 的一些关键区别。

表 1.2 计算机国际象棋与 RoboCup 的比较

	环境	状态改变	获取信息	环境确定性	传感器信息	控制方式
国际象棋	静态	回合制	完全	确定	符号式	集中
RoboCup	动态	实时	不完全	不确定	非符号式	分布

1.2.3 研究重点

RoboCup 挑战为智能主体的研究提供了一系列挑战计划，同时还提供了一个动态、实时、多主体平台通过比赛来检验研究成果。对于想设计一个参加 RoboCup 球队的研究工作者来说，根本问题是设计一个多主体系统，能够进行实时的反应，表现出目标驱动的理性行为。而目标与环境都是在动态地变化并且是实时的。由于足球比赛的状态空间极大，不可能用手工的方法来编码所有可能的情形和智能体的行为，因此使智能体能学习如何有策略的进行比赛变得极为重要。在这个方面的挑战包括以下的研究问题：

(1)在多主体合作及对抗环境中的机器学习，(2)多主体体系结构，为了团队合作而进行实时的多主体规划和规划执行，和(3)对手建模。

因此，以下三个挑战也就成为当前的研究重点——学习挑战、团队合作挑战和对手建模挑战。

1. RoboCup 学习挑战

RoboCup 学习挑战的目的是寻求全面综合的学习方案，它能用于需要适应环境的多主体系统的学习；并能用标准任务来评价所提供方法的优缺点。

学习是智能系统的重要方面，在 RoboCup 学习挑战中，任务是为了一组智能体创建一种学习和训练的方法。这个领域的学习可以分为以下几类：

- 单个智能体的离线技术学习（如拦截球、踢球）
- 智能体团队的离线合作学习（如传接球）
- 在线技术和合作学习（如适应性跑位）
- 在线对抗学习（如对预测的对手行为采取相应对策）

之所以区分离线和在线学习，是因为比赛一般只持续 10 分钟，所以在线技术，特别是学习某场特定比赛中的概念时，必须很快的产生结果。例如，如果一个队要学习针对某个对手调整它的行为，就必须要在比赛结束前提高自己的性能，这对学习算法的要求是很高的。因此，根据 RoboCup 中具体问题的特点可以选择不同的学习方法，如球员拦截、踢球行为比较固定，受比赛变化影响不大，所以可以采取离线学习；而球员适应性跑位，需随对手的策略实时调整，就只能通过在线学习来达到目的。这种学习特点可以应用到很多具有学习能力的多主体系统中。

2. RoboCup 团队合作挑战

RoboCup 团队合作挑战提出了在动态对抗环境中多主体团队合作中的实时规划、重规划、规划执行等问题。

在类似足球这样复杂、动态的多主体领域中需要极为灵活的协调和通讯来克服其中的不确定性。例如，团队目标的动态变化，团队成员因意外不能完成应尽的责任。不幸的是，以当前的智能体体系结构实现多主体系统经常依赖于预先规划的、领域相关的协调方法，不可能提供这种灵活性。所以建立一个适合团队实时规划和规划执行的体系结构就成为目前 RoboCup 团队合作挑战中的一个主要课题，另外将这样一个体系结构一般化以适用于其他非 RoboCup 应用也是一个重要环节。

团队合作挑战中基本的体系结构问题就是要建立这样一种体系结构，它可以支持团队行为的规划以及更重要的团队规划的执行。这种规划和规划执行可以通过一个两层的体系结构来完成，但是整个系统必须能够实时运作。在 RoboCup 仿真比赛中，传感数据每隔 75-300 毫秒到达，每 100 毫秒可以发送一条动作命令。环境以毫秒量级改变，所以规划、重规划和规划执行必须是实时完成的。

这里关键研究任务大致可列出以下几条：

- 对多主体对抗比赛中意外事件的规划
- 规划的分解和合并
- 团队规划的执行

3. RoboCup 对手建模挑战

智能体建模——为其它智能体的目标、规划、知识、能力或情感建模和推理——是多主体交互的关键问题。RoboCup 对手建模挑战研究在动态多主体领域内对一组对手建模。

RoboCup 中的建模问题可以分为三个部分：

- 在线追踪

通过观察对手动作对单个对手的目标和意图进行实时、动态的追踪。队员可以使用这种追踪来预测对手的行为并作出适当的反应。这种在线追踪还可以用于防止受骗。在线追踪可以为在线规划器或在线学习算法提供输入数据。

- 在线策略识别

球队的"教练"智能体可以从场外观察一场比赛，理解对方球队采用的高层策略。与在线追踪相比，教练可以进行更高层次的抽象分析，而不会有实时处理的压力，他的分析可以更详细。然后，教练智能体可以为他的队员提供数据来改变队形或使用某种策略。

- 离线审阅

"专家"智能体可以在比赛后观察球队的表现，来分析球队的优点和弱点，还可以提供专家评论。这些专家可以在人类足球比赛的数据库中训练得到。

1.3 本书安排

本书覆盖了从 RoboCup 基础知识到相关领域技术分析丰富的内容，既适合广大机器人足球初学者入门学习，又可作为已经开始相关开发设计者的进阶参考书，同时也可作为从事仿真机器人足球设计工作人员的技术手册。全书大体可以分为入门、进阶、专题三个部分，读者可以根据自己需要选择阅读。

- 入门篇（1-5 章）

系统介绍仿真机器人足球基本原理和简单实现主要面向初学者的入门学习。

第 2 章从仿真平台全貌入手，让读者了解仿真平台的设计思想，整个比赛的流程，使读者对整个仿真平台运作有个大体的认识，可以自行安装使用此平台。

第 3 章站在初学者角度，从平台基础入手，快速实现一个最简单球员程序的制作。

第 4 章在初学者对仿真平台已有了感性认识的前提下，从平台各部分设计原理出发，详细阐述了球员程序各基本部分的设计方法，使读者经过学习实践后达到可以自行设计比较完整的球员程序的目的。

第 5 章则从设计整个球队的角度出发，简单介绍了设计一支完整球队所涉及的问题及一些简单解决思想。

- 进阶篇（6-9 章）

从仿真机器人足球平台中抽取典型问题，结合典型球队的设计思想，分析各自特点、实现。适合具有一定基础的读者的进阶学习。

第 6 章通过对现有典型球队的体系结构分析比较，使读者了解目前球队总体结构的

主要设计思想

第 7-9 章分别从“信息获取与维护”，“球员复杂行为”和“团队复杂行为”三个球队设计中的重要问题，系统介绍、分析了从仿真机器人足球研究以来，实现的主要设计算法、思想。

● 高级专题（10-12 章）

根据目前仿真机器人足球研究进展，从中提出若干个重点专题进行介绍。可供相关领域研究人员参考。

第 10 章总结目前 RoboCup 中涉及的主要学习方法。

第 11 章系统介绍了仿真机器人足球教练智能体

第 12 章介绍目前仿真机器人足球领域新兴的 3D 比赛项目

批注 [Eagle2]: 网站资料索引应该怎么添加? 大段翻译原文应该怎么添加?

参考文献

<http://www.robocup.org/> and <http://www.robocupjunior.org/>

<http://www.rcccaa.org/>

<http://www.planetunreal.com/gamebots/>

[Mackworth. 1993] A. K. Mackworth. On seeing robots. In A. Basu and X. Li, editors, Computer Vision: Systems, Theory, and Applications, pages 1--13. World Scientific Press, Singapore, 1993.

[Kitano et al. 1995] Hiroaki Kitano et al. RoboCup in *Proc. Of IJCAI-95 Workshop on Entertainment and AI/Alife*, Montreal, 1995.

[Kitano et al. 1997a] Hiroaki Kitano et al. RoboCup: The Robot World Cup Initiative in Proc. of The First International Conference on Autonomous Agent (Agents-97)), Marina del Ray, The ACM Press, 1997.

[Kitano et al. 1997b] Hiroaki Kitano et al. The RoboCup Synthetic Agent Challenge 97. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, San Francisco, CA, 1997. Morgan Kaufman.

[Coradeschi and Malec 1998] Silvia Coradeschi and Jacek Malec. The Use of RoboCup (soccer simulation) for an AI programming course. in the Journal of Robotic Society of Japan, special issue in Robotics and Education, May 1998.

[Chen et al. 2003] Mao Chen et al. RoboCup Soccer Server. The RoboCup Federation, February 2003. Manual for Soccer Server Version 7.07 and later.

第二章、2D比赛平台

机器人足球仿真组比赛是在一个标准的计算机环境内进行的,比赛规则基本上与国际足球联合会的比赛规则一致。比赛的方式是由 RoboCup 委员会提供标准的机器人足球仿真软件平台,称作 soccer server[Noda et al. 1997]。这是一个真实足球的仿真系统。该平台支持多个虚拟球员在一个动态、不确定的多主体环境中实时地进行合作对抗。由于仿真平台摆脱了控制机器人所要研究的诸如物体识别、通讯、硬件设计等问题的限制,使得从事仿真机器人足球的研究者可以更好地把注意力集中合作、学习等在更高层的技术上。

从 1995 年完成第一个 soccer server 版本至今,已举办了七届正式世界杯比赛和一次世界杯预备赛。1996 年在日本大阪举行的世界杯预备赛[Kitano 1996]只有 7 支球队参赛,第二年在日本东京举办了第一届正式比赛[Kitano 1998]有 29 支球队参加了比赛。随后比赛规模不断扩大,每次比赛前都要有一个预选工作,从全世界报名参赛的球队中综合考虑比赛成绩和研究特点选出 40 支左右的球队参加正式比赛。另外,作为一项评测,起初每年参赛的冠军都必须不做更改地参加下一年的比赛,但后来由于 server 版本改动太大,失去可比性,从 2002 年开始就不再要求了。

本章主要将 RoboCup 仿真平台各个部分做一个简单的介绍,关于这些部分的细节可以查阅本书的后续章节。另外本章还包括获取 RoboCup 仿真软件包和安装使用这个软件的所有必要的信息,使用户可以快速使用该软件包搭建自己的试验平台。

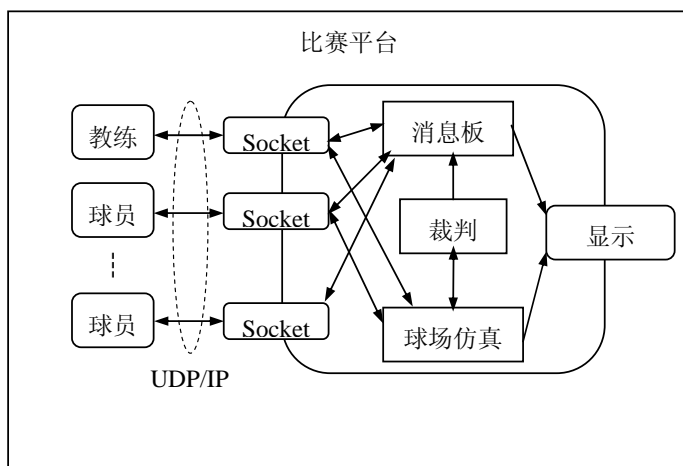
2.1 平台结构特点

机器人足球仿真比赛平台是一套系统能够让由不同语言编写的自主球员程序进行足球比赛。比赛的执行采用的是服务器/客户端(server/client)模式:服务器端程序 Soccer Server 提供了一个虚拟场地并且模拟包括球和球员在内的所有物体移动;每个客户端程序 Soccer Client 相当于一个球员的大脑,控制场上该球员的移动。服务器端和客户端之间都是通过 UDP/IP 协议进行信息交互的,也就是说,开发者可以使用任何支持 UDP/IP 协议的程序设计语言来设计球队程序。通过 UDP/IP 协议,客户端程序可以发送指令去控制相应的场上球员,而服务器端按照规则给每个客户端发送它所能获得的信息。每个客户端程序只允许控制一名球员。所以每队必须同时运行与比赛球员数目相等的客户端程序。客户端之间的通讯必须通过服务器端根据规则来进行转发,任何不经服务器客户端直接联系的行为都是违反规则的。当一场比赛开始时,双方 11 个独立的球员程序连接到比赛平台上场比赛。每个队的目标就是将球踢进对方球门同时阻止球进入自己的球门。

2.1.1 比赛平台(Soccer server)

比赛平台包括两个主要程序: soccerserver 和 soccermonitor。Soccerserver 作为一个服务器程序模拟所有球员和球的移动、和球员通讯以及根据比赛规则控制比赛进程。Soccermonitor 是一个程序将从 soccerserver 那里获得场上信息显示到一个虚拟场地上,可以有許多 soccermonitor 连接到 soccerserver 上。

比赛平台提供了一个虚拟的足球场地,由客户端程序控制的队员可以在场上跑动,踢球。Soccerserver 主要由球场仿真模块、裁判模块和消息板模块三个部分组成(图 2.1)。



• 图 2.1 比赛平台结构图

球场仿真模块计算球场上对象的运动，检测他们之间的碰撞。球场上的对象包括每队各 11 名队员、球、球门、标记及标志线等。其中球和球员都具有大小、位置、速度、加速度等属性，球员则还有方向、耐力等属性。球员与球的属性每个周期末计算更新一次，计算的依据是动力学定律。如果球员与球员或球之间发生重叠，则作碰撞处理。

裁判模块根据规则来控制比赛的进程。由于仿真比赛环境具有动态、实时、不确定、多主体对抗等特点，比赛不可能按照事先的设计按部就班地进行，还需要在比赛中有一个“智能”裁判。目前这个内嵌的人工裁判只是部分实现，可以检测一些简单的形势，如进球、界外球、越位等。然而，还是有一些很难检测的状态，如双方对峙，谁都不踢球，陷入僵局，这就需要一个人工裁判。所有的参赛队都必须遵守一个“绅士协定”包括不能利用系统漏洞等有碍比赛公平的做法。

消息板模块负责客户端之间的通讯。每个客户端程序通过 UDP 的 socket 来连接 server。同样，通过 socket，客户端程序可以发送命令来控制球员，也可以接收球员的感知信息。

Soccerserver 采用的离散化模式运行，即所有程序运行都是以仿真周期为单位在每个仿真周期（simulation_step，缺省为 100 毫秒）结束前，server 收集所有球员程序的行为请求，直到每个周期末才统一执行并更新场上信息，在每个周期的开始 server 根据各个球员的状态（包括可视范围、获得时间等）发送相应的已更新的场上信息，体现了球员感知信息和行动的异步性。如果一个球员在一个周期内发送了多于一条的独立行为请求，server 将随机选择一个执行。因此，球员为了保证执行自己的真实意图，每周期就只能发送一条独立行为请求；另一方面，如果球员在一个周期内没有发送行为请求，它将失去该周期的行动机会，对于这样一个实时对抗的环境这无疑是很不利的。

在 soccerserver 平台上比赛时，所有仿真比赛场景都可以通过一个可视化程序 Soccer monitor 显示在电脑屏幕上。它通过一个特殊的端口（缺省为 6000）直接和比赛平台连接，获得比一般球员程序更全面、更准确的信息，使得用户可以生动地看到比赛的整个过程，并且可以控制比赛的进程。目前的球场和球场上的对象都是二维的，任何对象都没有高度的概念。场上每个队员用一个圆圈表示，圆圈分为两半，亮的一面表示球员身体的朝向，另一半通过颜色的深浅变化表示球员的体力变化，另外从圆心引出一条线段表示球员脖子的朝向。球用一个实心原点或贴图表示。目前官方提供了两种不同的 monitor：rcssmonitor 和 rcssmonitor_classic。两个 monitor 都可以显示诸如比分、双方队名、场上所有球员和球的位置。它们也提供了和 server 简单的接口，如当比赛两边队员都上场后，monitor 上的“Kick-Off”

按钮就可以使人为裁判向 server 发送命令开始比赛。

Rcssmonitor 是从一下几个方面扩展了经典 monitor——rcssmonitor_classic 的功能：

- 可以放大场上的局部区域，这对于调试特别有用。
- 随时都可以显示所有队员和球的位置和速度。
- 多样信息都可以显示在 monitor 上，如某个队员的视角，体力或球员类型等。

其实进行一场比赛可以不需要 monitor，它只是方便观众了解比赛情况；当然如果需要的话，可以同时连接许多 monitor 到服务器端。

另外为了方便的重现比赛实况，平台还提供了比赛录像播放器（Logplayer）。Logplayer 可以被看做是一个录像机，用于重放比赛。当运行 Soccerserver 时，通过在启动配置文件中设置一些参数可以将比赛的所有数据以一种特定的格式存储在电脑硬盘中（这就好比按下了录像机的录制键）。然后，rcsslogplayer 程序捆绑一个 monitor 就可以用来反复多次地播放比赛录像。这对于分析球队、发现一个球队的优缺点是非常有用的。类似于一个录像机，logplayer 同样有播放、停止、快进、回卷等按钮，logplayer 也可以将比赛录像跳到指定的时间，如进球时。另外 Logplayer 还可以编辑录像，比如剪辑比赛中有趣的一段存为另一个录像文件。

2.1.2 球员客户端（Soccer Client）

一个球员客户端程序通过 UDP 接口连接到比赛平台。通过这个接口，客户端程序可以发送命令来控制场上的一个球员行动以及接收到这名球员的感知信息。换句话说，一个球员客户端就是球员的大脑：从服务器端接收到感知信息，并且发送命令到服务器端。

每个球员程序都是独立的进程，通过一个特定的端口和比赛平台连接。当一个球员程序和比赛平台建立好连接后，所有通讯信息都通过这个端口传输。一个球队最多可以连接 12 名队员包括 11 名球员（其中一个守门员）和一个场上教练。这些球员程序向比赛平台发送请求执行相应行为（如踢球、转身、跑步等），平台分析处理这些请求，相应的更新场上比赛状态。另一方面，比赛平台给所有队员提供他们可以感知到的信息（教练是一个特殊的队员，可以获得比普通球员更多的信息），如球员可以看到的视觉信息、球员自身的状态信息等。由于比赛平台实际上是一种以离散时间片（或称为周期）为时间单位工作的实时系统，球员程序必须在每个指定仿真周期内及时做出决策并及时将请求发送给比赛平台，否则将错过执行动作的机会。这就要求球员决策要有比较高的实时性。

正式比赛中每个客户端程序只能控制一个场上队员。因此，要组成一支球队就需要同样数量的程序分别控制每个队员。球员之间的通讯必须经过比赛平台按照 say 和 hear 命令协议（详见 3.2、3.3 节）执行，而且通讯环境具有单信道、窄带宽等特点。仿真比赛平台的一个目的就是评估多主体系统，智能体之间的高效通讯也是其中的一个判别标准。

为了尽可能模拟现实环境，比赛平台还加了很多限制，如 turn、kick、dash 等命令每周期至多只能执行其中的一个；用 say 命令说话只有一定区域内的球员能听到，而且频率不能太高。每个队员都有一定的视野范围，每次只能获得局部信息，即包括可视范围内的对象信息，而且是有随机噪音的；每个球员都有自己的体力值，随跑动衰减，每周期可以自动恢复一些，这样就限制球员要注意调整跑动速度，合理分配体力，也更符合现实。另外，为了反映出实际比赛中球以及球员运动的不确定性，Server 还引入了风及噪声的干扰及对行为参数的干扰，使比赛更趋于真实，正如现实比赛很难无风和噪声的干扰，快速跑动中的队员不太可能急转弯等等，但这同时也增大了准确建模的难度（有关建模的讨论在后续章节会陆续展开）。

在官方提供的仿真机器人足球源码包中还附带了一个演示程序叫 rcssclient，它只能实现

非常基本的功能，包括 server 建立连接，从 monitor 上即可看到球员正常上场；通过终端用户可以直接看到 client 从 server 接收到的消息，并且可以通过终端输入向 server 发送命令，直接控制场上球员。通过这个程序，可以使用户更好的了解整个仿真平台的运作方式、过程。

除了球员 client 外，还可以有教练 client (Coach) 连接到 Soccer Server 上。教练是一种有特权的 client，它不直接参与踢球，但可以获得比普通球员更多的信息，可以在场外通过特殊的渠道指挥球员，帮助它们更好地比赛。教练分为在线(online)和离线(offline)两种，在线教练可以在正规比赛中使用，它在比赛过程中能得到全局无噪声的信息，可以进行高层次的分析，并通过规则允许的方式给场上球员提供更多的信息和建议；离线教练现在更多地称为训练器 (trainer)，它比在线教练有更多的控制比赛的能力，不能在正式比赛时使用，主要用于球队程序的训练、调试。使用训练器可以改变比赛状态、给场上球员广播各种信息、移动球和球员到任何位置，并可以赋给它们一定的速度、方向等，这些在进行球员程序的自动学习和自动管理比赛起到重要作用。本书 11 章会详细介绍教练的相关内容。

2.1.3 平台特点

通过上面的介绍，可以看到，作为一个试验平台，这套仿真比赛平台提供了一个很好的全分布的、包括合作与对抗的多主体实时环境，非常有挑战性。其具体特点如下：

- **分布式多主体团队合作和对抗** 所有客户端程序分别控制场上的一名球员或教练，自主决策，分布运行，队友之间有合作，对手之间有对抗
- **动态、实时、不确定环境** 在服务器端，整个系统按照 100 毫秒的周期运转，所有球员都必须按照这个周期运行，意味着球员的所有决策都必须实时完成，由于多主体的存在，环境在动态的转变，无法预知（这也被称为“不透明转换” [Stone 1998]）。
- **感知和行为异步** 由于比赛时间以周期为单位离散，感知和行为就无法同步，所以光靠传统人工智能方法使用感知来激发行动是远远不够的
- **球员能力受限** 场上所有球员能力都是参照真实球员有所限制的，如体力、加速度、最大速度、惯性等
- **视觉受限** 每个球员的视觉都是局部的，收到球员视角、和视距的限制，也就是说球员在任何时刻都只能获得一部分球场上的信息。这就给球员正确分析场上形势，进而产生决策带来了困难
- **通讯受限** 球员之间的通讯环境具有单信道、窄带宽等特点，即每队球员公用一条信道，每个球员一个周期内只能“听”到队友一条消息，而且信道容量很有限（缺省为 10 字节）。这样，现有的一些团队合作理论就很难直接应用，因为目前大部分合作理论前提都是要求通讯是及时的、完全的
- **多噪声源** 为了真实模拟实际比赛，仿真世界里球员感知和动作都带有噪声，使得球员既无法精确地感知世界也不能完全按照它的意图影响世界
- **连接不可靠** 平台网络连接使用UDP/IP，不确保所有信息的正确及时，在网络繁忙时一些信息甚至会丢失，这也体现了比赛环境的不确定性，球员程序必须能够适应这一环境。

RoboCup 仿真比赛平台充分体现了人类足球的特点，也集中许多人工智能等领域关注的重点问题。用户可以在不同操作系统下，使用不同计算机编程语言，运用包括数学建模、搜索推理、数据挖掘、机器学习、动态规划等各种知识、技术来制造球队，并通过该平台进行实践、检验。很好地推动了相关学科理论的研究。

批注 [Eagle2]: 是否有这方面的文章说明？

2.2 比赛规则

在一个比赛中，通过比赛平台自带的自动裁判或者通过人为裁判来做一些规则限制，以确保比赛的顺利进行。下面将具体介绍这些规则如何作用，影响比赛。

2.2.1 自动裁判

- 中场开球 (Kick-Off)

在一个开球前，无论是半场开始前还是进球后，所有的球员都必须在自己的半场。为了确保这点，在进球后，裁判特别暂停比赛 5 秒，球员程序可以通过使用 `move` 指令将自己直接移动到自己一方的某个位置而不用浪费时间和体力跑回来。如果某个球员在进入开球状态之后仍然在对方半场，裁判将该球员移到所属半场的随机一个位置。

- 进球 (Goal)

当一队进球得分时，裁判要执行一系列任务。首先，它要通过广播一条消息向所有球员宣布进球。它还要更新场上比分，移动球到赛场中心，并且将 `play_mode`（程序中用以标识当前比赛状态）置为 `kick_off_left` 或 `kick_off_right`（左方开球或者右方开球）。最后，如“开球”部分介绍，它要将比赛暂停 5 秒钟等待球员移回到各自的半场。

- 守门员发球 (Goalie Free Kick)

当守门员扑住球后，即可直接发球。由于比赛场地是二维的，在对方干扰下，守门员就很难把球踢出到一个安全区域去，因此规则特别允许守门员发球可先使用 `move` 指令移动到禁区内任意对发球有利的位置（球也会随守门员一起移动），再迅速将球踢出。为了防止守门员滥用 `move` 指令，规则限制守门员一次发球最多只允许移动两次（`goalie_max_moves`），否则 `move` 指令无效。

- 出界 (Out of Field)

当球滚到了界外，裁判将球根据出的界不同移到一个合适的位置，包括边线、角球区和球门区，相应的将 `play_mode` 置为 `kick_in`（界外球）、`corner_kick`（角球）或 `goal_kick`（球门球）。在发角球时，裁判将球置于场地相应一角的坐标为（1 米，1 米）的位置上。

- 越位 (Offside)

一个球员在满足以下情况时被判越位：

1. 在对方半场，
2. 至少比两个防守球员更靠近对方球门，
3. 比球更靠近对方球门，
4. 距离球小于 2.5 米（这个值可以通过修改比赛平台参数 `offside_active_area_size` 调整）。

- 回传 (Backpasses)

就像在真实足球比赛中一样，守门员不允许扑队友传回来的球。一旦发生，裁判将判 `back_pass_l`（左方回传）或 `back_pass_r`（右方回传），然后让对方发一个任意球。当这种球发生在禁区内时，球将在最靠近守门员扑球位置的一个禁区角发。要注意的是，如果守门员不扑队友传过来的球就不算犯规。

- 发球违例 (Free Kick Faults)

当发一个任意球、角球、守门员发球或界外球时，发球者不允许传球给自己。如果发球者在踢出球后紧接着又踢球，裁判将判 `free_kick_fault_l`（左方发球违例）或 `free_kick_fault_r`（右方发球违例），并且对手既可获得一个任意球。

由于球员很多时候为了将球踢到期望的速度都不得不连续踢很多脚球,所以发球违例只有当满足以下情况才会判:

- 1、发球者再次踢到球
- 2、该球员在两次踢球间移动过(使用过 `dash` 指令)

所以诸如 `kick-kick-dash` 或 `kick_turn-kick` 这样的指令序列是完全合法的。而指令序列 `kick-dash-kick` 将被判为发球违例。

- 扑球违例 (Catch Fault)

当守门员扑球时球的位置在禁区外,裁判将判扑球违例 (`catch_fault_l` 或 `catch_fault_r`)。由对方在扑球位置发间接任意球 (`indirect_free_kick_l` 或 `indirect_free_kick_r`)。

需注意的是判罚扑球违例是以球的位置为准,由于守门员扑球是沿扑球方向 2 米长、1 米宽的矩形区域,所以即使守门员在禁区内扑球,球却有可能在禁区外,这时仍会被判扑球违例。

- 球员清除 (Player Clearance)

当 `play_mode` 为 `kick_off`, `free_kick` (任意球) 或 `corner_kick` 时,裁判将移除以球为中心的一个圆内的所有防守球员。这个圆的半径是比赛平台程序的一个参数,缺省为 9.15 米。这些移除的球员被放在这个圆周上。当 `play_mode` 为 `offside` (越位) 时,所有进攻球员要被移回到非越位位置。这些进攻球员包括处于越位位置和距离球 9.15 米圆内的所有球员。当 `play_mode` 是 `goal_kick` 时,所有的进攻球员将被移除禁区,并且在踢球门球时进攻球员不能再进禁区,只有当球出了禁区后, `play_mode` 才会被改变。

- 比赛状态控制 (Play-Mode Control)

当 `play_mode` 为 `kick_off`, `free_kick`, `kick_in` 或 `corner_kick` 时,裁判在球被踢动后立即将 `play_mode` 置为 `play_on` (继续比赛)。

- 半场 (Half_Time) 和终场 (Time_Up)

裁判当第一个或第二个半场结束的时候将暂停比赛。缺省的半场时间是 3000 周期,相当于 5 分钟。如果整场比赛打平就要开始第二个半场,比赛进入加时赛。加时赛采用“金球制”,或称“突然死亡法”,即加时赛中第一个进球方将赢得比赛。加时赛也分上下半场各 3000 周期,如果加时赛仍没有进球,双方即进入罚球大战 (`penalty mode`)。

2.2.2 人为干预

然而,并非场上所有情况自动裁判都可以处理,像故意阻挡等犯规行为是很难由自动裁判判断出,因为这与球员的意图相关。为了解决这种情况,比赛平台提供了一个接口可以供人为干预。人为裁判可以通过比赛平台的显示工具暂停比赛并让任何一方发任意球。以下是从 RoboCup2000 比赛开始采用的一些指导方针。

- 故意包围球

一旦一方将球包围住,由于球员身体阻挡效果,其他人是无法碰到球的,这将对比赛无法进行下去。

- 故意用过多的球员堵球门

由于比赛环境是二维的,球门宽度有限,所以一旦有过多球员堵门,球就很难射进,这对射门和守门的算法研究都是不利的。

- 在限定周期内没有让球投入比赛

目前这个规则已由自动裁判处理。如果一个球员没有在 `drop_ball_time` 周期

（缺省为 200 周期）内让球投入比赛，裁判将自动执行 `drop_ball`（即将球直接放到场上相应的一个位置，比赛正常进行）。如果一个球队总是不能及时让球投入比赛，人为裁判可以提前执行 `drop_ball`。

- 故意阻挡其他球员的移动
由于比赛平台对球员的碰撞自动处理，使得所有碰撞物体速度都会减慢，这样球员就可以通过碰撞阻止其他球员快速移动，这将使比赛无法正常进行。
- 守门员滥发 `catch` 指令
比赛平台限制守门员连续发扑球命令（`catch_ban_cycle`，缺省为 5 个周期），但是一旦守门员扑球成功，这个计数器就重置了。利用这个漏洞，守门员可以反复踢球、扑球，这样守门员就可以长时间地安全地将球移动到禁区内的任意位置。妨碍了比赛的进程。
- 用大量消息堵塞比赛平台的通讯信道。
一个球员程序不允许在每个仿真周期内发送超过 3 或 4 条命令给比赛平台。如果比赛平台被堵塞，或赛后有要求，将会检查是否有这种滥发行为。
- 不适当的行为
如果观察到某个球员以一种不合适的方式进行比赛妨碍了比赛的正常进行或违反了公平原则，人为裁判将暂停比赛并且给对方一个任意球。

2.3 比赛平台安装使用

RoboCup 仿真组官方主页在 <http://sserver.sf.net/>。该网站包括了 RoboCup 仿真组以及相关的很多有用的信息。开发人员可以从该网站下载到需要的资源。

2.3.1 机器人足球仿真源码包结构说明

官方提供了若干适合不同平台、不同需要的开发包，其中扩展名为 `zip` 的为 Windows 下的 `server` 执行代码包（目前还没有其他模块的 windows 开发包），在 Windows 系统中解压到任何目录既可使用；其他为基于 Linux 的系统，包括执行代码包的 `rpm` 文件，可在 linux 系统下安装使用，另外就是源码包，一个完整的机器人足球仿真源码包包括以下几个部分，当然每个部分也可以独立获得并安装、升级。

- **rcssbase** 供其他各种仿真机器人足球程序包使用的基本代码。
- **rcssserver** 是该仿真源码包的主要部分，执行实际的仿真工作。客户端程序和 `server` 通过 UDP/IP 协议发送命令和接收感知信息。
- **rcsslogplayer** 可以重放 `rcssserver` 录制的比赛录像（*.rcg 文件）。要注意的是 `rcsslogplayer` 只是用于控制录像的回放，显示还是需要通过 `monitor`。
- **rcssmonitor** 和 **rcssmonitorclassic** 通过连接 `rcssserver` 或 `rcsslogplayer` 来显示现场比赛或比赛录像。

2.3.2 安装

这里我们仅以 `rcsoccersim-*.tar.gz` 源码包为例，说明如何安装比赛平台。

首先从网站上下载机器人足球仿真源码包 `rcsoccersim-*.tar.gz`（书中采用目前最新的源

码包 `rcsoccersim-9.3.7.tar.gz`），然后将该源码包在一个 Linux 环境下解压到当前目录中：

```
-> tar zxvf rcsoccersim-9.3.7.tar.gz
```

同时一个目录 `rcsoccersim-*` 会自动会建立。进入该目录：

```
-> cd rcsoccersim-9.3.7
```

用户可以通过阅读 `README` 文件，了解更多的信息：

```
-> more README
```

另外目录下还有一个名为 `COPYING` 的文件包含了允许使用和修改此源码的许可。请务必在使用前先阅读。

```
-> more COPYING
```

根据 `README` 说明就可以进行快速安装：

```
-> ./configure
```

```
-> make
```

通过以上两步就可以产生需要的执行程序：

`rcsoccersim-*/rcssserver/src/rcssserver` 是比赛平台的执行程序，负责全场仿真和与球员程序交互使其能控制场上的虚拟球员。在这个目录中还可以找到一个简单的球员程序例子 `rcssclient`。

要想看到仿真比赛实际情况就需要启动一个仿真比赛的专用显示工具，`rcsoccersim-*/rcssmonitor/src/rcssmonitor` 正是这样一个程序。

要回放自己录制的或其他人提供的比赛录像，将要用到录像播放器——`rcsoccersim-*/rcsslogplayer/src/rcsslogplayer`。当然要看到实际回放内容，还是需要仿真比赛的专用显示工具。

下面将对每一个操作进行更详细的解释。

● 配置

在编译仿真机器人足球源码包前首先需要运行 `configure` 脚本文件，该文件会自动检测系统配置，定位安装需要的库文件，同时设置程序安装的路径。缺省配置会将程序组件安装到下面几个位置中：

`/usr/local/bin` 放置上面提到的可执行程序

`/usr/local/include` 放置所有头文件

`/usr/local/lib` 放置所有库文件

如果要将程序安装到上述缺省位置，需要有管理员权限。要修改安装的位置，只需使用配置的 `--prefix=DIR` 参数及其相关选项。详情可以参考具体帮助（执行 `configure --help`）。

● 编译安装

当 `configure` 执行成功后，就可简单的执行 `make` 指令进行源码的编译。成功完成编译后，可以通过执行 `make install` 将相应的文件安装到缺省的系统目录中或用户指定的位置，不同的安装位置自然需要相应的权限。

● 卸载

源码包也很容易卸载，只需进入源码包解压的路径，执行 `make uninstall` 即可。这将移除所有安装的文件，但不会删除安装过程中创建的目录。

2.3.3 使用

● 运行比赛

要启动服务器端，只需进入安装执行程序的目录中，输入 `“./rcssserver”`；如果该路径为系统路径（例如，安装缺省路径安装），则只需在任何目录下直接输入 `“rcssserver”` 即可。`Rcssserver` 执行后会检查用户的 `home` 目录是否有相关配置文件，如果没有，就会在用户目

录下创建 `rcssserver` 目录并产生 `server` 执行的配置文件 `server.conf` 和 `player.conf`，配置文件里所有值都赋为缺省值，可以通过修改这两个文件改变比赛的一些规则。当然通过执行 `rcssserver` 时加上 `-sfile` 和 `-pfile` 参数可以指定 `rcssserver` 要使用的配置文件。如果指定的配置文件不存在，同样 `rcssserver` 会自动建立相应的配置文件并将所有值都赋为缺省值。

类似于上面，可以通过输入 “`./rcssmonitor`” 或 “`rcssmonitor`” 启动可视化工具来观看比赛情况。

如果执行程序安装在系统路径下，可以直接执行 `rcsoccersim` 脚本文件来启动 `server` 和 `monitor`。而且通过这种方式启动，当 `monitor` 被关闭时，`server` 也会自动停止运行。

要想实际在比赛平台上运行一场比赛，还必须启动一些球员程序与 `server` 建立连接（每个队最多只允许 11 名球员和一个教练）。当这些球员程序都与 `server` 建立好连接时，用户就可以通过 `Monitor` 开始比赛。

关于如何设计一支完整的球队，将在后续章节陆续介绍。在 RoboCup 仿真组的官方网站上可以找到一些现有的设计好的球队程序，可以通过用户自建的比赛平台观看其他球队的现场比赛或与自己设计的球队进行比赛，来不断提供自己的设计水平。

另外，在前面章节也提到了，在每个发布的比赛平台中都附带了一个简单的球员程序，它可以通过一个叫做 `ncurses` 的接口来控制球员行动，或直接通过终端命令行输入来控制；当启动时使用 `-nogui` 选项时程序就会自动执行而不受外界控制。

客户端程序要想和服务器端建立连接就必须按照服务器端提供的网络地址和端口（缺省 `host=localhost`，`port=6000`）。当然这些参数可以在服务器端启动前修改，网络地址为启动服务器端机器的网络地址，端口可以在 `server.conf` 文件中修改。

● 停止比赛

正确的停止比赛的过程如下：

- 1 停止所有连接到 `server` 的球员程序
- 2 停止所有连接到 `server` 的显示程序
- 3 在启动 `server` 的终端窗口中按下 `CTRL-C` 来停止 `server`

安装这个步骤，不但保证了中止所有可见的运行中的程序，而且也确保了所有在后台运行的程序（例如 `server`）正常结束。当 `server` 没有被完全结束之前，是不能启动另外一个 `server` 进程的，除非使用不同的参数。另外，如果不是通过 `CTRL-C` 正常结束 `server`，由 `server` 生成的录像文件将不能正常关闭（如果使用压缩模式录像，文件将会出问题），而且文件名不能正确被修改，看到的将是名为 `incomplete.rcg` 和 `incomplete.rcl` 文件。

注：比较有效并且便利地结束终端的方法是使用它们的名字。一般使用系统命令 `kill` 来结束一个进程需要通过 `ps` 命令来查找要结束的进程号。一个清除所有具有同样名字的进程的简单方法是使用 `killall` 命令，比如：“`killall rcssserver`” 就可以结束由于以 `rcssserver` 命名的进程。

2.4 比赛流程

整场比赛的过程如下：

- 首先比赛裁判启动比赛平台（`Soccerserver` 和 `Soccermonitor`）
- 双方领队猜先，决定哪方先上场（先上场的一方先开球）
- 双方球员上场，即双方各启动 11 个球员程序（如果有在线教练还可以再启动一个在线教练程序）与比赛平台连接，连接上后通过发送 `init` 命令进行初始化，实现球员上场

- 当全部球员都准备好时，比赛裁判按下 Monitor 的开始按钮，向 Server 发送比赛开始命令，上半场比赛开始
- 上半场比赛为 5 分钟。当上半场比赛结束时，比赛平台自动暂停比赛，各队球员程序如需调整可以与 Server 断开连接
- 中场休息为 5 分钟。在此期间，参赛者可以修改各自的程序。
- 在下半场比赛开始之前，刚才断开连接的所有球员程序需要使用 reconnect 命令与 Server 重新进行连接。
- 当全部球员准备就绪时，裁判按下开始按钮，开始下半场比赛，由猜先输的一方开球。
- 下半场比赛同样为 5 分钟。下半场比赛结束时，server 自动停止比赛。
- 如果比赛结果为平局，加时赛开始。加时赛采用金球法（也称为突然死亡法），即任一方球队进球，比赛立即结束，进球方获胜。
- 加时赛同样分上下半场，如果还是平局，就通过罚球决胜。

罚球比赛的具体过程如下：

- 1 首先由自动裁判宣布在球场的哪一边执行罚球（penalty_onfield_l/r）
- 2 自动裁判将球移到起始罚球点，距对方球门 42.5 米，这个值可以通过修改比赛平台参数 pen_dist_x 来调整）
- 3 自动裁判随机选择一个队先来罚球，并宣布（penalty_setup_l/r）
- 4 自动裁判在规定时间内（pen_setup_wait，缺省为 100 周期）等待双方布置球员，然后按照下面规则检查球员是否移到正确位置：
 - 防守方守门员必须移到球门柱之间距球门 goalie_pen_line_dist 的位置
 - 另一方守门员必须移到球门线后面并且在禁区线外，唯一例外是这个守门员要执行罚球
 - 防守方球员和发球人的队友必须在球场中间的圆圈内
 - 球队中先罚球的一个球员必须要移动距离球 2 米内的一个位置。任何一个球员都不允许在同队所有 11 名球员（包括守门员）没有全罚过球之前又进行罚球。如果在规定等待时间内某个球队没有遵守上述要求，一次罚球即告结束。如果罚球方犯错，即判罚球失败，否则算进球。另外可以通过设置系统参数 pen_coach_moves_players 使得即使有球队没有按照要求正确布置球员，所有违例球员也会自动被放置到一个合法位置。这样就不会出现上述违例判罚。如果两队都遵守规则，比赛状态将变为 PM_PenaltyReady_Left/Right。
- 5 双方都准备好后，罚球方在规定时间内（pen_ready_wait，缺省为 50 周期）要执行踢球行为。如果在罚球进行中上面第 4 步中的其中一条规则没有遵守，罚球立即结束并算作罚球违例，并按上面提到的方法判罚。类似于上面，如果参数 pen_coach_moves_players 被设置，所有违例球员也会自动被移动一个合法位置，而不会被判罚。执行完第一脚踢球后，比赛状态将变为 PM_PenaltyTaken_Left/Right。
- 6 根据系统参数 pen_allow_mult_kicks 来决定罚球者是否允许再次踢球（缺省是允许）。当在规定时间内（pen_taken_wait，缺省为 200 周期）球被扑住或踢出球门底线，即判罚球得分或失败（取决于球是否踢到球门里）。如果在规定时间内球没有被扑住，也没踢出底线，也会判罚球失败。相应的比赛状态就会变为 PM_PenaltyMiss_Left/Right，否则就是 PM_PenaltyScore_Left/Right。
- 7 现在来检查哪方将获得最后胜利：
 - 通过规则规定的罚球次数（pen_nr_kicks，缺省为 5）已经有一方得分超出
 - 虽然罚球还没有执行到规定次数，但一方在规定的次数内已不可能赶上，比如

罚了 3 个球后比分是 3:0，则最后两轮罚球就没有必要进行了

- 在规定次数罚球后双方打成平手，则要继续进行额外的罚球，直到一方胜出，但罚球次数不能超过规定次数（`pen_max_extra_kicks`，缺省为 10）

当比赛决出胜负后，裁判将向全场宣布（`pen_winner_l/r`），并且比赛状态变为 `PM_TimeOver`，即比赛结束。

如果经过 `pen_nr_kicks + pen_max_extra_kicks` 轮罚球，双方仍然平手，裁判将根据系统参数 `pen_random_winner` 来决定是否需要随机产生最后的胜利者。如果不允许随机，则最终判双方平手（`pen_draw`）

当处于所有其他状态时，而且两队都还没有赢得比赛，裁判将等待一段时间（`pen_before_setup_wait`，缺省为 30 周期）自动回到第 2 步，继续进行罚球

参考文献

[Noda et al. 1997] Itsuki Noda, Shoji Suzuki, Hitoshi Matsubara, Minoru Asada, and Hiroaki Kitano. Overview of RoboCup-97. In Hiroaki Kitano, editor, RoboCup-97: Robot Soccer World Cup I, pages 20{41. Springer-Verlag, 1997.

[Kitano 1996] Hiroaki Kitano, editor. Proceedings of the IROS-96 Workshop on RoboCup, Osaka, Japan, November 1996.

[Kitano 1998] Hiroaki Kitano, editor. RoboCup-97: Robot Soccer World Cup I. Springer Verlag, Berlin, 1998.

Mao Chen et. al. RoboCup Soccer Server. The RoboCup Federation, February 2003. Manual for Soccer Server Version 7.07 and later.

[Stone 1998] Peter Stone. Layered Learning in Multi-Agent Systems. PhD Thesis, Computer Science Dep., Carnegie Mellon University, December 1998

第三章、快速入门

3.1 让球员上场比赛

Soccer Server 是一个允许竞赛者使用各种编程语言设计球员，进行仿真足球比赛的系统。为了实现这一目标，仿真平台以 Client/Server 架构方式运行，Client 与 Sever 之间仅通过 UDP/IP 协议通讯，这样只要编程语言提供了对 UDP 连接的支持，就可以用来开发球队。Server 提供一个虚拟场地，对比赛中足球和所有球员的运动进行仿真，根据比赛规则以离散的方式控制比赛的进行；Client 相当于球员的大脑，通过 UDP 协议向 Server 发送命令来控制球员的行为。

Rcssclient

为了在 Server 上进行一场比赛，用户必须连接一些球员 Client 到 Server。由于还没有开始设计可用的球员，我们暂时使用仿真平台发行版本中提供的示例球员程序 rcssclient。rcssclinet 使用 ncurses 界面或者命令行界面（如果没有安装 ncurses 支持或者运行时使用了 -nogui 参数），能够通过 UDP 协议同 soccer server 建立连接，把用户输入的命令发送到 server，并接收和显示 server 发来的信息。

使用 rcssclient 的命令格式为：

```
$rcssclient [serverhost [port]]
```

其中 serverhost 表示运行 soccer server 的机器名或 ip 地址，参数 port 表示 soccer server 使用的端口，serverhost 和 port 的缺省值分别为 localhost 和 6000。执行后 rcssclient 会根据参数同 server 建立 UDP 连接，然后等待用户输入命令。

init 命令

球员只同 server 建立网络连接是不够的，server 需要知道球员的一些信息才能使其正确的上场。因此，建立连接后，每个球员都要向 Server 发送下面格式的 init 命令来声明自己：

```
(init TeamName [(version VerNum)] [(goalie)])
```

init 为仿真 server 命令，TeamName 表示球员所在的队伍名称，VerNum 表示球员使用信息格式的版本；对于守门员，必须在 init 命令中包括 “(goalie)” ，这样 server 才会允许其执行守门员才可以使用的扑球命令。请注意每个球员最多只能有一个守门员或者没有守门员（没有规定必须要有一个守门员）。

例如，在 rcssclient 的窗口中输入(init MyTeam (version 9))命令并回车，monitor 上的一只队伍名就会变为“MyTeam”，并且有一个边线旁的球员被激活，这个球员就是对应刚刚使用 init 命令声明的 client。直到这时才彻底完成一个球员的上场，此后 client 发送到 server 的命令才会被正确执行。

注意 rcssclient 显示到屏幕上的信息，这些是 client 从 server 接收到的消息，这些消息包括 server 对球员命令的响应信息、server 的一些参数信息以及比赛中场上的各种信息（。。。。章介绍）。使用 rcssclient 发送 init 命令后的一个可能显示如下：

```
1 (init MyTeam (version 9))
2 (init 1 2 before_kick_off)
3 (server_param (catch_ban_cycle 5)(clang_advice_win 1)(clang_define_win
  1)(clang_del_win 1)(clang_info_win 1) (clang_mess_delay
```

WrightEagle

```
50)(clang_mess_per_cycle 1) (clang_meta_win 1)(clang_rule_win
1)(clang_win_size 300) (coach_port 6001)(connect_wait
300)(drop_ball_time 0) (freeform_send_period 20)(freeform_wait_period
600) (game_log_compression 0)(game_log_version 3) (game_over_wait
100)(goalie_max_moves 2)(half_time -10) (hear_decay 1)(hear_inc
1)(hear_max 1)(keepaway_start -1) (kick_off_wait 100)(max_goal_kicks
3)(olcoach_port 6002) (point_to_ban 5)(point_to_duration 20)(port 6000)
(recv_step 10)(say_coach_cnt_max 128) (say_coach_msg_size
128)(say_msg_size 10) (send_step 150)(send_vi_step 100)(sense_body_step
100) (simulator_step 100)(slow_down_factor 1)(start_goal_l 0)
(start_goal_r 0)(synch_micro_sleep 1)(synch_offset 60) (tackle_cycles
10)(text_log_compression 0) (game_log_dir
"/home/thoward/data")(game_log_fixed_name
"rcssserver")keepaway_log_dir "./" (keepaway_log_fixed_name
"rcssserver") (landmark_file "~/rcssserver-landmark.xml")
(log_date_format "%Y%m%d%H%M-")(team_l_start "") (team_r_start
"")(text_log_dir "/home/thoward/data") (text_log_fixed_name
"rcssserver")(coach 0) (coach_w_referee 1)(old_coach_hear 0)(wind_none
0) (wind_random 0)(auto_mode 0)(back_passes 1) (forbid_kick_off_offside
1)(free_kick_faults 1) (fullstate_l 0)(fullstate_r 0)(game_log_dated 1)
(game_log_fixed 1)(game_logging 1)(keepaway 0) (keepaway_log_dated
1)(keepaway_log_fixed 0) (keepaway_logging 1)(log_times 0)(profile 0)
(proper_goal_kicks 0)(record_messages 0)(send_comms 0) (synch_mode
0)(team_actuator_noise 0)(text_log_dated 1) (text_log_fixed
1)(text_logging 1)(use_offside 1) (verbose 0)(audio_cut_dist
50)(ball_accel_max 2.7) (ball_decay 0.94)(ball_rand 0.05)(ball_size
0.085) (ball_speed_max 2.7)(ball_weight 0.2)(catch_probability 1)
(catchable_area_l 2)(catchable_area_w 1)(ckick_margin 1)
(control_radius 2)(dash_power_rate 0.006)(effort_dec 0.005)
(effort_dec_thr 0.3)(effort_inc 0.01)(effort_inc_thr 0.6) (effort_init
0)(effort_min 0.6)(goal_width 14.02) (inertia_moment 5)(keepaway_length
20)(keepaway_width 20) (kick_power_rate 0.027)(kick_rand
0)(kick_rand_factor_l 1) (kick_rand_factor_r 1)(kickable_margin
0.7)(maxmoment 180) (maxneckang 90)(maxneckmoment 180)(maxpower 100)
(minmoment -180)(minneckang -90)(minneckmoment -180) (minpower
-100)(offside_active_area_size 2.5)(offside_kick_margin
9.15)(player_accel_max 1)(player_decay 0.4)(player_rand
0.1)(player_size 0.3)(player_speed_max 1)(player_weight
60)(prand_factor_l 1)(prand_factor_r 1)(quantize_step
0.1)(quantize_step_l 0.01)(recover_dec 0.002)(recover_dec_thr
0.3)(recover_min 0.5)(slowness_on_top_for_left_team
1)(slowness_on_top_for_right_team 1)(stamina_inc_max 45)(stamina_max
4000)(stopped_ball_vel 0.01)(tackle_back_dist 0.5)(tackle_dist
2.5)(tackle_exponent 6)(tackle_power_rate 0.027)(tackle_width
1.25)(visible_angle 90)(visible_distance 3)(wind_ang 0)(wind_dir
0)(wind_force 0)(wind_rand 0))
4 (player_param (player_types 7)(pt_max 3)(random_seed -1)(subs_max
3)(dash_power_rate_delta_max 0)(dash_power_rate_delta_min
0)(effort_max_delta_factor -0.002)(effort_min_delta_factor
```

WrightEagle

```
-0.002)(extra_stamina_delta_max 100)(extra_stamina_delta_min
0)(inertia_moment_delta_factor 25)(kick_rand_delta_factor
0.5)(kickable_margin_delta_max 0.2)(kickable_margin_delta_min
0)(new_dash_power_rate_delta_max 0.002)(new_dash_power_rate_delta_min
0)(new_stamina_inc_max_delta_factor -10000)(player_decay_delta_max
0.2)(player_decay_delta_min 0)(player_size_delta_factor
-100)(player_speed_max_delta_max 0.2)(player_speed_max_delta_min
0)(stamina_inc_max_delta_factor 0))
5 (player_type (id 0)(player_speed_max 1)(stamina_inc_max
45)(player_decay 0.4)(inertia_moment 5)(dash_power_rate
0.006)(player_size 0.3)(kickable_margin 0.7)(kick_rand
0)(extra_stamina 0)(effort_max 1)(effort_min 0.6))
6 (player_type (id 1)(player_speed_max 1.1956)(stamina_inc_max
30.06)(player_decay 0.4554)(inertia_moment 6.385)(dash_power_rate
0.007494)(player_size 0.3)(kickable_margin 0.829)(kick_rand
0.0645)(extra_stamina 9.4)(effort_max 0.9812)(effort_min 0.5812))
7 (player_type (id 2)(player_speed_max 1.135)(stamina_inc_max
33.4)(player_decay 0.4292)(inertia_moment 5.73)(dash_power_rate
0.00716)(player_size 0.3)(kickable_margin 0.8198)(kick_rand
0.0599)(extra_stamina 31.3)(effort_max 0.9374)(effort_min 0.5374))
8 (player_type (id 3)(player_speed_max 1.1964)(stamina_inc_max
31.24)(player_decay 0.4664)(inertia_moment 6.66)(dash_power_rate
0.007376)(player_size 0.3)(kickable_margin 0.88)(kick_rand
0.09)(extra_stamina 47.1)(effort_max 0.9058)(effort_min 0.5058))
9 (player_type (id 4)(player_speed_max 1.151)(stamina_inc_max
37.8)(player_decay 0.45)(inertia_moment 6.25)(dash_power_rate 0.00672)
(player_size 0.3)(kickable_margin 0.8838)(kick_rand 0.0919)
(extra_stamina 44.1)(effort_max 0.9118)(effort_min 0.5118))
10 (player_type (id 5)(player_speed_max 1.1544)(stamina_inc_max
34.68)(player_decay 0.4352)(inertia_moment 5.88)(dash_power_rate
0.007032) (player_size 0.3)(kickable_margin 0.8052)(kick_rand 0.0526)
(extra_stamina 47.1)(effort_max 0.9058)(effort_min 0.5058))
11 (player_type (id 6)(player_speed_max 1.193)(stamina_inc_max 36.7)
(player_decay 0.4738)(inertia_moment 6.845)(dash_power_rate 0.00683)
(player_size 0.3)(kickable_margin 0.885)(kick_rand 0.0925)
(extra_stamina 92)(effort_max 0.816)(effort_min 0.416))
12 (sense_body 0 (view_mode high normal) (stamina 4000 1) (speed 0
)(head_angle 0) (kick 0) (dash 0) (turn 0) (say 0) (turn_neck 0) (catch
0)(move 0) (change_view 0) (arm (movable 0) (expires 0) (target 0 0) (count
0))(focus (target none) (count 0)) (tackle (expires 0) (count 0)))
13 (see 0 ((f c t) 6.7 27 0 0) ((f r t) 58.6 3) ((f g r b) 73 37) ((g r)
69.4 32) ((f g r t) 66 27) ((f p r c) 55.7 41) ((f p r t) 45.2 22) ((f
t 0) 6.3 -18 0 0) ((f t r 10) 16.1 -7 0 0) ((f t r 20) 26 -4 0 0) ((f t
r 30) 36.2 -3) ((f t r 40) 46.1 -2)((f t r 50) 56.3 -2) ((f r 0) 73.7 30)
((f r t 10) 68.7 23) ((f r t 20) 66 15) ((f r t 30) 64.1 6) ((f r b 10)
79 37) ((f r b 20) 85.6 42))
14 (sense_body 0 (view_mode high normal) (stamina 4000 1) (speed 0 0)
(head_angle 0) (kick 0) (dash 0) (turn 0) (say 0) (turn_neck 0) (catch 0)
(move 0) (change_view 0) (arm (movable 0) (expires 0) (target 0 0) (count
```

```

0)) (focus (target none) (count 0)) (tackle (expires 0) (count 0)))
15 (see 0 ((f c t) 6.7 27 0 0) ((f r t) 58.6 3) ((f g r b) 73 37) ((g r)
69.4 32) ((f g r t) 66 27) ((f p r c) 55.7 41) ((f p r t) 45.2 22)((f t
0) 6.3 -18 0 0) ((f t r 10) 16.1 -7 0 0) ((f t r 20) 26 -4 0 0) ((f t r
30) 36.2 -3) ((f t r 40) 46.1 -2)((f t r 50) 56.3 -2) ((f r 0) 73.7 30)
((f r t 10) 68.7 23) ((f r t 20) 66 15) ((f r t 30) 64.1 6) ((f r b 10)
79 37) ((f r b 20) 85.6 42))

```

• 图 3.1 球员初始化信息

图 3.1 第 1 行是用户通过 `rcssclient` 向 `server` 发送的命令。接下来是从 `server` 接收到的信息，其中前 11 行属于球员的初始化过程，`Client` 和 `server` 初始化上场完成之后，`server` 开始向 `client` 发送感知信息（。。。章），其中第 12 和 14 行表示身体感知信息，13 和 15 行表示视觉感知信息。

在球员的上场初始化过程中，第 2 行是 `server` 对球员 `init` 初始化命令的响应。如果球员可以正确上场，则响应消息格式如下：

```
(init Side UniformNumber PlayMode)。
```

`Side` 代表比赛中球员所在球队所处的半场，其值是一个字母，可取 `l(left)` 或者 `r(right)`；`UniformNumber` 是球员的球员号，场上的球员是通过它们的队伍名称和球员号共同来标识的（详见。。。章）；`PlayMode` 是表示一个有效比赛模式的一个字符串（见。。。）。例如，第 2 行 (`init l 2 before_kick_off`) 表示球员球队的半场是球场的左侧 `l`，球员是 `2` 号，当前场上的比赛模式是 `before_kick_off`。

如果上场有问题，则会是如下出错消息：

```
(error no_more_team_or_player_or_goalie)。
```

可能的情况包括场上已有两支与球员不同的队伍，或者球员所在球队上场的人数超过了 11 个，或者球队中使用了不止一个守门员。

如果使用 `init` 命令时 `version` 参数使用了 7.00 或更高的值，球员的上场初始化过程中会接收到附加的 `server` 信息（本例中的第 3—11 行），分别为：

(sevrer_param Parameters...)	当前 server 使用的比赛相关参数
(player_param Parameters...)	当前 server 使用的球员相关参数
(player_type id Parameters...)	异构球员信息（详见。。。章）

reconnect 命令

有时会需要停止一个 `client`，修改其程序，在不重新开始比赛的情况下它再次上场，这时需要使用 `reconnect` 命令。`Reconnect` 命令用来向 `server` 声明某个退出的球员重新上场（只能在比赛处于非 `Play_On` 模式时使用，例如半场停止时）。

`reconnect` 命令的格式为：

```
(reconnect TeamName UniformNumber)
```

`TeamName` 和 `UniformNumber` 共同标识出要求重新上场的球员。如果可以正确上场，会得到下面格式的 `server` 响应消息：

```
(reconnect Side PlayMode)
```

否则，根据不同情况，可能会是如下消息：

(can't reconnect)	表示比赛处于 <code>PlayOn</code> 模式，不允许 <code>reconnect</code> ；
(error reconnect)	表示由于某种错误，球员未能成功重新上场；
(error no_more_team_or_player_or_goalie)	类似于 <code>init</code> 命令的响应。

如果重新连接的球员使用的是高于 7.00 的信息版本，同 `init` 一样会收到 `sevrer_param`，`player_param`，`player_type` 等信息。

Bye 命令

如果要球员断开同 server 的连接，可以向 server 发送一个 bye 命令：

(bye)

这个命令会通知 server 把球员从场上删除。对于 bye 命令，server 没有响应消息。

3.2 球员可发送的命令

正确上场后，球员可以向 server 发送命令，server 在每个仿真周期结束时执行这些命令，并根据这些命令的结果仿真计算出下一个周期场上的状况。球员可以使用的命令主要包括身体控制命令，通讯命令和数据请求命令。

身体控制命令

球员在场上的运动和比赛行为都是由几个基本的身体控制命令组合而成的，这里只简单介绍一下这些命令的格式和基本作用，参数的作用、命令具体的仿真模型等比较复杂，将在。。。章中的行为模型部分具体详细说明。

(turn Moment)

球员从当前位置转身 *Moment* 度，其中 *Moment* 的取值为-180~180。

(catch Direction)

守门员特有命令，向相对身体 *direction* 方向扑球，如果扑球成功，球将在守门员控制下知道被踢走。

(dash Power)

球员沿身体方向加速，*Power* 表示加速使用的力量大小，取值为-100~100。

(kick Power Direction)

沿 *Direction* 方向使用 *Power* 的力量踢球。其中 *Direction* 是相对于球员身体的方向，介于-180 和 180 之间；*Power* 介于 0 和 100 之间。

(move X Y)

使球员移动到场上坐标 (X, Y) 的点，只能在开球前或进球后使用，并只能移动到自己球队的半场。

(tackle Power)

铲球指令可以使球员距离球最远达 2 米时也能踢到球，并且不受中间物体阻碍，*Power* 参数同 **kick** 指令；但铲球后球员有 10 个周期不能动。铲球成功判定取决于一个与球位置相关的随机概率。

注意，server 中每个仿真周期，每个球员只能执行以上 6 个命令中的一个命令，如果同一周期一个球员有多个命令到达 server，server 会随机选择一个来执行。

(pointto Distance Direction) 或者 (pointto off)

指向某点指令使球员可以做出或取消一个指向某点的手势。第一种格式球员指向相对于球员当前面朝方向 *Direction* 角度，距离 *Distance* 的点。但实际上其他球员只能看到该球员指的一个大概方向。

(attentionto Team Unum) 或者 (attentionto off)

该命令可以使球员将注意力集中到一个指定球员身上，只能听到此球员的说的话，直到再次发此指令换一个注意的对象或解除这种状态。其中 *Team* 是指哪一边球队，*Unum* 是球员号码。

(turn_neck Angle)

球员头部独立与身体转过 *Angle* 度，其中 *Angle* 的取值为-180~180，但是球员头部相对于身体的角度必须介于-90 和 90 之间。Turn_neck 命令可以同其他身体控制命令在同一仿

真周期执行。

```
(change_view Width Quality)
```

改变球员的视觉模式，其中 *Width* 的取值为 **narrow**, **normal** 或 **wide**, *Quality* 的取值为 **high** 或 **low**。（有关球员视觉模式在。。。章）

通讯命令

```
(say Message)
```

球员之间的只能通过使用 **say** 命令广播消息来进行通讯，**say** 命令的内容 *Message* 以听觉感知信息 **hear** 形式到达。

如果 **say** 命令被成功执行，在 **server** 会返回响应消息：

```
(ok say)
```

否则如果命令中存在错误则会返回消息：

```
(error illegal_command_form)
```

数据请求命令

```
(score)
```

请求 **server** 发送比分，**server** 返回信息格式为：

```
(score Time Our Their)
```

Time 表示发送数据的时间，*our* 和 *their* 分别表示本方和对方的进球数。

```
(sense_body)
```

请求 **server** 发送身体感知信息。这个命令只是为了保持向下兼容，从 6.00 版本以后，**server** 每个仿真周期向球员自动发送身体感知信息，不再需要使用 **sense_body** 命令。

3.3 球员可获得的信息

球场中，一个球员只有了解周围的环境，才能据此作出各种决策，实现有效的比赛。在仿真平台中，球场上所有物体的运动都是由 **server** 进行仿真的，因此，在比赛过程中，为了获得场上情况，球员需要从 **server** 接收各种信息。

球员从 **server** 获得的信息主要有两种，一种是对球员命令的响应信息，通知球员命令是否正确执行等，这类消息与球员命令结合在一起介绍；另一种是球员的感知信息，分为视觉信息，身体感知信息和听觉信息，所有感知信息都包含一个时间标签，用来表示 **server** 发送信息的时间（仿真周期数）；与命令响应信息不同，感知信息不需要球员的请求，视觉信息和身体感知信息由 **server** 周期性地自动发送到球员，听觉信息在裁判、教练或球员发出后发送带球员。这里简单介绍感知信息的格式和特点，具体的表示和数据仿真计算方法在。。。章中感知模型的部分具体说明。

视觉感知信息

视觉信息是球员最重要的感知信息，它包含了球员可以看到的物体的信息。视觉信息的主要特点有：第一，视觉信息中的数据都是相对于观察球员的，而不是场上的绝对信息，决有相对性；第二，视觉信息获得的不是场上所有物体的信息，而只是在球员视野中的物体的信息，具有局部性；第三，视觉信息中物体的信息量受到与球员距离等因素的影响，具有不完整性；第四，视觉信息中的数据存在误差，具有不精确性。

视觉信息的格式为：

```
(see Time ObjInfo+)
```

```

ObjInfo ::= (ObjName Distance Direction
              [DistChange DirChange [BodyDir NeckDir]])
              |(ObjName Direction)
ObjName ::= (p [TeamName [Unum]])
              | (b)
              | (f FlagInfo)
              | (g Side)
              | (l LineInfo)

```

听觉感知信息

听觉感知用来接收其他球员或者教练通过 say 命令发送的消息，裁判广播的信息也作为球员的听觉信息接收。听觉感知信息的主要特点是：第一，仿真平台中的声音信息仿真的一个拥挤的低带宽环境，双方所有球员共用的是一个不可靠的频道；第二，如果是其他球员的声音，则只能得到发出声音球员的相对角度，而不能获得其球员标识。

听觉感知信息的格式为：

```

(hear Time Sender Message)
Sender ::= self | referee | online_coach_l
          | online_coach_r | Direction

```

身体感知信息

身体感知信息包含了球员自身的各种状态信息，信息格式为（详见 4.1.2）：

```

(sens_body Time (view_mode {high|low} {narrow|normal|wide})
  (stamina Stamina Effort)(speed Speed Angle)
  (head_angle Angle)(kick Count)(dash Count)
  (turn Count)(say Count)(turn_neck Count)
  (catch Count)(move Count)(change_view Count))

```

3.4 一般球员的框架结构

3.4.1 server 连接特性分析

在进一步介绍 Client 之前，我们先说明一下球员与 Server 进行连接的一些特性。

UDP 连接特性

Client 通过 UDP 连接和 Server 进行通讯。UDP 是一种非面向连接的不可靠传输协议。它具有以下几个特点：不保证数据的可靠到达，信息在传输过程中可能丢失；不保证数据的顺序到达，由于网络延迟等原因，后发送的信息可能会先到达接受方并先被处理；另外需要注意的是，UDP 连接没有流量控制机制，使用一个一定大小的接受缓冲队列。到达的信息在被读取之前是按照到达顺序存储在队列中。如果接收队列已满，则之后到达的信息会没有任何提示的被丢弃，而不会通知发送方进行重发。从 UDP 连接读取数据实际上是从接收缓冲队列中按顺序取出信息。如果球员不能按照信息到达的速度读出信息，而是慢与信息到达速度，则可能会接收到缓冲队列中过时的信息，而最新的信息则排在队列的后面，这就会导

致球员根据过时的信息来理解周围的环境，而产生错误的决策和行为。UDP 的这种特性就要求 Client 必须尽快的接收 Server 发送来的最新信息，避免信息在缓冲队列中排队，保证球员能够获取最新的环境信息，由此选择最适宜的行为。

Server 时序特性

Server 仿真一个实时系统，但是它是一种伪实时，采用的是一种离散的工作模式：在一个固定的“仿真周期”时间片中，Client 向 Server 发送球员的行为命令，Server 接收到这些命令后并不是立刻执行；相反，只有当这个仿真周期时间片结束时，Server 才会执行这些命令，并根据这些命令更新世界模型，生成下一个周期场上的状况。一个仿真周期中每个 client 只能执行一个主要（primary）行为命令，如果一个 Client 在一个仿真周期中发送了多个命令到 Server，Server 会随机选择其中一个来执行。这样会导致行为的不确定性，因此 client 要保证每个周期只发送最多一个命令到 Server。另一方面，如果某个周期 client 没有发送命令到 Server，球员就会什么都不作而浪费这个周期的行为机会，在一个足球比赛的对抗环境下，这会让对方获得主动，因此也是要避免的。由于 Server 和 Client 使用不同的时钟时间，这就要求 client 必须采取某种机制，根据有限的信息同 server 保持同步，以保证在正确的周期发送信息到 Server。

Server 另一个复杂的特性是 Server 发送感知信息和执行动作是异步的。在当前版本中，Server 每 100ms 执行一次 Client 发送的行为命令，但是每 150ms 才发送视觉信息给 client。由于为了利用所有的行为机会，client 需要每个仿真周期都执行一个命令，这就要求 client 在一部分周期中必须在没有新的视觉信息的条件下决定要执行的行为。这就要求 client 能够根据之前的信息预测当前的场上环境。信息获取和行为执行的异步性迫使 client 要在获取使用最新环境信息和不浪费执行机会两者中达到最佳平衡。

- 图 3.2 时序图示例及说明 ??可能和第 7 章重复

3.4.2 球员需要面对的问题

一个球员采用的框架结构是与它要解决的问题息息相关的，因此，为了设计球员的框架结构，我们先要知道一个球员需要面对的基本问题。

首先，仿真平台提供给球员的是一个动态实时的对抗环境，一支队伍所有球员努力完成共同的目标，因此要求球员之间可以分工协作；但是，球员之间只能通过听觉信息进行通讯，根据 3.3 中听觉感知的介绍，仿真平台中的声音信息仿真的一个拥挤的低带宽环境，双方所有球员共用一个不可靠的频道，这使球员间没有足够的时间使用通讯来协商复杂的团队计划，因此，每个球员应该能够独立决策行为进行比赛。

其次，在每个仿真周期，球员要从 Server 接收感知信息，对这些信息进行处理分析，并决策产生要执行的行为，发送相应命令到 Server。根据前面对 Server 时序特性的分析，球员需要面对的问题有：第一，尽快的接收 Server 发送的最新感知信息，避免信息在 UDP 接收缓冲队列中被排队，保证球员能够获取最新的环境信息；第二，设法保持同 Server 仿真周期的同步，保证决策的命令能够在正确周期发送到 Server；第三，尽量使用可获得的最新的环境信息来进行决策，但是由于 server 感知信息和执行动作的异步特性，为了效率，要求球员在没有视觉感知信息到达的周期中也要能够做出合理的决策。

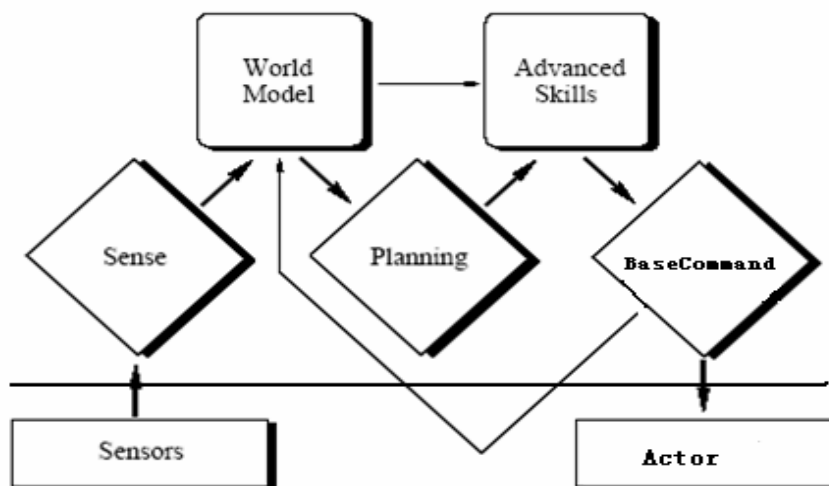
再次，球员主要通过从 server 获得的视觉感知信息来了解场上的状况。根据 3.3 中视觉感知模型的介绍，球员需要面对的问题为：视觉感知具有相对性、局部性、不完整性和不精确性，但是为了做出正确最优的决策，球员需要使用场上全部物体的绝对位置、速度等信息，并要尽可能保证这些信息完整和精确，因此要求球员能够使用一个包括全局信息的世界模型。

再次，根据 3.2 中球员命令的介绍，球员在场上的运动和比赛行为都是由几个最基本的身体控制命令组合而成的，如果要求每次决策只使用这些基本命令，会大大降低决策的性能，因此，要求球员能够为决策提供相对复杂的行为支持。

最后，当前使用的决策搜索算法，通常随着搜索时间的增加，获得最优决策的可能性也会增加。因此，为了提高球员决策的性能，在保证同 server 周期同步的前提下，应使用尽可能多的时间来进行决策。

3.4.3 球员基本框架

为解决球员面对的问题，通常球员包含的基本框架结构如图：



• 图 3.3 球员框架图

图中的粗箭头表示数据的流向和程序的流程，细箭头表示数据的流向。

首先，球员使用多线程的程序结构。线程是一个程序的执行体，在单处理器系统中，内核使用时间分片来实现线程的并发执行；多线程程序使用一种多任务、并发的生活方式，可以提高程序的并发度，从而提高程序运行效率和响应时间，同时具有线程间通信机制方便、改善程序结构等优点。在图 3.3 的框架结构图中，球员的感知部分和行为部分同其他部分处于不同的线程（具体见 3.4.4 的球员流程），这样，当 sense 线程接收过一条 server 信息并发送给其他线程后，就立刻转入等待下一条信息的到达，保证了尽快的接收 Server 的最新信息；而当命令发送时间到达时，Act 线程可以直接发送命令，而不需要等待其他部分的执行，使命令能够在正确周期及时发送；而当 Sense 线程因等待 server 网络信息而处于阻塞的时候，决策线程不需等待而可以继续执行，保证了球员使用尽可能多的时间来进行决策搜索。

其次，在图 3.3 的结构中，球员引入了世界模型 World Model 部分。世界模型中保存场上所有物体的绝对位置、速度等信息以及其他球员需要使用的信息，并采用多种更新方式尽可能的保持同周围场上环境的一致性：当有视觉信息到达时，把视觉信息中的局部相对信息转化为全局数据；对于没有包含在世界信息中的物体，或者当没有视觉信息到达时，则根据世界模型中记忆中的信息对物体的状态进行预测；并跟踪记录执行的命令，把命令的执行效果也包含到预测更新中（图中的 BaseCommand 到 World Model 的数据流）。这样球员的决策模块 Planning 就可以根据世界模型中的信息进行决策，而不是直接依赖于获得的感知信息，从而解决了球员面对的感知信息局部不完整性和 server 感知行为时序异步性等问题。世界模型具体更新方式见第 4 章和第 7 章。

最后，球员使用一个 Advanced skills 模块，为 Planning 决策部分提供高层行为（例如停

球，运球等）支持，并负责把决策产生的高层行为转化为基本行为命令和相应命令参数，使决策可以独立于底层基本行为，提高了球员的易用性和可扩展性。

3.4.4 球员基本流程

根据球员基本框架结构的讨论，一个使用 3 个线程来实现的球员流程见伪代码 3.1。

整个流程的接收，决策，发送 3 个基本任务由 3 个线程分别负责。**Sense** 线程接收 **Server** 发送的信息并对其进行解析，**Think** 线程处理这些信息并进行决策得出相应的行为，**Act** 线程则负责生成相应的命令及参数，并发送到 **Server**。这些线程与图 3.2 中的框架结构有如下对应关系：**Sense** 线程表示 **sensors** 和 **sense** 部分，**Act** 线程表示 **BaseCommand** 和 **Actor** 部分，而 **Think** 线程表示其他部分。

多线程的主要优点是它提高程序运行效率和响应时间，保证了信息的发送和接收，同时减少了球员用来处理 **server** 网络信息获取和发送的时间，使球员可以使用大部分时间来进行行为决策；但是多线程也会使球员实现更为复杂，需要正确解决线程间的同步问题。保证线程同步主要是由于 **Think** 线程和 **Sense** 线程必须互斥的访问世界模型 **World Model** 中的数据：**Sense** 线程解析从 **Server** 收到的信息并把它们添加到世界模型中，**Think** 线程根据感知到的信息和记忆中保存的信息来更新世界模型，并据此进行决策。如果两个线程可以同时访问世界模型，就可能导致数据的不一致（例如，**Sense** 线程还没有完全把信息写入世界模型，**Think** 线程就开始进行更新操作）。

```

{Think Thread}
Create Sense thread
Create Act thread
While server_is_alive do
    Block until READY signal arrives or 3 seconds have passed
    If 3 seconds have passed then
        Server_is_alive = false
    Else
        WAIT(lock)
        Update world model
        Determine next action
        Send commands to Act thread
        SIGNAL(lock)
    End if
End while

{Sense Thread}
while true do
    block until server message arrives
    determine send time t for Act thread    //发送时间决定见。。章同步方式
    set SEND signal to go off at time t
    WAIT(lock)
    Parse server message and send it to world model
    SIGNAL(lock)
    Send READY signal to Think thread
End while

```

```
{Act Thread}  
while true do  
    block until SEND signal arrives  
    convert commands to string messages  
    send messages to server  
end while
```

- 代码 3.1

在伪代码中使用了两个信号量机制伪函数 **SIGNAL** 和 **WAIT**，其操作分别为：**SIGNAL**(semaphore)：如果没有线程被信号量 semaphore 阻塞，则使 semaphore 加 1；否则被阻塞的线程中选择一个使其从 **WAIT** 函数后的代码开始继续执行。**WAIT**(semaphore)：如果信号量 semaphore 值大于 0，则使 semaphore 减 1，并使线程继续执行；否则挂起线程（线程被信号量阻塞）等待 **SIGNAL**。世界模型的数据由一个信号量 lock 来保护，保证线程对其的互斥访问。

参考文献

Mao Chen et. al. RoboCup Soccer Server. The RoboCup Federation, February 2003. Manual for Soccer Server Version 7.07 and later

Peter Stone. Layered Learning in Multi-Agent Systems. PhD Thesis, Computer Science Dep.,Carnegie Mellon University, December 1998

R. de Boer and J. R. Kok. The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team. Master's thesis, University of Amsterdam, The Netherlands, Feb. 2002.

第四章、球员智能体建模

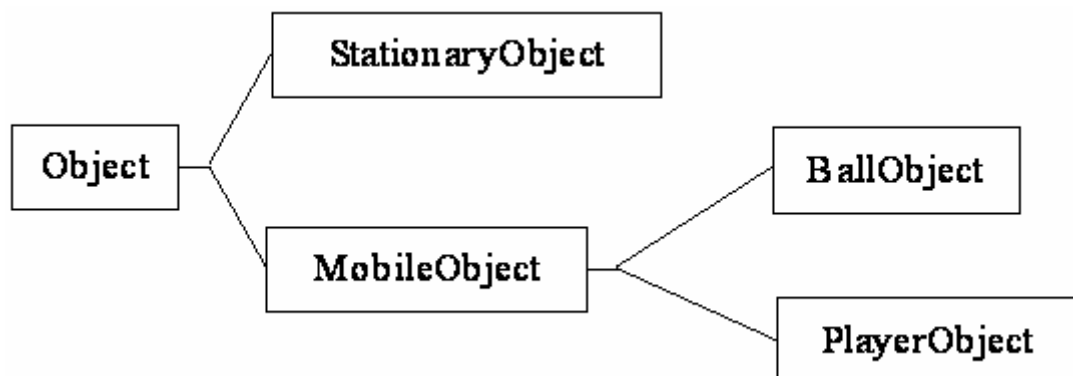
4.1 简单世界模型的构建

由于球员的行为不是单纯的直接根据从 **Server** 获得的信息，而是根据内部维护的世界模型，因此，保证选取动作时世界模型的准确和精确是非常必要的。世界模型的构建包括了以下几个方面：模型内对象的表示，**Server** 中运动和行为的建模，根据 **Server** 发送的感知信息进行更新，使用运动和行为的预测生成简单的行为等。

4.1.1 平台中的对象

场上对象结构

一个球员为了选择合适的行为，必须要知道自己场上位置速度等信息，同时也要了解求和其他球员的位置和速度等信息。因此，世界模型必须包含球场上这些不同物体的信息。这些物体主要分为两类，一类是具有固定的位置，例如标志，边线，球门等；另一类是动态物体，例如球和球员，位置会发生改变。使用面向对象的方法，这些物体在世界模型中的层次结构可以表示为下图：



Object:

- Global(x,y) position coordinates
- Confidence within [0,1] of the position accuracy

Stationary Object: nothing additional

Mobile object:

- Global(dx,dy) velocity coordinates
- Confidence of the velocity accuracy

Ball: nothing additional

Player:

- Team
- Uniform number
- Global facing angle
- Confidence of the facing angle

• 图 4.1 仿真平台中对象结构

详细说明如下：

Object: 在整个对象结构中，Object 是一个抽象基础父类，它封装了所有物体都包含的基本信息，最主要的几个属性是物体的类型，全局位置等，另外，由于信息的不完全性，球场上会有很多不可感知的物体，为了决策的需要，只记录可见的物体是远远不够的。这样，如果一个物体没有被看到，必须根据旧的数据对它的信息进行预测，不可见的时间越久，预测值与实际值之间存在的误差就会越大，因此，Object 中引入了一个位置可信度属性来表示这种预测的不确定性，球员可以根据这个属性的值来决定可以多大程度上依靠 Object 位置信息来做决策。

虽然感知信息和命令的参数都是使用相对的极坐标，世界模型中存储的 Object 位置使用全局坐标，因为全局坐标更容易存储和维护，尤其当球员在球场上四处移动的时候；而且考虑到固定物体在全局坐标中的位置是不变的，使用全局坐标更容易应用于决策。

Stationary Object: 在 Object 基础上没有增加其他的属性信息，表示场上的固定物体。

Mobile Object: 在 Object 基础上增加了速度信息，并且由于前面所述的原因，引入了速度信息的可信度属性，表示场上的可移动物体。

Ball Object: 在 Mobile Object 基础上没有增加其他信息，表示足球。

Player Object: 在 Mobile Object 基础上增加了球员所特有的信息：球员的标识信息（队名和球员号），球员头部全局角度，身体全局角度等。

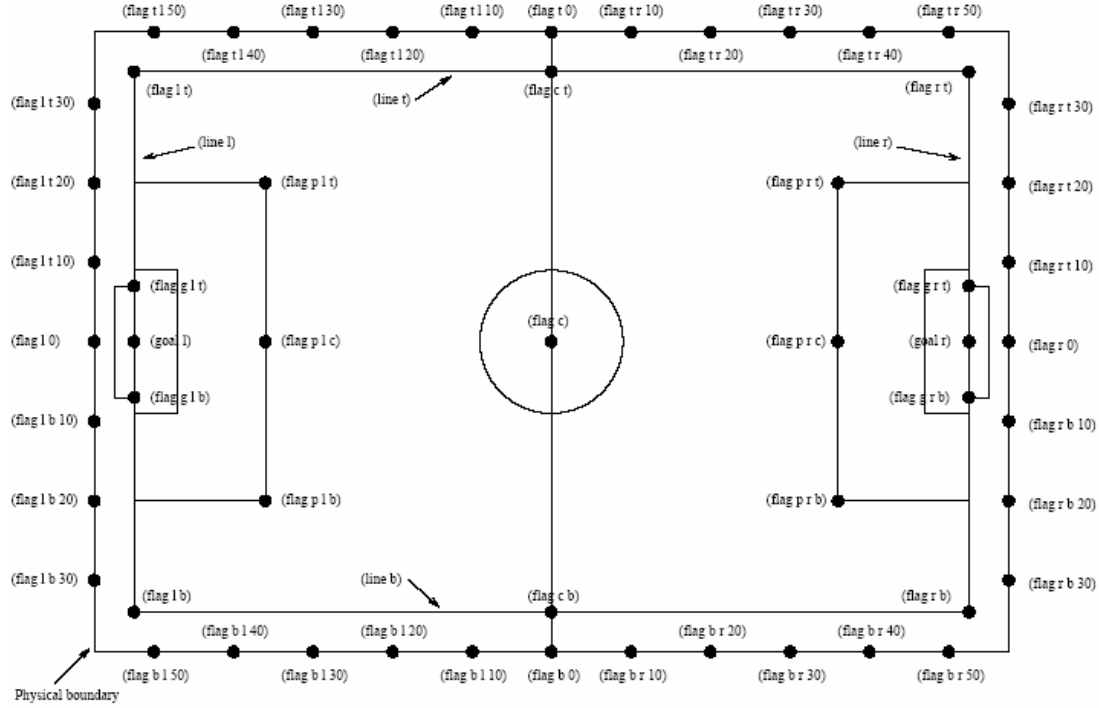
场上对象命名

球场上包含了各种物体，为了标识这些物体，在仿真平台中每个物体都有一个固定的名字。物体名称 ObjName 的格式为：

$$\begin{aligned} \text{ObjName} ::= & (p [\text{TeamName} [\text{Unum}]]) \\ & | (b) \\ & | (f \ c) \\ & | (f \ g \ [l|r] \ [t|b]) \\ & | (f \ p \ [l|r] \ [t|c|b]) \\ & | (f \ [l|r|t|b] \ 0) \\ & | (f \ [l|c|r] \ [t|b]) \\ & | (f \ [l|r] \ [t|b] \ [10|20|30]) \\ & | (f \ [t|b] \ [l|r] \ [10|20|30|40|50]) \\ & | (l \ [l|r|t|b]) \\ & | (g \ [l|r]) \\ & | (B) \\ & | (G) \\ & | (F) \\ & | (P) \end{aligned}$$

其中(b)表示足球；(p [TeamName [Unum]])表示场上的一个球员，TeamName 表示该球员所在的球队名，Unum 表示该球员的球员号；(g Side)表示球门，Side 可以取值为 l 或 r，表示左侧或右侧球门的中点；(l LineInfo)表示球场的边线，LineInfo 可以取值 l、r、t 或 b，分别表示左（left）边线，右（right）边线，上（top）边线和下（bottem）边线；(B)、(G)、

(F)和(P)等大写的名称表示球员邻域范围内的物体（见。。。节），只包含物体类型，而不知道物体的确切名称；((f *FlagInfo*))表示场上的标志，具体每个标志的名称见图 4.2:



• 图 4.2 仿真平台上标志位置与名称

4.1.2 感知模型

视觉模型

球员通过视觉感知获得视野中的物体距离，角度等信息。每 *sense_step*（目前 *server* 设置等于 150）ms，视觉感知信息由 *server* 自动的发送到球员处。视觉感知中的信息都是相对于球员的数据，球员不能直接获得自身或其他球员和球的全局位置等信息。为了维护一个全局的世界模型，球员必须从视觉感知推导出自身的全局位置，然后进一步计算出其他球员和球的全局位置。

视觉信息格式

从 *server* 发送的视觉信息遵循下面的基本格式：

(see *Time ObjInfo*⁺)

其中 *Time* 为 *Server* 发送这条视觉信息时所处的仿真周期数。

ObjInfo⁺ 表示一个或者多个 *ObjInfo*。而 *ObjInfo* 表示为

$$\begin{aligned} \text{ObjInfo} ::= & (\text{ObjName Distance Direction} \\ & [\text{DistChange DirChange} [\text{BodyDir NeckDir}]]) \\ & | (\text{ObjName Direction}) \end{aligned}$$

ObjName 见 4.1.1 节介绍，*Distance*，*Direction*，*DistChng* 和 *DirChng* 等则由 *server* 按以下公式计算出来：

$$p_{rx} = p_{xt} - p_{xo} \quad (4.1)$$

$$p_{ry} = p_{yt} - p_{yo} \quad (4.2)$$

$$v_{rx} = v_{xt} - v_{xo} \quad (4.3)$$

$$v_{ry} = v_{yt} - v_{yo} \quad (4.4)$$

$$\text{Distance} = \sqrt{p_{rx}^2 + p_{ry}^2} \quad (4.5)$$

$$\text{Direction} = \arctan(p_{ry} / p_{rx}) - a_o \quad (4.6)$$

$$e_{rx} = p_{rx} / \text{Distance} \quad (4.7)$$

$$e_{ry} = p_{ry} / \text{Distance} \quad (4.8)$$

$$\text{DistChange} = (v_{rx} \cdot e_{rx}) + (v_{ry} \cdot e_{ry}) \quad (4.9)$$

$$\text{DirChange} = [(-(v_{rx} \cdot e_{ry}) + (v_{ry} \cdot e_{rx})) / \text{Distance}] \cdot (180 / \pi) \quad (4.10)$$

$$\text{BodyDir} = \text{body_dir_abs} - a_o \quad (4.11)$$

$$\text{NeckDir} = \text{neck_dir_abs} - a_o \quad (4.12)$$

其中 (p_{xt}, p_{yt}) 是目标的绝对位置坐标, (p_{xo}, p_{yo}) 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_{xo}, v_{yo}) 表示队员自己的绝对速度。 a_o 是球员头部所朝向的全局方向。球员面向的全局方向是球员自己的身体角度 **BodyDir** 和头部角度 **NeckDir** 的之和。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。如果被观察者是球员, 则视觉信息中可能包含 **BodyDir** 和 **NeckDir**, 分别表示被观察球员身体和头部相对观察者头部的角度。如果两个球员的头部具有相同的全局角度, 那么视觉信息中的 **NeckDir** 就等于零; **BodyDir** 也类似。

在真实的足球场上, 球员只能看到自己视野中的球员、球等信息, 同时由于距离的关系, 有些的球员可能会看不清楚, 到其他物体之间的距离也只能是粗略的估计, 而且如果希望更清楚的观察场上状况, 就需要更多的时间。在 **Soccer Server** 中, 视觉模型也仿真了这些情况。球员并不能看到球场上所有的物体, 也不一定能看到可见物体的所有信息; 在这种情况下, 球员的视觉信息量由球员的视觉模式决定。

视觉模式

视觉模式包括视野范围、视觉质量和信息间隔三个部分。

server 上的参数 **sense_step** 和 **visible_angle**, 分别决定了球员基本的视觉信息到达时间间隔和球员正常的视野范围 (现在使用的是 150ms 和 90°)。但是实际比赛中视觉信息到达球员的间隔和 **Server** 视觉感知中包含的信息量由球员使用的观察模式决定。球员可以通过 **change_view** 命令调整使用的视觉模式, 根据需要获得相应的视觉信息间隔、视野范围和视觉质量等, 以满足决策的要求。**Change_view** 命令包含两个参数: **ViewWidth** 和 **ViewQuality**。**ViewWidth** 表示球员视野范围, 为 **wide**、**normal** 和 **narrow** 之一。球员实际视野的角度用如下公式计算:

$$\text{view_angle} = \text{visible_angle} \cdot \text{view_width_factor} \quad (4.13)$$

其中如果 **ViewWidth** 是 **narrow**, 那么 **view_width_factor** 等于 0.5; 如果 **ViewWidth** 是 **normal**, 那么 **view_width_factor** 等于 1, 如果 **ViewWidth** 是 **wide**, 那么等于 2。

ViewQuality 为 high 或 low。当 ViewQuality 设置为 high 时, SoccerServer 为观察者发送详细的目标位置信息。当 ViewQuality 设置为 low 时, Server 为观察者发送简化的目标信息, 只包含目标的相对方向。

另一方面, Server 为队员发送视觉信息的间隔随着 ViewWidth 和 ViewQuality 的变化根据下式计算:

$$View_frequency = sense_step \cdot view_quality_factor \cdot view_width_factor \quad (4.14)$$

若 ViewQuality 是 high, 那么 view_quality_factor 等于 1; 若 ViewQuality 是 low, 那么 view_quality_factor 等于 0.5。由这个公式我们可以看到, 视觉质量越高, 视角越大, 获得信息的时间间隔越大, 符合真实世界的特性。

邻域

球员有一个以 visible_distance 为半径的邻域, 如果某一物体在此范围内, 即使不在球员的视角范围内, 球员仍然可以“看到”(其实是感觉到)这个物体。不过球员只能知道对象的类型(足球, 球员, 球门或者是标志), 而不知道对象的确切名字。就是说获得的视觉信息中使用“B”, “P”, “G”和“F”来作为对象的名字, 而不是使用“b”, “p”, “g”和“f”。

可视信息的不完整性

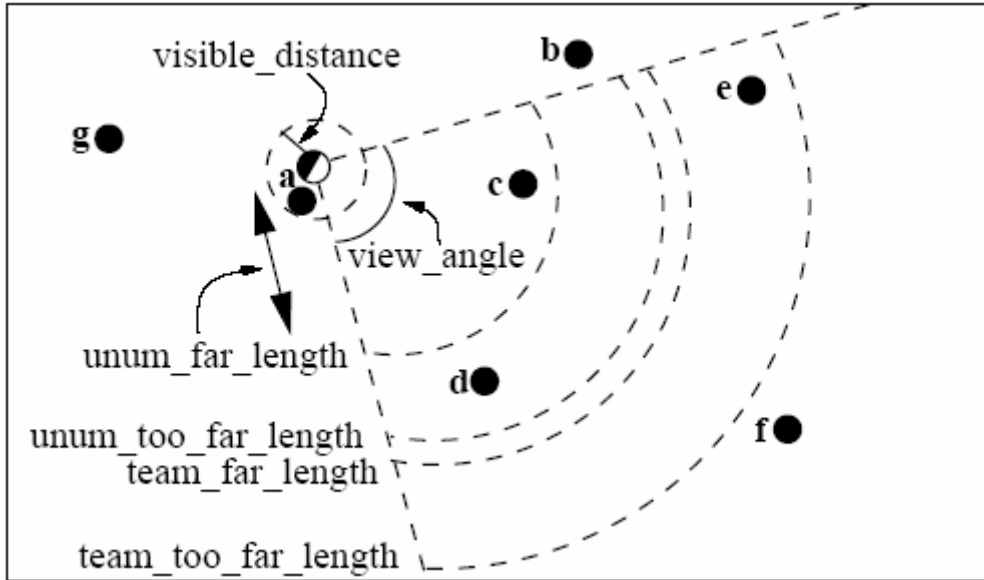
球员所获得的物体视觉信息的完整性由物体的距离所决定。球员到各种可视物体的距离 dist 与可以获得的物体信息的完整性有如下关系(以下讨论默认 ViewQuality=high, 因为如果 ViewQuality=low, 则视觉信息 ObjInfo 只包括 ObjName 和 Direction):

- 如果可视对象是 landmark(标志, 边线, 球门等), 则视觉信息中总是包括 landmark 的名字、到 landmark 的距离和 landmark 的相对角度。注意如果可视对象是边线, 那么表示比较特殊, Distance 表示的是球员视野角平分线与边线交点到球员的距离, 而 Direction 表示球员视野角平分线和边线所成的角度。(图表示)
- 如果可视对象是球员 p, 则距离 dist 和获得信息之间的关系为:
 - 对于任何距离, 视觉信息中总是包括球员的 Distance 和 Direction。
 - 如果 $dist \leq unum_far_length$, 那么球员号码和球队名称都可见。同时, 视觉信息中会包括所有 DistChange, DirChange, BodyDir, NeckDir 的值。
 - 如果 $unum_far_length < dist \leq unum_too_far_length = team_far_length$, 那么队名是可见的; 但是队员号码只有一定的概率可以看到, 这个概率根据 dist 是线性的从 1 到 0 减少的; DistChange, DirChange, BodyDir, NeckDir 也根据这个概率可能获得。
 - 如果 $dist > unum_too_far_length = team_far_length$, 那么球员号码是不可见的, 而且视觉信息中也不会包括 DistChange, DirChange, BodyDir, NeckDir。
 - 如果 $team_far_length < dist < team_too_far_length$, 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 $dist > team_too_far_length$, 那么队名是不可见的, 球员只被识别为一个匿名球员。
- 如果可视对象是球 b, 则距离 dist 和获得信息之间的关系为:
 - 对于任何距离, 视觉信息中总是包括球的 Distance 和 Direction。
 - 如果 $dist \leq unum_far_length$, 视觉信息中包括了球的 DistChange, Dirchange 信息。
 - 如果 $unum_far_length < dist \leq unum_too_far_length = team_far_length$, 视觉信息中包括 DistChange, Dirchange 信息的概率随着 dist 的增加从 1 线性减小到 0。



- 如果 $\text{dist} > \text{num_too_far_length} = \text{team_far_length}$, DistChange, Dirchange 信息变得总不可见。

在当前版本的 Server 中这些距离的关系是 $\text{unum_far_length} \leq \text{unum_too_far_length} \leq \text{team_far_length} \leq \text{team_too_far_length}$ 。

以上三点我们来看可以通过图。。。来看一下具体的例子：



• 图 4.3 物体距离与信息完整性关系

图中的  是我们考察的球员（浅色的半圆表示球员的前部）， 表示场上的其他球员。根据球员周围虚线表示的视野、距离等，我们可以判断：对于球员 a，处于视野外但是在球员邻域中，所以只能识别类型为球员“P”；对于球员 b 和 g，彻底处于视野外，对于球员不可见；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，虽然也处于视野中，但只能识别为一个匿名的球员了（DistChange, DirChange, BodyDir, NeckDir 等信息与此类似）。

信息中的噪声影响

Soccer Server 仿真真实世界复杂性的一个重要部分是随着距离增加，视觉信息的精确度会降低。这是通过在视觉感知数据中引入噪声来实现的。如果可视目标是球或者球员，则获得的目标距离值按如下方式进行量化计算：

$$d' = \text{Quantize}(\exp(\text{Quantize}(\ln(d), \text{StepValue})), 0.1) \quad (4.15)$$

$$\text{Quantize}(V, Q) = \text{ceiling}(V/Q) \cdot Q \quad (4.16)$$

其中 d ， d' 分别表示精确距离和相应获得的距离值，ceiling 表示一个取整函数。式中的 StepValue 表示量化的步长。如果目标是球或者球员，StepValue 使用 Server 中的参数 quantize_step；而如果目标是场上的标志或者边线，则 StepValue 使用参数 quantize_step_1。由上式可知队员是不能知道物体的精确位置的，距离越远噪声导致的误差会越大。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 DistChange, Direction 和 DirChange 则使用如下公式：

$$DistChange' = d' \cdot Quantize(DistChange / d, 0.02) \quad (4.17)$$

$$Direction' = Quantize(Direction, 1.0) \quad (4.18)$$

$$DirChange' = Quantize(DirChange, 0.1) \quad (4.19)$$

表 4.1 视觉模型有关参数和缺省值

Parameter	Value	Parameter	Value
send_step	150	team_far_length	40.0
visible_angle	90.0	team_too_far_length	60.0
visible_distance	3.0	quantize_step	0.1
unum_far_length	20.0	quantize_step_l	0.01
unum_too_far_length	40.0		

听觉模型

听觉感知用来接收其他球员或者教练通过 `say` 命令发送的消息。裁判广播的比赛信息也作为听觉信息被球员接收。Soccer server 中的听觉通讯仿真的是一个拥挤的低带宽环境，其中的双方所有球员共用一个不可靠频道。一个球员通过 `say` 发送的信息会立刻广播到一定范围内双方所有球员，不存在接收延迟。

从 server 接收的听觉消息的格式如下：

```
(hear Time Sender "Message")
```

其中 *Time* 表示 server 发送听觉信息时所处的仿真周期；*Message* 就是听觉消息的内容，听觉信息最长允许使用 `say_msg_size`（缺省为 512）个字节；*Sender* 根据信息发送者的不同有以下几种可能取值：

self: 信息的发送者是自己本人。

referee: 信息的发送者是裁判，从裁判处可能获得的听觉信息具体见。。。章节。

online_coach_left 或者 **online_coach_right:** 信息的发送者是在线教练。

Direction: 如果听觉信息的发送者其他球员，那么是 *Sender* 是发送者的相对方向，而不是信息具体的发送者标识。

为了仿真拥挤的低带宽不可靠频道，每个球员都具有一个的有限的听觉能力，最大值为 Server 参数 **hear_max**。每次球员收到一条听觉信息，它的听觉能力就会下降 **hear_decay**；而每个周期球员的听觉能力会增加 **hear_inc**。听觉能力不能为负数，所以球员的听觉能力至少有 **hear_decay** 时才能收到听觉信息。由于两支球队所有 22 个球员共用同一个通讯频道，如果一方使用大量的信息来占用这个频道，使对方球员的听觉能力下降，就可能导致对方无法正常的通讯；为了避免这种情况的发生，听觉模型中球员对两支球队的听觉能力被分离开各自计算。根据当前版本的听觉模型相关参数缺省值，一个球员最快每两个仿真周期才能从每支球队接收一条听觉信息。

如果一个周期有多个听觉信息可以到达，那么球员只能接收到其中随机的一个（当前是根据到达的先后顺序选取最早到达的），其他的会被 server 直接丢弃而不会发送到球员，因此听觉通讯是完全不可靠的；另外，球员的通讯范围是有限的，一个消息只能到达距离消息发送者小于 **audio_cut_dist** 的球员。但是裁判的广播信息具有特权，不受听觉能力和距离的限制，而总是能被发送到场上每个球员。

听觉模型的特征给球员之间的通讯带来很多问题。由于双方共用同一频道，队友间需要识别公有频道上的消息是发给哪个球员的；同时球员必须能够处理对方球员的干扰，例如，对方球员可能会记录听到的信息并延迟随机的时间后再发送，如果不能识别这些干扰，则球员可能会做出错误的行为决策。由于不同球员会看到场上不同的部分，队友间的通讯可以提高球员世界模型的可靠和准确性，但是由于听觉信息仿真的一个不可靠的频道，每个球员

应该能够不依靠通讯独立维护一个近似精确的世界模型。由于足球要求分工协作，队友间需要使用通讯来协商和调整战术计划，但是由于听觉信息不可靠，球员间必须通过某些机制来保证球队的球员都能使用相同的战术，同时也要求球员能够不依靠通讯，独立决策行为进行比赛。

表 4.2 听觉模型有关参数和缺省值

Parameter	Value	Parameter	Value
Say_msg_size	512	Hear_decay	2
Hear_max	2	Audio_cut_dist	50.0
Hear_inc	1		

身体感知模型 sense_body

身体感知信息球员当前的状态。Soccer server 每隔 sense_body_step（目前使用 100ms）就会自动的向球员发送身体感知信息。

从 server 发送的身体感知信息按照下面的格式：

```
(sense_body Time
  (view_mode ViewQuality ViewWidth)
  (stamina Stamina Effort)
  (speed AmountOfSpeed DirectionOfSpeed)
  (neck_angle NeckDirection)
  (kick KickCount)
  (dash DashCount)
  (turn TurnCount)
  (say SayCount)
  (turn_neck TurnNeckCount)
  (catch CatchCount)
  (move MoveCount)
  (change_view ChangeViewCount)
)
```

其中 *Time* 表示 server 发送身体感知信息时所处的仿真周期；ViewQuality 表示当前球员的视觉质量，取值是 high 或 low；ViewWidth 表示当前球员的视角范围，取值为 narrow, normal, wide 之一（见视觉模型）；Stamina 和 Effort 表示当前球员的体力信息（见 4.3.2 加速和体力模型）；AmountOfSpeed 是当前球员速度大小的近似值，它在球员当前真实速度 Speed 上使用公式 $AmountOfSpeed = Quantize(Speed, 0.01)$ 量化加入了噪声误差的影响，DirectionOfSpeed 是球员速度相对于球员头部的近似方向；NeckDirection 是球员头部与身体之间角度的近似值，它们都是在真实角度上使用公式 4.18 加入了噪声误差的影响；变量 Count 是截至到当前周期，server 已经执行的对应命令的次数总量。例如，DashCount=134，说明球员已经成功执行了 134 次 dash 命令。通过检查这些命令计数是否增加，可以判断球员上一周期发送到 server 的命令是否被成功执行。

身体感知模型中包含的信息可以用来更新世界模型，在需要使用上述参数时会给出更详细的说明。

表 4.3 身体感知模型有关参数和缺省值

Parameter	Value
Sense_body_step	100

4.2 复杂世界模型的构建

由于仿真平台的特性，只依靠 **server** 感知信息维护世界模型是远远不够的（见 3.4.2），我们需要使用更复杂的世界模型。

4.2.1 运动模型

在 **soccer server** 中，物体的运动是通过一个简单的逐步计算仿真的。每个仿真周期，**mobile** 物体通过下面的公式计算：

$$(u_x^{t+1}, u_y^{t+1}) = (v_x^t, v_y^t) + (a_x^t, a_y^t) + (\tilde{r}_1, \tilde{r}_2) + (w_1, w_2) : \text{加速度} \quad (4.17)$$

$$(p_x^{t+1}, p_y^{t+1}) = (p_x^t, p_y^t) + (u_x^{t+1}, u_y^{t+1}) : \text{位置变化} \quad (4.18)$$

$$(v_x^{t+1}, v_y^{t+1}) = \text{Decay} \times (u_x^{t+1}, u_y^{t+1}) : \text{速度衰减} \quad (4.19)$$

$$(a_x^{t+1}, a_y^{t+1}) = (0, 0) : \text{加速度重置} \quad (4.20)$$

其中 (p_x^t, p_y^t) 、 (v_x^t, v_y^t) 、 (a_x^t, a_y^t) 分别表示物体在周期 t 时的位置，速度和加速度向量。

$(\tilde{r}_1, \tilde{r}_2)$ 、 (w_1, w_2) 分别表示物体移动中受到的噪声向量和风向量影响。*Decay* 是一个 **server** 参数，用来表示物体速度的衰减，对于球员，*Decay* 等于 **player_decay**，而对于球，则等于 **ball_decay**。物体的加速度是由球员发送到 **server** 的命令作用产生的，例如球员的加速度由球员使用 **dash** 命令产生，而球的加速度是由球员成功使用的 **kick** 命令产生，命令对加速度的影响将在相应的行为模型中具体介绍。如果一个周期结束时两个对象位置发生重合，那么会把对象按照原来的运动方向后移使其不再重叠。然后两者的速度都乘以 -0.1 。注意，由于仿真平台使用离散的仿真周期来模拟实时环境，根据运动模型，足球有可能穿过球员，只要在周期的末尾，足球和球员没有冲撞就行了。

为了仿真真实世界中球及球员运动的不确定性，**soccer server** 在运动模型中所有物体的运动上加入了随机的噪声向量。在 (4.17) 式中，噪声向量包括两个随机数 \tilde{r}_i 。 \tilde{r}_i 取值范围为 $[-r_{\max}, r_{\max}]$ 。其中 r_{\max} 受物体速度和加速度的影响，通过下式计算：

$$r_{\max} = \text{Rand} \cdot \left\| (v_x^t, v_y^t) + (a_x^t, a_y^t) \right\| \quad (4.21)$$

其中 **Rand** 是一个 **server** 参数，用来表示物体运动中随机的噪声影响，对于球员 **Rand** 取值 **player_rand**，而对于球则取值 **ball_rand**。由公式 4.21 可知，物体的速度和加速度越大，运动中可能产生的噪声误差也越大。

另外，**soccer server** 还采用了一个风力向量来模拟一个自然中另一类影响。风在仿真模型中表示为向量 $(w_x, w_y) = \pi(\text{wind_force}, \text{wind_dir})$ ，其中 π 把极坐标转换为直角坐标。

而风力对物体运动的影响受物体质量的影响，具体的风力对速度影响的向量 (w_1, w_2) 根据下式计算：

$$(w_1, w_2) = \left\| (v'_x, v'_y) + (a'_x, a'_y) + (\tilde{r}_1, \tilde{r}_2) \right\| \cdot \frac{(w_x + \tilde{e}_1, w_y + \tilde{e}_2)}{\text{Weight} \cdot 10000} \quad (4.22)$$

其中 \tilde{e}_i 是 $[-\text{wind_rand}, \text{wind_rand}]$ 中的一个随机数，而 **Weight** 一个用来表示物体重量的 **server** 参数，对于球员取值为 **player_weight**，而对于球则取值 **ball_weight**。需要注意的是当前版本的 **server** 中，所有和风有关的参数都设置为 0，就是说缺省情况下物体运动没有风的影响。

表 4.4 运动模型有关参数和的缺省值。

Parameter	Value	Parameter	Value
Ball_decay	0.94	Palyer_weight	60.0
Ball_rand	0.05	Wind_force	0.0
Ball_weight	0.2	Wind_dir	0.0
Player_decay	0.4	Wind_rand	0.0
Player_rand	0.1		

4.3 基本行为模型

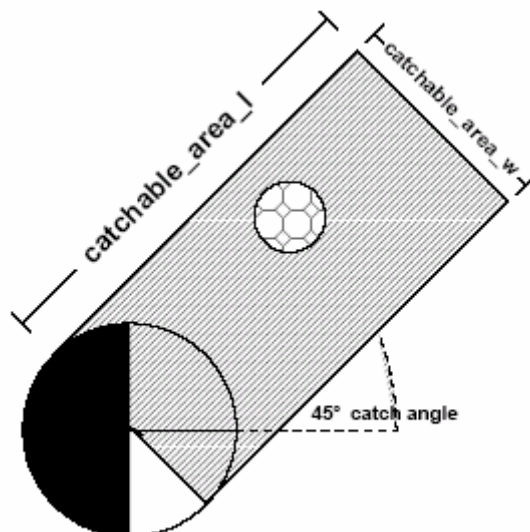
当希望执行一个动作时，球员会发送一个命令到 **server**，请求 **server** 执行，**server** 则根据行为模型来仿真动作结果，实现球员对周围环境的作用。

执行一个行为，主要包括三个部分：行为前状态，执行的行为，行为后结果。**Server** 行为模型提供了三者之间的数学转换关系，即球员行为的仿真数学模型。根据给定的 **server** 行为模型，如果已知其中的两个部分，就可以推知第三个部分。因此，行为模型主要用于三种用途。第一，行为结果预测，即计算执行一个给定动作后物体的状态，这个模型接收一个行为命令作为参数，返回球员执行这个命令后场上物体状态（如自身全局位置，速度，身体角度，头部角度，体力或者球的速度，方向等）的预测。这些预测是通过仿真 **server** 的运动模型和行为模型来计算获得的。行为结果预测对于维持 **wm** 的完整性和行为的决策是很重要的。第二，简单行为生成，即根据要得到的结果，计算发送的命令中使用的参数值，生成相应的命令信息。第三，行为前状态推测。这个与第一种用途属于逆向使用，即知道当前状态和上一个行为，求行为前的状态，例如 **sense_body** 信息的使用等。

4.3.1 扑球模型 catch

守门员可以通过发送 **catch** 命令到 **server** 来进行扑球动作。**Catch** 命令只有一个参数 **Direction**，用来表示守门员扑球的角度，**Dirction** 以度数作为单位，取值必须介于 **minmoment** 和 **maxmoment** 之间。

当有 **catch** 命令到达 **server**，**server** 会检查一些前提条件是否成立，来决定扑球动作是否能够成功。这些条件包括：首先，守门员是唯一可以使用 **catch** 命令的球员。其次，当前比赛模式必须是 **play_on**，最后，足球在守门员扑球范围和守门员本队禁区内。如果 **catch** 发送到 **server** 时这些条件不成立，则这次扑球会失败。守门员的扑球范围是在它扑球方向上长宽分别是 **catchable_area_l** 和 **catchable_area_w** 的矩形区域，图 4.4 显示了守门员使用 -45° 为参数使用 **catch** 命令时的扑球范围。



• 图 4.4 守门员扑球范围，使用命令(catch -45)

如果 catch 命令执行时，球处于守门员的有效扑球范围中，那么守门员成功抓到球的概率是 `catch_probability`。在真实足球比赛中，如果守门员扑球没有成功，通常都需要一定时间调整后才能再一次扑球（例如倒地后站起再次扑球）。类似地，在 `soccer server` 中，如果一次扑球没有成功，守门员必须经过 `catch_ban_cycle` 个仿真周期才能执行下一次 catch 命令，这个时段内发送的 catch 命令将被 server 忽略不执行。如果守门员成功的抓到了球，裁判会把比赛模式在一个时间周期内先变为 `goalie_catch_ball_x`，再成为 `free_kick_x`（其中 `x` 为 1 或者 `r`，用来表示抓球成功的守门员一方）。一旦守门员成功抓到球，在使用 kick 命令把球踢走前，可以使用 move 命令在禁区内持球移动 `goalie_max_moves` 次（move 命令使用见 4.3.5），更多的 move 命令 server 将会返回错误信息(error too_many_moves)而不再执行。

表 4.5 扑球模型 catch 有关参数和缺省值。

Parameter	Value	Parameter	Value
Minmoment	-180	Catch_probability	1.0
Maxmoment	180	Catch_ban_cycle	5
Catchable_area_l	2.0	Goalie_max_moves	2
Catchable_area_w	1.0		

4.3.2 踢球模型

当一个球员想要踢球时，它必须发送一个 kick 命令到 server。这个命令包含两个参数：踢球的力量 `Power` 和踢球的角度 `Direction`。踢球的力量用来决定对球加速的大小，取值必须介于 `minpower` 和 `maxpower` 之间；踢球的角度 `Direction` 是相对于球员身体方向的角度，以度数来表示，取值介于 `Minmoment` 和 `Maxmoment` 之间。

当 kick 命令到达 server，server 会对当前状态进行判断。如果球员没有处于越位的位置，并且球在球员的控球范围内，那么 kick 命令就会被执行。一个球员的最大控球范围定义为半径为 `ball_size+player_size+kickable_margine` 的圆，也就是说，当代表球和球员的圆边界之间的距离小于 `kickable_margine` 的时，球员可以踢到球。

踢球模型中一个重要的方面是，球获得的有效力量并不一定等于 kick 命令中使用的 `power` 参数，而是要受到球和球员的相对位置的影响。作用在球上的实际有效力量可以通过下面的公式计算：

$$act_pow = Power \cdot \left(1 - 0.25 \cdot \frac{dir_diff}{180} - 0.25 \cdot \frac{dist_diff}{kickable_margin} \right) \quad (3.26)$$

这里 $Power$ 是 $kick$ 命令中使用的参数， dir_diff 是球和球员身体方向之间的绝对角度， $dist_diff$ 是球和球员之间的距离（球员和球圆边界之间的距离）。根据公式，当球在球员身体朝向的正前方时，它对力量没有影响，而角度越大，实际有效的力量就越小，当最坏情况时，即球在球员的身后时，力量会减小 25%。球与球员距离对力量的影响与此类似。

踢球的有效力量被用来计算球获得的加速度 \vec{a}_t ：

$$(a'_x, a'_y) = act_pow \times kick_power_rate \times (\cos(\theta^t), \sin(\theta^t)) + (\tilde{k}_1, \tilde{k}_2) \quad (3.27)$$

其中 $kick_power_rate$ 是一个 $server$ 的参数， θ^t 是周期 t 中球获得的加速度的全局方向：

$$\theta^t = \theta_{kick_player}^t + Direction, \quad \text{其中 } \theta_{kick_player}^t \text{ 表示踢球球员周期 } t \text{ 时的全局方向,}$$

$Direction$ 是 $kick$ 命令中使用的第二个参数。

$(\tilde{k}_1, \tilde{k}_2)$ 的加入是为了在足球加速度中制造噪声。 \tilde{k}_i 是在 $[-k_{\max}, k_{\max}]$ 之间的一个随机

数，其中 k_{\max} 根据 $kick$ 命令中的 $Power$ 来计算：

$$k_{\max} = kick_rand \cdot \frac{Power}{max_power} \quad (3.28)$$

$Kick_rand$ 是 $server$ 中的一个参数。当前版本中，普通球员的 $kick_rand$ 值缺省为 0.0，即普通球员踢球时没有噪声影响；但是对于异构球员则不同， $kick_rand$ 值可能不为 0（见 4.4 异构球员）。

根据运动模型，从仿真周期 t 到仿真周期 $t+1$ ，球的加速度 \vec{a}_t 被用来更新球的信息：

1. \vec{a}_t 向量被规格化为最大是 $ball_accel_max$ 的值。

2. \vec{a}_t 被加到足球的当前速度向量 \vec{v}_t ，得到的 \vec{v}_t 向量将被规格化到最大是 $ball_speed_max$ 的值。

3. 速度向量 \vec{v}_t 规格化后，噪声 \vec{r} 和风力 \vec{w} 的影响也按照运动模型公式(4.17)增加到 \vec{v}_t 。噪声和风力具体说明见 4.2.1。

4. 更新足球的位置信息，足球的新位置 \vec{p}_{t+1} 是原先的位置 \vec{p}_t 加上速度 \vec{v}_t ：

$$\vec{p}_{t+1} = \vec{p}_t + \vec{v}_t。$$

5. $ball_decay$ 被应用到足球的速度中： $\vec{v}_{t+1} = \vec{v}_t \cdot ball_decay$ 。加速度 \vec{a}_{t+1} 被设为零。

在当前版本的 $server$ 中， $ball_accel_max$ 和 $kick_power_rate$ 参数的默认设置符合如下关

系 $\text{ball_accel_max} = \text{maxpower} \cdot \text{kick_power_rate}$ ，即当球处于最优位置（球员身体正前方，同球员边界相连），球员使用最大力量执行 `kick` 命令，可以使球达到可获得的最大加速度；同时由于 `ball_accel_max` 和 `ball_speed_max` 参数的默认值相等，因此只要一个 `kick` 命令就可能使足球达到最大速度。而在之前版本的 `server` 中，这是不可能的，为了获得球的最大速度，必须进行多次组合的踢球动作。但即使在现有参数设置下，由于球相对于球员的位置和踢球角度等会对球的加速度产生影响，由多次 `kick` 组成的复合踢球还是有用的，例如，停住球，把球踢到控制范围中一个更有利的点，然后踢向一个希望的方向（见 8.1）。

表 4.6 踢球模型 `kick` 有关参数和缺省值。

Parameter	Value	Parameter	Value
Manpower	-100	Kickable_margin	0.7
Maxpower	100	Kick_power_rate	0.027
Minmoment	-180	Kick_rand	0.0
Maxmoment	180	Ball_accel_max	2.7
Ball_size	0.085	Ball_speed_max	2.7
Player_size	0.3		

4.3.3 加速模型和体力模型

`dash` 命令被用来使球员按照它的身体方向进行加速。这个命令只有一个参数：加速力量 `power`。这个参数决定球员加速的大小，取值必须介于 `[minpower,maxpower]` 之间。当 `power` 是正数时，球员向前加速，相反，当 `power` 为负数时，球员向后加速。

为了防止球员总是以最大的速度奔跑，参照实际状况，`server` 采用了体力模型的限制。在体力模型中，每个球员都有一定有限的体力，会在执行加速命令 `dash` 时减少，同时，每个仿真周期会微弱恢复。体力模型包含三个部分：`stamina`，`effort`，`recovery`。体力值 `stamina` 代表球员当前剩余的体力，它介于 `[0,stamina_max]` 之间，执行 `dash` 命令会消耗体力。每个半场开始时，球员体力被置为缺省值 `stamina_max`。如果球员向前加速（`power>0`），体力值减少 `power`；如果是向后加速，体力值减少 2 倍 `power` 的绝对值，即向后跑相对更加费力。如果 `dash` 命令包含的参数 `power` 超过了当前剩余的体力值，`server` 就会将 `power` 相应降低为当前的剩余体力；作用 `effort` 代表球员加速的效率，介于 `[effort_min,effort_max]` 之间；恢复 `recovery` 代表球员体力恢复的速率，介于 `[recovery_min,1.0]` 之间。`Soccer server` 中这三个部分的具体算法如下：


```

//reduce stamina according to dash power
if Power<0 then
    Stamina = Stamina - 2·|Power|
else
    Stamina = Stamina - Power
end if

//if stamina is below recovery decrement threshold, recovery is reduced
if Stamina ≤ recover_dec_thr·stamina_max then
    if Recovery > recover_min then
        Recovery = Recovery - recover_dec
    end if
    Recovery = Max(recover_min,Recovery)
end if

//if stamina is below effort decrement threshold, effort is reduced
if Stamina ≤ effort_dec_thr·stamina_max then
    if Effort > effort_min then
        Effort = Effort - effort_dec
    end if
    Effort = Max(Effort, effort_min)
end if

//if stamina is above effort increment threshold, effort is increased
if Stamina ≥ effortc_inc_thr·stamina_max then
    if Effort < effort_max then
        Effort = Effort + effort_inc
    end if
    Effort = Min(effort_max,Effort)
end if

//recover the stamina a bit
Stamina =Min(stamina_max, Stamina + Recovery·stamina_inc_max)

```

- 代码 4.1

与 kick 模型类似，球员加速的实际有效力量并不一定等于 dash 的参数 power，而是受到球员当前体力状态的影响：如果 dash 命令到达 server 使用的参数 Power 超过了球员的当前剩余体力值，那么 Power 就会降低，然后提交命令球员的体力值减去 dash 的 Power 参数（如果 Power<0，则 2*|Power|），server 根据当前 effort 计算出实际有效加速力量：

$$act_pow = Effort \cdot Power \quad (3.29)$$

当前版本中最大 effort 的值是 1 (effort_max)，因此只要球员能够维持他的体力使 effort 不减少，那么实际有效力量就同参数 power 值相等。

有效力量被用来计算球员的加速度：

$$(a_x^t, a_y^t) = act_pow \times dash_power_rate \times (\cos(\theta^t), \sin(\theta^t)) \quad (3.30)$$

球员信息在 server 中更新的过程与 4.3.2 节中球信息更新的过程类似。

注意由运动模型公式 (4.20)，一个 dash 命令只会在一个仿真周期带来加速，之后球员的速度就会开始衰减，球员的加速度在下次执行 dash 命令之前都是 0。为了保持持续的速度，球员必须不断的使用 dash 命令。

表 4.7 加速和体力模型有关参数和缺省值。

Parameter	Value	Parameter	Value
Manpower	-100	Effort_inc_thr	0.6
Maxpower	100	Effort_inc	0.01
Stamina_max	4000	Recover_dec_thr	0.3
Stamina_inc_max	45.0	Recover_dec	0.002
Effort_min	0.6	Recover_min	0.5
Effort_max	1.0	Player_accel_max	1.0
Effort_dec_thr	0.3	Player_speed_max	1.0
Effort_dec	0.005	Dash_power_rate	0.006

4.3.4 调整方向模型

转身

命令 turn 用来改变球员的身体方向。这个命令只有一个参数：角度 Moment，表示球员要转身的角度；Moment 用度数为单位，取值必须介于 minmoment 和 maxmoment 之间。

如果球员没有运动，那么它实际转过的角度将等于 turn 的参数 Moment 的值；然而，如果球员在运动，惯性的作用会使其转身较为困难，因此，与前两个行为模型类似，球员实际转过的角度并不等于参数 Moment 的值。球员真正转过的角度使用公式：

$$act_ang = \frac{(1.0 + \tilde{r}) \cdot Moment}{1.0 + inertia_moment \cdot player_speed} \quad (3.31)$$

其中 \tilde{r} 是在 $[-player_rand, player_rand]$ 范围中的一个随机数，Moment 是 turn 命令使用的参数，inertia_moment 是 server 中用来表示球员惯性大小的一个参数，player_speed 表示转身球员的当前速度。在当前版本中，参数 inertia_moment 被设置为 5.0，根据上式，当球员的速度是 1.0 (player_speed_max) 时，它能够做到的最大有效转身角度是 $\pm 30^\circ$ 。但是，因为球员不能在同一个周期同时执行 dash 和 turn 两个命令，因此球员执行 turn 时的最大速度最多是 $player_speed_max \cdot player_decay = 1.0 \cdot 0.4 = 0.4$ ，这就意味着普通球员的最大有效转身角度（使用默认值）是 $\pm 60^\circ$ 。

表 4.8 转身模型有关参数和缺省值

Parameter	Value	Parameter	Value
Minmoment	-180	Player_rand	0.1
Maxmoment	180	Inertia_moment	5.0

转头

使用 `turn_neck` 命令，球员可以独立于球员身体来转动头部。`turn_neck` 只有一个参数：角度 **Moment**，表示球员转头的角度，**Moment** 用度数为单位，取值必须介于 `minneckmoment` 和 `maxneckmoment` 之间。球员的头部绝对方向也是他的视野方向，命令 `turn_neck` 改变球员头部相对他的身体的角度，同时也就改变了它的视野方向。注意头部角度是相对于球员身体的相对角度，如果不使用 `turn_neck` 命令，这个相对角度不变，即身体方向的变化也会导致头部绝对方向的变化，因此即使没有执行 `turn_neck` 命令，如果球员执行了 `turn` 命令，球员的视野方向也是会改变的。

与真实世界相同，球员不能无限制的朝一个方向转头，球员头部相对于身体的角度必须介于 `minneckang` 和 `maxneckang` 之间。如果一个 `turn_neck` 命令的 **Moment** 参数导致球员头部相对身体的角度超出这个范围，`server` 会把实际的头部相对角度调整到合法范围以内。

注意 `turn_neck` 命令与前几个行为命令两点区别，第一是虽然每个仿真周期只能执行一个 `turn_neck` 命令，但是 `turn_neck` 命令可以同 `kick`，`dash` 或者 `turn` 等其他命令在同一个仿真周期执行。第二是只要在合法范围内，`turn_neck` 命令实际转头的角度总是等于它的参数 **Moment**，没有噪声的影响，也不会象 `turn` 一样收到速度等其他因素的影响。

表 4.9 转头模型 `turn_neck` 有关参数和缺省值。

Parameter	Value	Parameter	Value
Minneckmoment	-180	Minneckang	-90
Maxneckmoment	180	Maxneckang	90

4.3.5 移动模型 move

命令 `move` 可以把球员直接移动到场上的任何一个地方。命令 **Move** 有两个参数：**X** 和 **Y**，用来表示目标位置的全局坐标值。由于目标位置是在场上，所以 **X** 的值必须介于 `[-pitch_length/2, pitch_length/2]` 之间，而 **Y** 必须介于 `[-pitch_width/2, pitch_width/2]` 之间。

正常比赛进行时（`playmode` 为 `play_on`）`move` 命令是不能使用的，它主要在两种情况下使用。第一种是在上下半场开始前（比赛模式是 `before_kick_off`）和进球后（比赛模式是 `goal_r_n` 和 `goal_l_n`）来设置球队基本队形。在这种情况下，只要比赛模式没有改变，球员可以被移动到自己半场的任何地点（就是说 $x < 0$ ），而且可以被移动任意多次。如果球员试图使用 `move` 命令移动到对方半场的话，那么将会被 `server` 移动到己方半场的随机位置。第二种情况发生在守门员成功抓到足球后（4.3.1 `catch` 命令）。如果守门员成功抓到（`catch`）了球，就可以使用 `move` 命令在禁区内持球移动，来寻找合适的传球机会。为了避免守门员过多的拖延比赛，在他踢球前只可以移动 `goalie_max_moves` 次，更多的 `move` 命令将不起任何作用，而且 `server` 会返回(error too many moves)信息。注意，如果守门员 `catch` 到足球，然后 `move`，再 `kick` 足球很近的一段距离，接着马上重新 `catch`，就可以绕开 `goalie_max_moves` 的限制，从而实现持续在禁区中安全地控球。为了避免这种问题，类似的用法被定义为守门员滥用 `catch` 命令，虽然自动裁判不能判定，但是比赛中的人类裁判可以判给对方任意球。

表 4.10 移动模型 `move` 有关参数和缺省值。

Parameter	Value	Parameter	Value
-----------	-------	-----------	-------

Pitch_length	52.5	Goalie_max_moves	2
Pitch_width	34.0		

4.3.6 通讯模型 say

球员可以使用 say 命令对其他球员进行广播消息。这个命令只使用一个参数：球员想要通讯的消息 Message。Message 是一个字符串，最大长度是 say_msg_size（缺省为 512）字节，可以使用有效字符在集合 [0..9a..zA..Z().+_*?_<>] 中（不包括方括号）。球员有一个有限的通讯传播范围：每个消息只能被到消息发送球员的距离小于 audio_cut_dist 的球员接收到。球员“说”的消息没有感知延迟，立刻传到可听范围内的两方所有球员。但是，球员有一个有限的听觉能力，这通过 server 的参数 hear_max、hear_inc 和 hear_decay 来计算（4.1.2 听觉模型）。根据当前 server 的缺省值，每个球员两个周期只能听到一个队友的一条消息。也就是说，虽然 server 对球员使用 say 命令发送消息的频率没有限制，甚至一个仿真周期中可以进行多次，但是实际使用超过球员听觉能力的频率是没有意义的。

表 4.11 通讯模型 say 有关参数和缺省值。

Parameter	Value	Parameter	Value
Say_msg_size	512	Hear_inc	1
Audio_cut_dist	50.0	Hear_decay	2
Hear_max	2		

4.3.7 其他行为模型

铲球模型

球员可以通过执行 tackle 指令来踢到用 kick 指令踢不到的球，铲球成功判定取决于一个与球位置相关的随机概率：

$$\text{fail_prob} = (\text{player_2_ball.x} / \text{tackle_dist})^{\text{tackle_exponent}} + (| \text{player_2_ball.y} | / \text{tackle_width})^{\text{tackle_exponent}}$$
或者

$$\text{fail_prob} = (\text{player_2_ball.x} / \text{tackle_back_dist})^{\text{tackle_exponent}} + (| \text{player_2_ball.y} | / \text{tackle_width})^{\text{tackle_exponent}}$$

这里当球在球员前面时是用 tackle_dist（缺省为 2.0），否则用 tackle_back_dist（缺省为 0.5）；Player_2_ball 是一个从球员到球的向量，相对于球员身体方向。当铲球指令成功，将给球沿自己身体方向一个加速度。

当该球员被其他球员看见时，他们也可以通过一个标志 ‘t’ 看到该球员的铲球状态，如下面的格式：

((p "<TEAMNAME>" <UNUM>) <DIST> <DIR> <DISTCHG> <DIRCHG>
<BDIR> <HDIR> [<POINTDIR>] [t])

((p "<TEAMNAME>") <DIST> <DIR> [<POINTDIR>] [t])

球员自己可以根据 sense body 信息里的胳膊信息了解自己的铲球状态，如下所示：

(tackle (expires <EXPIRES>) (count <COUNT>))

其中 EXPIRES 是铲球持续的时间，0 表示没有铲球；COUNT 该球员执行铲球的次数。



WrightEagle

Tackle 的执行效果类似于 kick，由参数 tackle_power_rate（缺省为 0.027）与 power 相乘得到。表 4.12 给出了 tackle 用到的参数。

表 4.11 铲球模型 tackle 有关参数和缺省值。

Parameter	Value	Parameter	Value
tackle_dist	2.0	tackle_cycles	10
tackle_back_dist	0.5	tackle_exponent	6
tackle_width	1.0	tackle_power_rate	0.027

Pointto

Attentionto

4.3.8 行为命令总结

表 4.12 仿真命令总结

Syntax	Arguments	executed	Frequency limit
(kick double double)	Power,direction	End of cycle	One of these per cycle
(dash double)	Power	End of cycle	
(turn double)	moment	End of cycle	
(tackle int)	power	End of cycle	
(move double double)	(x,y) position	End of cycle	
(catch double)	angle	End of cycle	
(say str)	Message string	Instantly	None
(attentionto str int off)	team unum	Instantly	None
(pointto double double off)	dist dir	Instantly	None
(turn_neck double)	Moment	End of cycle	One per cycle
(change_view str str)	Width,quality	Instantly	One per cycle
(score)	-	instantly	none
(sense_body)	-	instantly	-

4.4 其他模型

裁判模型与 play mode（怀疑和 2.2 会重复。。。）

为了使比赛按照规则进行，soccer server 仿真平台包含了一个自动的裁判来控制比赛。裁判根据场上不同情况调整比赛模式 **play mode**。每次比赛模式发生改变，裁判都会自动的向所有球员发送听觉信息来声明。这样每个球员就可以知道当前的比赛模式并依次作出决策。另外，裁判还要声明一些重要事件，例如进球或者犯规。下面是一些情况下裁判使用的规则和采取的行为：

开球 Kick_Off

每次开始比赛或者是有一方进球后，会进行开球。在 **kick_off** 之前，所有的球员都必须在她自己的半场。为了能够达到这样，在每次进球得分后，裁判把比赛挂起 5s 时间。在这个间隔中，球员可以使用 **move** 命令移动到某点，而不是跑向这个点，这样会既费时又费体力 **stamina**。如果在 5s 结束后，球员还呆在对方的半场，或者在这段时间中试图 **move** 到对方半场，裁判会把它移到自己半场的随机位置。

进球 Goal

如果一个球队进球得分，裁判要作很多事情。首先，她向所有的球员广播进球信息。然后更新比分，将球移到中点，并把比赛模式置为 **kick_off_x**（x 是 l 或 r，代表左半场球队或右半场球队）。最后，裁判将比赛挂起 5s，以便让球员在此期间回到自己的半场（如在“开球 Kick-Off”节所述）。

出界 Out of Field

当足球出界时，裁判把足球放到一个合适的位置，并且根据情况把比赛模式置为界外球 `kick_in`、角球 `corner_kick` 或者球门球 `goal_kick` 中的一种。如果是界外球，则裁判把球放在边线上球出界的地方；如果是角球，裁判将把足球放到场内正确的角球区，具体位置由 `server` 参数 `ckick_margin` 来决定；如果是球门球，足球会被放在小禁区靠近球出底线位置的前角。

越位 Offside

裁判在比赛过程中要 `enforce offside rule`。一个球员如果处于对方半场，并且当球传给它时，比球和少于 2 个对方球员更靠近对方的球门，则会被认为处于越位位置。由于球员可能从越位位置跑到非越位位置接球，因此判断越位与否的准确时间应该是球被踢出而不是接球时。在当前版本 `server` 中，裁判进行判定越位的实际时间是当球距离越位球员小于 `offside_active_area_size`（缺省为 5.0 米）。判定为越位后裁判会判给对方一个任意球 `free_kick`。

回传球

与真实中的足球规则相同，守门员不允许接（`catch`）队友回传给它的球。如果发生，裁判会宣布 `back_pass_side` 信息，并且判给对方一个任意球。由于这种回传球犯规只会发生在禁区中，任意球将被放在离守门员接球点最近的禁区角上。注意，如果守门员不使用 `catch` 来接球，而是使用 `kick`，则传球给守门员是绝对合法的。

任意球违规

在罚球门球 `goal_kick`、界外球 `kick_in`、任意球 `free_kick` 或角球 `corner_kick` 的过程中，发球球员不允许把球传给自己。由于为了把球加速到合适的速度，球员可能需要多次的 `kick`，所谓的把球传给自己指一个球员罚球时 1 第一个再次 `kick` 球，2 在两次 `kick` 之间进行了移动（`dash`）。所以，罚球时进行如 `kick-turn-kick` 或者 `kick-kick-dash` 这样的命令序列是合法的，但是类似于 `kick-dash-kick` 的命令序列构成任意球违规。如果违反了这条规则，裁判会宣布 `free_kick_fault_side`，并判给对方一个任意球。

清场 Player Clearance

当比赛模式是开球 `kick_off`、界外球 `kick_in`、任意球 `free_kick` 或角球 `corner_kick` 时，裁判把防守队员移出以足球为圆心，`offside_kick_margin` 参数（当前缺省为 9.15 米）为半径的圆形区域。被移出的球员随机放置在圆形区域的周围。如果比赛模式是越位 `offside`，所有造成越位的进攻方球员被移回到没有越位的位置。如果比赛模式是球门球 `goal_kick`，所有的进攻球员会被移到禁区外，当球门球发生时，进攻球员不能重新进入罚球区。

比赛模式控制 Play-Mode Control

当比赛模式是开球 `kick_off`、界外球 `kick_in`、任意球 `free_kick` 或者角球 `corner_kick` 时，裁判在足球因为 `kick` 命令产生移动时，马上把比赛模式设置为正常 `play_on`。对于球门球 `goal_kick`，当足球被踢出禁区后，比赛模式马上被设置为正常 `play_on`。

中场时间和终场时间 Half-Time and Time-Up

当上半场或下半场结束时，裁判暂时挂起比赛。每个半场的默认值是 `half_time` 个仿真周期（当前缺省为 3000，大约 5 分钟）。如果在下半场结束后，还是平局的话，会进行加时赛。加时中首先进球的队伍获胜，就是被称为“金球”规则或“突然死亡法”（通常加时只发生在比赛的淘汰赛阶段）。

裁判发送的每个信息都使用“（`referee String`）”的格式，其中 `String` 是表示信息内容的字符串。球员通过听觉感知接收裁判的信息。裁判的信息在球员的感知中具有特权，不受球员听觉能力的限制，不管一个球员从其他球员听到了多少信息，都可以接收到裁判广播的信息。

表 4.12 裁判信息及比赛模式变化。

Message	T_c	Subsequent play mode	Comment
---------	-------	----------------------	---------

Before_kick_off	0	Kick_off_side	比赛尚未开始
Play_on			比赛进行中
Time_over			比赛结束
Kick_off_side			声明比赛开始（人类裁判点击了 kick_off 按钮）
Kick_in_side		Play_on	球被踢移动后改变比赛模式
Free_kick		Play_on	球被踢移动后改变比赛模式
Corner_kick_side		Play_on	球被踢移动后改变比赛模式
Goal_kick_side		Play_on	当球被踢离开禁区后改变比赛模式
Drop_ball	0	Play_on	发生在球经过 drop_ball_time 后仍然没有被投入比赛
Offside_side	30	Free_kick_Oside	越位后给对方任意球
Goal_side_n	50	Kick_off_Oside	宣布 side 一方第 n 个进球
Foul_side	0	Free_kick_Oside	宣布 side 方犯规
Goalie_catch_ball_side	0	Free_kick_side	守门员接（catch）到球
Back_pass_side	0	Free_kick_Oside	守门员接回传球
Free_kick_fault_side	0	Free_kick_Oside	发生任意球违规
Time_up_without_a_team	0	Time_over	如果有一方队伍一直没有连接进入比赛
Time_up	0	Time_over	比赛结束
Half_time	0	Before_kick_off	上半场结束
Time_extended	0	Before_kick_off	下半场结束后比分相同，需要加时

side 为 l 或者 r，分别表示左半场和右半场的队伍；Oside 表示对手的 side； T_c 为从接到 Message 到比赛模式变为 subsequent play mode 之间的时间（仿真周期数）。

表 4.13 裁判模型有关参数和缺省值

Parameter	Value	Parameter	Value
Ckick_margin	1.0	Forbid_kick_off_offside	True
Offside_active_area_size	5.0	Half_time	300
Offside_kick_margin	9.15	Drop_ball_time	200
Use_offside	true		

异构球员

异构球员是 Server 升级到版本 7 时增加的新特性。在早期的版本中，场上所有球员的物理特性（physicall identical）和使用的球员参数都是相同的；而在新的版本中，每个球队可以从几个具有不同特性的球员类型中进行选择。这些球员类型在每次 server 开始新的比赛时自动随机生成。在一场比赛中，比赛双方使用同样的球员类型集合。生成的球员类型的种类数等于 server 的参数 player_type（缺省为 7）。在这些种类中，类型 0 指的是缺省的球员类型，每次都是相同的（所有参数都等于 server 中使用的球员缺省参数），而其他类型球员的参数每次都是不同的。

我们在 3.1 节看到，当一个球员连接到 server，server 会发送一些信息，这些信息中包括了异构球员的类型参数信息。对于每个可用的类型，server 会发送一条如下格式的消息：

```
(player_type id player_speed_max stamina_inc_max player_decay inertia_moment
```


dash_power_rate player_size kickable_margin kick_rand extra_stamina effort_max effort_min)

例如

```
(player_type (id 0)(player_speed_max 1)(stamina_inc_max 45)
(player_decay 0.4)(inertia_moment 5)(dash_power_rate 0.006)
(player_size 0.3)(kickable_margin 0.7)(kick_rand 0)
(extra_stamina 0)(effort_max 1)(effort_min 0.6))
. . . .
(player_type (id 2)(player_speed_max 1.135)(stamina_inc_max 33.4)
(player_decay 0.4292)(inertia_moment 5.73)(dash_power_rate 0.00716)
(player_size 0.3)(kickable_margin 0.8198)(kick_rand 0.0599)
(extra_stamina 31.3)(effort_max 0.9374)(effort_min 0.5374))
. . . .
```

这些信息包含了各球员类型使用的球员参数，由此定义了球员不同的能力。对于缺省球员类型的参数是固定的，而对于其他类型，这些参数是根据 server 中为异构球员定义的一些范围内随机选取的。在表 4.14 中，列出了 server 中使用的这些范围，并比较了缺省球员参数和异构球员的可能参数。例如，缺省球员的最大速度等于 server 参数 player_speed_max，而一个异构球员的最大速度则是 player_speed_max+player_speed_max_delta_min 和 player_speed_max+player_speed_max_delta_max 之间的一个随机值。同样，对于异构球员的体力恢复速度，即每个仿真周期体力增长的值，是在默认值 stamina_inc_max 的基础上加上一个介于 player_speed_max_delta_min*stamina_inc_max_delta_factor 和 player_speed_max_delta_max*stamina_inc_max_delta_factor 之间的值，由于 stamina_inc_max_delta_factor<0，所以异构球员的体力恢复速度小于缺省球员。这说明异构球员通过牺牲体力恢复速度来获得了更高的最大速度。其他情况参数也类似。

表 4.14 普通球员参数与异构球员参数比较

Player parameters		Parameters for heterogeneous players		
Name	Value	Name	Value	Range
Player_speed_max	1.0	Player_speed_max_delta_min	0.0	1.0-1.2
		Player_speed_max_delta_max	0.2	
Stamina_inc_max	45.0	Stamina_inc_max_delta_factor	-100	25.0-45.0
		Player_speed_max_delta_min	0.0	
		Player_speed_max_delta_max	0.2	
Player_decay	0.4	Player_decay_delta_min	0.0	0.4-0.6
		Player_decay_delta_max	0.2	
Inertia_moment	5.0	Inertia_moment_delta_factor	25.0	5.0-10.0
		Player_decay_delta_min	0.0	
		Player_decay_delta_max	0.2	
Dash_power_rate	0.006	Dash_power_rate_delta_min	0.0	0.006-0.008
		Dash_power_rate_delta_max	0.002	
Player_size	0.3	Player_size_delta_factor	-100	0.1-0.3
		Dash_power_rate_delta_min	0.0	
		Dash_power_rate_delta_max	0.002	
Kickable_margin	0.7	Kickable_margin_delta_min	0.0	0.7-0.9
		Kickable_margin_delta_max	0.2	
Kick_rand	0.0	Kick_rand_delta_factor	0.5	0.0-0.1
		Kickable_margin_delta_min	0.0	

		Kickable_margin_delta_max	0.2	
Extra_stamina	0.0	Extr_stamina_delta_min	0.0	0.0-100.0
		Extr_stamina_delta_m	100.0	
Effort_max	1.0	Effort_max_delta_factor	-0.002	0.8-1.0
		Extr_stamina_delta_min	0.0	
		Extr_stamina_delta_max	100.0	
Effort_min	0.6	Effort_min_delta_factor	-0.002	0.4-0.6
		Extr_stamina_delta_min	0.0	
		Extr_stamina_delta_max	100.0	

表的左侧是普通球员使用的参数值，右侧是异构球员使用的参数值的范围。从表中可以清楚的看出异构球员对哪些能力进行了折衷。

如何使用异构球员是由在线教练来控制的。在线教练能够选择要使用的球员类型，并根据需要改变场上球员的球员类型。当 `playmode` 处于 `before_kick_off` 时，教练可以随意改变球员类型；但是在比赛过程中，只允许进行 `subs_max` 次改变调整。另外，一个球队在场上的球员中，相同的球员类型最多只允许有 `subs_max` 个，而且必须有一个缺省球员作为守门员。每次教练使用 `change_player_type` 命令来改变一个球员的类型，这个球员的 `stamina`，`recovery` 和 `effort` 都恢复到新类型的初始值。有关在线教练的使用在第 11 章中有详细的介绍。

表 4.15 异构球员模型有关参数和缺省值。

Parameter	Value
Player_types	7
Subs_max	3

4.5 世界模型的更新

简单的更新模型

球员可以直接根据 `server` 感知信息中包含的内容对世界模型进行更新。一个使用视觉感知的简单世界模型更新过程包括了球员自身位置和角度更新，球和其他球员位置与速度的更新。

视觉信息的使用

视觉信息包含了当前仿真周期在球员视野中的可视物体的信息。这个信息默认每 `Send_step` ms（根据视野范围和视觉质量设置会有变化）自动发送到球员，包含了如下内容：

- 视觉信息所处的仿真周期 t 。
- 球场上可见物体的信息，这些信息是相对于球员的视角的，根据不同情况可能包含以下内容中的几种（4.1.2 中的视觉模型）：
 - 物体名称标示
 - 球员到物体的距离 r
 - 物体的相对角度 ϕ
 - 物体距离的改变 Δr
 - 物体角度的改变 $\Delta \phi$
 - 对象的身体角度 $\tilde{\theta}_{body}^t$
 - 对象的头部角度 $\tilde{\theta}_{neck}^t$

球员接收到视觉信息，就可以立刻对世界模型进行更新：首先对视觉信息字符串进行解

析处理，把相应数据存储在一个临时结构中。由于视觉感知中的信息都是相对位置，为了获取各物体的全局坐标，球员的自身信息必须首先更新。球员自身的全局位置和头部角度可以通过球员相对球场上的可视标志的位置来计算。由于视觉感知中其他物体的信息都需要根据球员自身位置信息按照公式 4.1-4.12 计算，因此自身定位算法应该尽可能的准确；球员在场上不同的位置和自身角度，会看到不同数目的球场标示，例如，当球员面对场中心时，它将看到很多标志，而当它靠近边线并且朝向场外时，它可能只看到很少的标志，因此要求定位算法必须能够足够健壮，可以只使用很少的标志；同时考虑到实时对抗系统的特性，算法要尽可能的快速。有很多方法可以用来定位，这里只简单介绍一个最简单的算法。

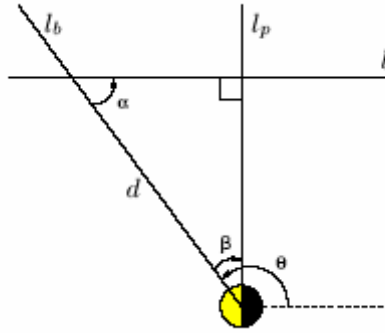
球员自身位置信息更新

这个自身定位算法使用一条边线和一个标志旗的现对位置来计算球员的全局绝对位置。边线的角度可以用来计算球员的全局头部角度，然后根据球员已知的标志旗的全局位置来计算球员的全局位置。假设 l_b 表示球员视野的角平分线，当球员看到一条边线 l 的时候，根据视觉模型的定义，视觉信息中包括了 l_b 和 l 交点到球员的距离 d 和 l_b 与 l 之间的夹角 α （如图）。

由图可知， β 可以通过下面的公式求得：

$$\beta = -\text{sign}(\alpha) \cdot (90 - |\alpha|)$$

（其中函数 sign 表示如果其参数为正数，则返回 1，否则返回-1）



• 图 4.4 使用边线计算球员头部全局方向

为了计算球员头部全局角度 θ ，我们需要知道 l_p 的全局角度。根据场上坐标方向的定义，可以很容易的推出与每个边线相应的 l_p 的角度。

这里我们只考虑了从场内看到边线的情况，如果球员的视角是从场外向场内时（如发边线球，角球的时候），则可以根据看到的较远的边线来进行计算。这样，就可以计算球员头部全局角度 $\theta = \text{orientation}(l_p) - \beta$

接下来可以通过视野中一个标志的已知位置来计算球员的全局位置。标志 f 的视觉信息中包括了到球员的距离 \tilde{f}_r 和相对于球员头部的角度 \tilde{f}_ϕ ，即 f 的相对极坐标 $(\tilde{f}_r, \tilde{f}_\phi)$ 。同时，我们知道 f 的全局坐标 (f_x, f_y) （作为已知信息保存在世界模型中），通过对比 f 的全

局位置和相对球员的位置，就可以计算出球员的全局位置 (p_x, p_y) ：

$$(p_x, p_y) = (f_x, f_y) - \pi(\tilde{f}_r, \tilde{f}_\phi + \theta)$$

其中 π 表示极坐标到直角坐标转换函数 $(x, y) = \pi(r, \phi) = (r \cdot \cos(\phi), r \cdot \sin(\phi))$ 。

场上运动对象信息更新

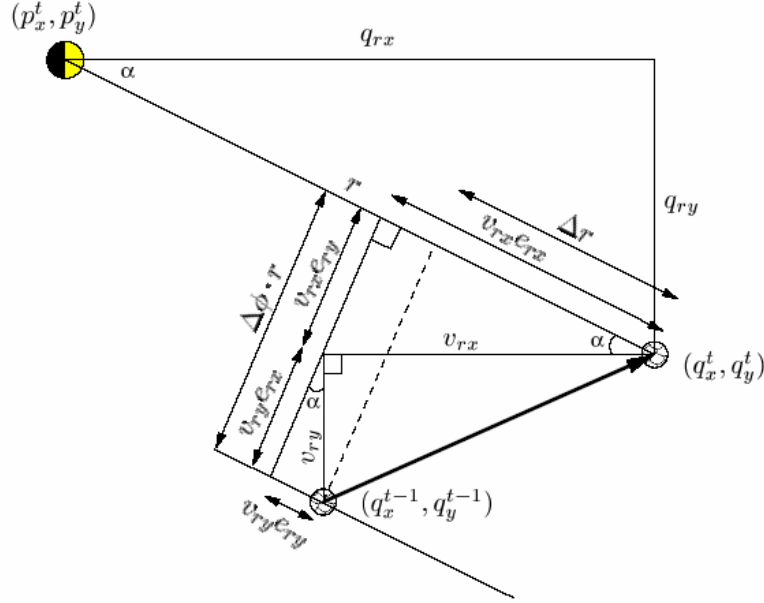
得到了自身全局位置 (p_x, p_y) 和头部全局角度 θ ，球员就可以使用这些信息来计算视觉信息中包含的动态对象（球或者球员）的属性，这些需要计算的属性主要包括：全局绝对位置 (q_x, q_y) ，绝对速度 (v_x, v_y) ，身体全局角度 θ_{body}^t ，头部全局角度 θ_{neck}^t ，其中后两个属性只属于球员。

动态对象的全局位置 (q_x, q_y) 可以通过结合球员自身的全局位置和对象的相对视觉信息来获得。视觉直接包含了对象极坐标表示的相对位置 $(\tilde{q}_r, \tilde{q}_\phi)$ ，而这个位置是相对与球员的头部视角的。因此， $(q_x, q_y) = (p_x, p_y) + \pi(\tilde{q}_r, \tilde{q}_\phi + \theta)$

如果可见对象的距离不是很大，那么视觉信息中也会直接包含对象的身体相对角度 $\tilde{\theta}_{body}^t$ 和头部相对角度 $\tilde{\theta}_{neck}^t$ 。由于这些信息是相对与球员自身的头部角度 θ ，因此对象的身体绝对角度和头部绝对角度为：

$$\begin{aligned}\theta_{body}^t &= \tilde{\theta}_{body}^t + \theta \\ \theta_{neck}^t &= \tilde{\theta}_{neck}^t + \theta\end{aligned}$$

在 soccer server 中，动态对象速度，尤其是球速的准确估计对于球员的决策非常重要。动态对象的速度估计也是对象信息计算中最复杂的部分。视觉感知中包含了一个运动物体距离的改变 Δr 和角度的改变 $\Delta \phi$ ，根据公式 4.1-4.12 如图可得：



• 图 4.5 运动物体 DistChange 和 DirChange 示意图

$$v_{rx} = \Delta r \cdot e_{rx} - \Delta \phi \cdot (\pi / 180) \cdot r \cdot e_{ry}$$

$$v_{ry} = \Delta r \cdot e_{ry} + \Delta \phi \cdot (\pi / 180) \cdot r \cdot e_{rx}$$

由于视觉感知是相对与球员自身头部全局方向 θ ，因此把相对速度向量 (v_{rx}, v_{ry}) 旋转 θ 就得到物体的全局速度 (v_x, v_y) 。

带预测的更新模型

由于仿真平台的特性，球员无法直接获得场上所有运动物体的信息，而且存在没有视觉感知信息到达的周期，因此为了维护一个包括全局信息的世界模型，球员必须能够合理的预测没有视觉信息物体的状态，这就要求在更新中对物体运动预测；同时，为了进行决策，球员需要知道执行不同行为的结果或者需要计算为获得一定结果需要使用的命令参数，这就要求球员能够预测行为结果。带预测的更新模型对于维护世界模型的完整性和决策的有效性都非常重要。

球运动预测的使用示例

使用运动模型，球员可以根据已有的场上运动物体信息计算其未来状态信息，但是由于其他球员执行的行为命令无法获知，预测其状态是非常困难的，因此通常预测集中于球的运动和球员自身的运动。

球的状态可以通过使用已知的球的运动模型来计算，Soccer Server 每个周期结束会把速度向量加到位置上并按照一定的比率衰减速度(4.2.1)。球速每个周期会衰减同样的比率，因此球速可以表示成一个等比数列。如果 a 表示某个周期球的速度，则 n 个周期后球的速度是 $a \cdot \text{ball_decay}^n$ 。由于球的移动是通过速度向量的叠加，我们可以表示 n 个周期中球移动的距离为

$$\sum_{i=0}^n a \cdot \text{ball_decay}^i = a \cdot \frac{1 - \text{ball_decay}^n}{1 - \text{ball_decay}}$$

注意，为了简化，预测中忽略了球运动中噪声的影响，也没有考虑 n 个周期中其他球员对球的影响（执行 `kick` 命令）。

球的最大速度为 `ball_speed_max`。在当前缺省设置的参数下，根据上两式可以算得，一次最优踢球（把球加速到最大速度）可以使球滚动大约 45 米距离，其中在前 15 个周期内，球可以达到大约 28 米距离，速度降为略大于 1，踢球后 53 个周期，足球移动距离是 43，其时速度小于 0.1。

进一步，如果知道了某个周期球的位置，则可以预测 n 个周期后球的全局位置

$$(p_x^{t+n}, p_y^{t+n}) = (p_x^t + p_y^t) + \pi(v_r^t \cdot \frac{1 - \text{ball_decay}^n}{1 - \text{ball_decay}}, v_\phi^t)$$

球员自身运动预测的使用示例

如果没有视觉感知信息到达，球员的身体感知信息也可以用来更新世界模型。利用运动模型，球员身体感知中的速度和头部角度可以用来计算它的全局速度和位置。根据公式。。。上一周期到本周期球员全局位置的移动为 (u_x^t, u_y^t) ，由公式。。。可知

$$(u_x^t, u_y^t) = \frac{(v_x^t, v_y^t)}{\text{Decay}}$$

为了计算相对上一周期球员位置移动的向量 (u_x^t, u_y^t) ，我们需要计算本周期的全局速度 (v_x^t, v_y^t) ，这可以利用球员身体感知信息中的相对速度 $(\tilde{v}_x^t, \tilde{v}_y^t)$ 来获得。如果忽略噪声的影响， (v_x^t, v_y^t) 和 $(\tilde{v}_x^t, \tilde{v}_y^t)$ 的模是相同的，只是方向不同，全局速度可以通过将相对速度 $(\tilde{v}_x^t, \tilde{v}_y^t)$ 旋转一定的角度获得。由于 $(\tilde{v}_x^t, \tilde{v}_y^t)$ 是球员相对它头部方向的速度，这个旋转角度应该等于球员头部的全局角度 θ^t 。这个角度可以在上一个周期的 θ^{t-1} （保存在世界模型中）的基础上根据上一周期执行的命令来预测。注意，这里我们需要知道上一周期发送的命令是否已经成功被 `server` 执行，可以通过比较本周期 `sense_body` 信息中包含的命令计数和上周期的计数来检查。为了计算 θ^t ，我们只关心成功执行的 `turn` 和 `turn_neck` 命令。用 α 表示上一周期球员实际转身角度。根据行为模型，球员实际转身角度 α 并不一定等于 `turn` 命令的角度参数，而要受到当时速度的影响。由于在上一周期执行 `turn` 命令，那么球员实际转身角度 α 只能根据上一周期的速度利用行为模型计算。由于球员不能在同一个仿真周期中同时执行 `dash` 和 `turn`，因此如果一个 `turn` 被执行，则球员的加速度 (a_x^t, a_y^t) 就会是 0，因此有

$$(v_x^{t-1}, v_y^{t-1}) = (u_x^t, u_y^t) = \frac{(v_x^t, v_y^t)}{\text{Decay}}。$$

则根据行为模型（忽略 \tilde{r} 的影响），

$$\alpha = \frac{(1.0 + \tilde{r}) \cdot \text{Moment}}{(1.0 + \text{inertia_moment}) \cdot \|(v_x^{t-1}, v_y^{t-1})\|}$$

用 γ 表示球员转头角度，它等于上一周期执行 `turn_neck` 命令时使用的角度参数。则

$$\theta^t = \text{normalize}(\theta^{t-1} + \alpha + \gamma)$$

得到球员头部的全局角度 θ^t ，把身体感知信息中的相对速度 $(\tilde{v}_x^t, \tilde{v}_y^t)$ 旋转 θ^t 得到

(v_x^t, v_y^t) ，而球员的全局位置可以通过前一周位置 (p_x^{t-1}, p_y^{t-1}) 和 (v_x^t, v_y^t) 使用运动模型获得。

简单世界模型更新过程

当更新球员的世界模型时，首先更新球员本身的位置，然后更新球和其他球员。

● 球员本身

在我们将球员世界模型更新到仿真周期 $t-1$ 后，可能会有周期 $t-1$ 和（或） t 的视觉信息（如果有更早的视觉信息，则不予采用考虑）。如果最近的视觉信息的 $t-1$ 周期，则球员本身的位置不做更新，直到其他对象更新完毕（因为视觉信息中它们的坐标是相对于球员 $t-1$ 的位置）。另一方面，如果最近的视觉信息的是 t 周期，或自上一次更新后没有收到新的视觉信息，则球员本身的状态可以立即更新。

更新球员状态的过程如下：

（1）如果收到新的视觉信息

球员通过可见边线和场上标志的相对坐标来精确地确定自身位置。

（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）也可以根据身体感知信息更新球员的状态信息。

（4）根据身体感知信息或动作结果预测更新体力模型

● 足球

在球员的决策过程中，足球的位置和速度对动作的选择是一个相当关键的因素，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球员的绝对位置和球相对于球员的相对位置来确定球的绝对位置。

（2）如果同时也包含了 `DistChange` 和 `DirChange` 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世界模型中的速度是否正确，并根据原来的速度预测更新当前球速。

（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。如果在上一周期球员踢球，则需要考虑行为的作用。

（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。听觉信息可以增加球的位置或速度的可信度，队友信息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，得到的信息也越精确）。

更详细的世界模型更新见第 8 章。

WrightEagle

Mao Chen et. al. RoboCup Soccer Server. The RoboCup Federation, February 2003. Manual for Soccer Server Version 7.07 and later

Peter Stone. Layered Learning in Multi-Agent Systems. PhD Thesis, Computer Science Dep.,Carnegie Mellon University, December 1998

R. de Boer and J. R. Kok. The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team. Master's thesis, University of Amsterdam, The Netherlands, Feb. 2002.

第五章、设计更高级的球员智能体

智能决策是机器人具有智能的集中体现，也是目前人工智能研究的一个重点，它充分体现了机器人的适应世界并且有意识的改造世界的能力。在机器人足球这个极为复杂的动态多主体环境中，一个能够根据环境动态调整，快速生成行动计划的决策就显得尤为重要，决策的优劣很大程度也决定了一支球队的强弱。对于想设计一支参加 RoboCup 比赛的球队研究者来说，根本问题是设计一个多主体系统，能够进行实时的反应，表现出目标制导的理性行为。由于足球比赛的状态空间极大，目标与环境都是在动态变化并且是实时的，不可能用手工的方法来编码描述所有可能的情形和制定智能体的所有行为，因此使智能体能学习如何有策略的进行比赛变得极为重要。因此在 RoboCup 挑战计划[Noda et al. 1997]中就明确了仿真机器人足球研究的重点问题就是：在多智能体合作及对抗环境中的机器学习，多主体体系结构，为了团队合作而进行实时的多智能体规划和规划执行，以及对手建模。

5.1 简单决策原理

在一个 RoboCup 仿真球员程序中，决策是指智能体在获得场上信息后，对场上形势进行分析，给出相应的行为执行方案，再由行为执行模块分解执行的过程。决策系统模块是球员程序的灵魂，是这个智能体的思维，是智能的集中体现，是评价一个智能体的智能程度最主要考虑的模块。由于场上状态过于复杂，智能处理难于下手，一般采取分层处理的办法，底层行为如带球、传接球、射门等相对固定，易于代码化，也可用离线学习的方法来优化；上层决策如跑位、技战术配合等由于场上形势的不确定性，很难明确给出，可以使用强化学习之类的在线学习方法，来学习提高。另外，还可预先制定一些既定战术如中场开球、发角球等以提高这些情况下的成功率。此模块可细分为：

1 **行为决策模块** 根据当前场上状态（包括实际感知和预测），以及赛前制定的合作协议，来决定当前的行为模式，并更新自身状态。行为决策模块将当前的行为模式以及该行为附带的参数(如果需要的话)发送给动作解释及执行模块，作进一步分解。

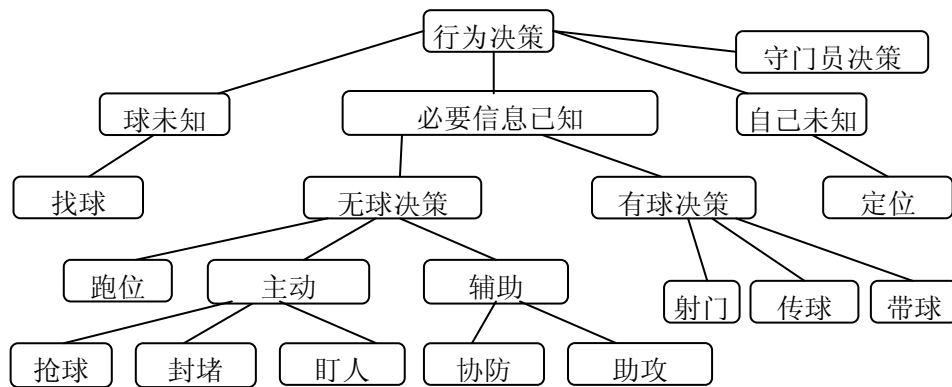
2 **合作协议模块** 存储在赛前制定的合作协议。可以是特定的技战术配合如中场开球、发角球等，还可以是比赛时的一些战术运用如交叉换位、下底传中等。

由上可看到，仿真机器人足球比赛是真正智能的对抗，通过这个具有普遍意义的平台，能很好地评价各种理论、算法和智能体的体系结构优劣，是检验智能体团队合作和多智能体学习研究进展的理想测试平台。

5.1.1 决策系统

行为树决策系统

行为树决策是一种典型的反应式决策系统，根据环境信息直接产生相应的行为。为了简化条件判断，可以对环境信息分类，逐层展开，就形成了一个决策树，如图 5.1 所示。树中所有叶节点为行为节点，每个行为节点都表示一个具体的行为；其它节点即为分支选择节点，对应不同的条件选择函数。



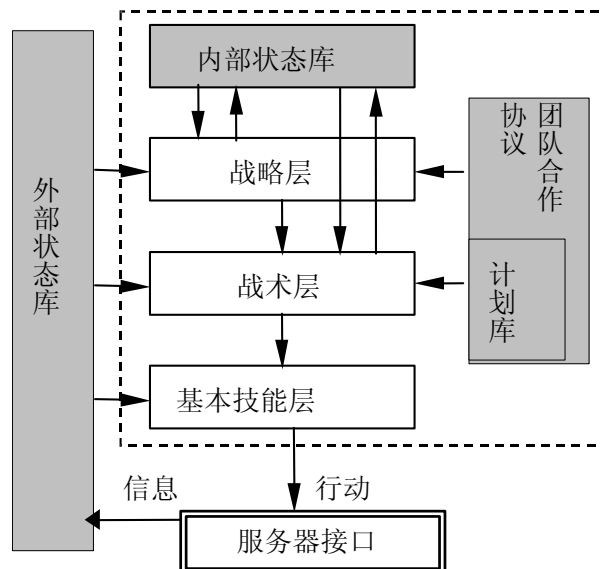
这种分层决策系统给出了一个比较简便易行的决策方法，根据环境变化可以比较快的做出反应，产生相应的行为，具有比较好的实时要求，但涉及到机器人足球这样一个复杂的决策环境，这种决策系统还有许多不足。

首先，上面的结构是单向分层决策，高层决策无法了解下层决策的实际能力，只能按一种预定模型来分析，不能实时调整，这就会导致上层决策与下层执行脱节，不能很好的完成目标。

其次，这种按照行为树的顺序决策方式不能很好地适应复杂的决策环境，实际上在这种方式下每个行为都被赋予了一个优先级，高优先级的行为总是被优先考虑，好处很明显，高优先级的行为总是被优先考虑执行。现在大部分球队设计时都引入了决策论的思想，进行全面的评估，不再是简单的按照行为树顺序决策。

分层决策系统

图 5.2 表示一个三层决策系统模块，依次为：战略层，战术层和基本技能层。



• 图 5.2 三层决策模型

决策系统是整个人体结构的核心模块。相当于人脑的思维模块，主要是根据“当前的场上形势”（来源于世界模型）来决定所要采取的行为决策。由于场上形势过于复杂，所以细

分为几个子层逐步处理。

1) 战略层

战略层是决策系统模块中的最高层，用来从全队利益出发实现高层决策，确定球员智能体的行为目标及总的行为方案。在这层中智能体决定诸如决定自己的角色，判断自己是该进攻还是该防守，要执行什么任务等问题，同时要明确当前达到的战略目标。这层充分体现了智能体的团队合作与规划行为（详细举例参见下节）。目前很多智能体的战略决策还是基于一些常规的分析法：

(1) 判定是否它处于是某一合作协议中（例如中场开球、界外球、角球等）。如果是，只需从合作协议库中提取出预先指定的计划，按角色来施行。虽说这些战术都是死的，但确实是经常会出现的，使用这种方法往往会取得较好的效果。另外也可以引入比赛进行中的一些合作协议配合。这属于动态的实时规划，难度较大。

(2) 判断自己的角色和当前队阵型。由此来决定自己的任务（不同的角色在不同的阵型下所要执行的任务自然也不尽相同）。战略层能根据当前的世界模型选择合适的阵型，确定自己的角色。

(3) 判断自己所处的行为模式（图 5.1）。根据当前的世界模型、队阵型和智能体所处的角色选择自己的行为模式，包括进攻模式、辅助进攻模式、防守模式、辅助防守模式、守门员模式（由于守门员处理比较复杂，又很重要，所以单独处理）。

2) 战术层

在战略层确定行为模式后，战术层要贯彻执行，制订出为了完成这一特定的目标，所应该采取的行为策略。即将这些行为模式具体化：如果智能体在执行某个合作协议，只需从合作协议库中提取出预先指定的计划，得到相应的执行命令。一般地，智能体分析当前世界模型，从对手模型中，把握对手的意图，依照行为树（图 5.1）来决定应采取的行为策略。包括抢占有利的位置、使用越位陷阱、抢球，拦截，带球，传球，射门，扑球（守门员），盯人，等等。

3) 底层技能层

底层技能层是决策系统层最低的，用以实现智能体的个人技能（如传球、带球等），即将这些具体的行为决策细化为比赛平台可以接受的执行指令。在每个周期，最终都是这一层生成本周周期最终所要执行的基本命令存到执行队列里，由程序中的接口模块负责将该指令发送到服务器端以实现对球员的控制。

在实际实现时，一般只是给智能体一些基本的行为规则，如球来了要去接球（守门员要去扑球），对方控球要上去封堵，不能让它轻易射门得分，还有按照行为树（图 4.4.3）决定自己的行为模式等等。但是这些还远远不够，怎样接球不会丢，怎样封堵最有效，什么时候处于进攻模式、什么时候处于防守模式，什么时候该传球等等都是无法直接给定的。随着场上形势的变化，所采取的行为也不尽相同。但采取每个行为都有个成功率的概念，智能体可以在实践中评价每个行为，修正每个行为在确定形势下的成功率，选取成功率高的行为，即可达到相对较好的效果。成功率的设置，一方面可根据规则，如球不在能踢的区域内（kickable_area 由 Server 给出）就不能踢球；一方面可根据常识经验，如在有守门员的情况下，离球门 25 米以外很难射门得分；另外还可以通过使用机器学习（包括离线的和在线的）来调整各行为的成功率。如在实现一些基本行为决策（如传球、射门）时可以采用一些离线学习方法如 C4.5 决策树方法来提高行为的成功率，在可使用一些强化学习方法，来提高决策的智能度，以获得较好的效果。

这种分层决策系统给出了一个比较简便易行的决策方法。通过分层处理,逐步简化问题,从而比较好地实现一整套决策任务。从整体看这是一个混合式决策系统,高层从长远利益考虑,制定全局决策,是慎思式的;底层简单的从环境得到相应信息,直接产生相应行为,是反应式的。这是混合式结构综合了两种方式的优点,既满足了实时决策的需要,又兼顾了长远利益,避免了反应式的短视问题。

上面给出两种典型的决策系统,目前大部分球队都使用的是分层决策系统。但具体层次及层次之间的关系又有些差异。如我国清华队将主要行为都独立成彼此并列的模块,各自进行评估,最后由仲裁模块进行全面权衡,选出最合适的行为执行[Yao et al. 2002]。具体介绍详见第6章。

5.1.2 决策模块分析

根据实际决策的不同,可以有许多种不同的分类。下面根据场上不同的形势,把决策分为进攻决策、防守决策和一般跑位决策三个模块加以分析。

1) 进攻决策模块

进攻决策是整个决策的重点,没有进球就无法赢得比赛。俗话说“进攻是最好的防守”,这里也充分体现的这一点。早期比赛由于踢球加速度比较小,球转移比较慢,很多球队都采取全攻决策,压制对手过不了半场。后来由于各球队技术普遍提高,加上球转移速度大大加快,这种策略发生了很多变化。现在阵地进攻型球队依然保持强大的压制对手能力,但与以往不同的是这种类型的球队进攻层次更多,推进更稳固,不会轻易被对手突破后防,但相对推进速度较慢,得分能力不强,毕竟在这种二维比赛环境中,一旦对手布防完成,要想进球还是比较困难。另外一类属于快攻型球队,进攻追求速度,专门利用对手的漏洞突破,让对手还来不及布防就破门得分,防守反击就是一个典型的打法。这种打法速度快,得分能力强,但相应的失误率就比较高,特别是一旦失误就容易被对手反击。

不管是哪一种进攻打法,准确的形势分析、高效合理的规划合作都是必不可少的。这里参与进攻的球员可以简单分为控球队员和助攻队员。

控球队员是进攻的中心,控球队员的选择往往决定了一次进攻的成败。合理的选择下一步的行为就显得尤为重要。如果简单用分支函数处理,就需要事先对可能出现的场上形势进行分类,然后针对不同的情况执行相应的行为。由于场上形势错综复杂,无法完全列出各种情况,就只能近似归类,这样就很难找到“最优”方案;目前更多球队采用的都是类似于决策论的方法[Yang et al. 2002],在给定的形势下分析各种可能行为(包括传球、射门、带球等)的成功率 p ,同时也给出执行该行为的收益 u ,然后通过计算得到执行一个行为的综合评价 b ,选择评价最高的执行即可,公式表示如下:

$$\max \sum p_{ij} * \mu_{ij} \rightarrow b_{best}$$

由于比赛环境是动态不确定的,所以无法准确预知其他球员的行为,这里无论是成功率 p 还是收益 u 的计算,都是一种基于期望的预测(EBP)[Yang et al. 2002]。我们在计算时实际已经假定所有队员都是按照程序里的“最优”决策方式进行,当然也可以根据球队的风格“高估”或“低估”对手。

助攻队员跑位点的选择以及对时机的把握都会对结果产生很大影响。助攻队员如果善于发现对方防守空档并选择合适的时机跑位,就能够配合控球队员出色地完成进攻任务。

对于跑位点的选择以及时机的把握都需要与控球队员协同。由于比赛环境限制，完全依靠通讯来协同是很困难的。很多队都是事先制定了战术，这样只要在比赛中由控球者选择合适的战术，然后通知参与者来实现协同。比较典型的如清华队的“scheme”机制[Jinyi Yao et al. 2003]，荷兰 UvA 队的协调图方法（CG）[Kok et al. 2003]。这些球队都给出了一种描述战术的方法，实现了队员间的一些配合。

这些战术由于都是实现制定的，所以并不总能够适用于所有的情况，队员还需有其他方法来适应场上多变的环境。一种简单的方法同样是根据使用前面提到的 EBP 方法，根据自己的决策算法预测控球队员可能采取的行为。由于是同一球队通常采用同一种算法，这种方法大多时候都比较准，除非两人获得的关键信息有较大差异。由于这种预测是近似算法，所以在实际使用时对算出的结果还要作些修正。另外要注意的是这种方法实际是根据当前状况查找可行的机会，我们更多的需要是无球队员能通过跑位来创造机会，这方面还需要通过战术或其他方式来帮助无球队员实现。

2) 防守决策模块

防守和进攻的关系正如矛和盾的关系，一支优秀的球队不仅需要强大的进攻得分，更需要稳固的防守保持住优势。防守不同于进攻，球在对方脚下，防守队员只能根据对方的行为及时采取相应的措施阻止对方。

针对防守对象和场景的不同，防守行为可以简单分为抢球、封堵、盯人三种。

抢球是在对方没有完全控制住球，双方处于拼抢状态时采取的行为，但由于对方处于主动，如果判断有误，很容易被对方晃过，所以安全的防守行为是封堵，虽然拿不到球，但也不让对手带球前进或轻易转移；盯人一般是指对无球进攻队员的盯防，破坏其可能的配合。

类似的，葡萄牙队将防守行为细分为[Reis and Lau 2000]

- 拦截球——以最快的速度抢到球
- 被动拦截球——根据场上形势，在尽可能占优的位置拦截球，不保证最快
- 封堵传球路线——盯住对方控球队员的可能传球路线，防止其通过传球突破防守
- 封堵对手——盯住对方控球队员，防止他带球前进
- 逼近球——即使不能抢到球，也要逼近对方控球队员，减少他的选择机会
- 盯防路线——选择合适的防守位置处于对手和我方球门之间

不管行为怎么细化，总的来说防守既需要个人优秀的盯防能力，还需要团队合作，制定合理的盯防分配策略，保证关键的进攻对手都有人盯，后场没有明显的防守漏洞。不仅如此，中国清华队还引入了两人协防概念，实现更有效的防守[Cai et al. 2002]。

守门员与普通球员不同，他的主要职责就是坚守球门，另外守门员专用的扑球指令使其比其他球员更容易抢到球，所以防守策略与一般球员也有所不同。

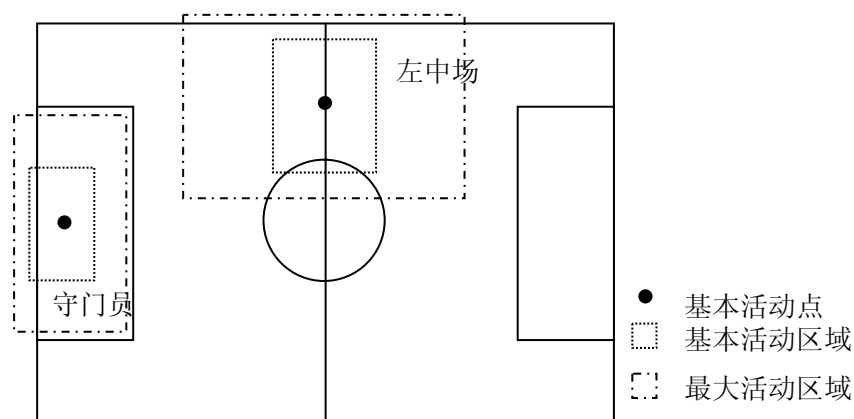
由于比赛环境是 2 维的，守门员防守相对容易一些。根据比赛平台模型，只要守门员发送扑球指令及时，球是不可能穿越守门员的。所以守门员的防守策略可以简单分为站位和出击。合理的站位可以使守门员覆盖尽可能大的球门区域，将对手射门机会降至最低；及时的出击扑球才能彻底瓦解对手的进攻。

合理的站位点原则上是在球到球门连线上，要保证无论球以多大速度射向球门任何位置，守门员都可以及时拦截到；同时考虑有威胁的对方助攻球员，站位点可以适当偏移。

出击关键是时机的把握，相对于普通球员，守门员的拦截行为更强调成功率。所以，守门员出击的判断一定要尽可能准确。这就使得很多时候守门员并不是选最快拦截点扑球，而是成功率相对比较高的位置扑球。当然如果情况危急，守门员也不得不冒险。

3) 一般跑位模块

一般跑位是指场上队员在没有明确进攻或防守任务的情况下，所遵循的一种移动原则。在 2000 年比赛以前，这个概念没有被明确提出，有不少球队并未专门设计，使得比赛中有很多球员追着球跑，比赛很混乱，双方总是挤成一团，就像小孩玩的街头足球一样。这种方式既耗费体力，又容易顾此失彼，特别是后来比赛规则修改，使得球的转移速度大大加快，这种方式就都无法应付了。后来很多球队学习了人类足球比赛的经验，开始引入“阵型”的概念，如 1998、1999 两届世界冠军卡耐基梅隆大学仿真足球队设计了一种方式，即给每个队员划定一块“责任区” [Stone and Veloso 1999]，限制队员的活动范围，在没有任务时就在活动中心待命，如图 5.3。



● 图 5.3 不同队员的活动范围

这种跑位方式使比赛看上去不再是一盘散沙，队员既节省了体力，又没有明显的漏洞。但是他的问题也很明显，就是球员站位过死，不能适应场上多变的形势，一旦比赛节奏加快，球员之间就会产生脱节，让对方有机可乘。

终于 2000 年世界冠军葡萄牙队提出了一个新的、更加合理的跑位模型——“基于形势的战略跑位 SBSP” [Reis and Lau 2000]，这个跑位模型将队员的跑位与场上的形势紧密的联系到一起。在计算每个时刻的跑位点时，球员要分析目前正采用哪种战术和阵型，并根据自己的角色计算出一个基本跑位点，然后再根据场上各种不同的形势，如球的位置、速度等信息，是进攻还是防守，还有比分等信息，以及角色的特点调整得到最终的战略跑位点。角色的特点包括对球变化的敏感程度、特殊的区域限制（如守门员不能出禁区）、保持在球之后（如后卫）等。这些都使得这种战略跑位更加灵活多变。

这种跑位系统使得球队的移动更像一个真实的足球队，球员保持这种场上分布可以比较好的覆盖球的移动，不至于产生大的漏洞。后来被各球队广泛采用。

关于这三个模块的划分也没有一个明确的界定，特别是对不同位置、不同角色判断也有所不同。例如，在一个强调进攻的球队而言，划分更倾向于进攻，当双方队员在拼抢时，周围队友也很有可能处于进攻模式寻找有利位置等待突破良机，而不会盯防对手；反之，一支打法保守的球队，一旦自己队员不控球就很有可能转入全面防守状态。所以，形势分析不但与决策队员所处的位置、场上所有球员、球的状态有关，还跟球队的战略意图、对手的特点密切相关，需要根据情况具体分析。

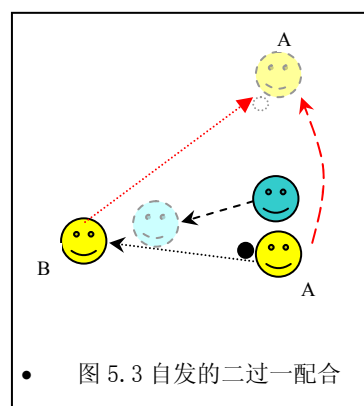
5.2 简单团队合作实现

足球比赛是一个集体项目，一个球队里包括许多球员智能体，这就涉及到如何协调这些智能体，发挥团队的优势，来更好地完成团队的共同目标，这需要团队间的合作。团队合作是目前多智能体系统研究的一个焦点问题，在类似足球这样复杂、动态的多智能体领域中需要极为灵活的协调和通讯来克服其中的不确定性[Kitano et al. 1997]，但现有的智能体团队很难应付这种环境：不确定性和通讯的受限使得团队成员无法确保信息一致，无法进行有效的合作；而实时性要求更使得传统的推理和规划无法应用到这些智能体中。在仿真机器人足球比赛中集中地体现了这些问题，机器人足球作为人工智能和机器人中的一个挑战性问题新的标准问题，是检验智能体团队合作和多智能体学习研究进展的一个理想的测试平台，所以团队合作也将被作为一个重点课题来研究。

5.2.1 自发的团队合作

团队合作可以是自发的，也可以是有规划的。前者主要是当智能体在掌握了一定的基本技能（如能较准确的将球踢到指定点）、能够进行一些简单的反应式行为决策（如抢球、简单传球、跑位等）之后，在某种情况下就会产生自发的团队合作如二过一配合：控球队员 A 控制球在前进过程中被对方阻挡，此时另一名队员 B 处于一个无人防守的位置，队员 A 发现队员 B 位置比较安全，就会将球传给他，这时防守队员就会继续去抢队员 B 的球，队员 A 就可摆脱防守队员迅速前进，然后队员 B 再把球传回给队员 A，从而过掉防守队员，形成二过一的配合（图 5.3）。

这种现象并非偶然，正如一个训练有素的职业队员，即使刚加入一个陌生的球队，他也能与其他队员进行一些有效的配合，这并非是预先的约定，也不是长期形成的默契，而是队员本身长期积累的经验，是对场上形势的一种把握。这里模拟的“队员”同样也应具备这样的素质。当它们具备了基本的技能，并且知道一些基于反应式的行为规则，它们就能由此来推导出在特定形势下所应采取的行为，并在不断的学习积累中强化提高。尤其在这种动态不确定环境下，规划不易形成，这种合作就显得尤为重要。由于目前所给的常识规则还不够、也存在一些不确切的地方，就给球员智能体的推导带来了困难，只能完成一些比较明显、简单的合作。更有效的自发产生的合作还在进一步的研究中。



5.2.2 有规划的团队合作

自发合作虽然也有它的不足，它只能完成短期的、局部的、比较明确的目标，而对于长期的、全局的目标就很难胜任了，这就需要规划。没有规划，可以说团队合作就不完整，顶多也只能算是一支“业余球队”。全局规划使全队上下有了一个共同的目标，所有的行为都将为这个目标而服务。全局规划又可划分为一些子规划，以几个队员为一个单元，制定局部的子目标，做一些局部合作。这样逐步细化最终可使每个队员的行为都明确化，这样就使智能体的行为决策更明确、更有效。这种有规划的合作实现也有两种情况，一种是预先建立一

些合作协议，如我方中场开球，当某一特定的条件满足时，如比赛模式（Play_Mode）为我方中场开球（My_Kick Off），即按照该协议执行。此方法在条件比较明确时常常采用，在比赛中由行为决策模块来引用合作协议，作为智能体之间合作的规范，效果还比较好。但是，由于它是相对比较固定，容易让对手找到对付的办法，所以不易过多采用；另外，场上环境的复杂性和实时性，使得有许多情况，是很难确定其触发条件的，这就需把球队看成一个多智能体系统，建立全局的规划。所有队员有一个一致的长期目标，再向下细分为若干个子目标，将智能体分组划分为几个单元来完成这些子目标，以实现小范围的团队合作，使问题简化。逐步实现最终目标。这种规划实现起来比较困难，其根本原因是当前的智能体体系结构中缺少了团队合作的理论，也不能适应实时性的需要[Kitano et al. 1997]。可以考虑选择一些比较好的智能体体系结构如 BDI 模型，在其上来实现规划。

5.2.3 团队合作的一些结构表示

为了实现团队合作，就需要有些合适的结构单元来表示。美国卡耐基梅隆大学球队 CMU98 最早提出了一个团队合作结构，称为更衣室协定[Stone and Veloso 1999]，主要由三部分组成：可以根据协定灵活多变（包括互换）的角色（Role）、由角色组成的阵型（Formation）和在特点场景执行多步、多球员的合作协议（Set-Play）。决策系统的决策依靠世界状态模型和这些结构。

角色 就是一个球员在比赛中承担的责任，包含一个球员所应执行的内部和外部行为的规范，任何行为的执行条件和相关参数都依赖于球员当前的角色。极端的讲，一个顶层行为完全可以写成一个全分支函数，对应每一个特定的角色都有不同的行为模式，然而，这种作法不太现实，因为角色可以影响到球员任何一层的决策行为，即使是在某个行为子树中的一个行为参数。

角色可以严格定义，完全确定一个球员的所有行为；也可以是灵活的，给球员一定的自由度可以让它自己适应角色需要。例如，考虑一个智能体去控制一个钟的整点报时。一种方法为这个智能体严格知道一个角色，就是让它每小时让钟响一次；另一种方法，可以给智能体一个灵活些的角色，让智能体保证每小时钟响的次数在 25% 到 75% 之间，这种条件下，智能体就要考虑在一个可调整的范围内成功满足这个角色要求。当满一个小时时，具体怎么做由它自己决定。通过这种给定参数范围或行为选项的方式，智能体在实现一个角色职责时就有了更大的灵活度。在机器人足球领域中，角色可以一个位置，如中场；在医院维护领域，一个角色是医院侧厅的地板，智能体必须保证其清洁；而在网络搜索领域，角色又可以一个用于搜索的服务器。

在机器人足球领域中，所有智能体都有一些特定的相对独立的职责，每个角色记录都包括了角色的活动区域，初始位置等，当然，不同角色的活动区域可以互相重叠。每个角色有不同的行为模式，这主要是通过行为树（图 5.1）中的条件来体现的；当执行一些合作协议时，有时也会需要它们的位置、任务发生一些变动。

阵型 通过引入阵型可以实现球员智能体之间的合作。一个阵型是由定义了一组特定角色集合的任务空间组成。阵型包括了和球队球员数相同数目的角色，这样每个角色都对应由一个球员充当。如 433 阵型就是 1 个守门员、4 个后卫、3 个前卫和 3 个前锋的集合，每条线上又可以细分为左、中、右等，总共 11 个不同的角色。另外，还可以定义子阵型，或称为单元，不包括整个球队。一个单元包括从阵型中抽取的角色子集，一个队长和在角色中的

单元内部交互。

考虑一个包含 n 名球员的球队 $A=\{a_1, a_2, \dots, a_n\}$ ，任何阵型都可以用以下形式表示：

$$F = \{R, \{U_1, U_2, \dots, U_k\}\}$$

其中 R 是一个角色集合 $R=\{r_1, r_2, \dots, r_n\}$ ，而且 $i \neq j \Rightarrow r_i \neq r_j$ 。要注意角色数量要保持和场上球员数一致（事实上是有可能定义了一些冗余角色，即 r_i 定义的行为和 r_j 相同，这里 $i \neq j$ ）。每个单元 U_i 是一个角色的子集： $U_i = \{r_{i1}, r_{i2}, \dots, r_{ik}\}$ ，这里 $r_{ia} \in R$ ， $a \neq b \Rightarrow r_{ia} \neq r_{ib}$ 并且 r_{i1} 是队长或者叫做单元领导。这个映射 $A \rightarrow R$ 是不固定的：角色可以由不同的异质球员充当。一个单独的角色可以是任何数量的单元和阵型的一部分。

单元是用于出来解决局部问题的。与其要考虑整个队来处理一个局部问题，不如针对它由相关角色组成一个局部单元来处理。队长是一个拥有特权的单元成员，它可以指挥其他单元程序行动。

在球员智能体实现中，角色和阵型的引入是独立的。更衣室协定定义了一个初始阵型，一个初始的球员到角色的映射，以及为了能动态变换阵型而设的实时的触发器。在任何给定的时间，每个球员智能体都明确知道当前球队所使用的阵型。智能体在当前阵型中保持着从队员到角色的映射 $A \rightarrow R$ 。所有这些团队合作的结构信息都存储在智能体的内部状态中。它可以通过智能体的内部行为转换。

由于球员智能体都是全自主的，加上环境的限制（信息局部性、通讯受限），使得无法保证每个队员都清楚无误的知道自己当前的角色，自然也不清楚准确的映射关系 $A \rightarrow R$ 。事实上，球员智能体唯一需要准确知道的是自己当前的角色。因此，在实际实现时可以进行简化，一般不需要每周都更新维护场上所有队友的角色分配信息。合理利用有限的通讯，可以逐步的互相通告协调各自的角色信息，直到所有的角色达成一致。

另外考虑到球场上瞬息万变的情况。可以设计几套不同的阵型以应付各种不同的情况和不同的对手。每个阵型都明确指定场上所有角色和球员的映射关系，当场上的情况满足预定的阵型变化触发条件时（如比分、球的位置、比赛状态等比较明显的客观信息），整个球队就会自动转换阵型，以适应新的情况。

合作协议 作为更衣室协定的一部分，球队可以定义多步、多球员的计划在合适的时机执行。特别是如果存在总是重复发生的特定状态时，球队在这些状态发生之前制定的计划就显得十分有意义。合作协议中存储了各角色站位、任务安排（例如发中场球、角球、任意球、球门球等），一个完整合作协议主要由以下几个部分组成：

- 一个触发条件指明激活一个合作协议所处的状态集；
- 合作协议的角色集 $R_{sp}=\{spr_1, \dots, spr_m\}$ ， $m \leq n$ 定义了合作协议中参与者要采取的行为。每个合作协议角色 spr_i 包括：
 - 一个要执行的合作协议行为；
 - 一个终止条件指明一个球员智能体应该终止充当合作协议角色恢复它的正常行为所处的状态集。

合作协议定义在更衣室协定中以保证球队所有球员都知道。注意这里一个合作协议不需要涵盖整个球队： $m \leq n$ 。更衣室协定也包括一个一般化的函数来将阵型中的角色映射到合作协议中的角色： $F \rightarrow R_{sp}$ 。这样合作协议中的角色就不需要预先指定到具体的球员，而是在球队当前阵型中由任意可以胜任这些角色的球员充当。

决策系统的战略层将主要运用这些结构来表示、实现高层的规划。具体的决策依据是用行为树来表示的(图 5.1)，每个叶结点表示最终在战术层中贯彻执行行为决策。而中间节点则是对场上形势的分析。当然，目前的合作还很低级，必须选择一些比较好的智能体体系结

构如 BDI 模型，建立完整团队合作的理论，能把新的精神状态作为团队合作的基础，如团队目标、团队规划、共有信念和联合承诺等[Kitano et al. 1997]，才能实现更高层次的团队合作。

5.3 更有效的获取信息

5.3.1 信息更新采集

前面介绍的决策机制对于信息的依赖性较强，同时由于 RoboCup 仿真环境中信息采集的局限性，使得信息的采集始终是不完全的，这就要求一方面程序设计必须考虑提高球员信息采集的效率，另一方面必须采取新的方法来弥补信息不足带来的弊端。如对球、球员的运动建模，以通过有限的、不准确的信息推测球、球员的位置，从而弥补信息的不完全。

RoboCup 仿真环境中，信息获取的主要手段是视觉信息的获取。每个队员所能得到的视觉信息就是在其可视范围内的所有内容，球员的可视范围是其面向方向的视野。球员的视野宽度有 45 度、90 度和 180 度三种，较窄的视野宽度所获得信息的频率较快。目前 RoboCup 仿真环境中规定 45 度视野宽度每 0.75 周期获得一次视觉信息，90 度视野宽度每 1.5 周期获得一次视觉信息，180 度视野宽度每 3 个周期获得一次视觉信息。球员的面朝向受到球员身体朝向的限制，面朝向与身体朝向的偏差不能超过 90 度。球员可以通过转脖子的动作来改变朝向，转脖子的动作可以伴随其他动作进行。

通过上面描述可以看到，在足球这样一个复杂的环境里，必须以一种非常“聪明”的方式合理调整传感器才能保证准确的跟踪场上状态的变化，另外，对于不同的情况智能体所关心的场上信息往往是不同的。如果一个球员智能体正在控球并且靠近对方的球门，那么对方守门员的位置、速度等信息对该球员来说就是非常重要的；而对于一个在中场控球的球员来说，显然这些就不太重要了。

RoboCup2000 仿真组的冠军葡萄牙队 (FCPortugal) 当时采用了一套球员感知策略称为 SLM (Strategic Looking Mechanism) [Reis and Lau 2000]。这种观察机制综合考虑球员当时的角色、阵型、位置以及场上所有物体的位置速度信息的置信度来决定每周周期球员所要观察的方向。对于每一个可能的观察方向，都有一个效用值来评价其可能带来的收益。这个效用值是通过计算一次观察可以带来全场物体置信度的增加总和得到。当然，考虑到决策的不同需要，根据场上形势，球和自己的位置等信息，在求和时会对不同的物体使用不同的权重。当选择好最佳观察方向后，即可通过 turn neck 指令调整球员视角，以获得期望的信息。

在保证了决策出的行为动作完成的同时，通过调整面向角和视野宽度来获取更多的视觉信息，采集最有用的信息，对决策无疑是有巨大帮助的。然而仅仅通过转脖子动作来满足这种视觉需求是不够的，由于受到视野和脖子转角的限制，转脖子动作所能满足的视觉需求仅仅只是身体朝向上附近一定范围内的信息。为了能够更加全面的采集，必须考虑转身动作，来调整身体朝向。但是这样一来，必然与决策出的动作相冲突，这就需要信息采集与决策出的动作相协调；另外采用不同的视野会导致获得信息的时间也不同，这就需要球员根据需求的紧迫程度来权衡选择。

为了能够满足上述要求，就需要有个更全面的评价机制。如中国科大蓝鹰队在决策时对决策所用到的所有信息分别给出一个关心程度。然后，将可以同时在一个视野内满足的视觉信息合并成为一个需求，对所有的需求给出优先级，最后满足可以实现的最高优先级的视觉需求。取得 RoboCup2001 仿真组冠军的清华队 (TsinghuaAeolus) 在其设计框架中有专门的

一个视觉仲裁模块（Visual Mediator）来处理上述主动信息采集问题[Yao et al. 2002]。

当信息采集和动作发生冲突时，只能满足两者之中的一个，这就要求比较究竟哪一个更加关键，更加重要。由于信息采集行为给出了其优先级，而行为选择也有它的一套评价机制，因此由两个不同的评价体系评价出的结果，要比较其关键性，就有一定的困难。

下面给出一个简单的解决方法：通过当前信息采集需求对决策的影响，来确定该信息是否为关键信息；而所执行的动作，也根据其所能达到的效果确定来判定其是否为关键动作。对于关键动作，一般原则上是不能中断的，但当需求信息对动作的效果有重大影响时，无疑信息获取就更为关键了，就必须满足信息获取的需求。通常而言，如果信息需求的紧迫性还没有达到相当的程度，关键动作是不能中止的。简单的说，这里的做法就是当动作与信息采集发生冲突时，牺牲动作和信息采集中重要性较低一个。而这个判断可以通过认为直接给定。

5.3.2 通讯

[假期重点可以看这里](#)

由于一个仿真球队是一个分布式系统，每个球员都是自治的智能体，因此智能体无法完全了解任何时候队友的信念和意图。智能体除了通过队友的行为模式、预先制定的合作协议来做出部分判断外，还可以利用通讯来增进彼此的了解，提高合作的成功率。良好的通讯是进行有效合作，在对抗中取得优势的基础。

在仿真平台中，通讯是实时的，并且是受限的，具有单信道、窄带宽、不可靠等特点。单通道意味着每队球员使用同一个通讯信道，互相有干扰，每个球员每周期只能听到一条队友通讯信息，因此需要做好发送消息的选择，有主次的进行通讯，避免干扰队友的通讯；窄带宽是说球员的通讯容量受到限制，目前比赛规则规定一次只能发不超过 10 字节的信息，这使得一条信息的内容十分有限，为了提高信道利用率，可以对信息进行适当的编码；不可靠的通讯信道会导致丢失消息，所以智能体在行为决策时要考虑到这一点，只能利用这些消息，而不能依赖这些消息。比如在进行传接球配合时，发起者，也就是控球者在通知接球者准备好后，不能指望等到接球者回应再执行，一方面通讯有延迟，这一来一回就要耽误 2 个周期，另外消息传递过程中还有可能丢失，所以在实际执行时都是先通知接球方注意，随即立刻执行，假定队友已经接收到消息，并做好准备。

在这样一个受限的通讯环境中如何稳定、高效地交互信息也就成为 RoboCup 中又一个重要的挑战课题，具体可表述为以下几个方面：

1. 队员需要一种有效的判断方法来确定在这个单通道内哪个信息是要发送给哪个队友的（不但要确定自己最应该发送什么消息，而且还要判断是否干扰队友的通讯）
2. 由于每个周期一个球员只能听到一条信息，所以应该避免多名队员“同时说话”。虽然有很多通讯需要许多队员的回应（如多人战术协调），但如果大家一起回应，必然大部分回应会丢失
3. 由于通讯带宽很窄，要设计一个高效的编码算法使得在一条有限的通讯中尽可能多包含一些内容，同时还要保证编码后的每个字节都是合法的传输字符，如 0-9、a-z 等
4. 由于通讯不可靠，球员设计必须保证足够的健壮性：球员的行为不能依赖从队友接收到的消息，这样才能确保即使丢失消息，程序一样可以正常运行
5. 尽管每个球员都是自主行动，通讯又不可靠，球队应该最大可能地保证所有球员全局战略（如阵型）的一致性

关于这个通讯环境的特点及挑战问题总结见表 5.1。

表 5.1 RoboCup 通讯环境特点及挑战

通讯环境	挑战
窄带宽	高效的信息编码
不可靠	对丢失信息的健壮性
单通道	消息目标确定及区分, 抗干扰能力
多主体、团队	多主体同时响应
	全队协同

在目前的球队中, 智能体之间的通讯主要分为以下几类:

- 1 要求(通报)位置信息: 包括己方队员、对方队员和球的位置、方向及速度等。
- 2 通报自己正在执行的行为模式。例如告诉其他智能体自己正在上前拦截对方的拿球队员。这类通讯有利于智能体之间的分工合作, 避免几个智能体执行同样的任务。
- 3 要求(响应)某个智能体执行某个任务。例如, 一个智能体在传球的时候, 会向队友发出要求他注意接球的消息。
- 4 分析当前形势, 提出合适的行为规划。

5.4 小结

设计球员决策是 RoboCup 仿真研究的核心内容, 上面仅给出了一些可行的方式。更多的相关研究仍在进行中。如德国的 BrainStormer 球队[Riedmiller et al. 2001], 一直致力于强化学习应用的研究, 并且已实现的一些底层行为如踢球、拦截等, 效果都挺不错, 目前正逐步扩展到高层行为以及团队合作的学习中。

另外充分利用平台的特色, 最大程度发挥其作用也是设计决策要考虑的问题。比较典型的如异质球员的使用, 2001 第一次参加比赛的荷兰 UvA 队就是充分发掘这个特色, 合理分配场上队员的属性, 设计了与之相适应的进攻防守策略, 进攻时充分利用边锋比对方后卫跑得快的特定, 贯彻下底传中打法, 取得了优异的成绩[Boer et al. 2002]。

总之, 球员决策包含了非常丰富的内容, 在设计时要注意全面分析问题, 逐层分解, 针对具体问题具体解决。详细内容见后续章节。

参考文献

[Noda et al. 1997] Itsuki Noda, Shoji Suzuki, Hitoshi Matsubara, Minoru Asada, and Hiroaki Kitano. Overview of RoboCup-97. In Hiroaki Kitano, editor, RoboCup-97: Robot Soccer World Cup I, pages 20{41. Springer-Verlag, 1997.

[Kitano et al. 1997] Hiroaki Kitano et al. The RoboCup Synthetic Agent Challenge 97. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, San Francisco, CA, 1997. Morgan Kaufman.

[Stone and Veloso 1999] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artificial

Intelligence, 1999.

[Reis and Lau 2000] Luis Paulo Reis and Nuno Lau. FC Portugal Team Description: RoboCup 2000 Simulation League Champion. Univ. of Porto & Univ. of Aveiro. In Stone P., Balch T., and Kaetschmar G., editors, RoboCup 2000: Robot Soccer World Cup IV. pp.29-41, Springer Verlag, Berlin, 2001

[Riedmiller et al. 2001] M. Riedmiller et al. Karlsruhe brainstormers—a reinforcement learning approach to robotic soccer. In Peter Stone, Tucker Balch, and Gerhard Kraetschmar, editors, RoboCup-2000: Robot Soccer World Cup IV. Springer Verlag, Berlin, 2001.

[Yao et al. 2002] Jinyi Yao et al. Architecture of TsinghuAeolus. RoboCup-2001: Robot Soccer World Cup V, Andreas Birk, Silvia Coradeschi, Satoshi Tadokoro editors, LNAI, Springer Verlag, 2002

[Yang et al. 2002] B. Yang et al. WrightEagle Simulator Team. In: Pre-Proceedings of RoboCup-2002 (CD version), Fukuoka, Japan

[Cai et al. 2002] Yunpeng Cai, Jiang Chen, Jinyi Yao and Shi Li. Global Planning from Local Eyeshot: An Implementation of Observation-Based Plan Coordination in RoboCup Simulation Games. Robot Soccer World Cup V, Andreas Birk, Silvia Coradeschi, Satoshi Tadokoro editors, LNAI, Springer Verlag, 2002

[Jinyi Yao et al. 2003] Jinyi Yao et al. Technical solutions of TsinghuAeolus, in RoboCup Symposium 2003, Padova, Italy

[Kok et al. 2003] Jelle R. Kok, et al. UvA Trilearn 2003 Team Description. In Proceedings CD RoboCup 2003, Springer-Verlag, Padua, Italy, July 2003.

[Boer et al. 2002] R. De Boer, J. Kok, and F. Groen. UvA Trilearn 2001 Team Description. RoboCup 2001: Robot Soccer World Cup V. Springer-Verlag, Berlin, 2002.

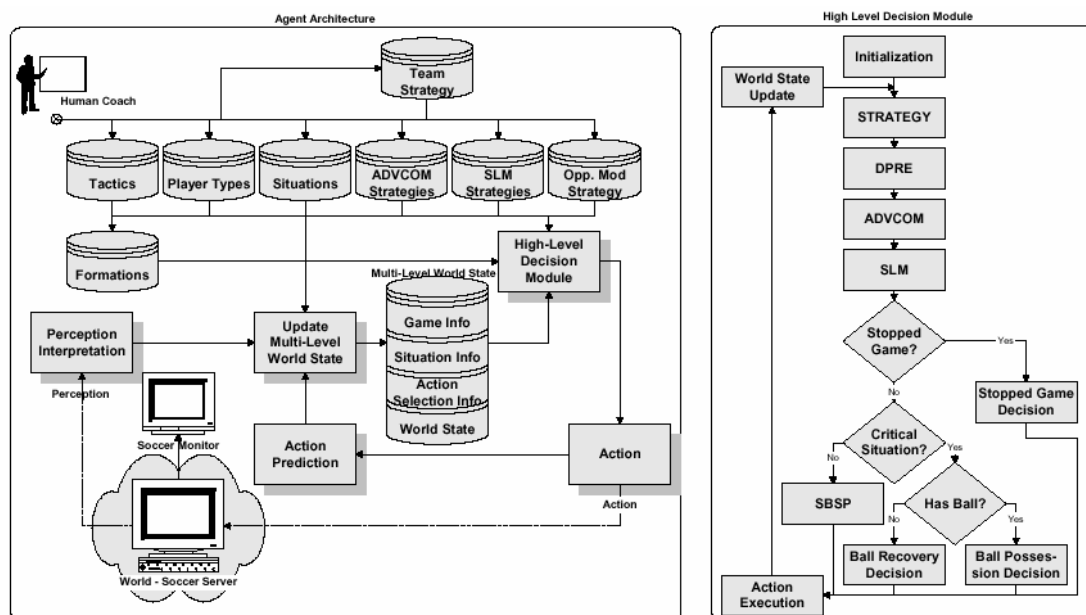


图 2：智能体结构和高层决策模型的控制流程

智能体的主要控制循环使用感知解释和动作预测来更新世界模型（9），然后使用高层决策模型来决定下一步的动作。FC Portugal 的信息模型是一个四层结构的数据模型：

全局形势信息 — 高层信息比如比分、时间、比赛策略（射门，成功传球，控球等）和对方行为，用来确定每一时刻的战术；

形势信息 — 和阵型选择相关的信息，和 SBSP, SLM(Strategic Looking Mechanism 策略上的观测机制)和 ADVCOM(Intelligent Communication Mechanism 智能通讯机制)机制相关的信息；

动作选择信息 — 一套高层参数，用来确定动态形势，选择适当的控球或开球行为；

世界状态 — 底层信息，包括球员和足球的位置和速度。

高层决策模型不仅决定了球员动作，而且决定了战术、阵型、站位和每一时刻的球员类型。这个高层决策模型（图 2）初始化以下内部结构后，进入智能体的主控制循环。这个循环通过允许 STARTEGY 模型开始，智能体使用比赛的全局信息确定当前战术，使用形势信息来确定阵型。然后，智能体分析 DPRE（Dynamic Positioning and Role Exchange 动态站位和角色变换）的可能性，执行智能通讯分析和策略观测模型。如果进而死球情况，执行死球比赛站位。如果活动形势没有被确定，就表明智能体不需要马上进而活动行为，执行 SBSP（Situation Based Strategic Positioning 基于形势的策略站位）。如果形势是活动的，而且智能体是控球的，就会选择一个恰当的控球动作，否则就会选择一个恰当的开球防守动作。还有不是决策部分的构成了循环：动作执行和多层世界状态更新。

6.3 TsinghuAeolus 中国清华大学仿真球队[chenjbs]

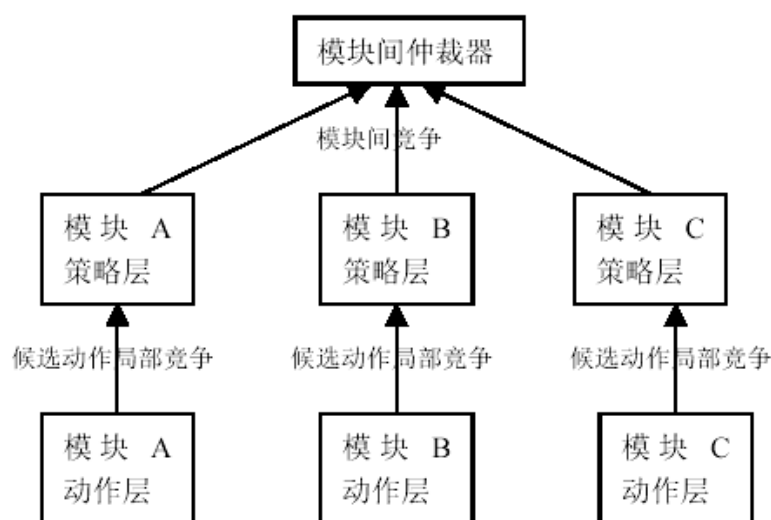
在 RoboCup 仿真环境中，由于状态空间过于庞大以及反馈延迟过大，不融入任何先验知识的决策设计很难取得良好的效果。即使对于一些非常简单的概念如带球、截球、射门、传球等，要让智能体自动地从与环境的交互中总结出来并组织成可用的知识，将是十分困难的。从人类自身获取知识的实际情况来看，通过教学活动所获得的知识占了很大的比例。基于以上考虑，TsinghuAeolus 队认为，一个融入人类的先验知识并且能在一定程度上自主学习的决策系统，是一个有价值而且可行的设计目标。Tsinghua 队主要利用“框架”和“建议”

的形式表达相关的人类知识。比如，人类往往自然而然地把决策分解为进攻和防守，如果还有点足球知识的话，可以更细致地分解为带球、截球、传球、射门等等。在 Tsinghua 队的决策体系中，这些概念都是事先假定的，也就是以所谓框架的形式存在。作为建议的人类知识通常以限定求解方向的形式存在，也就是一种启发式知识主要用来加速搜索。还有一部分人类知识通过教练的方式加以体现和运用，如角色、角色行为的定义、阵形、定位球配合等等。

环境的复杂性使得设计者不可能完全通过先验的知识来确定智能体每一时刻的行为，而且人类的知识往往不能完全适合模拟环境。智能体本身具有一定的学习和适应能力，能够自己从环境中获取知识，也成为决策系统是否成功的一个重要因素。目前流行的自适应学习算法有 BP、动态规划、Q-learning、遗传算法等。当运用于复杂决策时，这些学习算法普遍面临的问题是所能处理的问题形式单一，学习能力有限。如何有效的把这些学习算法获取的知识运用在决策体系中，使之能够处理形式复杂的各种决策场景，也是 TsinghuAeolus 球队设计中的一个难点。

● 决策系统结构

在纵向，TsinghuAeolus 队采用了分层的结构，分成基本动作层和策略层两层；在横向，TsinghuAeolus 按足球的各个基本的技战术动作来划分模块，分成截球、传球、带球、射门、跑位、视觉、通讯几个模块。每个模块由底层的基本动作和上层的决策策略构成。底层评价在不考虑战术收益情况下动作的可行性（feasibility）。而上层策略层结合战术收益，对候选动作做进一步的评价和仲裁。模块内部的候选动作先进行局部竞争，然后再提交给更上层的仲裁器进行模块之间的仲裁。概括的说，总体的结构是一个分层、分模块的多级评价仲裁器，如图？（图号）所示。



TsinghuAeolus 球队决策系统

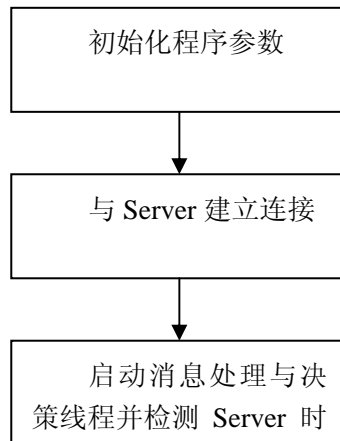
TsinghuAeolus 采用多级仲裁和局部竞争，使得模块的评价器对同一模块内候选动作的评价具有合理性，但是在模块之间就可能失去可比性了。尽管如此，模块的评价结果至少表明了该候选动作在目前环境中的适应程度，即在目前状况下执行该动作的“好坏”，只不过不同模块用来评价好坏的标准不同。模块之间的仲裁器可以看作是一个平衡标准的动态加权器，对不同模块之间的评价结果进行动态加权，使它们具有某种意义上的可比性。这个全局

的动态仲裁器，可以根据实际的比赛效果来学习调整，也可以由人类足球专家提出建议。比如设计者可以有意的加重传球动作的权值，引导智能体实现多传球的战略意图。

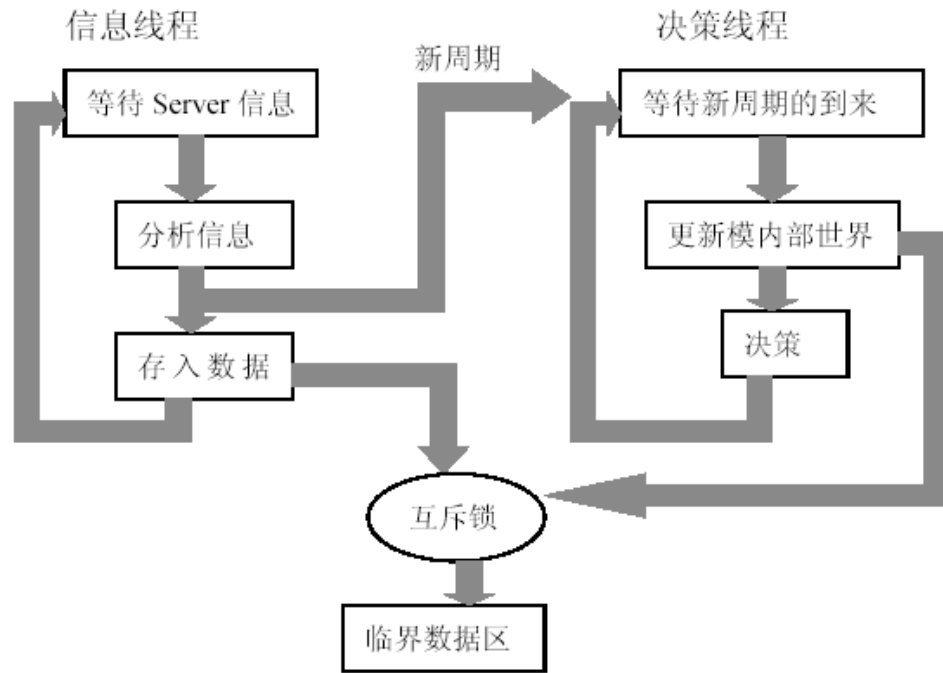
● 程序结构

RoboCup 仿真组比赛采用 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。球队与 Server 之间的通讯依照规范进行，各队结构大同小异。

TsinghuAeolus 的程序支持 Windows/Linux 双系统，它的启动流程如图？所示。



TsinghuAeolus 程序由信息处理和决策两个线程组成。信息处理线程接收和分析 Server 发送的信息，并将分析好的数据存到一个指定的数据区。Server 和 Client 采用异步的方式运行，每一个信息头都有一个时间标签用于同步。信息线程通过判断收到信息的时间标签，来判断是否进入一个新的仿真周期，如果进入新的仿真周期，则激发新仿真周期事件。决策线程做完一个周期的决策后，将等待新的仿真周期事件。新的仿真周期到来后开始从分析模块存储的数据区中读取信息数据，并通过更新模块对当前状态进行估计，把数据转化成内部世界模型，然后进行决策。参见图？。



执行流程

信息线程和决策线程有一块共同的临界数据区，TsinghuAeolus 程序采用互斥锁来实现互斥访问临界数据区，这一设计保证了内部世界模型在一次决策过程中始终不变，从而避免了许多潜在的问题。但是，这个设计也存在一定的弊端，在决策过程中间，Server 可能发送新的消息，决策不能根据最新的信息做出判断。

检测 Server 的时钟程序每 2 秒钟被触发一次，它是用来检测 Server 是否还在正常工作，或者通讯是否运转正常。程序中维护一个 Server 空闲时间的计数，每当信息线程收到 Server 的一个信息，这个计数被清零，当时钟程序被触发时计数加 1。当计数累计到一定大小时，时钟程序断定 Server 已经不存在或是网络连接出了故障，这时它将中止整个程序。

6.4 Brainstormer 德国卡尔斯鲁厄仿真球队

6.5 WrightEagle 中国科技大学仿真球队

6.6 小结

参考文献

[Stone 1998] Peter Stone. Layered Learning in Multi-Agent Systems. PhD Thesis, Computer Science Dep.,Carnegie Mellon University, December 1998

[Stone 1999] Peter Stone et. al. The CMUnited-99 champion simulator team. In M. Veloso, E. Pagello, and H. Kitano, editors, RoboCup-99:Robot Soccer World Cup III, pages 35–48. Springer Verlag, Berlin, 2000.

[Reis and Lau 2000] Luis Paulo Reis and Nuno Lau. FC Portugal Team Description: RoboCup 2000 Simulation League Champion.Univ. of Porto & Univ. of Aveiro. In Stone P., Balch T., and Kaetschmar G., editors, RoboCup 2000:Robot Soccer World Cup IV. pp.29-41, Springer Verlag, Berlin, 2001

[chenjbs] 陈江. 清华大学仿真机器人足球队设计实现. 学士毕业论文, 2002

第7章 信息获取与维护

前面的章节已经提到，客户端（client）从服务器端（server）接收的信息有视觉（vision）信息、听觉（aural）信息和感知（sense-body）信息等。一般来说，客户端都有一个世界模型（world model），这是一个可以保存和获取所有客观信息的对象结构，对于仿真组的 client 来说，这些信息包括各个队员的位置、速度、身体朝向、异构类型，球的位置和速度，以及队员自己的信息比如体力等。本章的将向读者介绍如何把从 server 接收的字符串格式的原始信息转化成客户端决策程序的世界模型。

本章分为以下几个小节：

- 1) 信息获取的一般方法
- 2) 提高信息处理的精度
- 3) 信息不确定时的猜测
- 4) 主动信息采集
- 5) 实现一个简单的 client

本章提到的各种算法，源代码可以通过以下几个途径获得：

- ✧ 各参赛队公布的代码里面
- ✧ server 的源代码里面
- ✧ 作者实现的测试用 client 源代码

7.1 信息获取的一般方法

对于 client 来说，最基本的信息就是场上物体的位置信息，最基本的信息获取就是定位，也就是通过看到、听到和感觉到的信息推算出队员或者球的位置和速度等。这一节介绍最简单的定位和预测的概念和算法。

7.1.1 根据视觉信息定位

场上的物体（object）分为动态物体（dynamic object）和静态物体（static object）：动态物体指队员（player）和球（ball）；静态物体指地标（flag）和边线（line）。在现有的参数情况下，场上有 1 个球、22 名队员（每边各 11 个）、145 个地标和 4 条边线，图 7.1 是球场的一部分，上面有一个队员，一个球，空心的小圆圈代表地标。

静态物体的位置是确定的，client 通过边线和地标这两种静态物体就可以推测出自己的位置信息。

边线的作用是获得 client 的视角中心线的朝向（一般称为脖子朝向，neck angle）。server 信息中的边线信息结构为“((l 编号) 距离 角度)”，编号用来区别看到的边线是上、下、左、右四条边线中的哪一条，角度是边线朝向与 client 脖子朝向的夹角。client 在边界内的时候，只能看到一条边线，但如果在界外的话则会看到两条边线（这样可以通过看到边线的数量确定是在界内还是在界外）。边线是静态物体，本身的角度是固定的，通过编号就可以得知其角度，通过边线的角度和其与 client 脖子朝向的夹角就可以确定 client 的脖子朝向。在感知信息里面可以知道 client 的脖子朝向和身体朝向（body angle）的夹角，通过这两个

角度就可以确定 client 身体的绝对朝向。

知道 client 脖子朝向和某一个地标的相对位置和相对角度之后，就可以通过地标的绝对位置反推出 client 自己的绝对位置，如果还能看到地标的相对速度，也能反推出 client 的绝对速度。但是一般情况下看到的地标不只一个，这时候可以选择距离最近的一个地标作为参考，因为距离越近观察误差就越小，下一节的蒙特卡洛定位算法（Mento Carlo Localization）则尽可能的使用看到的所有地标来提高定位精度。

我们来看下面的例子（运行环境为 Gentoo Linux 2.6.9）：

首先启动 server 和 monitor：

```
peter@gentoo ~ $ rcsoccersim
```

然后启动 server 代码中提供的示例 client 程序（sample client）：

```
peter@gentoo ~ $ rcsscclient
```

在 input 窗口里面输入“(init team-name (version 9.0))”就可以连上 server 了，输入一系列命令让这个 client 移动到场地里面（因为在场外地边会看到两条线，不是一般情况），坐标为（0，20），随意转动身体和脖子：

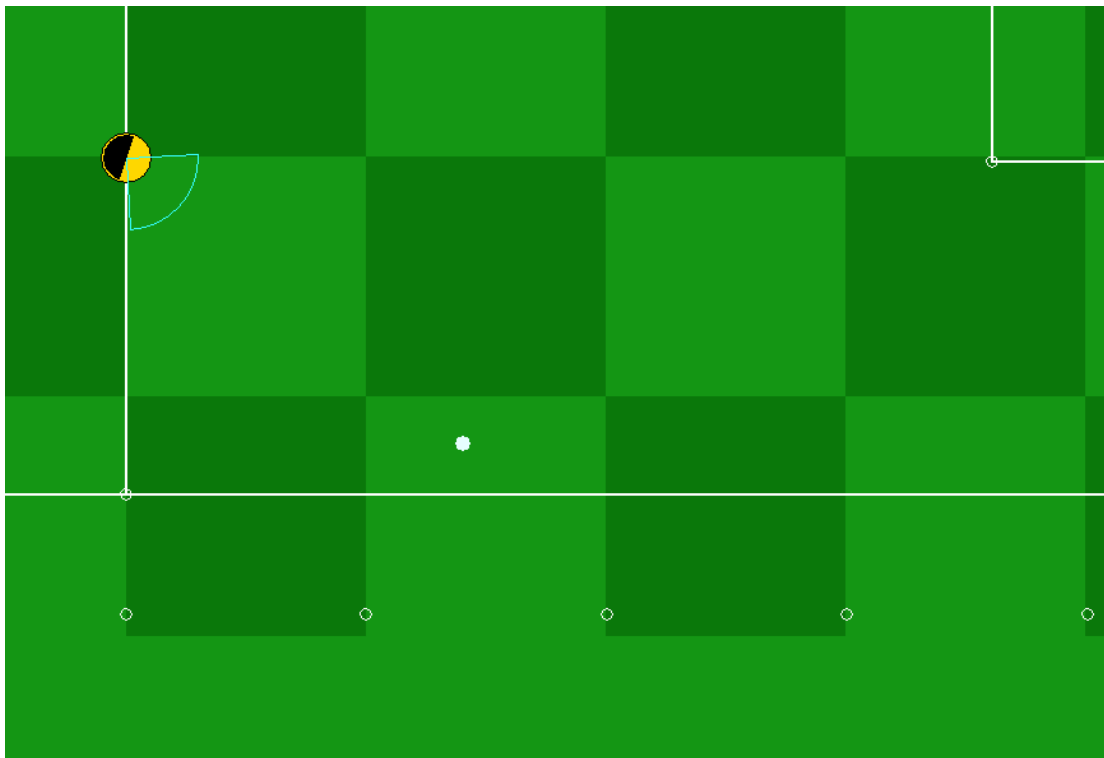
```
(move 0 20)
```

```
(turn 20)
```

```
(turn_neck 20)
```

场景如图 7.1 所示。然后就会看到下面的视觉信息：

```
(see 2026 ((f r b) 54.6 -25) ((f p r b) 35.9 -40) ((f b r 10) 21.5 22 0 0)
((f b r 20) 27.7 3 0 0) ((f b r 30) 35.5 -8) ((f b r 40) 44.3 -15) ((f b r 50)
53.5 -19) ((f r b 20) 57.4 -40) ((f r b 30) 58.6 -30) ((b) 18.2 -2 0 0) ((l b)
21.8 -40))
```



• 图 7.1，球场的局部场景

上面的信息是说在第 2026 周期看到了 9 个地标和一条边线，首先根据边线信息来确定视角的绝对朝向，看到的是下边线“(l b)”距离 21.8 米，角度-40 度，那么脖子角度计算

方法如下：

```
lineRelativeAngle = (line_dir < 0 ) ? (90 + line_dir ) : (line_dir - 90)
= -40 + 90 = 50
neckGlobalAngle = lineGlobalAngle - lineRelativeAngle
= 90 - 50 = 40
```

而此时的感知信息：

```
(sense_body 2026 (view_mode high normal) (stamina 4000 1) (speed 0 -40)
(head_angle 20) (kick 0) (dash 0) (turn 1) (say 0) (turn_neck 1) (catch
0) (move 1) (change_view 0) (arm (movable 0) (expires 0) (target 0 0) (count
0)) (focus (target none) (count 0)) (tackle (expires 0) (count 0)))
```

里面“(head_angle 20)”说明当前的脖子角度相对身体角度夹角为 20 度，因此身体朝向绝对角度：

```
bodyGlobalAngle = neckGlobalAngle - head_angle
= 40 - 20 = 20
```

然后找距离最近的地标，是 (f b r 10)，距离 21.5 米相对脖子朝向夹角 22 度，这个地标的绝对位置是 (0, 39)，根据相对位置和相对角度就可以推出 client 的位置：

```
playerPos = flagPos - polar2vector(flagDist, neckGlobalAngle + flagDir)
= (10, 39) - polar2vector(21.5, 40 + 22)
= (10, 39) - (10.094, 18.983)
= (-0.094, 20.017)
```

这个位置和 client 的真实位置 (0, 20) 非常接近。

位置和角度确定了，然后就可以确定 client 的速度，有两种方法可以获得，一种是通过看到的地标的相对速度反推得到，另一种是利用感知信息里面的“(speed speedValue speedAngle)”项来获得。虽然两种方法都可以，但是精度不一样，地标的相对速度和地标的相对距离有关系，精度随距离增加而变差，有时候还有可能距离太远看不到地标的相对速度；感知信息的速度精度只和 client 的速度大小有关系，速度越大精度越小，所以一般使用后者来确定速度，除非通过地标获得的速度的精度很高，后一种方法读者可以参考源代码。

例如在某个周期感知信息中速度信息为“(speed 0.41 -11)”，当时的脖子角度为-4 度，client 的速度计算如下：

```
playerVel = polar2vector(speedValue, neckGlobalAngle + speedAngle)
= polar2vector(0.41, -4 + -11)
= (0.39603, -0.106116)
```

client 自己的信息确定之后，其他物体，包括队友、对手、球等也都可以通过视觉信息里面的相对位置和相对角度推算出来。举例说明，同样是 2026 周期，在视觉信息里面有球的信息：“((b) 18.2 -2 0 0)”，可以知道球与 client 距离 18.2 米，相对脖子朝向角度-2 度，那么球的位置：

```
ballPos = playerPos + polar2vector(ballDist, neckGlobalAngle + ballAngle)
= (-0.094, 20.017) + polar2vector(18.2, 40 - 2)
= (14.248, 31.222)
```

球的速度可以根据视觉信息中的相对速度推算，例如某个周期球的信息为“((b) 2.2 6 0.22 2.8)”，client 的速度为(0.39603,-0.106116)，脖子角度为-4 度，则球的速度计算如下：

```
relPos = polar2vector(1.0, ballAngle)
= polar2vector(1.0, 6)
```

```

= (0.994522, 0.104528)
relVel = (distChg * relPos.x() - (dirChg * PI / 180 * ballDist * relPos.y()),
distChg * relPos.y() + (dirChg * PI / 180 * ballDist * relPos.x()))
= (0.207557, 0.12992)
ballVel = playerVel() + relVel.rotate(neckGlobalAngle)
= (0.39603, -0.106116) + (0.207557, 0.12992).rotate(-4)
= (0.612143, 0.00900887)

```

7.1.2 信息的简单预测

视觉信息并不一定每个周期都有，根据 **server** 内部的更新机制，视觉信息的周期和时钟周期不是同步的，而是和视角的宽度（**view width**）还有视觉信息的质量（**view quality**）有关系。从前面的章节可以知道，有三种视角宽度（**narrow, normal, wide**），两种视觉质量（**low, high**），视角宽度或者视觉质量增加一个量级，视觉信息的周期增大一倍。比如，在现有的参数下面，缺省的是普通视角宽度和高质量，这时视觉信息的周期是 150 毫秒，而时钟周期是 100 毫秒。这样有的时钟周期里面就没有视觉信息，但是 **client** 每个周期都要决策计算，这个时候就需要预测没有视觉信息的这个周期的物体信息，才能进行正确的决策。

预测的方法就是通过现有的信息（包括位置、朝向、速度等）模拟 **server** 内部的更新机制。在感知信息里面，**server** 会告诉 **client** 命令执行成功与否，命令执行成功，感知信息里面对应的项数会值增加 1，否则不会增加，比如前面的 2026 周期的感知信息，**move**、**turn** 和 **turn_neck** 的值都是 1 说明这几个动作都只执行了一次，而前面的例子也只发了一次，意味着动作正确执行了。动作执行之后，就可以计算其效果，举例如下：

周期 1，**client** 的位置 (-0.094, 20.017)，速度 0，身体朝向绝对角度 20 度，执行命令（**dash 100**），周期 2 没有收到视觉信息，进行预测。从感知信息知道 **dash** 命令执行成功，那么周期 1 末尾的加速度：

```

accel = effort * power * dash_power_rate
= 1 * 100 * 0.006 = 0.6

```

周期 1 末尾速度：

```

vel = old_vel + polar2vector(accel, direction)
= (0, 0) + polar2vector(0.6, 20)
= (0.564, 0.205)

```

周期 1 末尾也就是周期 2 的位置：

```

pos = old_pos + vel
= (-0.094, 20.017) + (0.564, 0.205)
= (0.47, 20.222)

```

从周期 1 到周期 2 速度会有所衰减，周期 2 开始的速度：

```

vel = vel * player_decay
= (0.564, 0.205) * 0.4
= (0.226, 0.082)

```

同理，转身、转脖子等其他动作的效果也可以得出。

在比赛中，经常碰到看到球却看不到球速度的情况，因为只要球在 **client** 周围 3 米以内，就能感觉到球的位置，但球要在 **client** 视野之内才能看到速度。球的速度对于控球非常重要，尤其是在射门和带球的过程中，这时候就需要预测球的速度。

预测的方法有多种。传统的是通过球的运动模型和踢球模型来预测。比如，连续两个周期得到球的位置，那么球在上一个周期末的速度：

```
ballVel = ballPosThisCycle - ballPosLastCycle;
```

如果在上一个周期，队员发送了 kick 命令，则球的速度变为：

```
ballVel = ballVel + ballKickVelInc;
```

经过衰减及之后，球在当前周期的速度：

```
ballVel = ballVel * ballDecay;
```

还有一种情况，就是球与队员产生碰撞，球的速度受到影响，这种情况要根据两者的位置判断是否发生了碰撞，再进行处理。

7.1.3 时钟周期的同步

既然视觉信息和时钟周期不同步，而且信息的传输使用的是没有连接的 UDP 协议，并不保证所有的消息都能正常到达，那就存在一些同步问题，比如正在进行决策处理的时候来了视觉信息，或者正在等待本周期的视觉信息，但是此信息却在传输过程中不见了。目前仿真组的程序同步方式有两种，一种是信号量（signal），另外一种线程（thread），信号量类似于中断，系统在某种消息到达时，把当前正在执行的内容压栈，进入一个事先定义好的函数入口，此函数执行完之后，把先前入栈的内容弹出，继续执行；线程就简单一些，接收 server 消息是一个线程，进行决策是另外一个线程，运行是独立的，利用互斥锁进行同步。线程对于信号量来说，优势比较明显，所以大多数仿真组参赛队特别是近几年参赛的队都是用线程的，信号量只能在 Peter Stone 的 CMUnited97、98、99 的经典代码里面才能看到。线程方法中，消息线程的执行流程非常简单，就是等待消息，消息来了之后处理消息，处理完了继续等待；决策线程的执行流程稍微复杂一些。整个流程如图 7.2，带箭头的虚线表示触发事件：

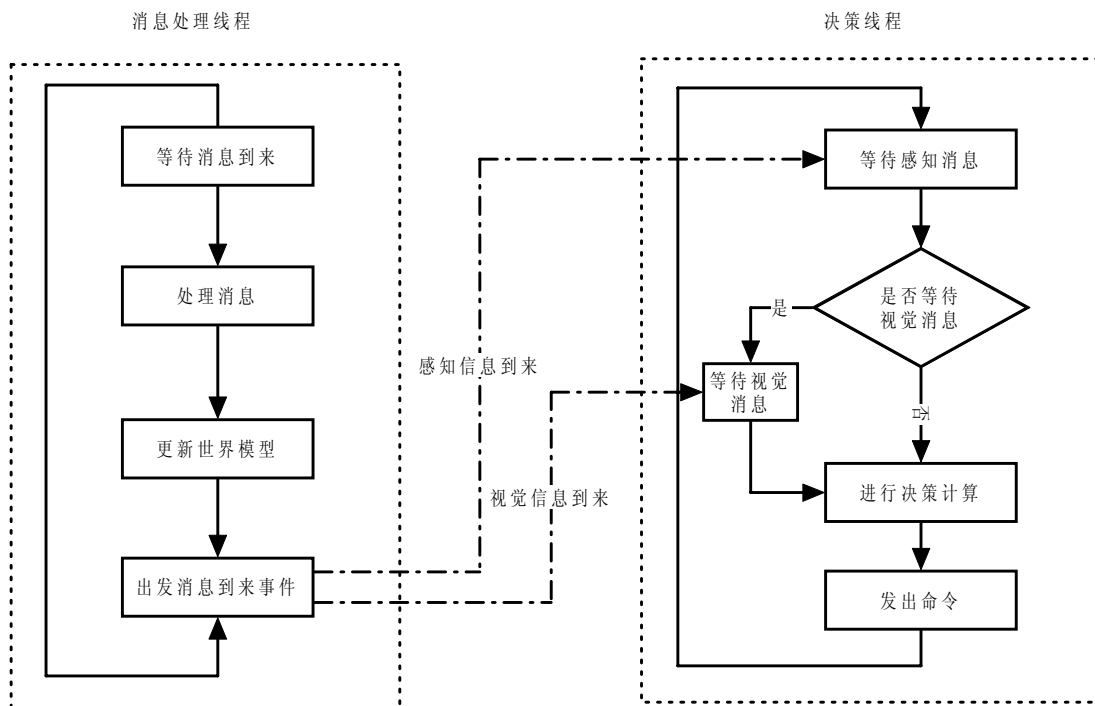


图 7.2 client 程序双线程执行流程

在本书提供的源代码里面，很好的体现了这个流程，不过还有一种问题没有解决，就是 UDP 协议里面数据丢失问题，解决方法是等待事件的时候设置一定的时间，超过这个时间就不再等待。由于这种情况出现概率很低，也可以不处理，造成的结果是一个周期没有进行

决策。

7.2 提高信息处理的精度

在仿真 server 里面为了模拟真实情况，人为的加入了一些随机噪声，比如运动物体的速度在每个周期都会有一定比例的偏差，速度越大偏差越大，如果把球的偏差参数设得足够大，甚至有可能可以看到球的运行路线是随机线路。这一小节的主要内容是分析随机噪声对定位精度的影响并给出一些解决方法。

7.2.1 误差的产生

通过 server 的文档和源代码可以知道噪声的原理，首先是运动噪声，再就是观察噪声。在 server 的参数里面有 player_rand 和 ball_rand，这两个就是队员和球的运动噪声的大小。在 server 的原代码里面，可以看到运动噪声的产生方法，展开如下：

```
maxrnd = randp * vel.r() // 误差和物体的速率成正比
noise = vector(drand(-maxrnd, maxrnd), drand(-maxrnd, maxrnd))
vel += noise // 在原有的速度基础上加入噪声
```

也就是噪声均匀分布在一个半径最大值和速度大小成正比的圆内，比如在当前的参数下 player_rand 为 0.1，那么一个队员在速度 1 米/秒的情况下，他的运动方向最大可以偏差 6 度，在数值上可以偏差 0.1 米/秒，当然这是极限情况下。也就是说，如果在某个周期没有看到视觉信息，只能通过预测来定位，而且前一个周期队员的速度达到了 1 米/秒之后，由于运动误差产生的定位误差平均就有 5 厘米。

观察误差，则是 server 在产生视觉信息的时候，把看到物体的距离和相对的角度值进行取整计算，从而产生的截断误差。距离的截断方式如下：

```
double quantize(double v, double q){
    return ((rint((v)/(q)))*(q));
}
double calc_quantized_dist(double dist){
    return quantize (exp ( quantize ( log ( dist ), 0.01 ) ), 0.1);
}
```

这样，经过 calc_quantized_dist，物体的相对距离从 0.95 米到 1.05 米，计算之后的距离都是 1 米，也就是说最多有 10% 的误差。角度的截断误差计算就比较简单，只是把浮点数角度的取整成整数，所以角度的最大误差就是 0.5 度。

从看到的物体的距离和角度反推出实际的距离和角度只能确定知道在某个范围内，而不能具体到某个值。可以构造截断取整的反函数，通过看到的距离计算出一个距离区间，如下：

```
double inv_quantize_min(double d, double qstep)
{
    return ( rint( d / qstep ) - 0.5 ) * qstep;
}
double inv_quantize_max(double d, double qstep)
{
    return ( rint( d / qstep ) + 0.5 ) * qstep;
}
Range inv_quantize(double d)
```

```

{
    return Range(
        exp( inv_quantize_min( log( inv_quantize_min(d, 0.1) ), 0.01)),
        exp( inv_quantize_max( log( inv_quantize_max(d, 0.1) ), 0.01))
    );
}

```

角度的反推简单很多，只要加上正负 0.5 度就可以了。还是前面的例子，看到的最近的地标距离 21.5 米，角度 22 度，实际上，有可能均匀分步在距离 21.43 到 21.65 米，角度 21.5 到 22.5 度之间的区域里面。这样，通过这个地标来定位，最大误差可能有 15 厘米。有个方法可以减少这个误差，就是不仅仅利用一个地标来定位，而是利用所有的地标，把所有地标反推出的 client 位置的可能区域进行叠加，取交集，再求交集区域的中心点，就得到 client 的位置，这样可以有效的降低观察误差对定位精度的影响，下一小节将详细介绍。

有的数据不是直接通过观察得到的，而是有直接观察到的数据计算得来，同样误差也会叠加。比如身体的绝对角度是通过视角的绝对角度和感觉到的脖子角度相减得到的，那么身体绝对角度的误差就是其他两个角度的误差的和，也就是正负 1 度。

7.2.2 蒙特卡洛定位（Mento Carlo Localization）算法

蒙特卡洛定位算法是美国华盛顿大学的 Dieter Fox 最先提出用于机器人定位的算法。用在环境地图已知但是传感器输入有限的环境中，基本思想是用很多样本来分别猜测机器人的状态（位置，方向等），通过机器人的移动，去掉不合理的样本，利用合理的样本再产生新的样本，这样一直迭代下去，很快就可以准确定位。这个算法有个前提，就是观察的误差是符合高斯分布的，这样才能越来越准。读者可以去他的网站看他们的研究成果，地址是 http://www.cs.washington.edu/ai/Mobile_Robotics/。

在 RoboCup 其他组的比赛中，这个算法应用很广，比如 AIBO 机器狗的定位。仿真组的环境对于蒙特卡洛算法可以说是大才小用，2003 年的冠军 UvaTrilearn 用了这个算法简化算法，本书所附的测试用 client 源代码实现了一个较为复杂一些的版本。实验证明效果很好，基本达到了定位的极限。下面结合整个世界模型更新流程讲述这个算法。

首先是样本的数据结构，定义如下：

```

struct client_sample{
    Vector pos; // 位置
    Vector vel; // 速度
    Angle body_angle; // 身体的绝对朝向
    Angle head_angle; // 脖子与身体朝向的相对角度
};

```

算法的步骤如图 7.3，具体如下：

第 1 步，初始化样本集。一般情况可以通过看到的边线、地标和感觉到的脖子角度得到，如果想精度更高可以使用更多的地标。比如前面的例子，距离的范围（dist_range）是 21.43 到 21.65 米，角度的范围（dir_range）是 21.5 到 22.5 度，视角的绝对角度的范围（neck_global_range）是 39.5 到 40.5 度，脖子角度范围（neck_relative_range）是 19.5 到 21.5 度，如果是在运动中，速度的大小和方向和有一个范围。在这个区域里面产生 N 个均匀分布的样本：

```

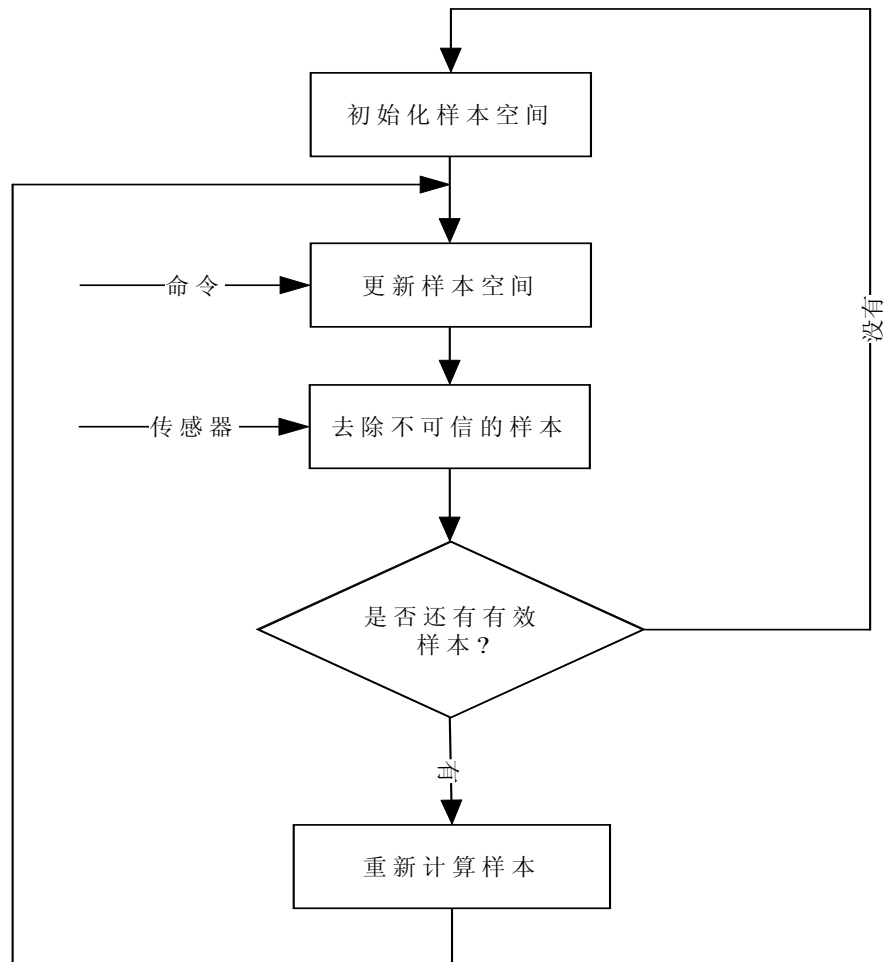
for ( i=0; i<SAMPLE_SIZE; ++i )
{
    sample[i].pos = flag.pos() - Polar2Vector( dist_range.rand(),

```

```

dir_range.rand());
sample[i].neck_angle = neck_relative_range.rand();
sample[i].body_angle = neck_global_range.rand() - sample[i].neck_angle;
sample[i].vel = Polar2Vector(speed_range.rand(), sample[i].body_angle +
sample[i].neck_angle + speed_angle_range.rand());
}

```



• 图 7.3 蒙特卡洛定位算法

第 2 步，模拟更新样本空间。对每一个样本，都采用和 server 一样的算法，执行命令，更新速度、位置和角度，也同样加入运动误差。对于单个队员的来说只有三个命令对样本里面的数据有影响，分别是 dash、turn 和 turn_neck，比如 dash 命令的模拟更新是这样的：

```

accel_mod = power * effort * dash_power_rate;
for ( i=0; i<SAMPLE_SIZE; ++i )
{
    sample[i].vel += Polar2vector(accel_mod, sample[i].body_angle);
}

```

当前，为了清晰起见，在上面的代码里一些小的细节没有给出，比如要判断队员的体力，不同类型的队员 dash_power_rate 也不一样，倒退的时候需要双倍的体力等，读者可以参考所附源代码。命令更新之后是位置更新和速度衰减：

```

for ( i=0; i<SAMPLE_SIZE ; ++i )
{
    noise_range = player_rand * sample[i].vel.mod();
    sample[i].vel += noise(noise_range);
    sample[i].pos += sample[i].vel;
    sample[i].vel *= player_decay;
}

```

第3步，处理新信息，去掉不合理的样本。新的信息的在误差范围上就有了新的限制。通过新看到的地标、边线还有感知到的脖子角度可以把误差范围之外的样本去掉，比如通过看到的地标就可以去除一些不合理的样本：

```

Range dist_range = (getMinMaxDistQuantizeValue(flag.dist(),
getQuantizeStepL(), 0.1 )); // 通过看到的地标距离反推出实际距离的范围
Range dir_range = (getMinMaxDirQuantizeValue( flag.dir() )); // 通过看到
的地标的相对角度反推出实际角度的范围
sample_size = 0; // 新的样本个数
for (i=0; i<SAMPLE_SIZE; ++i )
{
    Vector player_to_flag = flag.pos() - sample[i].pos; // 地标到当前样本的向
量
    double dist = player_to_flag.mod(); // 地标到样本的距离
    if ( !dist_range.inRange(dist) )
    {
        continue; // 此样本超出距离范围，去掉此样本
    }
    double dir = normalizeDegreeAngle(player_to_flag.dir().degree() -
sample[i].body_angle - sample[i].neck_angle); // 地标在样本局部坐标中的角
度
    if ( !dir_range.inRange(dir) )
    {
        continue; // 此样本超出角度范围，去掉此样本
    }
    sample[++sample_size] = sample[i]; // 保存有效样本
}

```

第4步，补充样本。第3步之后，一些样本因为不合理被去掉，若此时样本集里面没有足够的样本了，则回到第1步；否则通过剩余的样本组合出新的样本到样本集里面。组合的方法如下：

```

for (i=sample_size; i<SAMPLE_SIZE; ++i )
{
    random_template = rand () * sample_size; // 随即选取一个有效样本，复制为新
的样本
    sample[i].pos = sample[random_template].pos;
    sample[i].vel = _sample[random_template].vel;
    sample[i].body_angle = sample[random_template].body_angle;
    sample[i].neck_angle = sample[random_template].neck_angle;
}
sample_size = SAMPLE_SIZE;

```

从上面的步骤可以看出，这个算法非常简单，但是计算量和样本空间的大小成正比，所

以虽然理论上样本越多越好，但样本的数量要根据 CPU 的运算能力来确定。在作者使用的机器（Intel Pentium 4, 2.0GHz）上，500 个样本是比较理想的值。UvaTrilearn 的算法里面没有引入身体角度和脖子角度两项，对样本的需求小一些，但至少也要 100 个样本才行。

7.2.3 蒙特卡洛定位的结果分析

我们可以用下面的例子来看蒙特卡洛算法的优点。队员在位置（-10, 0），每个周期发送（turn 20）命令来改变视角方向。分别使用普通定位算法和蒙特卡洛算法，并比较结果。为此作者编写了一个 Python 模块 RoboCupSim.so，只要在模块所在的目录里面输入下面的命令就可以了。

```
peter@gentoo ~ $ python
Python 2.3.4[GCC 3.3.4 20040623 (Gentoo Linux 3.3.4-r1, ssp-3.3.2-2,
pie-8.7.6)]
Type "help", "copyright", "credits" or "license" for more information.
>>> import RoboCupSim
>>> client = RoboCupSim.Client()
>>> client.connect("name", "localhost", 6000)
>>> client.send(" (move -10 0)" )
>>> client.send(" (turn 20" )
>>> client.output()
```

得到的结果如下表：

表 7.1 两种定位算法的比较

身体角度	普通算法	蒙特卡洛算法
0	(-10,0)	(-9.99978,0.000589024)
20	(-10,0)	(-10.0026,0.00115291)
40	(-10,0)	(-10.004,0.000945896)
60	(-9.78513,-0.12479)	(-10.0002,0.000543275)
80	(-9.78513,-0.12479)	(-9.98153,0.000850441)
100	(-9.80476,0.420467)	(-9.99646,0.00161069)
120	(-10.1532,-0.0336964)	(-9.99104,0.00181503)
140	(-10,-3.18398e-15)	(-9.99456,0.00141412)
160	(-10,-3.18398e-15)	(-10.0054,0.000693275)
180	(-10,-3.18398e-15)	(-10.0087,0.00032752)

从上面的结果可以看出，蒙特卡洛算法的结果非常稳定，最大误差只有两个厘米；而普通算法则跟看到最近的地标的远近相关，最大误差达到了 20 多个厘米。在运动中，蒙特卡洛算法的优点更加明显，作者曾经做过实验，让队员在场上以 1 米/秒的速度随机跑动，把蒙特卡洛算法得出的位置和 server 记录下的队员实际位置比较，得到平均误差为 5 厘米，而队员的平均运动误差也是 5 厘米，说明此定位算法已经趋进定位精度的极限了。

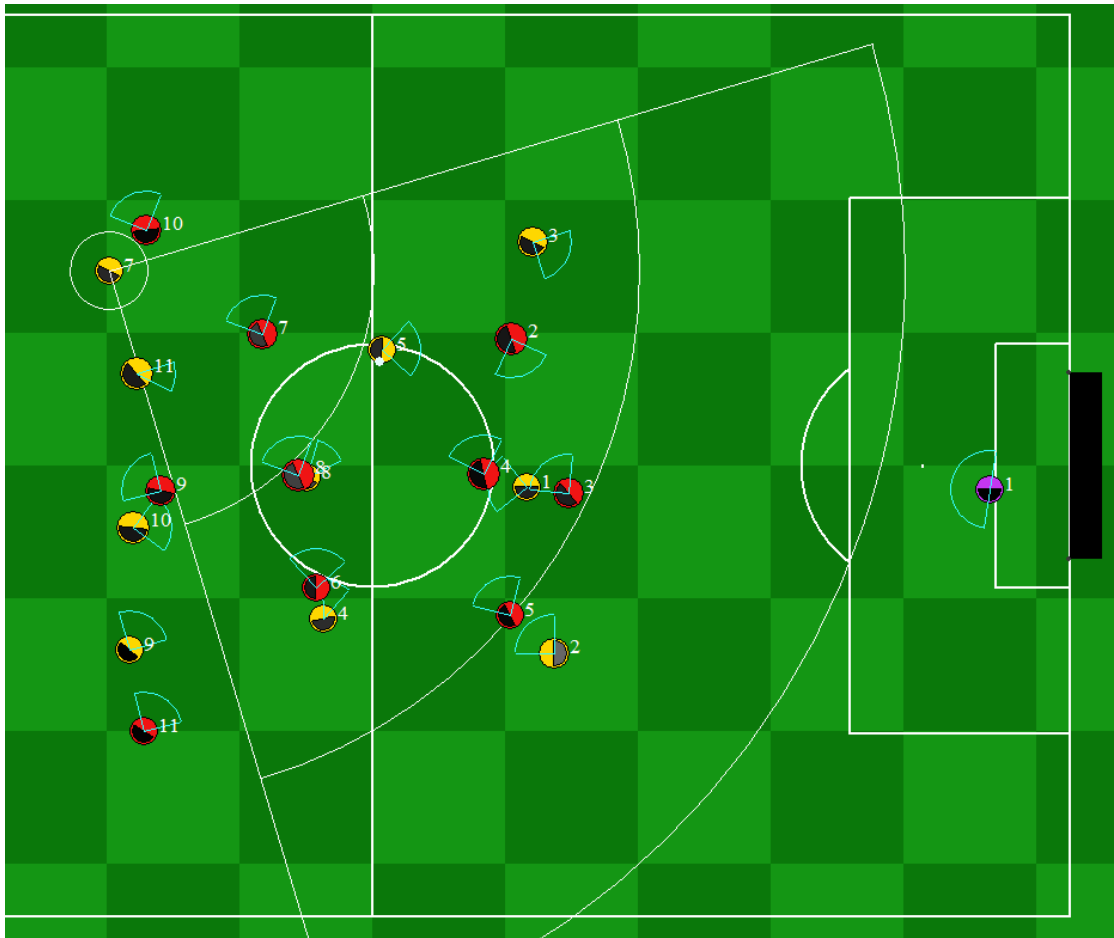
7.3 信息不确定时的猜测

仿真组 `server` 的设计原则就是要一定程度的加入噪声和不确定信息，增加决策程序的难度和比赛的不确定性。其中除了误差之外最明显的就是队员的边和号码，距离越远，信息量就越少，经常能够看到不知道是哪一边或者不知道是哪个号码的队员的情况。这个时候就需要猜测来提高信息的准确性。

7.3.1 不确定信息的产生

我们知道，视觉模型除了与时钟周期不同步这一个特点以外，还有另外一个导致信息不确定的特点，就是看到物体的信息质量会受到相对距离的影响，距离远了，就会看不到队员的号码，再远甚至看不清是哪一边的队员。在 `server` 的源代码中有三个与此有关的参数，分别是 `UNUM_FAR_LENGTH`，`UNUM_TOOFAR_LENGTH`，`TEAM_FAR_LENGTH` 和 `TEAM_TOOFAR_LENGTH`。这几个参数是硬编码在程序里面的，而不是像其他参数一样可以通过修改配置文件改变，这可能是由历史原因造成的。对于队员看到的场上的其他队员，从参数的名字很容易知道各自的意思：当距离小于 `UNUM_FAR_LENGTH` 时，可以看到队员的号码，距离从 `UNUM_FAR_LENGTH` 到 `UNUM_TOOFAR_LENGTH` 看到号码的概率越来越小，距离小于 `TEAM_FAR_LENGTH` 可以看到队员属于哪一边，而超过这一距离直到 `TEAM_TOOFAR_LENGTH`，看到边的概率也就越来越小，再超过就完全看不清是哪一边的了。其实，这几个参数并不是只应用在地上队员的号码和边上面，对所有物体的视觉信息的质量一样起作用，但这不是本节的重点。

队员的视野如图 7.4 所示，浅色一边的 7 号队员当前的视角是 90 度的普通视角，视野中有三条线，分别是距离队员 `UNUM_FAR_LENGTH`、`UNUM_TOOFAR_LENGTH`、`TEAM_TOOFAR_LENGTH` 的范围。实际上应该有四条线，但是因为 `UNUM_TOOFAR_LENGTH` 和 `TEAM_FAR_LENGTH` 的值相等，两条线重合，就只要三条线就可以了。这样，深色的 7 号队员在 `UNUM_FAR_LENGTH` 范围内，一定可以看到边和号码；深色的 2 号队员在 `UNUM_FAR_LENGTH` 和 `UNUM_TOOFAR_LENGTH` 之间，有可能会看到号码，看到的概率与距离有关，距离越远越容易看不到；浅色的 2 号队员在 `TEAM_FAR_LENGTH` 和 `TEAM_TOOFAR_LENGTH` 之间，可能会看不到是哪一边；深色的 1 号队员，距离在 `TEAM_TOOFAR_LENGTH` 之外，一定看不到边和号码。



• 图 7.4 队员的视野

这样，场上的队员在比赛过程中，看到的其他队员有时候是不确定的，经常看到没有号码或者没有边的未知队员，这样的信息不经过处理，就无法应用在决策中。这个处理算法的目的就是维持一个尽量准确的场上队员未知信息表，信息越准确，则对于比赛形势的把握越准确，传球等动作成功率就越高。最有用的例子就是身体周围三米以内的队员，如果没有在视角之内，就会感觉到一个未知队员存在，但是不知道号码，也不知道是哪一边的，此时如果不能准确猜测那个队员的信息，甚至把对方误认为己方，就会无缘无故的被对方偷袭。

7.3.2 简单的猜测方法

在 CMUnited 提供的源代码中，作者根据前 N 个周期的信息猜测当前的未知队员。比如当前周期看到一个未知队员，从前 N 个周期的记录里面查找符合条件（比如当前周期没有看到此队员，和未知队员属于同一边而且距离最近）的已知队员，通过测算距离和队员能力，看是否可以移动到当前位置。如果条件符合，则把此未知队员的信息赋给选定的已知队员。

对于未知队员的识别，各个队有不同的方法，但也大同小异。在 UvaTrilearn 提供的代码中，实现未知队员识别的在 `WorldModel::mapUnknownPlayers` 函数中，读者可以参考，下面是做了一些修改并加入了注释的版本：

```
double      dDist, dMinDist; // 记录位置的临时值
posAgent = getAgentGlobalPosition(); // client 自己的位置
ObjectT     o, o_new, objTmp; // 记录类型的临时值
```

```

int         index;
for( int j = 0; j < iNrUnknownPlayers; j ++ )
{ // 对所有的未知队员依次遍历
    pos = 获取位置队员 i 的位置;
    dMinDist = 1000.0; // 用来表示距离次未知队员最近的可确定队员的距离
    o = 获取位置队员的类型;
    o_new = OBJECT_ILLEGAL;
    if( ! SoccerTypes::isOpponent( o ) )
    { // 看到的边是己方或者未知
        for( int i = 0 ; i < MAX_TEAMMATES ; i++ )
        { // 依次遍历己方队员，找一个距离此未知队员最近的，并记录最近距离
            objTmp = Teammates[i].getType();
            if( 此队员信息有效且本周期没有看到 )
            { // 比较此队员是否更近
                dDist = 此队员和未知队员的距离;
                if( dDist < dMinDist )
                { // 保存最近的距离和队员类型
                    o_new    = objTmp;
                    dMinDist = dDist;
                }
            }
        }
    }
}

if( ! SoccerTypes::isTeammate( o ) )
{ // 看到的边是对方或者未知
    for( int i = 0 ; i < MAX_OPPONENTS ; i++ )
    { // 依次遍历对方队员，找一个距离此未知队员最近的，并记录最近距离
        objTmp = Opponents[i].getType();
        if(此队员信息有效且本周期没有看到)
        { // 比较此队员是否更近
            dDist =此队员和未知队员的距离;
            if( dDist < dMinDist )
            { // 保存最近的距离和队员类型
                o_new    = objTmp;
                dMinDist = dDist;
            }
        }
    }
}

if( 找到的队员是确定的且能够在这段时间内到达未知队员的位置 )
{
    将未知队员的信息附加在找到的队员身上;
}
else if( 可以确定未知队员是己方的 )
{
    将此队员放入己方未知队员集合中;
}
else if( 可以确定未知队员是对方的 )
{

```



```

        将此队员放入对方未知队员集合中;
    }
    else if( 这个队员不知道属于哪一边, 但是距离很近 )
    {
        // 此时要猜测, 尽量的认为是对方队员, 防止被偷袭
        将此队员放入对方未知队员集合中
    }
} // end for
iNrUnknownPlayers = 0;

```

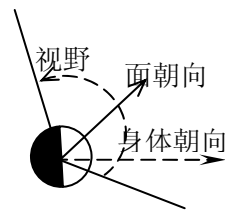
上面的方法其实非常简单, 对于大部分情况还是比较有效的。但是, 完全识别未知队员有很大难度, 尤其是猜测目标是对方的队员, 或者是两个队员距离很近的时候, 也很难分清, 有时候要根据其意图。

7.4 主动信息采集

比赛中球员获取信息有多种方式, 其中视觉信息是最直接可靠、也是最多使用的一种方式。前面章节已经提到, 仿真比赛中, 每个球员都有自己确定的“视野”, 比赛平台每隔一段特定的“视觉延迟”时间, 就会发送给球员他所能“看”到的信息。如果不做专门处理, 这种信息总是“被动”地接收, 即使信息更新做得再好, 也只能保证这些“看”到的信息符合真实情况, 但却不一定是决策需要的信息。由于比赛中信息感知和行为执行是异步的, 这就使得球员在执行决策时无法立刻获得自己需要的信息。为了解决这种被动获取与决策所需的信息不协调的问题, 就必需引入主动获取方式。

7.4.1 信息采集分析

在 RoboCup 仿真环境里, 球员有身体朝向和面朝向两个概念。球员观察的范围由面朝向决定, 视野是一个以面朝向为中心, 正负各 1/2 视野宽度的一个扇形, 如图 7.5 所示。并且球员距离观察目标越远, 看得越不清楚。视野宽度可以为 45 度、90 度和 180 度。较窄的视野宽度所获得信息的频率较快。目前 RoboCup 仿真环境中规定 45 度视野宽度每 0.75 周期获得一次视觉信息, 90 度视野宽度每 1.5 周期获得一次视觉信息, 180 度视野宽度每 3 个周期获得一次视觉信息。另外还有信息质量的影响。上面给的数据都是获取信息为高质量的情况, 如果获取信息为低质量时, 周期进一步缩短一半, 但这种模式只能知道目标的方向, 无法确定目标的位置, 决策时大多没办法使用, 所以一般不考虑。



• 图 7.5 球员视野示意图

在实际比赛中, 球员可以通过 `change_view` 指令改变视野宽度, `turn` 指令改变身体朝向以及 `turn_neck` 指令改变面朝向和身体朝向之间的夹角。身体朝向决定了每个队员加速（dash）的方向, 即每个队员每周期只能沿身体朝向加速, 同时平台模拟了惯性原理, 对转身动作的最大幅度做了限制, 在静止情况下, 球员可以一次转 ± 180 度角, 但随着球员即时速度增大, 转身的最大幅度越来越小, 当球员达到全速时（10 米/秒）, 一次最多只能转 ± 30 度角（详见 4.3.4 节中的转身模型）。另外由于 `turn` 指令与 `dash`、`kick` 等指令互斥, 不能在同一周期发送, 这就大大限制了其作为主动信息采集指令的使用, 否则会与其他决策模块产生冲突。`turn_neck` 指令的限制相对较小, 没有惯性的限制, 也不会与其他指令冲突, 不过球员的面朝向和身体朝向最多只能成 90 角, 即 `turn_neck` 指令相对身体最多只能转 ± 90

度角。由于在实际指令如 `turn`、`dash` 等执行后，球员身体朝向都会发生一些变化，而平台会根据所有行为更新完之后，球员的视角来决定要发送的信息，所以一般动作对观察能力的限制就集中体现在它们对做完动作后身体朝向的限制。主动信息采集模块很自然放在所有决策模块之后执行，并且要对产生的决策效果做出准确的预测，然后根据需求计算出最终实际要转动脖子的角度。考虑到脖子转动的限制，可以得到在不转身情况下球员理论上可以观察到的最大角度为：

观察到的最大角度 = 视野宽度/2+90 度（脸和身体朝向的最大夹角）

根据上面的结论，一种简单的处理就是在每周期决策后预测后续周期需要的信息，然后调整脖子角度，使下个视觉信息来时的视野能覆盖观察目标。比如，当球员准备给队友传球时，为确保成功率，可以事先看一下传球方向是否安全。

当然有时要观察的目标角度不满足上面的公式，这就要考虑使用 `turn` 指令，为了不和其他决策模块冲突，可以设置一个标志，只有当其他决策模块优先级低于采集模块时，采集模块才能使用 `turn` 指令。这通常发生在决策所需的重要信息（如球）丢失时，要优先考虑找到球，这时考虑 `turn` 指令所能转动的实际角度值，可视角度可以进一步扩大。

7.4.2 信息采集算法

1. 观察模式

根据平台规则，每个队员在获取视觉信息时都是用一种视野宽度的模式观察，不同视野宽度带来的观察效果也不同。由前面介绍也可看出，越窄的模式观察延迟越小，但观察范围小，宽模式观察的延迟大，但观察范围大。在不同情况下对得到信息的频度和广度的要求一般是不同，甚至于在一次信息采集仲裁中，不同信息请求对这种频度和广度的要求也不尽相同。这种不同会对仲裁算法产生很大影响，如果再加上视觉质量，就会产生 6 种组合，在评估时就必需考虑这 6 种不同观察模式所付出的额外代价，这样就会使得仲裁算法更加复杂。

一般为了简化运算，在做采集策略之前先确定本次采集使用的观察模式，然后在这个给定的视野宽度下产生最终决策。观察模式的给定首先看如果有某决策模块需要某种特定的观察模式，就优先满足。否则，由主动信息采集模块根据一些原则自行决定当前应采用的观察模式。这些原则主要是具体分析场上形势，根据对信息需求的渴望程度，调整频率，并使视野宽度尽可能大。一种典型做法是根据球员距离球的远近来调整频率，因为比赛总是围绕球在进行，当球离得比较远时，球员的决策就比较简单，在球逼近之前有足够的时间做出反应；反之，一旦球离得很近，球员对产生形势变化的敏感度就大大提升，需要大量相关信息使之能够快速做出正确的决策。

根据比赛规则，由于球员每周期只能执行一个动作，所以过于频繁的获取信息也没有意义，比如用窄视角模式，每 0.75 周期就可以获得一次视觉信息，这样就会在有的周期获得两次视觉信息，造成浪费。所以在确定获取信息的频率后，还要合理分析计算，选择合适的视野宽度最有效的获取信息。目前比较常用的一种方案是普通视角—窄视角—窄视角，这样获取视觉信息的周期依次为 0—1.5 周期—2.25 周期—3 周期...，这样以 3 周期为一次循环单位不断循环，从而保证每周期都可以获得视觉信息，同时又可获得最大视野范围。

2. 信息请求的产生

一般来说在一个周期内会有多个目标信息的需求，比如防守的时候既要一边注意着球的动向，又要一边盯着对手。这样就无法像上节所讲的那样直接确定最终目标角度。这时就需要给出一种仲裁算法，可以处理多个信息请求。在实现这种仲裁算法之前，首先要考虑这些信息请求具体如何产生，并且已何种方式表示。

考虑一般一个信息请求就是一个具体的目标,要获取该目标信息就是使在下次获得视觉信息的时候,该目标被当时的视野范围覆盖。所以每个信息请求可以简单表示为该目标相对球员的角度,为了方便后面计算,还可以保存能观察到该目标的最大视角取值范围。视角取值范围的确定一方面考虑球员当时的视野范围,另外还要考虑实际目标可能的位置分布,因为一般要观察的目标信息都是不确定的,无论是球还是球员都有可能发生移动,所以实际可看到的范围需要通过一个 **buffer** 变量修正, **buffer** 变量的确定主要是根据对目标行为的一些预测,最大不超过其可行动范围,如一位对手两周周期没看,则他的可能位置必定在已知位置为中心,半径为球员最大速度 $\times 2$ 的圆内。具体信息请求的数据结构可表示为:

```
typedef struct{
    AngleDeg angle; //中间的角度,即目标相对于球员角度,再根据当前视野宽度
                    //以及其他要考虑的因素计算出观察的取值范围
    AngleDeg leftbuf; //能观察到该目标的视角最大取值范围
    AngleDeg rightbuf;
    float score;      //优先级评价
}VR_ACTION;
```

在事先已确定采用的观察模式下,存储一个视觉请求的过程如下:

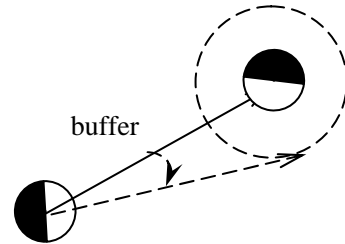
```
VR_ACTION NewVisualRequest(AngleDeg ang, float score)
{ //根据所给的视觉目标角度(相对球员自己)
  //以及该目标的优先级(可以按照某种算法自动产生)
  //产生新的信息请求结点,并返回
  vr.angle = ang;
  //由给定的当前视野范围算出可以观察到的最大视角范围
  vr.leftbuf = ang - (viewwidth - buffer);
  vr.rightbuf = ang + (viewwidth - buffer);
  vr.score = score;

  return vr;
}
```

要产生每周信息请求,一种方法是信息采集系统自行产生,简单的办法就是对每个关心的目标比如球,都用一个置信度表示其不确定程度,然后将这些目标按照置信度排序。如果置信度低于某个值,就提出信息采集请求,并根据其置信度按照一种方法生成评价 **score** 值。这种方法实际是在保证所有关心目标的平均置信度尽量高,可以满足一般的需要。但由于场上形势千变万化,不同球员关心的目标不可能一样,同一个球员在不同形势下关心的目标也不尽相同,而且对不同目标关心的程度也不是一样的,这样完全由信息采集系统自行决定就比较困难。

另外一种方法就是完全由各决策模块根据各自的决策分析提需求,信息采集系统只是直接将这些请求存储下来,供后续算法分析。这样能比较好的适应场上多变的形势。但由于各决策模块出发点不同,提需求的标准也就很难统一,特别是要给出一致的优先级评价就变得很困难。

比较好的折衷办法就是信息采集系统给出一种表示目标关心程度的方法,由各决策模块每周提交不同的关心目标,系统维护一个各目标关心程度的列表。然后类似于前一种方法,每周自动产生信息请求。这样即便于维护,又能比较好的适应比赛。另外考虑到实际有很多信息请求在已确定的观察模式下是无法满足的,一般就不再考虑了。这样可以减少仲裁模



• 图 7.6 目标范围示意图

块过多不必要的检查，提高了效率。

3. 仲裁算法实现

当系统在已确定的视野宽度下将所有信息请求保存在一个信息请求队列之后，下面要做的工作就是设计一个仲裁算法，从这些信息请求中选出“最佳”的视角调整方案。一个简单的处理方法是按照某种方法给这些信息请求排序，总是优先满足排在前面的信息请求。具体做法是，根据当前的视野宽度，可确定观察每个目标的最大取值范围，首先从队列头取出第一个请求，确定基准范围，然后依次从队列中取信息请求，将该请求对应角度的取值范围与基准范围求交集并更新基准范围，如果交集为空，则忽略此请求，继续往下取，直到队列取空。最终此视角范围的角平分线即为观察目标角度。使用上节给出的信息请求的数据结构，实现伪代码如下：

```
VR_ACTION DealVisualRequest(VR_QUEUE VR_Queue)
{ //根据所给的信息请求队列（已按优先级排好序），计算出最优观察角度，并返回
  If (!VR_Queue.isEmpty())
  {
    pvr = VR_Queue.getHead();
    vr = pvr->data; pvr = pvr->next;
    while (pvr)
    {
      //求两个信息请求范围的交集，如果交集不为空，则更新 vr
      getIntersection(vr, pvr->data);
      pvr = pvr->next;
    }
    return vr; //已检查完所有请求，根据 vr 即可算出最终
  }

  return NULL; //没有信息请求，返回空
}
```

上面给出了一种主动信息采集的简易算法，其中优先级的评价主要根据场上的形势及该目标的价值。在 2000 年葡萄牙队（FCPortugal）设计的一套球员感知策略 SLM（Strategic Looking Mechanism）[Reis and Lau 2000]中，这个评价是通过计算一次观察可以带来全场物体置信度的增加总和得到，同时考虑到决策的不同需要，根据场上形势，球和自己的位置等信息，在求和时会对不同的物体使用不同的权重。这种观察机制综合考虑球员当时的角色、阵型、位置以及场上所有物体的位置速度信息的置信度来决定每周期球员所要观察的方向。

2001 年清华队（TsinghuaAeolus）在其设计框架中有专门的一个视觉仲裁模块（Visual Mediator）来处理上述主动信息采集问题[Yao et al. 2002]。在这个视觉仲裁模块里定义了视觉动作概念：一个视觉动作指的是通过一个一般动作（turn、dash、kick、catch）和一个转脖子动作（turn_neck）的组合，将面朝向转向一个特定方向。一个视觉动作也可以用做完组合的动作后他所能观察到的视野范围表示。对于一个视觉目标来说，有许多视觉动作可以保证能看见它（只要视觉目标在视野范围，就可以看见）。当考虑多个目标的时候，问题就转化为如何选取一个最佳视觉动作能充分利用视野宽度，覆盖尽可能多的目标。由于不同视觉动作可能获得的效果相同，即他们包括的视觉目标可能是相同的。据此可以对这些视觉动作划分等价类：所有覆盖的视觉目标的相同的视觉动作互相等价。这样每一个视觉请求就是以视觉目标为左边界或右边界的视觉动作所在的等价类。最终只需在这些等价类里面找最优解即可。这里所谓的优先级评价就可以理解为如果能观测到的目标越多，则观察利益越大。

在具体实现中，这种评价策略没有什么标准，完全根据具体情况自行定制，但最终算出这种优先级关系应该尽量符合实际情况。

4. 算法小结

综上所述，下面给出一种主动信息采集的简要流程。

- 1) 各决策模块在决策过程中提需求
- 2) 系统根据场上形势，综合各模块对转身、观察模式等的特定要求，确定要采用的观察模式（通过发送 `change_view` 指令及时生效）
- 3) 系统根据各决策模块提的需求，根据当前采用的观察模式，产生要处理的信息请求
- 4) 根据生成的信息请求列表，执行仲裁模块，综合评价产生“最优”观察目标
- 5) 最后根据这个最优观察目标计算实际需要调整的脖子或身体角度，发送 `turn_neck` 或 `turn` 指令，完成一次主动信息采集处理

7.5 实现一个简单的 client 程序

这一节，作者要介绍一种新的实现 client 程序的思路，也就是尽量利用现有的资源，使用 server 提供的函数库，最快的达到可以进行比赛的目的。

7.5.1 server 提供的函数库

RoboCup 仿真比赛平台提供了 `rcssbase` 库，可以实现网络连接、参数读取、序列化等一些通用的功能，是仿真比赛 server 的底层库。除了其官方的主页 (<http://sserver.sf.net/>) 之外，还提供了几个有用的函数库 (<http://rccparser.sf.net/>)：

- ✧ `Rccparser` 是一个对 server 信息进行分析的库，可以把文本信息转化为数据结构。
- ✧ `Rcclient` 是一个简单的 client 实现类，封装了 `Rccparser` 和 `rcssbase` 里面的接口，通过它可以很方便的实现一个 client。
- ✧ `Rccserializer` 是一个序列化库，可以把命令转化为文本发送给 server。

源代码的获得方法有两种，一种是下载压缩包，另外一种是从 CVS 仓库里面获得，两种方法对应的网站上都有介绍，安装方法一般都写在 `README` 或者 `INSTALL` 文件里面。这些库都是面向对象的，使用 C++ 语言，读者可以自行察看其源代码了解继承结构。这些库的最新代码都在 CVS 里面，作者下面提到的例子也都是针对最新的代码。

下面是 `Rcclient` 里面提供的例子，实现一个最简单的 client（为了节省篇幅，作者省略了一部分无关代码）：

```
#include "rccplayer.hpp"
#include <rccserializer/rccplayerserializer.hpp>
class TestClient : public rcc::Player
{
public:
    TestClient() {}
private:
    virtual void doBuildInit( int unum )
    {
        move( -10, -10 ); // 初始化时移动到场地内一点
    }
    virtual void doBuildCompressionOK( int level )
```

```

    {
        setCompression( level );
    }
    virtual void onStart()
    {
        init( "test", 8 ); // 队名为 test, 协议版本号为 8
    }
};
int main()
{
    rcss::net::Addr addr( 6000 );
    addr.setHost( "localhost" );
    TestClient player;
    rcc::UDPPlayer udpplayer( player, addr );
    udpplayer();
}

```

上面的代码实现的功能非常简单，就是建立网络连接，告诉server加入一个队名为test的新队员，采用的协议版本为8.0，把此队员移动到某位置，然后就不停的接收信息。`rcc::Player` 是Rcclient提供的队员实现类，它的父类是RccParser提供的`rcc::Parsr`类，例子中的TestClient类继承自`rcc::Player`，只要实现父类中已经定义好的虚接口`doBuildXXX`，就可以得到server信息中的数据。如TestClient的`doBuildInit`函数，就是当server发送“(init side unum playmode)”信息的时候被调用。`rcss::net::Addr` 是 `rcssbase` 提供的网络地址描述类，可以提供server地址和端口号的信息。而`rcc::UDPPlayer`则是一个使用UDP作为输入输出的client实现，它其实是一个函数子（functor），它运行的时候从网络收发信息，再把信息发送给真正的client实现。

7.5.2 实现 server 接口类

实现一个 client，最基本的就是和 server 进行通信，这就需要一个 server 接口类。这个类的作用就是提供一个直观的接口用于其它模块和 server 的通信，这个类提供以下接口：

- ✓ 发送队员的命令
- ✓ 获取 server 的各种参数（server 参数，player 参数和 player type 参数）
- ✓ 获得当前的 server 时间
- ✓ 获取队员的边和号码
- ✓ 获取当前的看到的物体的列表
- ✓ 获取队员的感知信息数据
- ✓ 信息响应处理对象的注册接口
- ✓ 以及其他用于调试的接口

这样实现的好处就是，和 server 的通信功能被限制在一个很小的范围之内，client 的其它模块只要知道这个类的接口和数据结构定义就可以方便的和 server 进行连接，避免了无关的代码耦合。

在作者提供的测试用代码中，这个接口类被称为 `Kernel`，实现这个接口的类是 `KernelImpl`。之所以称之为 `Kernel` 是因为这是一个主动类，它不停的从 server 接收信息，对信息进行分析并将其转化为数据结构之后，唤起相应的信息响应处理对象（sense handler），而且这是一个最简的 client 实现框架。`Sense handler` 其实是设计模式中的 `Visitor` 模式，`Kernel`

中可以注册若干这种对象句柄，等相应的信息到来之后，就调用这些对象，然后这些对象再通过访问 `Kernel` 提供的各种接口对信息进行进一步的处理。比如收到视觉信息之后，分析出看到了哪些物体，每个物体的相对位置信息，然后调用 `sightHandler` 里面的所有对象。

`KernelImpl` 的实现，读者可以参考原代码中的 `kernelImpl.h` 文件。

7.5.3 实现世界模型

在上面的 `Kernel` 的基础上可以建立一个简单的世界模型。世界模型是整个 `client` 的中心，它对其它高级决策模块提供了所有的客观信息，包括每个周期球和每个队员的位置和速度，当前的比分，每个队员的类型信息等。世界模型的算法主要是已知信息的定位和未知信息的预测，前面几节都已经提到。

其中队员自己进行定位的是 `AgentFilter` 类，对球定位的是 `BallFilter` 类，这两个类对于队员或者球维持了一个长期的状态，输入是每个周期看到的信息（如果没看到则要输入标志没有看到的信息）、感知到的信息以及每个周期的命令，输出是每个周期的位置和速度等信息。其他队员的定位用到的类是 `PlayerFilters`，这个类同前面两个类相似，但是输入不包含感知信息和命令，因为无法获知其它队员的感知信息和发送的命令。

原始的视觉信息经过 `GlobalWorld` 类进行处理，将看到的队员局部坐标系的信息转化为全局的球场坐标系中的数据。原始的感知信息则通过 `SelfProperty` 类进行处理。命令通过 `EstimateFromCommand` 类进行处理。这些类和上面提到 `AgentFilter`、`BallFilter` 以及 `PlayerFilters` 都有交互的数据输入。

队员和球的信息汇总成 `EstimatedSelfAndBall` 类，再加上其他队员的信息，一起组成 `EstimatedWorld` 类。这些类是世界模型对外的接口，其他模块可以根据需要调用，比如只要知道感知信息，就只调用 `SelfProperty` 类，若只要知道球和队员自己的信息，就调用 `EstimatedSelfAndBall` 类。

整个世界模型的结构如图 7.7。通用的数据结构定义在 `world_define.h` 头文件中。主要包含了以下结构：

- ✓ 静态物体结构，只有位置信息
- ✓ 动态物体结构，包括位置和速度
- ✓ 有身体朝向的结构，在动态物体基础上增加了身体朝向信息
- ✓ 队员结构，是带有边和号码的动态有身体朝向的物体
- ✓ 感知信息结构，包括体力、效用、脖子角度以及各种命令的计数值等

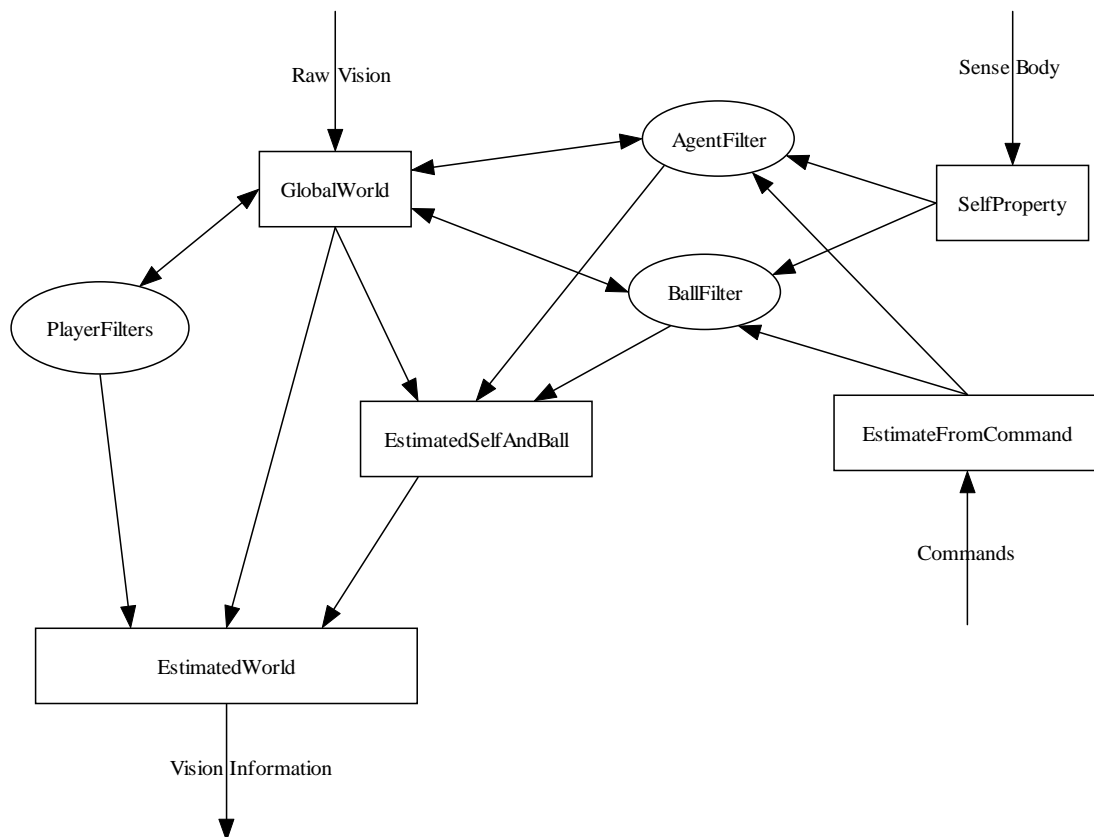


图 7.7 世界模型结构图

下面是一个简单的可以踢球的 client，用来演示世界模型的接口调用：

```

Kernel& kernel = Kernel::instance();
EstimatedSelfAndBall& self_and_ball = EstimatedSelfAndBall::instance();
//
SelfProperty& self_property = SelfProperty::instance();
while( true )
{
    Time current_time = kernel.currentTime();
    if( current_time.valid() && last_time < current_time )
    {
        // 新的周期
        last_time = current_time; // 保存时间标志
        const OrientedMobileObject& player =
self_and_ball.player(current_time);
        const WEWorld::MobileObject& ball =
self_and_ball.ball(current_time);
        if( player.pos().valid() && ball.pos().valid() )
        {
            // 队员和球的信息都是可信的
            const Angle player_dir = player.orientation();
            const Vector player_to_ball = ball.pos() - player.pos();
            const Angle ball_dir = player_to_ball.dir();
            const Angle dir_diff = ball_dir - player_dir;
            char buf[20];
            if( player_to_ball.mod() < 0.6 )

```



```

{
    // 球在可踢范围之内，把球踢向对方球门
    Vector their_goal(52.5,0);
    Vector player_to_goal = their_goal - player.pos();
    Angle angle_diff = player_to_goal.dir() -
    player.orientation();
    sprintf(buf,"(kick 50 %.2f)", angle_diff.degree());
    kernel.send(buf);
}
else if( dir_diff.degree() > 20 || dir_diff.degree() < -20 )
{
    // 转身向球
    kernel.send("(turn_neck 0)");
    sprintf(buf,"(turn %.2f)", dir_diff.degree());
    kernel.send(buf);
}
else
{
    // 走向球的位置
    kernel.send("(dash 100)");
    sprintf(buf,"(turn_neck %.2f)", (ball_dir -
    player.headOrientation()).degree());
    kernel.send(buf);
}
}
else
{
    // 信息不确定，转身搜索定位标志和球
    kernel.send("(turn_neck 0)");
    kernel.send("(turn 30)"); // scan field
}
}

```

7.5.4 其他工具类

实现一个 `client`，除了上面提到的几个部分之外，还要有一些辅助的工具类，比如参数读取、几何运算和时间比较等。

在 `server` 的源代码中，参数读取的接口类是 `rcss::conf::GenericBuilder`。实现参数读取的功能，需要定义一个这个类的子类。然后实现相应的虚函数，分别对应不能类型的参数，作者设计的 `GenericParam` 类如下：

```

class GenericParam : public rcss::conf::GenericBuilder{
public:
    virtual bool doBuildParam( const std::string& module_name, const
std::string& param_name, int value ); // 整数型参数
    virtual bool doBuildParam( const std::string& module_name, const
std::string& param_name, bool value ); // 布尔型参数
    virtual bool doBuildParam( const std::string& module_name, const
std::string& param_name, double value ); // 浮点数值型参数
    virtual bool doBuildParam( const std::string& module_name, const

```

```
std::string& param_name, const std::string& value ); // 字符串型参数
..... // 以下省略
};
```

真正读取参数的类是 `rcss::conf::Parser` 类，这个类的构造函数的参数是 `GenericBuilder` 类的指针，使用时调用 `parse` 函数，参数是文件名：

```
void GenericParam::readFromFile(const std::string& file_name)
{
    rcss::conf::Parser parser(*this);
    parser.parse(file_name, std::cerr); // 读取参数
}
```

`server` 提供的类只负责参数的读取，参数的保存还要子类自己实现，子类重写的虚函数就是用来实现这些功能的。

其它工具类，比如向量（`vector`），角度（`angle`），直线（`line`）等几乎所有的 `client` 源代码中都有实现，细节大同小异，请读者自行参考源代码。

7.5.5 使用动态脚本语言

动态脚本语言（比如 `Python`、`Perl`、`Ruby` 等）相对静态语言（比如 `C`、`C++`、`Pascal` 等）有类型灵活、修改方便、无须编译、移植性好等优点。`Boost C++` (<http://www.boost.org/>) 库里面有绑定 `C++` 对象到 `Python` 语言的工具，可以很方便的用 `C++` 语言编写 `Python` 模块。在作者的测试用 `client` 中，就写了一个 `Python` 的模块，用来实现一个简单的 `client`，基本的实现如下面的代码所示：

```
class PyClient
{
public:
    void connect(const std::string& team, const std::string& host, short
port)
    {
        .....
    }
    void send(const std::string& msg)
    {
        .....
    }
    void output()
    {
        .....
    }
};

#include <boost/python.hpp>
using namespace boost::python;
BOOST_PYTHON_MODULE(RoboCupSim)
{
    class_< PyClient >("Client")
        .def("connect", &PyClient::connect)
        .def("send", &PyClient::send)
        .def("output", &PyClient::output)
```

```
        i  
    }
```

上面的程序代码定义了一个 C++ 类 `PyClient`, 其中有三个类方法 `connect`, `send` 和 `output`。然后使用 `boost::python` 提供的方法定义了一个 Python 模块, 模块名 `RoboCumSim`, 这个模块里面有个类, 名为 `Client`, 其中也有三个方法, 分别同 `PyClient` 的三个方法对应。

将上面的代码编译成 Python 模块, 在 Python 脚本或者交互环境里面 `import` 这个模块就可以使用了, 使用方法参看前面本章蒙特卡洛定位一节最后的定位结果比较部分。

参考文献

[S. Thrun et al. 2001] Robust Monte Carlo Localization for Mobile Robots, Artificial Intelligence, 2001

[Chen et al. 2003] Mao Chen et. al. RoboCup Soccer Server. The RoboCup Federation, February 2003. Manual for Soccer Server Version 7.07 and later.

[Reis and Lau 2000] Luis Paulo Reis and Nuno Lau. FC Portugal Team Description: RoboCup 2000 Simulation League Champion. Univ. of Porto & Univ. of Aveiro. In Stone P., Balch T., and Kaetschmar G., editors, RoboCup 2000: Robot Soccer World Cup IV. pp.29-41, Springer Verlag, Berlin, 2001

[Yao et al. 2002] Jinyi Yao et. al. Architecture of TsinghuaAeolus. RoboCup-2002: Robot Soccer World Cup V, Andreas Birk, Silvia Coradeschi, Satoshi Tadokoro editors, LNAI, Springer Verlag, 2002

The RoboCup Client Parser, <http://rccparser.sourceforge.net/>

Boost C++ Libraries, <http://www.boost.org/>

Python Programming Language, <http://www.python.org/>

UvaTrilearn Team, <http://www.science.uva.nl/~jellekok/robocup/>

第8章 球员行为分析

由于 Robocup 仿真组模拟平台只提供实现一个球员的一些基本技能,如:kick,dash 等等。这往往不能满足主体设计人员建立一只能力很强球队的需求。主体还需要利用 Server 提供的基本技能,组合这些原子动作生成比较复杂的行为,比如踢球,带球,射门等等。这些都将在本章节中被引入。其中每个部分都有很多中实现方法。当然编者还不能说某种方法最好,因为很多往往还取决于实现细节,直到 Robocup2004 各队仍然采用各自不同的方法满足了各自的需要。这些就留给读者去深究吧。

8.1 快速踢球

回顾前面的章节读者知道,在仿真平台上物体移动的方式是这样的:每个周期 Object 有一个位置 pos,一个速度 vel.当下个周期到来是,仿真平台更新 Object 的位置速度:

$$vel = vel + acc$$

$$pos = pos + vel$$

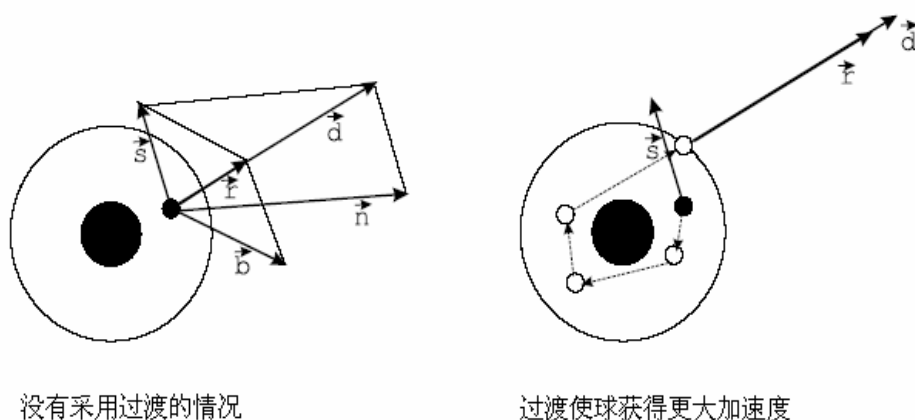
$$vel = vel * \alpha$$

其中 α 是速度的衰减因子,它随 Object 的类型不同而不同, acc 是执行 dash 或者 kick 后形成的加速度,默认情况下为 0。因此,为了使球按照我们所要求的方式进行运动,主体采用 kick 命令来改变球的加速度。然而模拟平台限制了球员踢球的能力,球员可获得的最大加速度为 2.7 米/秒²。尽管 Agent 已经可以把球从零速度在一个周期加速到最大速度:2.7 米/秒,但是只要我们比较一下实际需要和踢球模型里面的限制,它仍然不能满足我们对球控制的需要。比赛中,主体常常有要对进行高速变向的情况。在这种情况下,主体用几个周期连续踢球的模式获得比较大的加速度,以满足高层决策需求。

在 Robocup 仿真平台上,Server 设定球员的踢球加速效率为 0.027,而每个球员的最大踢球力量为 100,同时还受踢球范围,球到主体身体距离和球相对主体身体角度影响。但自 2002 年以来引入的异构球员,使球员踢球的方式上有些区别。不同球员类型踢球时加速度会随 Server 动态随机生成的两个参数变化: kickable_margin,kick_rand。详细内容,请读者回顾一下前面踢球模型和异构球员的相关章节。

在 Robocup 仿真平台中,主体的踢球范围为 kickable_margin,主体的身体半径为 player_size.由仿真平台设定,当两个 Object 重叠时产生碰撞,两个物体分别被反向弹开并以原速度大小的 10% 反向运动。由于运动物体在仿真平台中运动时,会有一定的随机误差。这些误差给碰撞判断造成一定的困难,影响 Object 的位置速度信息更新,更可能导致主体在踢球时发出错误的命令。主体应该尽量避免碰撞以保证对球控制的精确。

下面我们简单描述一下本节要解决的问题。如图 8-1(左) 所示, d 向量表示期望的目标速度向量, s 为当前的速度向量, b 为球在正确方向上能获得最大速度时的加速向量, n



• 图 8-1 采用过渡方法和未采用过渡倒脚的比较

为主体需要的加速向量 ($\mathbf{n} = \mathbf{d} - \mathbf{s}$)。通常高层决策模块要求主体能把球朝某个确定的方向踢去，因此保证出球方向的正确性是最重要的。同时由于踢球误差和模拟平台的动态随机误差，出球速度一般不能完全精确，只要能保证一定的精度，踢球模型就算是成功的。当设计模型不采用多次踢球的模型时，从图中可以看到，试验中的出球速度不到期望的出球速度的一半。

在这种情况下，只有采用多个踢球命令才能获得比较大的加速度。如图 8-1（右），多个踢球，每次踢球后，球仍然停留在可踢范围之内，尽管每次球速与期望速度不一致，但是最后一次踢球把球加速到正确的方向并踢出可踢区域。。这样经过 4 个中间的过渡步骤，主体使实际出球速度与期望的出球差距控制在一定范围内，保证了精确性。

注意上述例子中，我们所指球的位置都是相对于主体的，实际情况下主体也是运动的。所以如下我们列举，实际踢球规划时所需要四个具体参数：

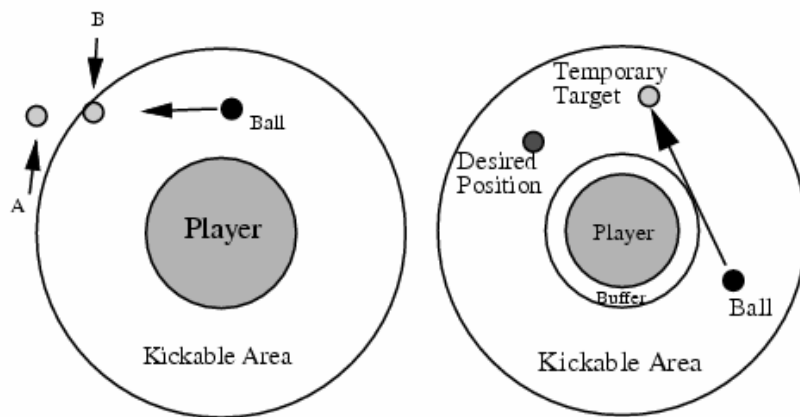
- 主体的速度向量，
- 球对主体的相对位置向量，
- 球速的向量，
- 期望出球速度向量。

当给定以上参数后，主体的踢球模块要能确定中间过渡步骤的具体个数和确切动作。当这些计算结束后，最后一踢就可以轻易地解决。然而为了使主体能获得比较好踢球，中间过渡步骤的选择在求解时是一个非常困难的问题。

8.1.1 最早 CMUnited 的踢球方式

在最早做出比较好的踢球方式，是 CMUnited-98。Peter Stone 和 Patrick Riley 最早完成的踢球方式是对踢球模型做分析，然后手工编码完成了当时的踢球动作。实际上在他们完

成的模型中对踢球采取了分模式编码。从前面我们已经明晰：1)主体是通过向足球提供加速度改变球的运行方式，2)为了连续踢球，必须在把球提到理想速度前，需要把球控制在以身体为中心半径为“kickable_area”的圆形里面。3)为了避免主体与足球的碰撞，主体身体半径里面是不能放球的。因此，球实际上只能在一个圆环里面（如图 8-2）。



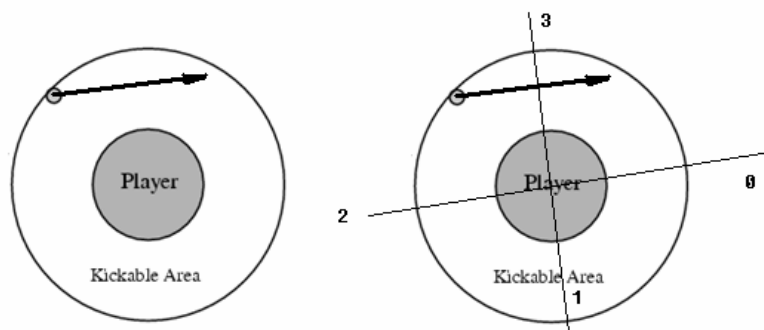
• 图 8-2

为了实现整个踢球模型，主体需要有能够把球从一侧调整到另外一侧的技能，称为 turn_ball。为了避免碰撞，实现增加了一个 buffer。计算 turn_ball 的目标是通过一条从球当前位置发出的射线来计算的。

另外一个比较重要的技能是要能够把球从任意角度踢出。首先要能确定踢球的目标速度，对于射门，或许最高速度是个合适的值。而对于射门，为了使接球队员能比较顺利的接到，可能稍微低一点的速度比较好。因此，主体需要考虑到球在运动中随时间的速度衰减。

有时，为了把球踢到较高速度，主体需要连续的踢球。Peter Stone 的做法是，把球踢到身后，然后连续两次或者更多次的加速。在图 8-2(左)里面，踢球 A 会使球在下个周期不能被踢到。如果出球速度到了我们需要的速度，那么这个动作就是合适的。而如果需要采用 B 的方式，我们可以把球停留在踢球范围内，以期在为了能加速到更大速度。为了区别这两个方式，显然对踢球动作后果的预测成为了关键。

Peter Stone 编码的踢球模块在调整球时，可以通过传递参数调整球。通常调整到的目标是与出球方向成 $\pi/2$ 的角度。如图将有 3 个角度可以用来调整：



• 图 8-3

8.1.2 根据踢球模型搜索，并利用 Hill-Climbing 优化

2000 年 FC Portugal 利用快速传递的打发，在仿真联赛横扫千军，夺得冠军。FC Portugal 是在 CMUnited-98 的开源代码上进行开发的，而 CMUnited 已有的踢球动作模块不能提供比较好的踢球结果。于是，Luis Paulo 和 Nuno Lau 在大量复用 CMUnited 的代码同时，利用搜索和爬山法改进了踢球模块。

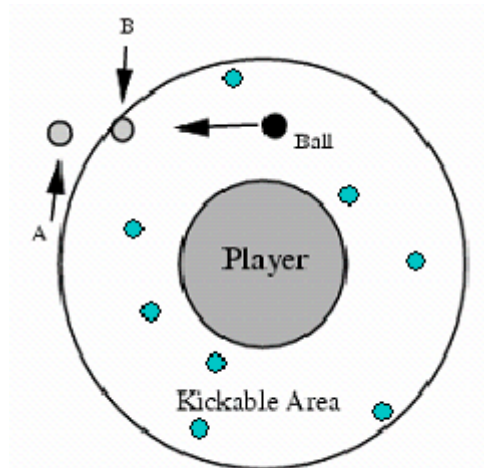
由于 FC Portugal 的研究并没有注重底层技术，而且在 Internet 上可以得到大量的底层源码，FC Portugal 的大部分技术都是由 CMUnited99 的源码改编而成。然而，其中关键的技术，是基于优化技术实现了更强大的踢球。球员动作的成功率很大的依赖于球员处理足球的能力。其中最重要的基本技能是大力踢球能力，就是说在大部分的形势下，球员能以最快的速度踢球。当足球在球员的可踢球圆周内时，球员就可以执行踢球指令，这种服务器的踢球模型允许球员在连续的周期内持续踢球的可能性。如果这种可能性被很好的研究，球员就能提供给足球比正常踢球更高的速度。然而，找出加速序列是相当困难的。

Luis Paulo 和 Nuno Lau 研究了通过解析法和学习技术。最后我们作为一个优化问题建模，然后使用了一些简单（但是非常有效）的搜索法则来寻找加速序列。在比赛中，通过在线优化，在每种形势和每一时间，球员决定如何才能大力踢球。基于服务器是动态的，因此 FC Portugal 还实现了一个预测器，根据足球的初始位置/速度和球员的初始位置/速度来预测经过几次踢球后，足球的位置/速度。使用这个预测器来评估结果，在线搜索可以分成两步：

- 1) 是通过一个随机的踢球序列得到“最好”的踢球序列；
- 2) 是对前一步得到的序列进行改善。

随机搜索使用下面的运算法则。在足球可踢范围内，执行随机的踢球动作。然后把最后一次踢球用解析法确定的踢球动作取代。随机踢球之后的最后一个踢球动作是没有意义的，因为这次踢球仅被用来调整最后的角度，因此，可以用解析法确定。每个踢球序列被一个评估函数分析，此评估函数考虑足球的最后速度，踢球序列长度和对方球员也可能踢到球的周期数。最后选择最高评估值的踢球序列。还使用了简单的启发式方法以搜索到更好的解决方案。仅仅使用随机搜索，考虑静止的球员和它前面的一个足球样本，结果是很

好的，对于 500 次的随机反复，平均速度（超过 2000 次的测试）达到了 2.25。最后选择了 500 作为随机搜索的反复次数，这样可以使计算时间保持较少水平（使用奔腾 500MHz 大概是 1.5ms）。



蓝色小点即为可能的随机中间点

• 图 8-4

随机搜索得到的最好的踢球序列又经“爬山”（hill-climbing）运算法则改善。随机噪声干扰这个序列，然后考虑一个新的踢球序列是否更好。如果是改善了结果，最后一轮的干扰噪声在我们的“爬山”中被重复使用，因为搜索空间的那个方向必须被重新测试。如此进行几个反复。爬山效应以较少的时间代价（对于相同的计算机配置不到 0.5ms）改善结果。对于一个静止的球员和其前面足球的样本，即使将传感器噪声也计算在内的安全可踢范围（不是使用已经给定的理想踢球区域限制），这种方法允许我们在 3 到 4 个周期内将足球加速到平均 2.47。基于优化技术在线搜索最好的踢球序列是解决如何在仿真比赛环境中更大力地踢球的一个较新颖有效的方法。这种方法使得 FC Portugal 球员可以在任何方向踢球（不用转球指令），仅花费几个仿真周期而且能够大力踢球

8.1.3 基于范例的学习方法

对于给定球员的速度向量，球相对与球员的位置向量以及球的速度向量，踢球技能必须找到一个踢球序列，并执行它按照所期望的方式改变球速。如果不可能把球踢到期望速度，那么踢球模块也应该尽量使最后踢出的球速与期望球速的大小差别最小。这都是在保证出球方向与目标方向一致的前提下完成的。

对于球员来讲，存在大量的踢球动作可以选取。为了简化问题，我们在可踢圆环内引入四十个等距离点。只有那些把球踢到这四十个中间过渡点的踢球动作在考虑中。而这些点的密度已经足够大，以确保踢球计划的质量，事实上每一步这四十个中间过渡点中的绝大多数都会被考虑到。

通过引入这些点，问题被大大简化了，因为搜索空间被缩小了。然而，它仍然不可能

为了寻找最优踢球计划实时地进行穷尽搜索，因为这在当前的时间约束下是很难完成的。为了处理这种复杂性，球队实现采用了基于范例的学习。该方法主要被德国 ATHumbt 队所采用。ATHumbt 队还采用该方法应用于其他几个方面。

一个范例包含了球员速度向量的大小，球的相对位置，期望球速向量以及一个包含 40 个非负实数的列表，该列表描述了这四十个点作为中间过渡点的效用值，并被称为(Point Description List,PDL)。为了提高效率，当前球速并没有作为输入。这个点描述列表被用来选择可达的点。只有这些点，它们的描述可以计算得到。

基于范例的学习需要一定大小的范例库支持。这里使用了四个范例库: CB_2 , CB_3 , CB_4 , CB_5 。范例库 $CB_i(i \in \{2,3,4,5\})$ 用来选择含有 i 个过渡踢球的序列中第一个过渡踢球动作。选择出来的踢球动作将称为踢球计划的一部分。

假定，我们给出如下四个参数：

1. 球员速度向量的大小，
2. 球的相对位置，
3. 期望球速向量，
4. 可以采用的点。

CB_i 返回一个评价可达点的数值列表,即 PDL。PDL 中的数值表示了当前选择点作为第一个过渡踢球时的目标出球速度大小的期望误差。比如，假设 CB_3 描述了一个点 P 的值为 0.2。这意味着如果主体第一次踢球把过渡到 P 点，而剩下的三次踢球(注意是 CB_3 ,还有两个过渡踢球和最后一脚出球)是最优选择的情况下，最后出球速度向量的大小与期望出球向量大小的差的绝对值可以控制在 0.2 之内。当然，还有一个前提就是，最终实际出球方向必须与期望的出球方向一致。

期望误差与当前球速向量无关。它仅仅与踢球后的速度相关，这个速度可以用选中的踢球过渡点和球当前位置决定。因此没有必要把当前球速引入到范例库中。

每次选择最有希望的过渡点,使用 PDL 可以直接生成踢球计划。如果踢球计划仅有一个过渡踢球，可以实时地计算一个合理的 PDL,因为这没有太多的计算量，整个计算非常快。所以我们不必再找一个范例库 CB_i 。为了计算需要多少个过渡踢球，计算过程将参考所有的范例库。如果在 $CB_{(i,j)}$ 的 PDL 中没有比 CB_i 的 PDL 好很多的情况下，那么 i 的值就可以确定了。

使用基于范例学习的方法还有一个重要的过程，必须要考虑范例的生成。对于四个范例库: CB_2 , CB_3 , CB_4 , CB_5 。范例库,每个范例库都含有 26400 个范例。范例库的生成是通过动态规划实现的。 CB_i 中每个范例的 PDL 值都可以使用已经生成的 $CB_{i,j}$ 的 PDL 值。

由于 Robocup 仿真联赛使用模拟服务器的实时性，范例的获取必须要非常的快。并非所有生成的范例都可以放到范例库中使用，因为范例获取的过程实在是太长了。因此，只有最优代表性的范例才可以被选中，并在每个范例中使用。如下将演示如何选取最优代表性的范例：

1. 首先随机选择 n 个范例，并把他们从获取到的范例集合中删除。
2. 从剩余的获取范例集中随机选取另外一个范例，将它从范例集合中删除，并添加到范例库中。

3. 检查范例库中的每个范例，评价使用范例库中的范例生成其中一个范例的效用值。
使用其他有最小误差的范例生成的这个范例将从范例库中删除出去。
4. 跳转到 2)，除非获取到的范例集合为空。
5. 最后我们得到了含有 n 个范例的最具代表性的范例库。

我们已经有从获取到的范例集合生成范例库的方法了。但是单个范例的获取仍然是我们面临的一个问题。接下来我们介绍如何获取范例。

一个范例库的查询包含如下四个部分：

- 球员速度向量的大小 p
- 球相对球员位置的笛卡儿坐标 (b_x, b_y) ,
- 期望球速向量的极坐标 (b_{len}, b_{dir}) ,
- 可达点索引的列表。

设个五维向量表示输入 $in(p, b_x, b_y, d_{len}, d_{dir})$ 。而输出可以用我们要查询点的 PDL 表示，它是范例库的返回值。每个范例的 PDL 用下面公式生成：

$$PDL = \begin{cases} PDL_j, \exists j, j \in \{1, \dots, n\}, d(in, in_j) < 0.01 \\ \frac{\sum_{i=1}^n \frac{1}{d(in, in_i)} PDL_i}{\sum_{i=1}^n \frac{1}{d(in, in_i)}}, otherwise \end{cases}$$

这里 $in_i (i \in \{1, \dots, n\})$ 是范例库中第 i 个范例而 $PDL_i (i \in \{1, \dots, n\})$ 是范例库中第 i 个范例的输出, d 是一个距离函数，它给出了输入部分和查询条件之间的距离。

上面的公式基于如下想法计算 PDL。如果当前踢球情况跟范例库里面一个已经存在的范例的输入条件几乎相同，那么它的 PDL 就跟范例库中那个范例的 PDL 相同。否则，范例库中所有范例都会被用来计算这个 PDL。某个范例的影响仅仅依赖于该范例的输入和当前踢球情况的输入之间存在的差别。

实验使用了好几个距离函数，下面给出一个效果最好的距离函数：

$$\begin{aligned} & d((b_{x1}, b_{y1}, d_{Len1}, d_{Dir1}, p_1), (b_{x2}, b_{y2}, d_{Len2}, d_{Dir2}, p_2)) \\ &= (100 * d_1^2(b_{x1}, b_{x2}) + 100 * d_1^2(b_{y1}, b_{y2}) + 100 * d_2^2(d_{Len1}, d_{Len2}) \\ &+ d_3^2(d_{Dir1}, d_{Dir2}) + d_4^2(p_1, p_2))^{\frac{1}{2}} \end{aligned}$$

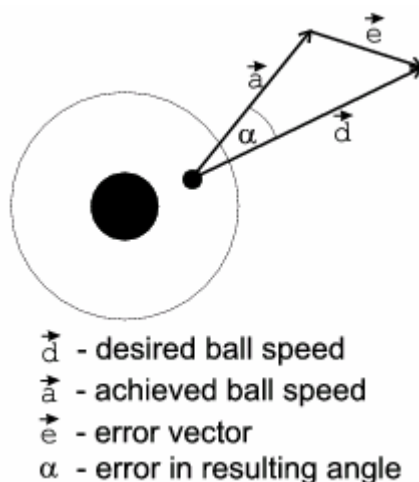
这里解释一下几个参数的意思：

- $(bx1, by1), (bx2, by2)$ 是球的相对位置，
- $(dLen1, dDir1), (dLen2, dDir2)$ 是极坐标的期望出球速度向量，
- $p1, p2$ 是球员速度向量的大小，
- $d1, d2, d3, d4$ 是正规化参数差别的函数（最大差别为 10）。

利用范例学习的方法比我们以前看到的方法都要复杂许多。ATHumbt 队对这个方法和

他们以前使用过的踢球方法做了对比测试。相比而言，旧的方法可以生成一个简单，有效的踢球计划。这种方法下通常生成的过渡踢球都是在期望出球方向上的，最后一脚则把球加速到所需要的速度。为了对基于范例的方法进行测试，ATHumbt 队的开发者随机选择了 165 个踢球情况作为测试例。把新旧两种方法都同时应用到这些测试例上。实际比赛中，球员常常需要把球踢到最大速度。因此，我们要求收集的三分之二踢球情况把球踢到最大速度。

在测试时，我们需要观察两组输出，一组是误差向量的长度和误差向量的绝对方向为了是比较能有代表性，测试中我们采用了不同的球员速度和误差上限。



• 图 8-5 踢球测试中用到的参数

由于运算复杂度上的关系，测试采用了小型范例库。但是测试给了我们一个非常不幸的消息，踢球新方法踢出的不合格踢球数大大高于旧的方法。平均来讲，新方法踢出的不合格踢球高于旧方法 63%。针对这个结果，我们可以用如下两个理由进行解释：

在旧方法下，在最后一次把球踢出之前，球员已经把球的速度改变到期望方向上，同时还处于踢球半径之内。因此，即使没有控制好球的速度，使它除了踢球方向，我们依然保证它的运动方向是没有问题的。在新方法下，踢球没有对球的运动方向做这么强的约束。因此，球员往往在倒脚调整球的过程中就把球踢出了可踢区域，这往往得到一个错误运动方向的结果，显然是个不合格踢球行为。另外，新的方法对于主体的世界模型信息的准确度要求比较高。而模拟平台在球运动过程中又添加了随机误差，主体是实际运行中又常常受到网络问题的阻挠。这些使得新的踢球效果表现没有想象中好。

旧的踢球方法往往能比新方法更快的生成踢球计划。当时间要求比较高的时候，新的方法就不能完成踢球。

但是从踢出的球速大小来看，新方法确实要新方法要好的。在 497 个测试场景中有 295 个场景新方法可以比旧方法踢出更快的球。而主体运动速度比较低情况下，测试结果还可以更好。

一个主要的问题是，当前的实现是基于小范例库的。我们希望随范例库增大能取得更好的效果。而另一个问题是前面我们选取样本是 40 个控制点中某些点接近于边界，当调整

到这些点是常常踢出边界。

总的来说，这里引入了一种新的方法，虽然它有些缺陷，但也有非常吸引人的地方

8.1.4 强化学习方法

8.1.5 动态规划方法

相对于前一种方法而言，我们在设计主体踢球模块时，还可以采用另外一种稍微简单的方法。WrightEagle 在 2003 实现了一个比较快的踢球模块。自从 7.0 版本模拟平台的推出依赖，由于踢球力量到加速度的转化率（`kick_power_rate`）提高到 0.027 以来，把球高速踢出比以前容易多了，在 4 个周期内都能把球加速到 2.5 左右。

使用搜索和强化学习的方法来设计踢球模块，使 Agent 能拥有快速踢球的能力，但是都存在这样的问题，Agent 始终无法确切的知道球将在哪个周期被踢出去。球的运动对于所有 Agent 来讲都有巨大的影响。当球的运动方式发生改变的时候，往往能改变 Agent 的决策，Agent 转而执行其他决策。出球时间对于预测其他队员的行为具有非常大的意义，因为出球意味着球的运动方式确定了，不再受 Agent 控制而可以改变。这里介绍的在这种可以确定出球的周期，显然对于决策非常有帮助。

离散化球队相对球员的位置空间，用来表示过渡踢球的中间状态。以 10 x 10 的格子离散球员周围以球员为中心，边长为 $2 \times \text{kickable_area}$ 的正方形。当主体没办法一次把球踢到目标速度时，它会把球努力调整到这些点之一。而对于可以一脚出球的情况我们根本不必用到这些点，直接通过向量运算就知道。

相对于一脚出球来说，有一个过渡踢球的情况稍微复杂一点。不过，10 x 10 的格子数目规模相对于现在的计算机来讲还是可以承受的。因此简单使用了搜索算法即可解决问题。

动态规划非常合适求解三个周期的踢球情况。首先，求解过程需要离散化踢球模型。Agent 把身体周围连续的 360 度，离散地划分为 36 个方向。Agent 需要保留的结果是确定第一个过渡点 A 时，再经过一个过渡点可以踢出到某个方向 α 的最大速度。因此，计算结果将保存到一个大小为 $36 * 100$ 的列表。

首先计算两个过渡点的出球速度。如果确定了第二个过渡点的位置，那么 Agent 就知道了球在第二个过渡点的速度同时还知道 Agent 在第二个过渡点对球的实际控制能力。Agent 给球的加速度与 Agent 踢球力量之比称为 `kick_rate`，它只与 `kick_power_rate`，球到 Agent 身体距离，以及球相对于 Agent 的角度相关：

$$ep = \text{kick_power} * \text{kick_power_rate}$$

$$ep = ep \cdot (1 - 0.25 \cdot \frac{\text{dir_diff}}{180^\circ} - 0.25 \cdot \frac{\text{dist_ball}}{\text{kickable_margin}})$$

$$(0 \leq \text{dir_diff} \leq 180, 0 \leq \text{dist_ball} \leq \text{kickable_margin})$$

有了这些，Agent 能够计算出第二个过渡点在给定方向所能踢出的最大速度。然后计算三个过渡点的情况。这可以在已经给定子问题的解的情况下进行。Agent 计算一个点到

两点情况的第一个点的可达性，并复制它的出球速度到新表即可。

通过试验，三个过渡点的情况已经能完全满足实际使用的需求。因此不需要太多迭代。更重要的是这些计算并不耗费太多时间，因此当使用异构球员时，可以在比赛开始前实时计算他们。

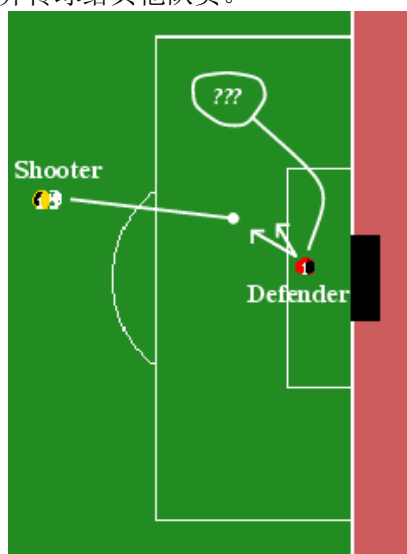
使用生成的数据时，采用距离出球方向最近的离散方向，去该方向出球速度最大点作为调整点。

8.2 截球模型

8.2.1 提出问题

跟年轻球员一样，Agent 在学习比较复杂的策略前需要学习如何控制球的技术。在多主体系统里面通常，Agent 的个人行为能力还没有具备就要求 Agent 熟练地与球队其他成员配合显得太过于苛刻了。

拦截一个移动的球是 Server 端常常被要求执行到的一个任务。除非 Server 给出信息在没有守门员抱住球的情况下，球是静止的，那么这个技能对任何踢球动作都是需要具备的。它被 Agent 用在几乎所有的角色上：守门员和后卫需要拦截一个射向球门或者是传球的球，而中场和前锋则反复拦截球，并传球给其他队员。



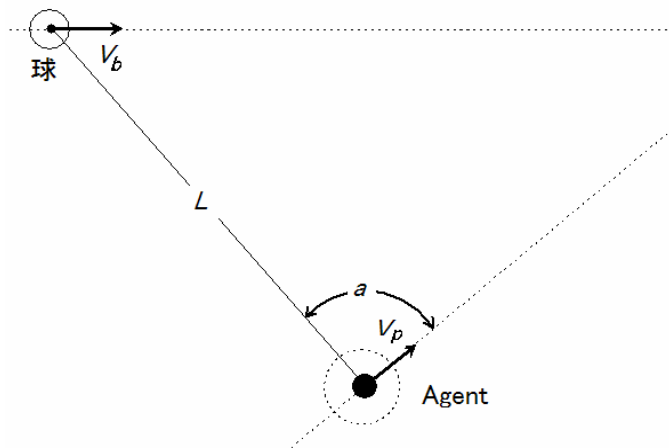
• 图 8-6 常见的拦截球情况

由于球的运动噪音和 Agent 有限的感知能力，拦截球仍然是一项非常困难的任务。如前面的章节所述，每个 Agent 有着有限的视角，并且只能在一定的离散时间内接受到视觉信息。通常当一个 Agent 尝试着去拦截一个移动的球的时候，这个球也只是有个大概的移动方向而已。如图所示，Agent 非常难以选择。

通常 Agent 有两种方法可以采用来拦截一个移动中的球:

根据球经过的路径, 位置, 分析并估计出球的速度, 并由此来预测球在未来的移动
收集一些成功拦截的例子, 并利用人工智能里的学习技术生成拦截球技术。

在这里, 我们将同时介绍两中技术用于计算拦截球。假定球速, 位置以及 agent 的速度和位置, 最后可以计算出 Agent 的拦截位置, 拦截时间(系统的时间是离散时间, 默认为 100ms 一个周期, 但很多 Agent 计算时以连续时间计算)。



• 图 8-7 Agent 分析计算拦截球

8.2.2 分析计算求解拦截

拦截球问题主要是指: 对某个确定球员, 当 Agent 知道他地具体信息, 如何利用转身(turn)和加速(dash)命令控制球员最终控制球。监督下地学习方法收集足够多的成功例子, 利用这些经验以生成拦截球的技术。但是最近, 由于性能上的优势, 分析计算方法则被广泛使用。

最早使用该方法的是 CMU 大学的 Robocup 仿真球队。

跟踢球的情况相似, 可以假设, 球的初速度, 方向, 以及给定球员的速度, 位置信息等等, 在计算拦截球的过程中假设球不会被再踢一次。为了计算 Agent 拦截一个移动目标, 可以假设, 时间是连续的, 而 Agent 的移动速度是固定的。可以证明这个算法可以收敛到正确的拦截时间上。

对任意球的位置 B_0 , 初速度 V_0 , 队员位置 R_0 , 我们定义拦截时间为函数 $I(B_0, V_0, R_0)$, $I(B_0, V_0, R_0)$ 表示拦截队员所可以利用的最小时间, 到达某个位置, 并踢到球。球以 V_0 初速度移动, 经过时间 t 之后的瞬间速度为 $V = V_0 e^{-t/\tau}$ 。而球员的速度 V_p 被设定为仿真平台内的最大值——每秒一米。这个速度衰减参数 τ 会不断调整以适应模拟平台的离散模拟时间—— $\tau \approx 2s$ 。 $I(B_0, V_0, R_0)$ 抽象定义了模拟平台的本质和球员转身和加速的时间。然而 $I(B_0, V_0, R_0)$ 可以相对简单的用浮点数进行数值化计算。而且这种表示方法也可以采用启发式方法来调整适应球员的转身和加速。

为了计算 $I(B_0, V_0, R_0)$ ，我们先考虑 t 时间后球的位置，表示为 $P(t)$ 。它可以用如下公式计算：

$$P(t) = B_0 + V_0(1 - e^{-t/\tau}) \quad (1)$$

拦截之间满足一定条件，球员在拦截之间内可以行动的距离等于球员的初始位置到拦截时球位置的距离。这个条件用 $\|X\|$ 表示，它表示了向量 X 的长度：

$$V_r t = \|R_0 - P(t)\| \quad (2)$$

只要能计算到最小的 t ，使它满足条件(2)。一个简单的想法是利用牛顿迭代求解(2)。特别的是，当一个快速移动的球距离那里非常近的时候，球有可能会穿过拦截的球员，而球员会在更远处当球速降低下来时再次拦截到球。这几种情况都满足条件(2)，但我们只要求得最短那个时间。

为了求解它，可以采用牛顿迭代的修改版算法。在该算法里面，从 t^0 开始，由 t^i 迭代计算出 t^{i+1} 。由 t^0 计算 t^{i+1} 时，如果 t^0 不大于拦截时间，那么 t^0, t^1, t^2, \dots 序列单调增并收敛到拦截时间(对于条件 2 的最小值)。实际上，我们只要方程 2 中的距离小于某个值(比如 0.1 米)时，我们可以停止计算。通常可以在三到四次迭代后到达中止条件。因此，这样的计算非常快，可以在每个模拟周期内计算很多次。

为了形式化该问题，我们定义 $g(t)$ 如下：

$$g(t) = \|P(t) - R_0\| - V_r t \quad (3)$$

现在可以把条件 (2) 改写为 $g(t) = 0$ 。注意，如果球员直接跑向 $P(t)$ ， $g(t)$ 为球员在 t 时间和球的距离。标准的牛顿方法更新 t^{i+1} 为 s^{i+1} ， s^{i+1} 由如下定义：

$$s^{i+1} = t^i - g(t^i) / g'(t^i) \quad (4)$$

注意，如果 $g(t)$ 为正，而 $g(t)$ 随 t 变大而变小，那么倒数 $g'(t^i)$ 为负数，更新使得 $s^{i+1} \geq t^i$ 。不幸的是，对于某些特定函数，它仍然可能是 $g(t)$ 局部最小，而不是求得拦截时间。为了避免出现这个局部最小值， $g(t)$ 和 $g'(t^i)$ 都需要为正，(4)更新后 $s^{i+1} < t^i$ 。如果我们设置 t^{i+1} 为 s^{i+1} ，那么可能会遇到这个局部最小值。当拦截发生在球穿越球员之后速度降低时，公式(4)也可能会出现同样的情况。我们得到的 s^{i+1} 远远大于拦截时间。

在修改更新函数以得到所需的收敛特性前，我们定义倒数 $g'(t^i)$ 为给定的表达式：

$$\begin{aligned} g'(t) &= \frac{P'(t) \cdot (P(t) - R_0)}{\|P(t) - R_0\|} - V_r \\ &= (e^{-t/\tau} V_0 \cdot U(t) - V_r) \end{aligned} \quad (5)$$

如果 $\|P(t) - R_0\|$ 为 0，那么运动到了球员的初始位置。但是，这一定发生在球员拦截到球之后。因此，在拦截前，我们可以假设 $\|P(t) - R_0\|$ 非零， $U(t)$ 被定义为一个单位向量。我们可以重写方程 (4)，如下：

$$s^{i+1} = t^i + g(t^i) / (V_r - e^{-t^i/\tau} V_0 \cdot U(t^i)) \quad (6)$$

因此，更新规则改为如下其中 s^{i+1} 如规则(6)定义：

$$\begin{aligned} t^i + g(t^i)/(V_r - e^{-t^i/\tau} V_0 \square U(t^i)) & \quad \text{if } (V_0 \square U(t) < 0) \\ t^{i+1} = t^i + g(t^i)/(V_r - e^{-t^i/\tau} V_0 \square U(s^{i+1})) & \quad \text{if } (V_0 \square U(t) > 0) \text{ and } (g'(t^i) < 0) \\ t^i + g(t^i)/V_r & \quad \text{其他} \end{aligned}$$

我们现在可以证明， $t^{i+1} \geq t^i$ ， $t^i \leq I(B_0, V_0, R_0)$ ，并且 t^i 可以收敛到 $I(B_0, V_0, R_0)$ 。当 V_0 等于零时，我们采取更新跪着的最后一个条件（而且只需要一次计算，多次迭代并不能是计算更加有效）。因此，我们假设 V_0 非零，这不会使计算结果失去广泛的效力。假设 R_\perp 是球运行路线上某点，它距离球员初始位置最近，也就是 R_0 在球路上的影射。这里把 t^0 设置为，球员从 R_0 移动到 R_\perp 的时间。首先，我们考虑这种情况：当球员到达 R_\perp 时，球在 t^0 时尚未到达 R_\perp 。这可以用条件 $V_0 \square U(t_0) < 0$ 描述，当前仅当满足该条件时，会出现这种情况。这里，球员会比球更快到达 R_\perp 。只要球没有经过 R_\perp ，那么 $V_0 \square U(t_0)$ 的值就为负数，并且 $g'(t)$ 的值会随时间减小。同时 $V_0 \square U(t_0)$ 也会随时间减小。这隐含地描述了，当拦截点在 R_\perp 之前时， $g'(t)$ 随时间递减但绝不会小于 V_r 。当我们越来越靠近解的时候，倒数的大小越来越小，但仍然不等于零，牛顿迭代方法具有以上提到的性质。当拦截点在 R_\perp 之前，更新规则总是采用第一种方法计算，它直接对应着经典的牛顿迭代方法。因此，所有的属性仍然成立。

如果 $V_0 \square U(t_0) = 0$ ，那么球员与球同时到达 R_\perp 点，也就是说 R_\perp 即拦截点。在这种情况下 t_0 就是拦截时间。现在假设 $V_0 \square U(t_0) > 0$ 。在这种情况下，球经过 R_\perp 点时，球员尚未到达 R_\perp 。这时，拦截点超越了 R_\perp ，直觉上来说当球员只有在球速下降后才能拦截到球。在这里 $V_0 \square U(t_0) > 0$ 时，更新方式是不变的，倒数 $g'(t)$ 重视大于 $-V_r$ 。因此，在该情况下，球在 t^0 时穿过了 R_\perp 使用第三个更新规则绝对不会得到错误的结果。另外，只用第三条规则，还可以保证收敛到拦截时间。这是，因为迭代步长为正，计算过程 t 决不会超过拦截时间。但是步长有可能会到达 0。如果 $g(t)$ 到达 0，那么此时得到的结果即拦截时间。因此，只要第二个更新规则没有出错，那么第二个规则也能收敛到拦截时间。注意由于 $V_0 \square U(t_0) > 0$ ，它是随时间单调增的。事实上当我们选择第二个规则更新时， $V_0 \square U(t_0)$ 变大表示我们会得到 $t^{i+1} \leq s^{i+1}$ 。另外对于 $t \in [t^i, s^{i+1}]$ ，我们可以得到：

$$-g'(t) \geq V_r - e^{-t/\tau} V_0 \square U(s^{i+1}) \quad (7)$$

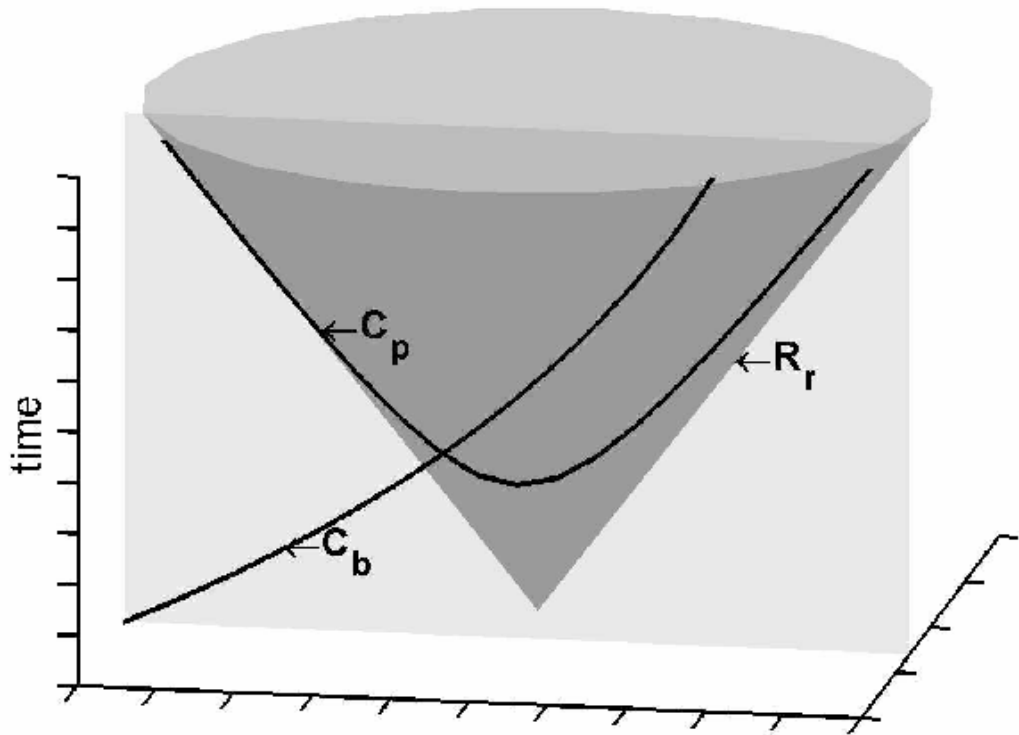
由于公式 (7) 为 $g'(t)$ 给出一个界。我们可以用第二个更新规则克服穿越问题。

8.2.3 分析计算方法的改进

分析计算被大多数球队所采用并不断得到改进。这主要介绍清华 TsinghuAeolus 队对该方法的改进。

简化后的拦截球模型包括球的初始位置 B_{p0} ，初始速度 B_v 和它的方向，以及接球者位

置 P_r 。另外一些因素则在此后用启发式方法添加进去，比如球员身体方向，速度等等。如果没有随机噪音，球的移动路径是固定的。接球者在 t 周期内可达的范围为一个以他初始位置为中心的圆。圆的半径随时间线性增长，斜率为球员的最大速的 v_p 。如果把 t 作为第三个维度，那么我们可以得到拦截过程的一个三维图像：

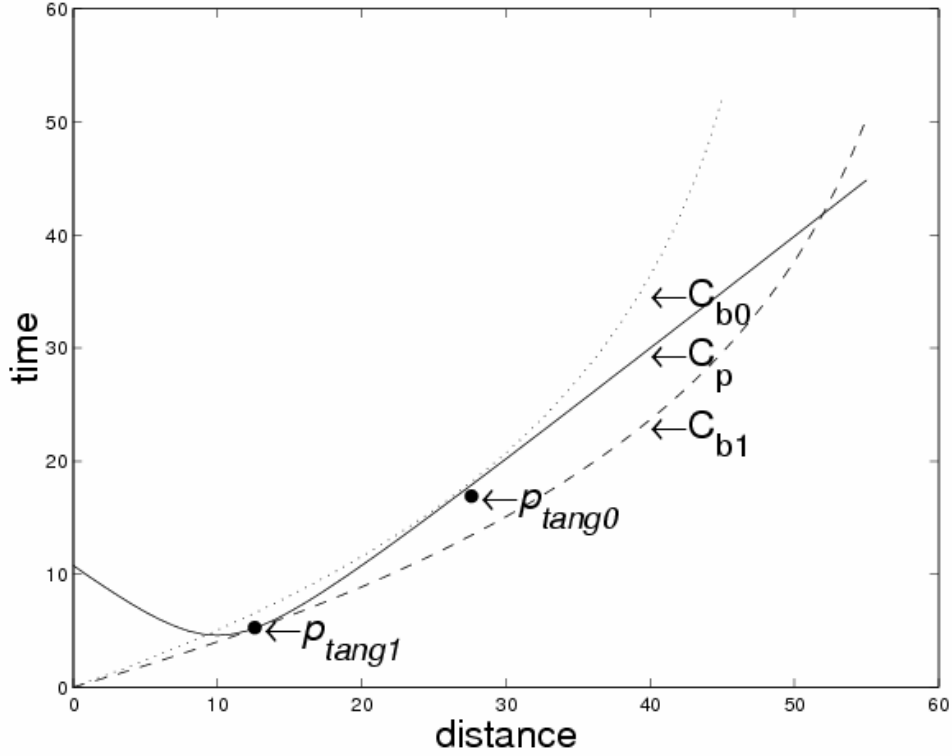


• 图 8-8 拦截球解析示意图

球的运行路径 C_b 是一条弧线。而接球者的可达区域为一个锥型 R_r ， R_r 和 C_b 相交部分则是可以完成拦截的点集，表示为 S_{ol} 。在上一节讲述拦截的最短时间，但是并不是所有的最短拦截时间都是最佳的拦截策略。假设在某个场景里面，一个球员位于球的前面，而球以非常快的速度向他运动。此时转身去接球并带球前进比直接在正前方拦截球并带球前进相比，并不是最好的选择。而且这种情况在比赛中会被频繁的遇到。因此，计算出 S_{ol} 是非常有必要的。在三维坐标里面， C_b 在一个垂直平面内，该平面与 R_r 相交与双曲线 C_p 。如图可以看出， R_r 和 C_b 最多可以有 3 个交点。

为了求解这个问题，假设 C_p 和球速的方向不变，而 B_v 则是可变的。改变 B_v 的同时， C_b 将会与 C_p 相切。对应的 B_v 值为 $b_{\tan gv0}$ 和 $b_{\tan gv1}$ 。而切点分别是 $P_{\tan gv0}$ 和 $P_{\tan gv1}$ 。当 B_v 落在区间 $(0, b_{\tan gv0})$ 或者 $(b_{\tan gv1}, +\infty)$ 时，很直观，这时仅有一个交点在 C_p 和 C_b 之间。

而当 $B_v \in (b_{\tan gv0}, b_{\tan gv1})$ 时, 则有三个交点。特殊情况下, B_v 等于 $b_{\tan gv0}$ 或 $b_{\tan gv1}$ 时, 仅有两个交点。如果给定 C_p , 合适的初值和迭代方法可以用牛顿迭代计算出 $P_{\tan gv0}$ 和 $P_{\tan gv1}$ 。那么对应的 B_v 是相对容易计算的。



• 图 8-9 几种可能的情况

回到最初的问题, 已知 C_b 与 C_p , 我们采用牛顿迭代计算交点。 C_p 是首先在顶点 p_s 处被分为两部分。显然, 在 C_b 左边不会有多于一个的交点, 如果存在一个交点, 则可以用标准牛顿迭代求出。而对于 C_b 的右边, 我们假设存在 3 个交点 p_0 , p_1 , p_2 。我们只讨论如下情况。而另一种情况可以用相对比较简单而且近似的方法完成。很显然 p_0 位于 p_s 和 $P_{\tan gv1}$ 之间, p_1 则处于 $P_{\tan gv0}$ 和 $P_{\tan gv1}$ 之间, 而 p_2 处于 $P_{\tan gv0}$ 到 $P_{+\infty}$ 之间 (在 C_p 点的 x 值足够大)。如果用 $f_{cb}(x)$ 表示函数 C_b , $f_{cp}(x)$ 表示 C_p 右边。标准的牛顿迭代方程如下:

$$xi+1 = xi - \frac{f_{cp}(x_i) - f_{cb}(x_i)}{f'_{cp}(x_i) - f'_{cb}(x_i)} \quad (1)$$

但是它有可能出现超越的现象。 $f'_{cp}(x)$ 的值在区间 $(0, v_p)$ 中。当把 $f'_{cp}(x)$ 替换为 1 时可以得到最值, 因此我们修改更新规则。

$$x_{i+1} = x_i - \frac{f_{cp}(x_i) - f_{cb}(x_i)}{0 - f'_{cb}(x_i)} \quad (2)$$

$$x_{i+1} = x_i - \frac{f_{cp}(x_i) - f_{cb}(x_i)}{v_p - f'_{cb}(x_i)} \quad (3)$$

采用 p_s 当作初值时, 利用(2)迭代, x_i 收敛到 $x(p_0)$ 。在迭代中, $x_{i+1} < x_i$, $x_i > x(p_0)$, 这些并不难于证明。同样如果用 p_s 作初值, 而才用(3)迭代则会收敛到 p_1 , 采用(2)迭代 $p_{+\infty}$ 则收敛到 p_2 。

另外一个相关的问题是这对传球选择非常重要。因为在传球前, 有必要知道穿过某点所需要的最小速度。这里 C_p , 对应一个点 p , 而球的速度方向是已经给定的。问题转为计算最大的 B_v 值, 因此在 C_p 和 C_b 之间至少有一个交点 p 点之前。最重要的是 C_p 和 C_b 相交于 p 或者 $P_{\tan g 0}$ 。假设交点是 p 或者 $P_{\tan g 0}$, 显然可以轻松计算出 B_v 。最大那个就是我们寻找的答案。

拦截球问题是一个非常基本, 但又非产重要的问题, 这里采用分析计算的方法提供了一种精确计算拦截点的方法, 这有利于设计更强的对抗技能。

8.2.4 二分法求解拦截球

对时间 t 进行二分法求解拦截时间比前面的迭代更简单一些。二分法计算计算时, 需要对时间 t 设置取值范围。而最快拦截的拦截点被分为两种情况:

球在球员初始位置在球路上的投影前, 那么很显然时间 t 取值范围在, 从球员初始位置到投影的时间到球运行到投影位置的时间之间。即, t_{\min} 设为球员到 R_{\perp} 的时间, 而 t_{\max} 设为球运动到 R_{\perp} 的时间, 所以二分计算的范围是 (t_{\min}, t_{\max}) 。

如果球运动到 R_{\perp} 时, 球员还不能到 R_{\perp} , 那么拦截地点一定是在超过 R_{\perp} 的某点。这时, 可以取 t_{\min} 为球运动到 R_{\perp} 的时间, 而 t_{\max} 则可以简单的设置, 由于球场场地大小有限, 所以球员跑动覆盖整个场地的时间是有上限的, 比如 100 个周期等。余下的工作可以用二分迭代算法计算。

```
Int intercept(t_min, t_max)
Int i ;
while (t_min + 1 < t_max)
t = (t_min + t_max) / 2 ;
    如果 (球员不能在 t 之前到达 t 时刻球所在位置)
        t_min = t ;
    否则
        t_max = t ;
end while ;
```

这里还有许多可以细化考虑的问题, 比如 t_{\min} , t_{\max} 等变量的初值, 合理选择初值

可以减少迭代次数，降低计算量。

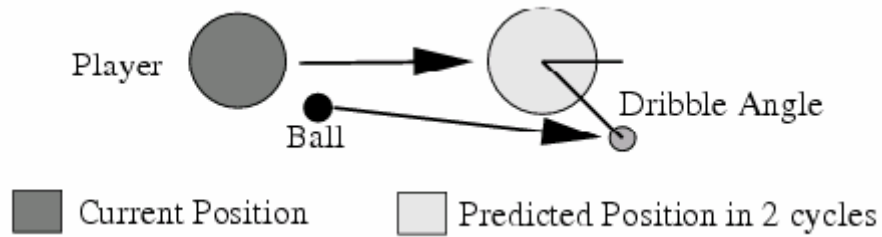
8.3 带球模型

Robocup 比赛和实际的足球运动一样，有时需要球员能控制球向前推进：控制节奏或者等待队友接应，又或者寻找射门机会等等。这种技能称之为带球，实现它的最初想法显得非常简单：利用踢球来控制球，或者 dash 控制身体，交替使用这些动作而已。更复杂一点就是加入转身动作，使带球能够改变前进方向。

每个周期，Agent 检查如果执行 dash 动作，球会不会在下个周期仍然在可踢范围之内。如果是这样，那么采取 dash，否则必须执行 kick 来控制球的运行。

8.3.1 简单实现

这里我们简单介绍 CMU 实现的带球模型。它不考虑躲避对手，也不考虑改变前进方向等等因素，只要求每个周期能控制到球。



• 图 8-10 简单带球中的对象移动模型

如上图，CMU 实现中，每次踢球的结果是球在下一个周期位于相对 agent 某个固定的角度上——dribble angle。这主要是考虑到方便在下一个周期控制球。通常 dribble angle 取值域为 $[-90, 90]$ 。

首先，我们看看 Agent 两个周期后的位置

$$p_{new} = p_{current} + v + (v * pdecay + a)$$

这里 p_{new} 为球员的新位置，而 $p_{current}$ 为球员当前的位置， v 是球员的当前速度， $pdecay$ 被设为服务器参数 `player_decay`，而 a 这是 Agent 控制身体 dash 之后产生的加速度。通常 $a = power * dash_power_rate$ ，在选择 `power` 是最好参考 Agent 当前的体力状况。

在已知 p_{new} 的基础上，由于通常球会被放到跟 Agent 相对固定的某个角度上，同时还要在 Agent 踢球范围内，因此得到球运动后两个周期的位置 p_{target} 。那么可以用如下公式计算球的运行轨迹：

$$traj = \frac{p_{target} - p_{ball}}{1 + bdecay}$$

$traj$ 是球运行的轨迹, p_{ball} 是球当前位置, $bdecay$ 被设为服务器参数 $ball_decay$ 。如果由于踢球模型的限制, 不能把球提到目标位置, 那么 Agent 则需要对球进行调整。这时 Agent 把球提到身体周围的某个位置, 从而可以在下一个周期正常踢球。

可以看出, 这里 Agent 带球的模型相当简单, 可以轻松预测球和球员的位置速度。而且计算被限制在太小的范围, 仅仅两个周期而且每个周期都要重新规划。

基本带球技能模型里面有一个重要的参数控制带球: $dribble_angle$ 。这里介绍智能带球, 它主要用于计算 $dribble_angle$ 以躲避场上的物体。直观的说, Agent 要使球远离对方球员。因此如果对方球员在右侧, Agent 会把球放到左侧。

Agent 需要考虑它所知道的周围所有对方球员。每个球员对于选择 $dribble_angle$ 都有一定的权重。比如一个对方球员在 45 度, 那么 Agent 会更趋向于选择 -90 作为 $dribble$; 如果对方球员在 -120 度, 那么 $dribble_angle$ 可能会选择 60 度。这样加权每个对方球员后, 计算出 $dribble_angle$ 。当然离 Agent 越近, 方向越靠 Agent 正前方的对方球员权重越大。

8.3.2 考虑更多的周期

在 CMU 模型里面只考虑 2 个周期, 其实在实际比赛时, 往往需要能考虑更多周期, 并连贯执行以获得较快的带球速度。一般在考虑多个周期的时候, 带球行为可分为两种:

球始终控制在可踢范围之内;

球会被提出 Agent 控制区域, 但可以预测在某个时间重新控制球

对于第一个方法的实现变得非常简单, 而第二个带球模式则需要比较精准的拦截球预测模型。以下先就第一个带球模型展开。



• 图 8-11 连续控球模型

为了每个周期都能踢到球, Agent 必须合理选取下一次踢球的位置, 并且小心计算球的运行轨迹。N 个周期后 Agent 的位置是 p_p^n , 球的最终位置取某个相对 Agent 固定的位置是 p_b^n :

$$p_b^n = p_p^n + V^*$$

由于球的运动方式已知:

$$p_b^n = p_b^0 + V_b^0 * \frac{1 - (q_b)^n}{1 - q_b}$$

这里 p_b^0 是球的当前位置, q_b 是球的衰减因子, 可从 server 的参数中得到。由于 p_b^0 和 p_b^n 已知, 那么当前需要的球速向量 V_b^0 的方向就是已知的, 仅仅需要求解它的大小, 所以:

$$|V_b^0| = |p_b^n - p_b^0| * \frac{1 - q_b}{1 - (q_b)^n}$$

由此，我们得到了为了实现目标的出球速度。然而该速度能不能符合我们的需要，则需要一步一步的模拟检验。

我们可以模拟球和 Agent 的行动，以检查 Agent 能否踢到球和 Agent 与球之间的碰撞问题。在求解上面方程过程从，需要预测知道 Agent 的移动。这里假设 Agent 在第一个周期踢球之后，全力加速前进。Agent 的位置可以如下求出：

$$\begin{cases} p_p^1 = p_p^0 + V_p^0 \\ V_p^1 = V_p^0 * Q_a \end{cases} (n = 1)$$

$$\begin{cases} p_p^n = p_p^{n-1} + V_p^{n-1} \\ V_p^n = V_p^{n-1} * Q_a + a \end{cases}$$

Q_a 是 Agent 的速度衰减因此，注意较高版本引入异构球员后，不同球员类型的参数是不一样的。 a 是 Agent 的单周期最大加速度。求解以上方程可以得到

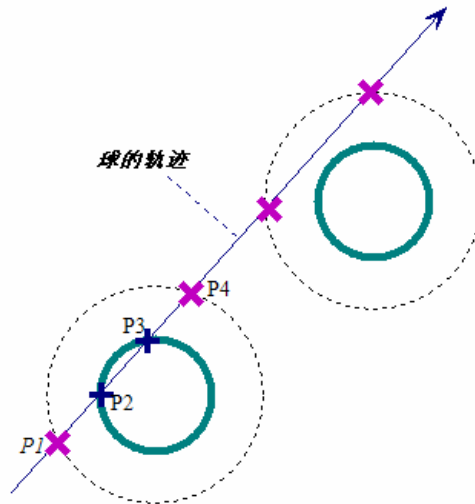
$$V_p^n = (V_p^1 + \frac{a}{Q_a - 1}) * Q_a^{n-1} - \frac{a}{Q_a - 1} (n > 1)$$

$$P_p^n = P_p^0 + V_p^0 + V_p^1 + \dots + V_p^{n-1}$$

$$= P_p^0 + V_p^0 + V_p^1 + \sum_{i=2}^n V_p^i$$

$$= P_p^0 + V_p^0 + V_p^1 + \{ (V_p^1 + \frac{a}{Q_a - 1}) * (\frac{1 - Q_a^{n-1}}{1 - Q_a}) - (n - 1) * \frac{a}{Q_a - 1} \}$$

由上面这些计算，可以顺利的求解出，需要的踢出速度。但是，有的 V^* 的选择使球会被提出 Agent 控制范围，或者是与 Agent 发生碰撞，需要重新选择 V^* ，这显得很麻烦，而且选取合理的周期数也是个问题。下面的方法同样可以用来求解出球速度。



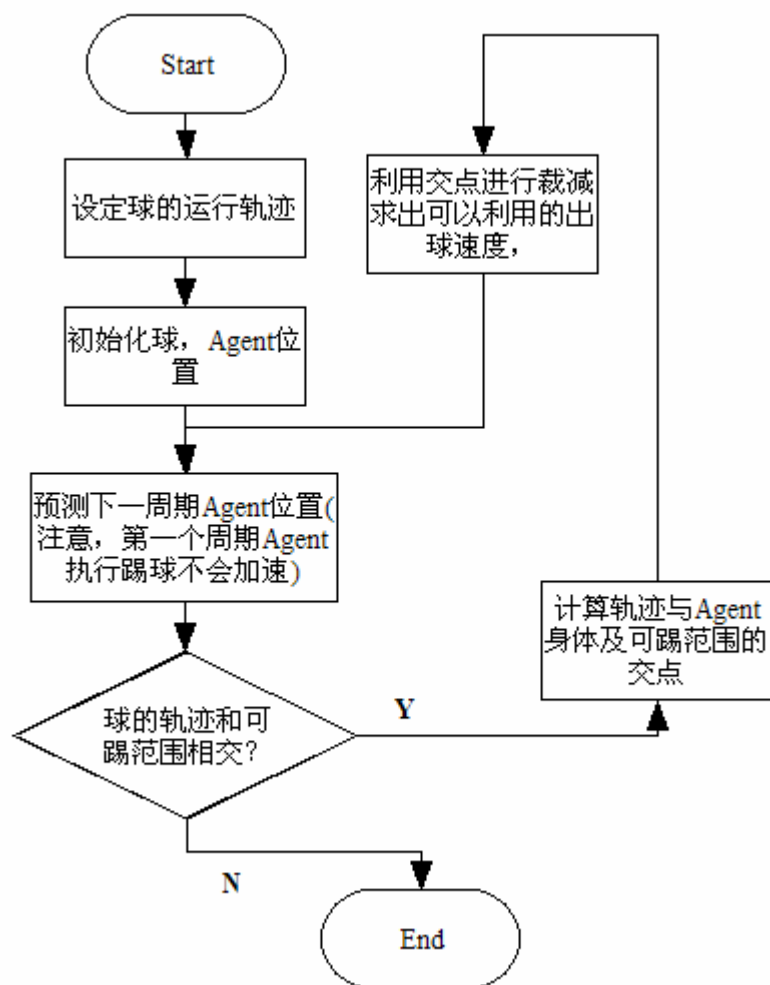
• 图 8-12 球的运动轨迹和 Agent

如上图，球的运行轨迹和 Agent 的关系只有两种：或者有 4 个交点，或者有 2 个交点。对于少于两个交点的运动轨迹，说明球已经出了 Agent 的控制范围。

如果球的的位置落在 $[P1, P4]$ ，那么说明球对于 Agent 来说是可以随时用踢球来控制的；如果球的的位置落在 $[P2, P3]$ ，那么说明球会与 Agent 产生碰撞。其中 $P1, P2, P3, P4$ 这四个点每个对应着一个出球的速度，设为 $V1, V2, V3, V4$ 。速度在 $V1$ 和 $V2$ 之间以及 $V3, V4$ 之间的这一段为合理的出球速度。对于只有两个的情况，凡是速度在 $V1, V4$ 之间的都是合理的速度值。

理论上来说，在 Agent 的每一个位置，都将会得到四个或者两个点，我们会得到这样一个或者两个合理的速度取值范围。最终取他们的交集，在交集的范围内可以随意选择一个值。

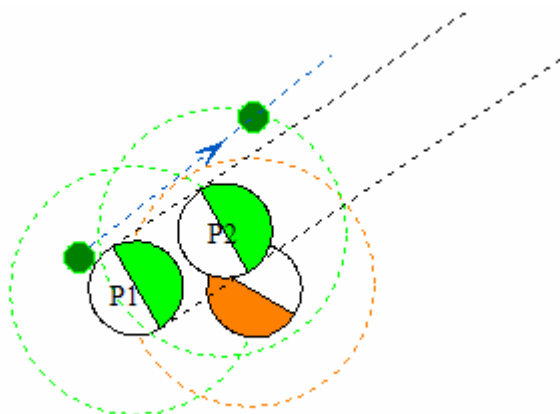
在实际编码中，可以采取边裁减边计算的原则以简化计算和表示。计算 p_p^0 和 p_p^1 的解，取他们的最大子集为可选范围，继续计算 $p_p^2, p_p^3, p_p^4 \dots$ 直到速度的取值范围缩小到一定程度，取该范围的中点即可。仿真服务器在运行时会对所有运动物体添加随机偏差，剪裁计算的结果往往不能精确反映活动物体在场上的运动情况。因此在计算时，采用稍微苛刻的条件可以得到比较好的效果。计算流程图如下：



• 图 8-13 计算带球时出球速度流程

8.3.1 考虑对手等因素

在设计 Agent 个人行为时还可以采用把球踢离身体一定范围，并利用拦截球的技术在对方控球前抢到对球的控制权。该方法在选择下次拿球的具体位置需要计算比较多的拦截。往往 Agent 周围没有对手干扰时可以简化不少计算。而 Agent 周围有对方球员时还需要考虑 Agent 与对方球员的身体碰撞等细微因素。



• 图 8-14 图示 Agent 带球时与对方球员发生碰撞

8.3.4 利用强化学习

采用强化学习(reinforcement learning)也是比较常见的 Agent 行为设计技术。德国卡尔斯鲁勒大学的 Brainstormer 球队利用该技术设计的带球行为实战效果非常好，带球快数而精确。

8.4 射门技术

作为全队进攻的最后一环，Agent 对机会的把握能力往往能决定正常比赛的结果。在比赛中 Agent 有可能出现的问题有：

出现比较好的射门机会，Agent 错误判断为小概率进球机会。

Agent 随意地进行射门，被对方守门员得到，浪费进攻机会。

Agent 射门选择的方向错误，被对方化解。

因此有必要对 Agent 射门行为进行优化，提高 Agent 的射门判断能力。这里，我们简单定义射门如下：在给定的情况下，寻找一个点，使球射向该点时进球概率最大。这样我们就定义了一个优化问题。我们可以把这个优化问题分解为两个独立子问题：

当给出球门内某点时，需要求得把球射向该点后球能够在无干扰的情况下滚进球门的概率。在给定情况下，守门员拿到球的概率。

由于子问题是独立的，当向球门内某点射去时，射门成功得分的概率等于以上两个概率的积。

8.4.1 对球运动的噪音建模

这一小节，我们将展示如何计算当 Agent 把球射向门内某个点时，球会滚进球门的概率。为了这个目的，首先我们要计算球到目标点的运动偏差。这些偏差是有每个周期添加到速

度向量上的随机误差产生的。我们可以把球运动的过程看成是一个散射的过程，因为球每一步移动都是一次散射。该过程可以被看成是一个 Markov 过程，它遵循 Markov 性质：未来状态完全由当前装决定。但是这个散射过程的分析求解还是非常困难的，原因有二：

移动噪音并非是在随机范围内的均匀分布，它以来于球的速度。

该过程收到几何约束，因为球必须最后进门

这些困难使对该过程计算显得非常困难。因此组后我们决定利用试验来学习球运动的统计规律。我们将给出对受几何约束的连续时间 Markov 过程分析求解的一个近似解。这应该是一个相对简单的方法，在其他应用中也同样可以应用。

这个解决方案要求包含对实验中的累计噪音的估计。因此，我们用一个球移动距离的函数来计算球在运行轨迹正交方向上的偏差。而这个函数可以通过重复实验来学习。这里我们把球放到球门前 0 到 32 米远的位置，并让一个球员把球踢向球门。在每个位置，球员都要以最大力量把球踢向球门正中 1000 次。在每次实验中，我们需要记录球进入球门时的 Y 坐标。这个数据会用来计算采样的标准偏差 σ 。为了能得到一个比较好的拟和结果， σ 函数如下表示：

$$\sigma(d) = -1.88 \cdot \ln(1 - d/45)$$

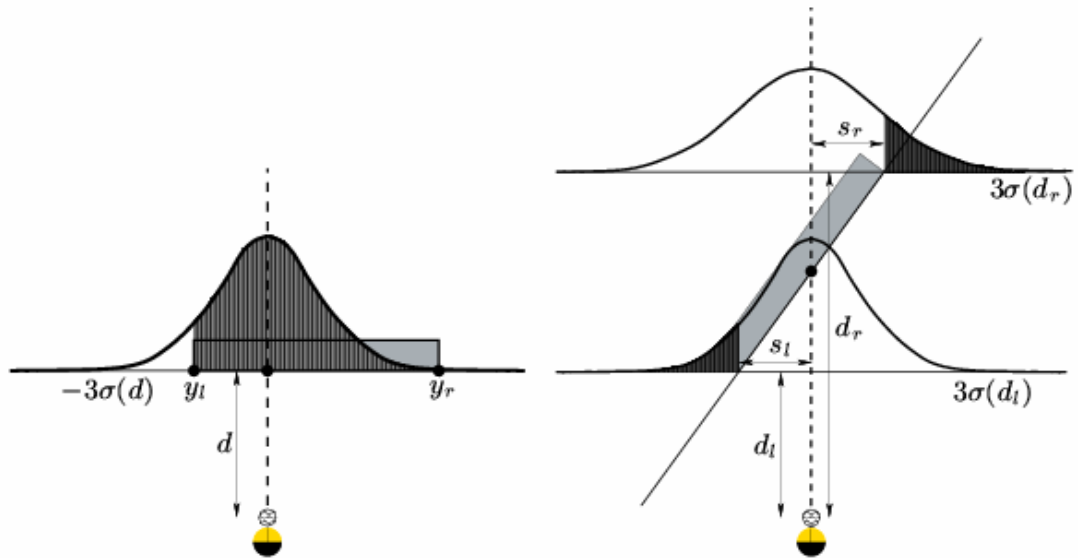
(1)

下一步就是计算球到达球门时的偏差分布。这里我们采用了概率论的一个基本结论——中央极限定理：在 $N \rightarrow \infty$ 时，N 个随机变量 x_i 求和的分布，是高斯分布。高斯函数 g 必须有两个参数，一个零平均和一个标准差 $\sigma = \sigma(d)$ 。采用这一模型，当我们知道球从某个位置射向球门时，我们可以计算球最终会滚进门的概率。这个概率等于图中高斯函数下方阴影的面积。假设左右两个门柱的 y 坐标分别用 y_l, y_r 表示，并且 $y_l < y_r$ 。那么计算该阴影面积可以用积分：

$$P(goal) = \int_{-\infty}^{y_r} g(y; \sigma) dy - \int_{-\infty}^{y_l} g(y; \sigma) dy = G(y_r; \sigma) - G(y_l; \sigma)$$

这里 $G(y; \sigma)$ 是密度函数 $g(y; \sigma)$ 的分布函数。

当然实际射门是未必是如图那样正面垂直射向球门，我们还要计算当球以某个角度射向球门的情况下球进入球门的概率大小。



• 图 8-15 垂直射向球门和以某个角度射向球门的进球概率分布

这种情况或多或少跟前一种情况类似。由公式可知，球运行的偏差仅仅跟球移动的距离有关系。因此沿球门线的进球概率分布不再是高斯分布，这给全概率的精确计算制造了麻烦。然而由图可以看出，我们要求的概率等于概率分布的某个特定形状面积。由于，这里球运行的距离不一样，所以计算球进门的概率不能再用上面的公式计算。这里我们计算进球概率采用另一个公式：

$$P(\text{goal}) = 1 - P(\text{not_goal}) = 1 - P(\text{out_of_left}) - P(\text{out_of_right})$$

这里 $P(\text{not_goal})$ 表示，球偏离球门的概率，球要么从左边偏离球门，要么从右边偏离。这个概率比较容易计算。高斯分布的两端表示了球偏离球门的情况。如图所示，当球到达左门柱时，球移动距离为 d_l 。因此可以利用这个距离计算出球从球门左侧偏出的概率。而球从右侧偏离出球门的概率可以类似的技术求出。当球移动到右门柱时的移动距离为 d_r 。球从左侧偏出球门概率等于左侧的阴影面积：

$$P(\text{out_of_left}) \approx \int_{-\infty}^{-s_l} g(y; \sigma(d_l)) dy$$

这里积分上限为 $-s_l$ ， s_l 表示从左门柱到球门的最短距离。

8.4.2 计算守门员得到球的概率

第二个子问题可以表述如下：给定球门内的某点，求得守门员在球滚进门前拦截到球的概率。我们采用实验的方法来学习这个概率。我们采用了 Robocup2000 的冠军 FC Portugal2000 的球队守门员作为学习对象。因为看上去它是目前最好的守门员程序，而且可以找到二进制代码。

为了把这个问题变换到一个正规的数学框架里面，我考虑到拦截球可以看成两个仅有

两个类的划分问题：给定目标点和守门员以及球的位置（输入的特征向量），预测球的运行是否会被拦截（能被拦截或者不）。此外，我们对在某种预测结论下的后验概率有兴趣。我们做了许多实验。每次实验中，会有一个球员把固定位置上的球射向球门。而守门员会被随机放置到球周围的某个位置。这样执行实验 10000 次得到一个实验数据和球是否射进球门的结论结合。得到的数据经过分析说明了分类的相关特征如下：

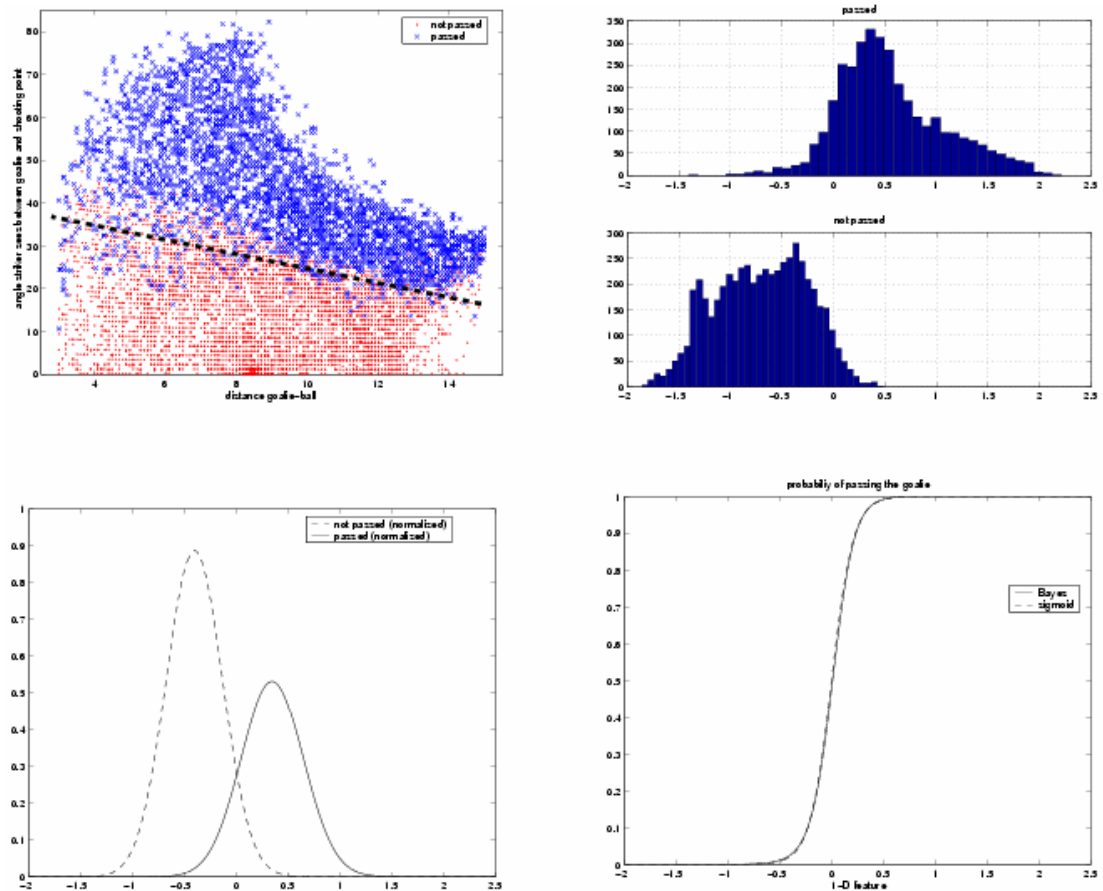
守门员和球进门的位置之间的角度

球到守门员之间的距离

这两个值可以组成一个二维的特征向量空间。记录下的数据集合用图描述。它说明在两类之间存在一个线性的判别函数。我们对布尔分类符使用线性回归方法可以得到这个线性的判别函数。这个过程可以得到最优的费舍尔线性判别式，而费舍尔线性判别式具有这样的性质：他能最大化组间变量和组内变量两个类变量。最后我们得到的判别函数如下：

$$u = (a - 26.1) * 0.043 + (d - 9.0) * 0.09 - 0.2$$

其中距离值 d 的定义域在 $[3, 15]$ 。这个现象可以如下解释：对于一个新的(角度，距离)对 (a, d) 采样减去 $(26.1, 9.0)$ ，然后计算差和向量 $(0.043, 0.009)$ 的内积。得到的向量跟判别边界正交这里判别边界添加了偏移量 -0.2 。在公式中向量 (a, d) 等于零到两个类别的边界。得到的分布图如下：



- 图 8-16 守门员拦截球实验的数据集合和计算后的统计材料

交点 (a_i, d_i) 对于分割线是正交的，可以得到一个一位的点 u_i ，它可以比较好的分割描述两个类别。柱状图分类两个类的分布。我们没有参数化建模两类分布，我们注意到相关范围就只是两个柱状图的相交部分。显然，当 $u \leq -0.5$ 式，没有拦截到情况的后验概率将为 0；而对于 $u > 0.5$ 的情况将被设置为 1。后验概率因此可以用一个 sigmoid 函数表示。在拦截与没有拦截到区域相交的部分，我们用高斯函数来表示。对于某种情况 C ，给定的高斯函数模型条件密度函数为 $P(u|C)$ 。采用这个模型，我们计算后验概率 $P(C|u)$ ，计算方法为 Bayes 规则：

$$P(C|u) = \frac{P(u|C)P(C)}{P(u|C)P(C) + P(u|\bar{C})P(\bar{C})}$$

它将是一个类 sigmoid 函数。因为这只是一个简单的两类分类问题， \bar{C} 则指另外一类情况， $P(C)$ 是指 C 类的先验概率。在图中，用非拦截类划分后验概率，最后采用 sigmoid 拟和

$$P(\text{pass_goalkeeper}|u) = \frac{1}{1 + \exp(-9.5u)}$$

8.4.3 选择射门点

在计算完球进门的概率和被守门员得到的概率后，两个条件独立的假设可以使我们计算出给定条件下得分的概率。最后得到的全概率，就像一个钟一样的形状，它表示了球能进入球门。这个曲线总是有两个局部最大值，对应着左右两个门柱的点。可以简单的用爬山算法(Hill-climbing)求出这两个点。两点中，概率最大的一个将被选来作为射门的目标点。

第九章、高层决策

高层决策作为RoboCup仿真球队设计的重点，是最受人关注的部分。本章将分四个部分介绍阵型与跑位、行为选择、防守策略和团队合作的主要方法。

9.1 阵型与跑位

在现代足球运动中，为了适应攻守战术的需要，全队队员在场上的位置排列和职责分工，称为比赛阵型。在RoboCup足球仿真比赛中，设计者借鉴人类足球的知识，引入阵型的概念，使场上球员位置分布更合理。如同人类足球比赛一样，RoboCup比赛中，球员通过跑位保持阵型的完整，尽可能攻守平衡。在RoboCup仿真比赛中，常见的阵型包括442、433、424、532、352等。如图所示的两支球队，都采取了433的阵型。



图9.1.1 阵型433

基于角色的阵型描述

基于角色来安排阵型，是非常自然的想法，人类足球中的阵型安排也总是强调前锋、中场、后卫等角色的分配。在机器人足球中，引入这些角色的概念来对阵型进行描述，帮助球队建立团队的模型。下面将以卡内基梅隆大学的机器人足球队CMUnited为例，介绍基于角色的阵型描述方法，本节主要参考[Peter Stone Thesis,1998]中的相关内容。CMUnited中，对于球队中的 n 个成员 $A = \{a_1, a_2, \dots, a_n\}$ ，阵型以下面的形式定义：

$$F = \{R, \{U_1, U_2, \dots, U_n\}\}$$

其中 R 表示角色的集合 $R = \{r_1, r_2, \dots, r_n\}$, U 是 R 的子集, 表示同一 U 集合中的角色同属一个小组, $U_k = \{r_{k1}, r_{k2}, \dots, r_{kn}\}$, 一般 r_{k1} 是 U_k 的组长。每一个成员都维护由 $A \rightarrow R$ 的一个映射, 任何时刻每个成员都清楚地知道自己在阵型中的角色。对于每种阵型的描述, 每个成员事先知道, CMUnited的设计者称之为关门协议 (Locker-Room Agreement)。在比赛中阵型以及 $A \rightarrow R$ 的映射都可以实时调整, 如图为动态调整的一个例子。

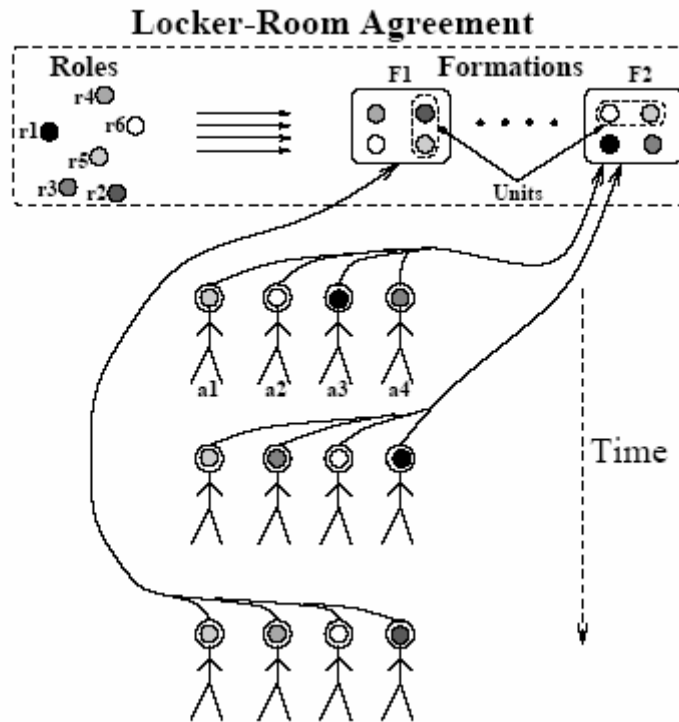


图9.1.2 阵型的实时调整

图中表示了两种阵型, $F1 = \{r2, r4, r5, r6, \{r4, r5\}\}$, $F2 = \{r1, r3, r5, r6, \{r5, r6\}\}$

一开始阵型为F2, $A \rightarrow R = \{\{a1, r5\}, \{a2, r6\}, \{a3, r1\}, \{a4, r3\}\}$

之后阵型不变, $A \rightarrow R = \{\{a1, r5\}, \{a2, r3\}, \{a3, r6\}, \{a4, r1\}\}$

最后阵型变为F1, $A \rightarrow R = \{\{a1, r5\}, \{a2, r4\}, \{a3, r6\}, \{a4, r2\}\}$

在球队的具体实现中, 阵型的描述需要对跑位做出具体的定义, CMUnited球队对跑位点的描述包括三个部分: 职责点 (home coordinates), 职责范围 (home range), 最大范围 (maximum range)。职责点是球员缺省的位置, 球员可以根据情况再职责范围内调整职责点, 球员的位置要尽量保证不离开最大范围。如图为守门员与前卫的位置描述。

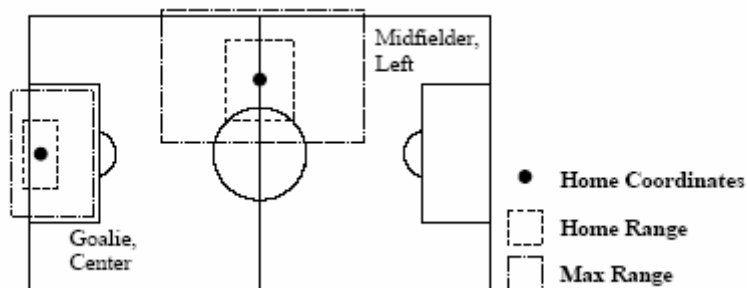


图9.1.3不同角色的位置描述

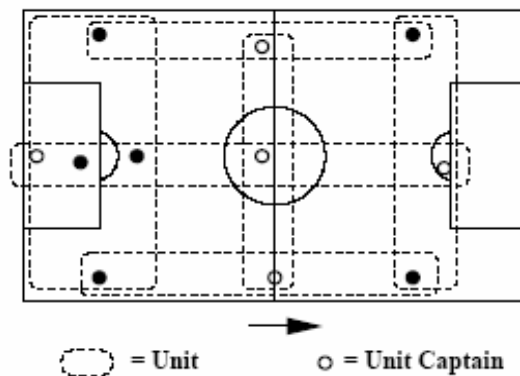


图9.1.4 433阵型的分组

前面阵型的定义中包括了若干小组，在足球的背景中，前锋、后卫和前卫就可以看作是三个小组，也可以将左中右路视为小组。如图表示了433阵型中分组的情况，同一角色，可以同时属于多个分组，定义分组组长，是为了便于协调小组成员，形成合作。

在比赛中，球队的阵型可以动态调整，在这个问题上，有三种基本策略：

固定阵型：在关门协议中定义好阵型，永远不变。

实时调整：在场上特定时刻，根据所有队员都知道的实时变化调整阵型，如：根据比分变化调整。

通讯协调：由场上某一队员决定调整，通过通讯告知其他队友。

后两种调整都改变球员的内在状态，实时调整的触发机制和通讯协调的通讯协议都在关门协议中预先定义好。

在实际的比赛中，球员的跑位需要有一定的灵活性，要求能够适应场上一些不同情况。CMUnited球队在这个问题上采用了三种方法：

对手盯防：根据关门协议，给不同角色的球员分配不同的对手球员进行盯防，不再仅仅依赖缺省的职责区域

基于球的跑位：根据球位置的实时变化，调整球员的跑位。

基于吸引子和排斥子的策略跑位（strategic position by attraction and repulsion, SPAR）：将场上跑位的相关概念形式化，通过某种原则计算决定跑位。这种方法主要考虑无球队员跑位以利于传球完成。

CMUnited球队基于角色进行阵型描述，较好的引入了人类足球中的许多概念，取得了不错的效果，之后的许多方法实际上都是在CMUnited球队工作的基础之上完成的。

基于形势的策略描述

在CMUnited球队基础之上，FC Portugal球队进一步推进了对于真实足球中概念的引入。[Reis & Lau, 2000]借助对于形势的判断，考虑战术和球员角色的区别，FC Portugal球队战略的定义建立在球员类型集合和战术集合的基础上。这里的球员类型不是指Server提供的异构类型，而是球员一系列倾向的定义。战术集合定义了几种特定阵型，用于不同形势下选用，如进攻、防守、攻防转换等等不同形势下可以选用442、433等等不同的阵形。如图9.1.5，描述了FC Portugal球队策略。

基于形势的策略跑位（Situation Based Strategic Positioning, SBSP）

FC Portugal的跑位建立在严格区分场上形势的基础上，场上形势被分为两类，主动模式（Active situation）和策略模式（Strategic situation），主动模式指的是控球、拦截球、定位球等情形，其他的情形就是策略模式，在策略模式下的跑位就是SBSP。

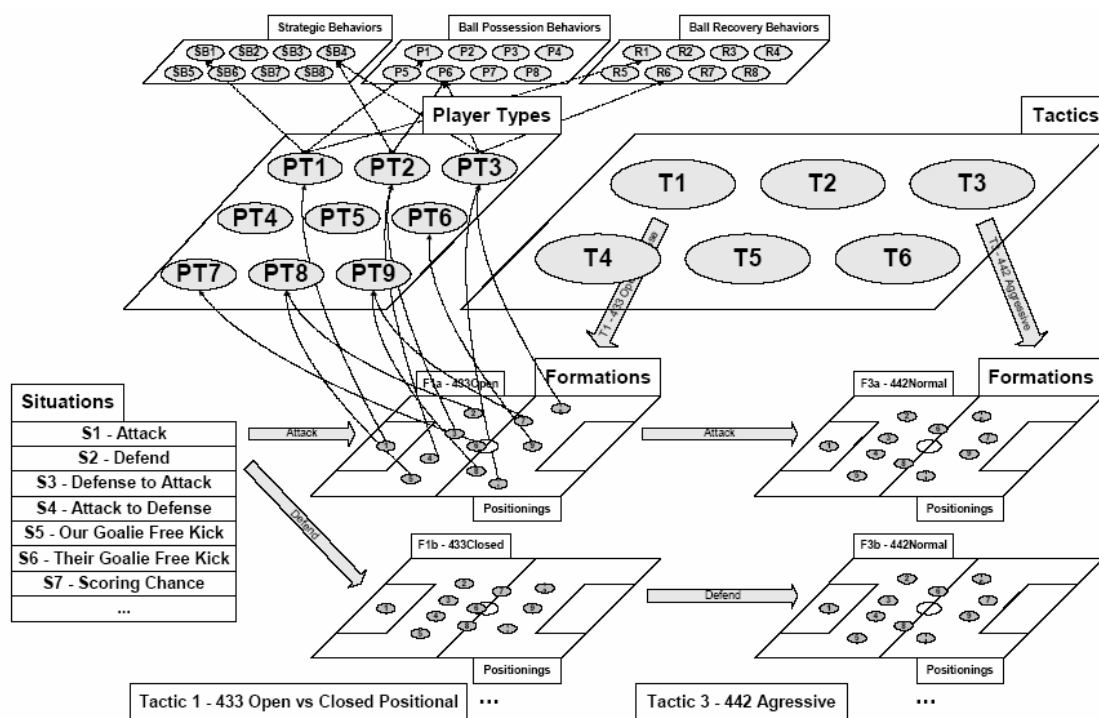


图9.1.5 基于战术、阵型和角色的全队策略描述

在确定形势在策略模式之下时，球员需要分析当前使用的战术及阵形，根据自身类型确定跑位，计算出基本的跑位点。这种位置的计算是根据球的位置和速度、当前的形势（进攻、防守、得分机会等等）以及球员自身的策略。球员策略包括球的吸引参数、区域约束、特殊职责约定、保持在球后的约定、越位约定、特定位置的吸引参数等等。下面给出了一个简化的SBSP算法，这里没有考虑球速、特定位置吸引等因素：

```

/* Variables: P - Player Positioning, BSP - Base Strategic Position,
PT - Player Type, BallP - Ball Position, SP - Strategic Position */
Vector SBSPPosition(Positioning P) {
    Vector BSP = getPosition(Tactic, Formation, P);
    PType PT = getPlayerType(Tactic, Formation, P);
    SP = adjustSBSPPosition(BSP, PT, BallP);
    return SP;
}

Vector adjustSBSPPosition(Vector BSP, PType PT, Vector BallP) {
    SP = BSP + (BallP.X*BallAttraction.X(PT), BallP.Y*BallAttraction.Y(PT));
    If alignsOnDefenseLine(PT) then SP = adjustToDefenseLine(SP, BallP)
    If behindBall(PT) then SP = adjustToBehindBall(SP, BallP)
    If offsidePosition(SP) then SP = adjustToOnside(SP)
    If not inside admissibleRectangle(PT) then
        SP = adjustToAdmissibleRectangle(SP, admissibleRectangle(PT))
    return SP;
}

```

在建立了SBSP之后，球队就可以像真实的足球队一样，合理的在场上保持一个合理的分布。

球员类型的差异

真实的足球比赛中，球队是千差万别的，这些差异包括球员不同的素质、不同角色分配等等。RoboCup Soccer Server提供了球员的异构，不同的球员类型意味着不同的基本身体素质，如最大速度、加速度、踢球范围等等，这使得球队可以根据不同的需要来调整球员配置。FC Portugal球队借用了球员类型的概念，在其球员类型的概念中，进一步包括了球员的策略特征，分为三类strategic、ball possession和ball recovery。

Strategic特征是SBSP跑位的参数，不同的球员具有不同的参数。这一特征决定了球员活动的范围，有些球员总是在后场活动，而有些则活动在前场。同时不同的参数决定了球员的倾向，有些更积极参与防守，有些更多在可能得分的位置活动。

Ball possession特征决定了球员控球时的行动。这些参数决定了球员射门、传球、带球等等的倾向性。有些球员可能更优先考虑保持球权，有些则总是尝试突破对方防线得分。

Ball recovery特征决定了在主动抢球的情形下球员的行动。这类特征定义了球员做抢球、盯人、拦传球线路等行为的倾向。对于中后卫这类球员可能更多的考虑盯防传球线路，而较少考虑有风险的抢球。

位置与角色的动态交换（Dynamic Positioning and Role Exchange, DPRE）

DPRE是在CMU的Peter Stone工作基础上，对于球队适应性的进一步加强。在FC Portugal球队中，球员不仅可以交换其位置，同时还可以交换他们的球员类型。这种交换是基于一种效用评估，只有当交换可以取得正的收益时，交换才会进行。效用的评估是根据球员到其策略跑位点的距离远近、相关位置的重要性以及相关球员对相关位置的适合程度等因素进行的。

位置的重要性是根据当前阵形及球的位置计算得来。举例而言，一个球队的战略中包括两个阵形（进攻和防守），在防守状态下，中后卫的位置被赋予了非常高的重要性，而在进攻状态下中锋的位置则重要性非常高。

虽然通过直接的协同合作就可以实现DPRE，但是利用通讯可以使交换准确度更高。FC Portugal球队通过实现位置与角色的交换取得了不错的成绩，DPRE对于提高球队的适应性有着明显的帮助。

9.2 行为选择

在RoboCup仿真球队决策中，合理的选择最恰当的行为是最为一个重要的环节。合理的行为选择对于球队的表现有着决定性影响。[S.Russell & P.Norvig, 2002]

在仿真机器人足球队中，球员决策的基本结构如图9.2.1，球员Agent的实现一般都将Server获得的信息转换为内部状态表示的世界模型，在行动选择模块确定具体行动后，动作执行模块将动作细节发送给Server，改变外部环境。

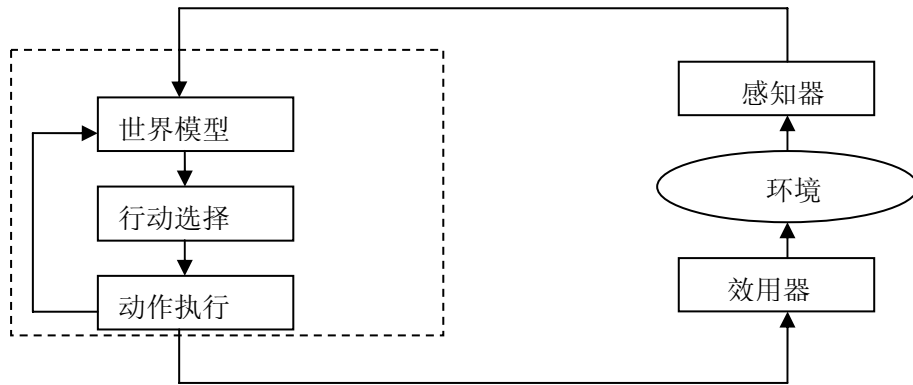


图9.2.1 RoboCup球员决策基本结构

行为选择的实现方法主要有基于规则的反应式方法、基于目标的决策、基于效用的决策方法以及各种方法的混合应用等。本节主要介绍基本的三种方法。

基于规则的选择策略

基于规则的行动选择结构如图9.2.2，球员根据预先设计者给定的条件-行动规则，匹配当前场景合适的规则，选择相应行动。比如预先给定规则：

```
If(BallKickable() && ShootDistance() < 20 && ShootAngle() > 30) then SelectAction(Shoot);
```

则在球员可以踢到球，且离球门距离小于20米，射门角度大于30度时，选择射门动作。

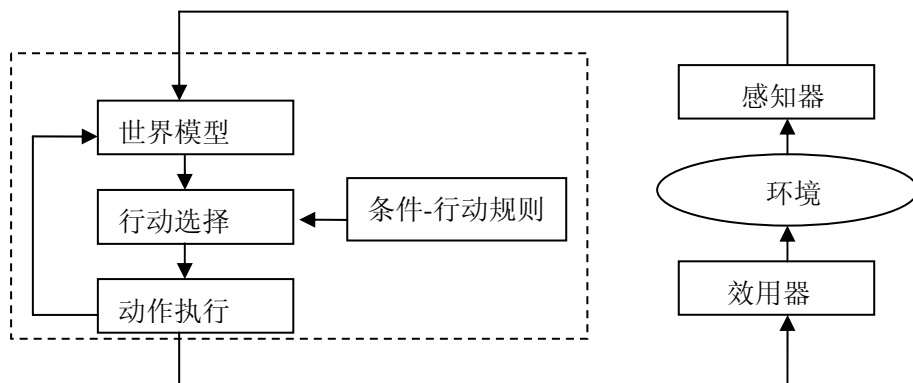


图9.2.2 条件-行动规则决策基本结构

对于简单的行动选择，可以给出一个简化的行为选择实现如下：

```

Void SelectionMethod()
{
    //控球的情形
    If(BallKickable())
    {
        If(Shootable()) SelectAction(Shoot);
        Else if(PassableToForward()) SelectAction(PassToForward);
        Else if(PassableToMidfield()) SelectAction(PassToMidfield);
        Else if(PassableToDefender()) SelectAction(PassToDefender);
        Else if(Dribblable()) SelectAction(Dribble);
        Else SelectAction(Clear);
    }
}

```

```

}
//拦截球的情形
Else if(BallInterceptable()) SelectAcion(Intercept);
//跑位的情形
Else SeclectAction(Position);
Return;
}

```

上面的实现仅仅只是一个十分简单的策略，实际比赛中的球队往往采取了较为复杂的策略，采用基于规则的行动选择机制，就需要极大的扩充if-then规则的数量，对更多的细节加以刻画。毫无疑问，基于规则的行动选择十分简单，执行速度也很快，但是直接的建立规则十分复杂，规则数量及其庞大。

基于目标的行动选择

在决策中，引入目标通过一定的推理来进行行动选择，这将使球员的行动更理性。目标的达成往往需要一个行动序列的执行，球员的行动将更连贯。在仿真机器人足球中，比赛的最终目标是进球和防止对方进球，球员运行时执行的所有目标都是最终目标的子目标。运行中，通过对更长远目标的分解，可以获得当前可以达成的目标。基于目标的决策结构如图9.2.3，在运行中，行为选择的依据是行动对于选定目标的达成有帮助。所谓“有帮助”，可能是规划产生行动序列能达成目标，也可能是行动能部分满足目标。举例而言，在机器人足球仿真比赛中，为达成进球目标，可能产生目标下底传中，任务可分解为任务序列传球到边路、带球突破和传中，因此选择向边锋传球这样的行动。

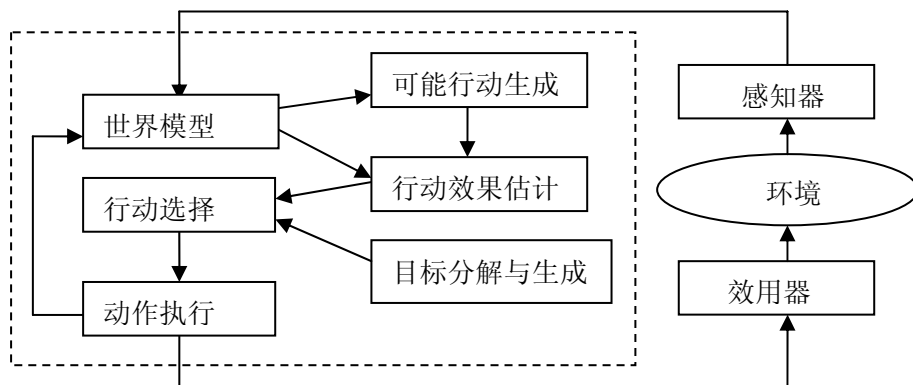


图9.2.3基于目标的决策基本结构

下面给出一个简化的选择算法：

```

Void SelectionMethod()
{
    //产生可能行动
    ActionList al = GeneratePossibleActions();
    //产生目标
    If(!IsGoalPossible(CurrentGoal)) CurrentGoal = GenerateGoal(CurrentState);
    //评估行动效果
    EvaluateAction(al);
}

```

```

//选择对目标CurrentGoal合适的行动
ActionType action = SelectActionForGoal(CurrentState, al, CurrentGoal);
SelectAction(action);
Return;
}

```

这里针对目标通过规划产生行动序列，规划的算法可以多种多样，取决于设计者对于仿真机器人足球中知识的描述。基于目标的决策，连贯性比较好，但是完备的规划效率比较低，要在仿真比赛中实时完成有一定难度，仿真机器人足球环境是动态不确定的多主体环境，其中的逻辑表示和推理也都有较大难度，都是值得进一步研究的课题。

基于效用的选择

基于效用的决策方法，也是Agent决策的基本框架。按照决策论的思想，在决策中，首先根据场上形势，给出当前状态下所有可能的行为，而后分析行为可能的结果状态以及转向这些状态的概率，并对所有的结果状态进行效用评价。选取期望效用最大的策略，依照下面的公式：

$$\max \sum p_{ij} * \mu_j \rightarrow b_{best}$$

公式中*i*、*j*表示不同状态， p_{ij} 为状态*i*到*j*的期望概率， μ_j 为状态*j*期望效用。

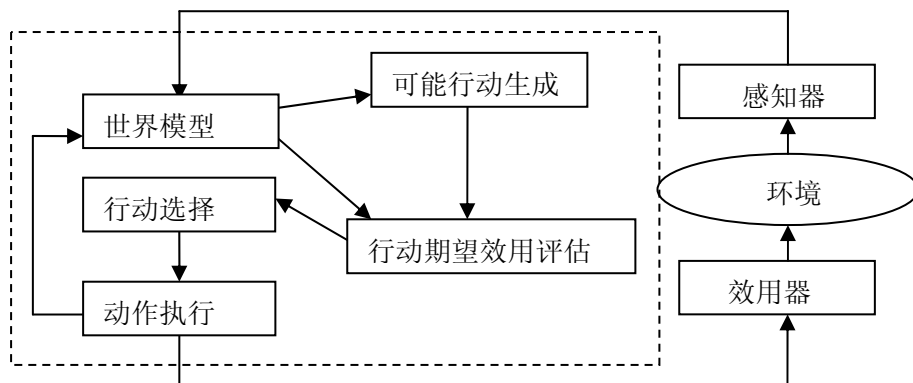


图9.2.4 基于效用的决策基本结构

基于效用的决策框架如图9.2.4，期望效用的评估通常包括两个部分，一部分是对于状态转移概率的评估，一部分是对于达成状态效用的评估。

状态的表示

在RoboCup仿真比赛的环境中，每一个周期的场上状态如果完整表达的话，需要把场上22个球员和球的位置、速度等相关信息都表示出来，这将复杂化分析的困难，而且，实际上不是所有的信息对于特定的分析都有关。因此，有必要对于状态进行简化。

在状态中对于决策而言最关键的内容，是控球者的相关信息，简化后的状态表示中，核心的部分是控球者和控球位置。场上形势的变化就可以简化为下面的表示：

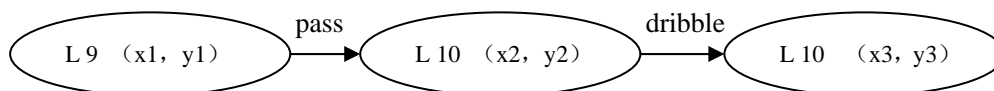


图9.2.5 简化的状态表示

对于基本的形势分析而言，上面的状态表达已经提供了足够的信息，但对于决策仍然是不充分的。在此基础上，需要增加决策相关的队友和对手球员的详细信息。由于，RoboCup 仿真环境中信息采集有较大的局限性，每一个球员根据其视角的不同，只能看到其视野内的信息，而且并非每一个周期都会得到视觉消息。因此球员信息通常都不能保证及时取得。在具体的决策中，相关联的球员信息通常都要根据不同的需要采取各种方法加以估计，以提高决策的正确性。

转移概率的计算

在实际中，由当前状态经过某一行行为，可能实现的状态是多样的。如图：

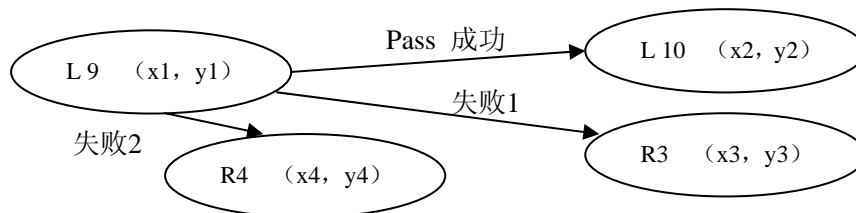


图9.2.6 状态转移

左边9号向10号的传球行为，可能的结果有多种，可能成功传给10号，也有可能被对方的球员拦截。按照决策论的思想，需要对于可能的行为如传球，加以分析，考虑其可能结果并分析达成概率。

但是显然，没有办法计算所有可能的不同状态的达成概率，也不能给出行为完成时所有球员的位置，甚至是行为完成后控球者的位置也不能准确给出。但是在上面提到的简化的状态中，进一步模糊化控球者位置这一属性后，行为可能的结果状态就相当少了，概率分析就成为可能。对于行为结果的概率分析，实际上是在特定条件下，不同球员的拦截球分析。

状态效用的评估

状态的评价对于球队的效果而言是至关重要的。在RoboCup仿真比赛中，对于Agent而言，其目标就是进球与阻止对方进球，状态评价实际上是提供了球场上不同状态对于此目标达成的帮助大小的量化表达。

在一般的考虑下，对于本方控球的状态，状态评价取正值，对方控球取负值。评价值的取值，应该由当前状态到达入球或被入球状态的“距离”决定。在无限前瞻周期的情况下，对于任何状态，都可以进行比较准确的评价，但在实际中前瞻周期受到较大的限制，只能利用特定的评价函数来进行评价，本身带有比较一定的主观性。

基于效用的决策

下面给出一个简化的选择算法：

```
Void SelectionMethod()  
{  
    //产生可能行动  
    ActionList al = GeneratePossibleActions();  
    //评估行动达成概率  
    EvaluatePossibilityOfAction(al);  
    //评估行动达成后效用  
    EvaluateUtilityOfAction(al);  
    //评估期望效用  
    EvaluateExpectedUtilityOfAction(al);  
    //选取最大期望效用行为执行  
    ActionType action = GetMaxUtilityOfAction(al);  
    SelectAction(action);  
    Return;  
}
```

基于效用的行为选择，实际上难点在于状态转移概率的计算和状态效用的评估。对于RoboCup仿真决策这样复杂的问题，效用评估和概率计算更多依赖于设计者对于问题的理解和经验。当然对于决策中一些局部问题可以通过学习获得概率与效用，但整体而言，概率与效用的评估仍然是十分复杂的问题。

9.3 防守策略

在Robocup仿真球队的决策中，防守是十分重要的一环，防守的好坏很大程度上决定了球队的比赛效果。RoboCup仿真组的比赛中，强队一定都有不错的防守。防守的决策与一般

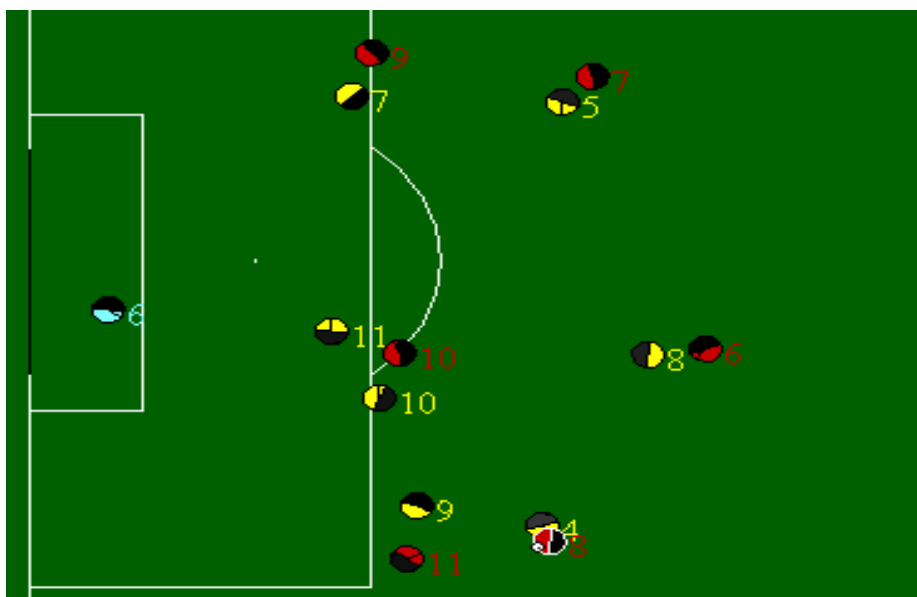


图9.3.1 协调一致的防守

行为决策有所不同，相较于控球行为着重考虑个体对于环境的影响，防守更关注对手可能得行动，同时防守效果更多地依靠所有防守球员的协调一致。

如图9.3.1，黄色一方的防守球员对于红方进攻球员做了全局的看防，在防守中协调一致的防守能够阻断对手可能的推进，将风险降至最低。本节主要介绍防守策略一般的实现方法。

防守体系的一般想法

在很多情况下，球员要能够根据场上的形势，选择符合全队最佳利益的行为。RoboCup仿真机器人足球是一个分布式的系统，就是说不存在一个全局的指挥官，所以每个队员必须独立的进行决策，同时球员与球员之间的通讯是受到严格限制的，并且是不可靠的。在这种情况下，如何协调队员的行为，使他们能够合理的选择自己的任务，并避免与同伴执行重复的行为并且减少相互之间的干扰，使全队的行动一致，形成一个整体来防守，这是一个非常关键的问题。

防守的基本行为

防守体系的建立，首先离不开球员的基本防守行为。在防守中，球员的行为包括如下的几种基本行为：

1. 截球（GetBall）：就是预计球的未来位置并加以拦截。
2. 拦截（Block）：阻止对方控球队员，使其难于向前推进。
3. 抢断（Tackle）：从对方控球队员脚下抢下或者铲断球。
4. 盯人（Mark）：盯防对方可能造成威胁的无球队员。
5. 盯线路（MarkLine）：盯防对方可能传球的线路。
6. 协防（AssistDefense）：辅助本方其他球员进行防守。
7. 跑基本位置（Formation）：跑向个人球员的基本阵型位置。

当然在设计中可以包括更多的基本防守行为，这里仅列出了上面的几种。可以发现，拦截球的行为我们给出了三种，这是因为，足球比赛中对于球的争夺是关键，细分争夺球的行为，对于问题的求解有很大的好处。GetBall行为，实际上发生在球实际上没有被对方控制是，Tackle行为是对于对方控球者的积极抢断行为，Block则是相对消极的做法。

防守行为的效果评价

选择合适的防守行为，必须对于防守行为的效果进行评估。防守行为评估取决于防守对象的重要程度、防守行为达成可能以及防守的代价三个因素。

防守对象的重要程度实际上由其位置决定，综合考虑防守对象当前的位置以及未来可能的控球位置，可以给出适当的评估。根据人类足球的知识，位置越靠近球门、射门角度越大的球员越有威胁性，同时越是无人看防的球员威胁性越大，综合几个因素，调整权重，可以给出所有对方球员的重要程度评价价值。

防守行为达成可能性，是对于球员的防

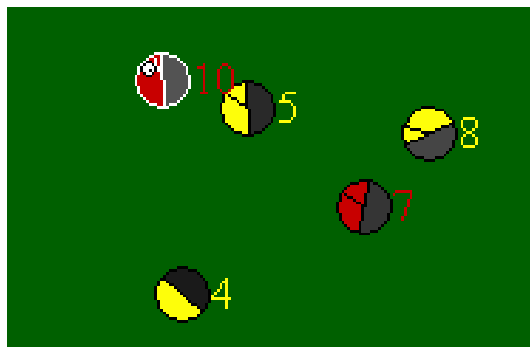


图9.3.2 拦截行为

守行为执行能力的评价。如Block行为，如果Block对象位置较远，可能防守球员根本无法实现拦截的目的，如图9.3.2，图中黄色5号对于红方进攻球员的拦截就难于实现。防守行为达成可能性的评估，需要对于具体的防守行为根据场上实际的位置关系具体计算。

前面我们曾经介绍过阵型与跑位的实现，在真实的足球比赛中，阵型的概念是非常重要的，阵型对保持攻防队形、进行攻防转换都有非常大的作用。真实的足球比赛中，每个队员的活动区域是相对稳定的，每个队员根据场上的情况不同，适时调整自己的位置，但一般而言变化的幅度都不大，在防守中保持整体阵型，对于形成整体的防守是十分重要的。但是在防守行为的执行中，通常都会对于阵型带来一定的破坏，如主动的抢断可能使阵型紊乱。这种对于阵型的偏离就是防守行为执行的代价。此外，防守行为如果需要大范围的跑动，对于球员体力的影响也是需要考虑的代价。

综合几种因素，可以对于防守球员的防守行为进行评估，防守行为的选择建立在这些评估值基础之上。

防守行为的选择

对于防守行为评估之后，每个球员可以根据评估值的大小选择评价最大的行为执行。但这样一来，多个球员间很容易出现冲突。如图9.3.3，黄色的防守方球员5号和9号有可能都选择了对于对方10号的Block动作。

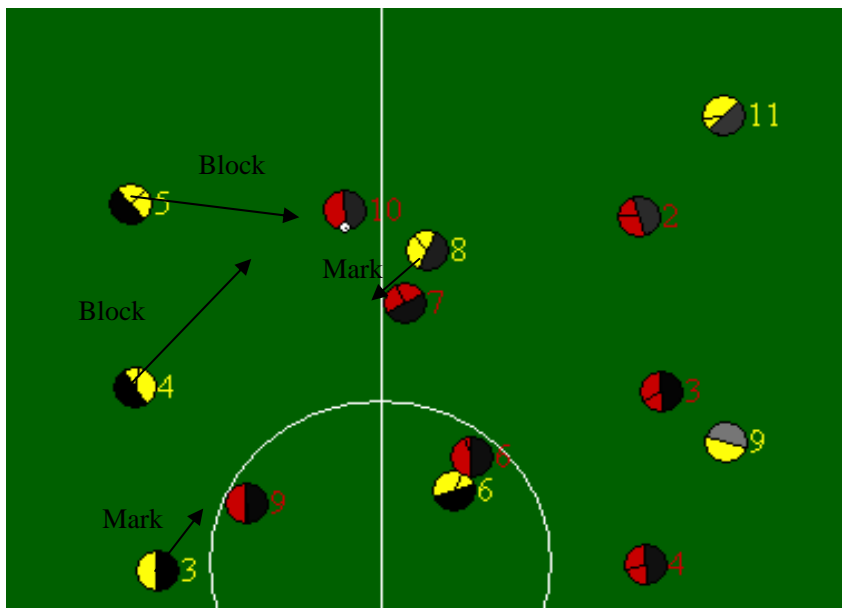


图9.3.3防守行为的冲突

在防守中，如果每一个球员从个体角度出发，选择对于自身有利的行为，显然会发生这种情况。为了解决这个问题，球员需要从全局的角度考虑，使全队的防守行为评估和最大。每个球员利用自身观察得到的信息，计算全队的防守行为安排使得全队防守评估和最大，自己则选择全队行为安排中自身的相应行为。

防守安排的简化算法如下：

/*预先定义

```
vector< vector<DFTask> > tasks;
```

```
bool covered[12];
```

```
vector<DFTask> scheme;
```

```
vector<DFTask> OptimalScheme;
```

WrightEagle

```
vector<DFTask> ResultScheme;

float      MaxIncome;

*/

void Plan()
{
    //产生所有任务
    tasks.clear();
    for (unsigned int i = 0; i < analyser.teammate.size(); i++)
    {
        vector<DFTask> arr;
        DFTask t;
        for (unsigned int j = 0; j < analyser.opponent.size(); j++)
        {
            if (GenerateTask(analyser.teammate[i], analyser.opponent[j], t))
                arr.push_back(t);
        }
        if (GenerateTask(analyser.teammate[i], Unum_Unknown, t)) arr.push_back(t);
        tasks.push_back(arr);
    }
    //产生全队安排
    memset(covered, false, sizeof(covered));
    ResultScheme.clear();
    scheme.clear();
    MaxIncome = -1.0;
    AllCovered = false;
    GenerateScheme(0, 0.0);
    OptimalScheme = ResultScheme;
    return;
}

//产生任务的函数，仅考虑了Formation,Block,Mark三种行为
bool GenerateTask(Unum teammate, Unum opponent, DFTask& t)
{
    if (opponent==Unum_Unknown){
        t.teammate = teammate;
        t.opponent = opponent;
        t.type = DF_Formation;
        return EvaluateFormation(teammate, opponent, t.income, t.pos);
    }
    else{
        t.teammate = teammate;
        t.opponent = opponent;

        if (-analyser.BallTracer.controller == opponent){
            t.type = DF_Block;
        }
    }
}
```

```

        return EvaluateBlock(teammate, opponent, t.income, t.pos);
    }
    else{
        t.type = DF_Mark;
        return EvaluateMark(teammate, opponent, t.income, t.pos);
    }
}

return true;
}

//全队安排的生成
void GenerateScheme(unsigned int t, float SumIncome)
{
    if (t < tasks.size()){
        for (unsigned int i = 0; i < tasks[t].size(); i++){
            if (tasks[t][i].opponent != Unum_Unknown){
                if (covered[tasks[t][i].opponent]) continue;
                covered[tasks[t][i].opponent] = true;
            }
            scheme.push_back(tasks[t][i]);
            GenerateScheme(t + 1, SumIncome + tasks[t][i].income);
            scheme.pop_back();
            covered[tasks[t][i].opponent] = false;
        }
    }
    else {
        if (SumIncome > MaxIncome) {
            ResultScheme = scheme;
            MaxIncome = SumIncome;
        }
    }
}

```

上面的动态规划算法，仅仅考虑了Mark、Block、Formation三种基本动作，简单实现了防守任务的全局最优分配。在实际的RoboCup仿真球队比赛中，由于每一个球员所观察的信息不尽然相同，因此防守中仍然可能发生冲突。通过调整防守行为的执行与评估，可以在一定程度上增强系统的健壮性，尽量减少由于局部信息不一致而引起整体安排发生较大变化。此外通过通讯进行协调，也可以减少防守任务分配的不一致。

9.4 团队合作

在RoboCup仿真机器人足球中，除了前面介绍的防守需要团队的协调外，在进攻中，通常也需要几个队员共同的合作。这一类的合作与防守中的合作有所不同，进攻中控球者相比其他球员，对于球的变化有直接的影响力，也就决定了合作的走向，而防守中，所有合作者处于同等的地位；此外，进攻中的合作，对于球员间一致性的要求较高，合作参与者需要同

时动作，而防守时，合作者略有延迟，对于实际效果影响并不大。本节将介绍，进攻中球员合作的实现。

战术的表示

在人类足球中，有许多多人配合的战术，如：二过一、撞墙配合、下底传中等等，在仿真机器人球队的实现中，也可以借鉴人类足球知识，利用战术来进行有效的团队合作。

多人配合的战术，是多个Agent的行动序列，战术是对于已经知道的较好行动序列的总结。战术执行的关键，在于执行过程中相关条件的描述。球员必须知道战术在何时适用，何时战术终止，战术最终期望达到的效果怎样，战术的过程如何，那些球员参与战术等。战术的表示需要包括如下内容

- a) **战术的参与者。**战术参与者可能是确定的球员，如规定参与者号码；也可以是符合某一条件的球员，这种情况下，执行时需要动态确定参与战术的球员，有时一个战术可能会有多种参与者组合情况，如二过一配合，可以使前锋与前锋配合也可以是前锋与后卫配合等。
- b) **战术执行条件**，其中包括两类条件，一类是触发条件，只有满足触发条件才可能执行战术；一类是持续条件，对于开始执行的战术，只有满足持续条件才可能继续执行。在任何决策周期都只有两类前提条件之一发生作用，但是满足战术执行条件并不一定执行战术，是否执行战术，取决于决策的最终结果。
- c) **战术过程与结果**，描述战术的过程和结果，可以使用前面提过的状态简化表示，但战术执行的中间步骤与结果，通常都不确指一个状态，而对应于一个状态集合，如：战术的最终结果状态可能要求本方球员控球且有一定的射门角度，而并不限定在某一个特定位置，甚至于不限定是哪一个球员拿球。因此在这里，战术经历状态的表示需要因子化，是一种抽象状态的表示。
- d) **战术的动作描述。**由于战术执行的过程并非确定，而仅仅需要满足某些约定，因此战术执行时具体的动作并不确定，如战术规定的传球可能会根据具体的情况在参数上有较大的区别。另外，战术执行的过程也并非确定，如图 9.4.1，战术过程 S1-S2-S3 执行中，S1-S2 有两个动作可以执行 dribble 和 Pass，dribble 动作并没有直接推进战术，但战术可能仍然满足持续条件。在足球比赛中，经常会出现这种情况，球员需要采取 dribble 这类行动来等待和创造机会来达成战术意图。何时 pass，何时 dribble，取决于球员在具体情况下，对于行为结果的判断。

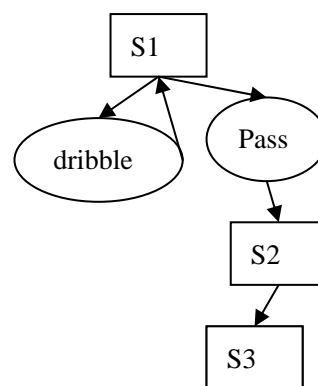


图9.4.1战术的过程

一个简单的例子

二过一是常见的战术，可以简单做如下定义：

触发条件：

$\text{IsController}(S1, x) \wedge \text{IsController}(s, x) \wedge \text{BallInArea}(s, \text{Area})$

表明触发时的控球者和第一个状态定义的控球者是同一个人，且球位于区域Area内

持续条件：

$\text{IsController}(s, x) \wedge \text{IsActionMember}(x)$

表明控球者始终是当前的行为相关者

战术过程：S1-S2-S3，

三个状态满足如下约束：

$$\text{IsController}(S1, x) \wedge \text{IsController}(S3, x) \wedge \neg \text{IsController}(S2, x) \\ \wedge \text{IsController}(S2, y) \wedge \text{DistanceLessThan}(s, x, y, 15)$$

表明状态S1、S3的控球者都是球员x，状态S2控球者为y，x和y不是同一个人，且x和y间距离小于15米。

战术的行为策略：如图示

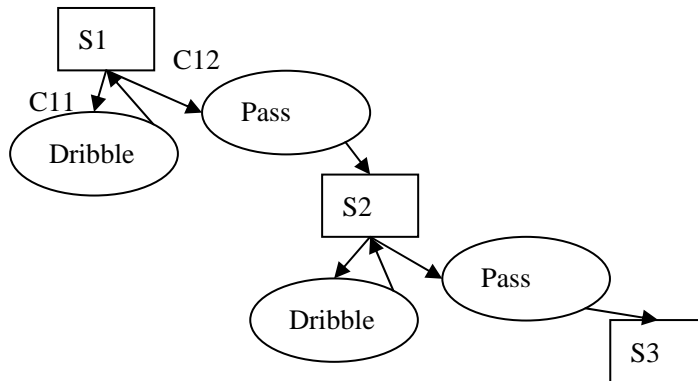


图9.4.2 二过一战术

图中C11表示

$$\neg \text{Passable}(S1, x, y) \wedge \text{Dribblable}(S1, x) \wedge \text{IsController}(S1, x) \wedge \text{IsController}(S2, y)$$

也就是说状态S1的控球者x，在状态S1下可以带球，但无法将球传给S2的控球者y

C12表示

$$\text{Passable}(S1, x, y) \wedge \text{IsController}(S1, x) \wedge \text{IsController}(S2, y)$$

表示状态S1的控球者x，在状态S1下可以将球传给S2的控球者y

上面的表示中，s、x、y均为变元，将当前状态S带入其中，给其中变元x、y赋值，这里x、y是我方球员。当x、y有多组不同取值满足条件，则此战术有多种实现方式。如，当前控球者是9号，10号、11号都满足二过一的实现条件，则有与10号配合、与11号配合两种二过一模式的战术执行。

战术系统的实现

战术系统的实现，需要预先定义可能的所有战术，在执行中，根据具体情况，对于预先定义的战术原形实例化，选定具体战术后，根据战术过程的描述，解释产生具体行动。战术系统的基本框架如下：

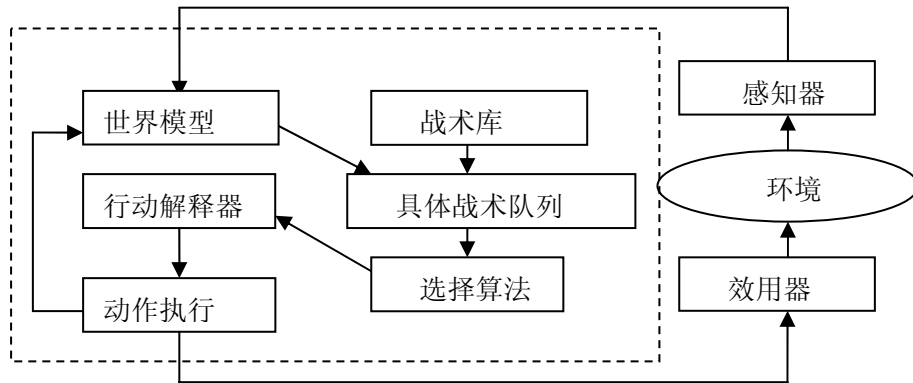


图9.4.3 战术系统的基本框架

由抽象战术原形声称具体战术的算法如下：

```

void GenerateTactics(list<Tactics>& taclist, Tactics TacticsOfProto[])
{
    for(int i = 0; i < NumOfTactics; i++)
    {
        if (!TacticsOfProto[i].StartCondition()) continue;
        TacticsOfProto[i].AssignPlans()
        for(int i = 0 ; i < TacticsOfProto[i].NumOfPlans; i++)
        {
            Tactics tac = TacticsOfProto[i].clone(i);
            if(!IsRelatedToMe(tac)) continue;
            taclist.push_back(tac);
        }
    }
    return;
}

```

在战术执行过程中，每一个周期都会判断战术是否满足持续条件，如果满足，则由决策进行评估决定是否继续执行。

基于效用的决策与战术库结合

在9.2节，我们介绍了行动选择的基本方法，战术库的思想实际上是一种基于目标的决策和反应式规则的结合。在RoboCup仿真球队的实现中，更多的是采用基于效用的决策方法，可以将战术也纳入到效用评估中，仍然利用期望效用进行决策。

系统可以看作是四元组 $\langle S, A, T, E \rangle$

S: 状态集

A: 行为集，包括战术库中行为，可以是计划，不是环境中最基本的操作。

T: $S \times A \rightarrow \Pi(S)$ 是状态转移函数。 $T(s,a,s')$ 表示状态s下执行行为a到达状态s'的概率。

$E: S \rightarrow R$ 是对状态的评价函数，达成状态 S 对于总目标的帮助大小的度量。 $E(S)$ 的输入不需要完整的状态表示，可以是 S 的简化表示。

与基于效用的MDP框架不同，这里将行为集合扩充为可以是计划，可以是一般行动。具体实现上，一般行为可以直接执行，而战术一类的抽象行为，需要由解释器解释产生环境中可执行的具体行动。

行为集合 A 中定义的行为，都不是RoboCup仿真平台所提供的最基本的行为。如图9.4.4， A 中的行为可以分为三种：

1. Action是对于环境中基本操作的组合。
2. Combo-Action是在考虑对抗的环境中对Action的组合。
3. Tactics是战术行为，是多个球员Combo-Action的组合。

在RoboCup仿真足球中，多脚的踢球就是对于环境中基本操作一脚踢球的组合，是这里定义的Action；在考虑对方球员的干扰影响后，合理的利用多脚踢球及其他Action避免传球被对手拦截，这样的传球动作也就是Combo-Action；在这些传球之上，可以形成的二过一战术配合，这是Combo-Action构成的Tactics。所有这些行为，都可以作为行为集合 A 中的元素。

行为集合 A 中的行为评估都采取了统一的方法。

当前状态 s 下执行行为 a 的评估值为：

$$V(s, a) = E(B) = \sum_{s_i \in S} b(s_i) E(s_i),$$

B 为 s 下执行 a 后的信念分布，

$$b(s) \in B, \quad b(s_i) = T(s, a, s_i)$$

依照公式上面的公式，计算各个行为的评价值 V ，选取评价值最高的行为执行。对于函数 $T(s, a, s_i)$ 和 $E(s)$ 的计算仍然是基于效用决策的难点，由于战术行为的达成有较大的延迟，因此对其转移概率与效用的评价更加困难。实现中，可以考虑利用战术本身定义的战术执行过程，仅考虑战术顺利执行和中途停止的情况，简化问题。

在不考虑战术的效用评价中，决策完全依赖于效用评价，由于效用评估函数的局限性，通常只能对于行为的短期结果做出评估，而难以体现长远利益。如：禁区前沿直接对禁区内前锋的传球评估往往高于通过传接的过渡行为，但很多情况下，直接的传接所形成的威胁要远小于过渡后的进一步的传接。通过引入战术，能将人类足球的知识应用到球员的实现中去，通过战术，来评估，可能带来较好长远利益的行为。

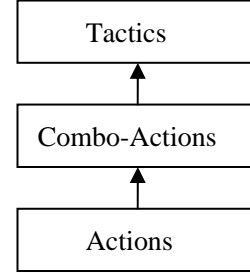


图9.4.4 行为的分层

多Agent的协调一致

在球员的合作中，多个Agent能够步调一致的采取行动是十分重要的。在[Jelle Kok,2003]中，提供了一种基于协作图（Coordination graph, CG）的方法来解决这个问题。

定义 G_1, \dots, G_n 为合作中的 n 个Agent，Agent G_i 选择行动基 A_i 中行动 a_i ， X 为抽象状态集合，定义文字 c 为所有可能的状态语动作组合， $c \in C \subseteq X \cup A$

我们可以定义规则 $\langle p; c: v \rangle$ ，函数 $p: C \rightarrow R$ ， $p(x, a) = v$ 表示在 x 状态下，Agent采取行动 a ，获得的评估为 v 。

对于多个Agent协同动作的求解，实际上就是求解动作组，使得评估值之和最大。求解

的算法是计算一个Agent的行为，使相关的Agent取得最大的可能评估，然后从图中去掉计算过的Agent更新协作图，计算下一个Agent。

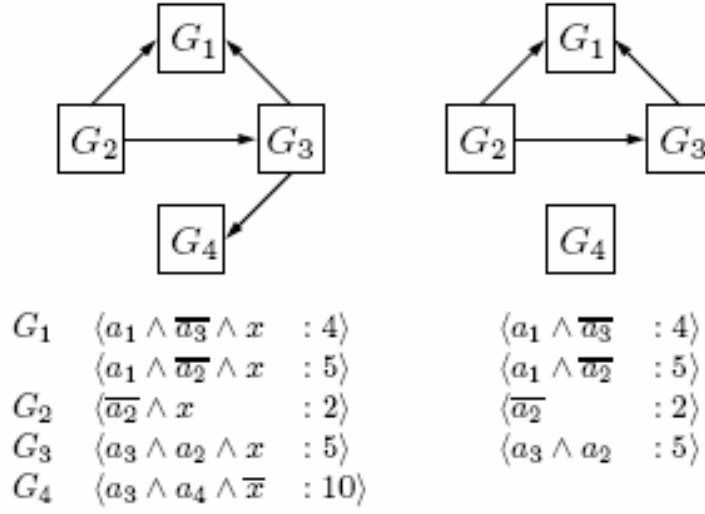


图9.4.5 初始协作图（左）和 $x = \text{true}$ 时的协作图（右）

如图9.4.5，为一个有四个Agent的协作图，其相互作用关系如左图，当状态 $x = \text{true}$ 时，可简化为右图，此时G4没有任何作用，可以去除。考虑消去G3，则对于所有G1、G2的行动，我们可以得到 $\langle a_2 : 5 \rangle, \langle a_1 \wedge \bar{a}_2 : 6 \rangle$ ，将此规则更新到G3的父节点G2，此时G3可以消去，同样消去G2，得到条件策略 $\langle a_1 : 11 \rangle, \langle \bar{a}_1 : 5 \rangle$ ，此规则更新到G1，此时G1是唯一一节点，因此可得G1选择行动 a_1 ，反向推倒，我们可以得到最佳的行动为 $\langle a_1, \bar{a}_2, \bar{a}_3 \rangle$ 。

在RoboCup合作中，我们首先定义如下行动

- $\text{passTo}(i; \text{dir})$: 传球给队员 i 的 dir 方向， dir 取值范围 $D = \{\text{center}; \text{n}; \text{nw}; \text{w}; \text{sw}; \text{s}; \text{se}; \text{e}; \text{ne}\}$ 。
- $\text{moveTo}(\text{dir})$: 向 dir 方向移动。
- $\text{dribble}(\text{dir})$: 向 dir 方向带球。
- score : 射门。
- clearBall : 向对方方向无人能控球的地方将球踢出。
- moveToStratPos : 返回阵型基本点。

一般的而言，对于场上每一个球员我们都可以定义如下的简单规则：

$$\langle p_1^{\text{passer}}; \text{HasRoleReceiver}(j) \wedge \neg \text{IsPassBlocked}(i, j, \text{dir}) \wedge a_i = \text{passTo}(j, \text{dir}) \wedge$$

$$a_j = \text{moveTo}(\text{dir}) : u(j, \text{dir}) \rangle$$

$$\langle p_2^{\text{passer}}; \text{IsEmptySpace}(i, n) \wedge a_i = \text{dribble}(n) : 30 \rangle$$

$$\langle p_3^{\text{passer}}; a_i = \text{clearBall} : 10 \rangle$$

$$\langle p_4^{\text{passer}}; \text{IsInFrontOfGoal}(i) \wedge a_i = \text{score} : 100 \rangle$$

$\langle p_5^{receiver}; HasRoleInterceptor(j) \wedge \neg IsPassBlocked(j,i,dir) \wedge a_i = moveTo(dir) : u(i,dir) \rangle$

$\langle p_6^{receiver}; HasRolePasser(j) \wedge HasRoleReceiver(k) \wedge \neg IsPassBlocked(k,i,dir) \wedge$

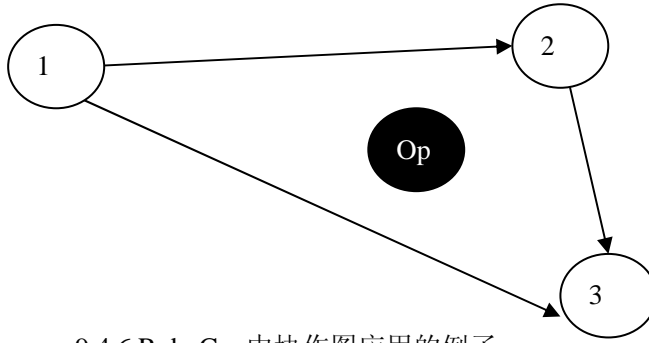
$a_j = passTo(k,dir2) \wedge a_k = moveTo(dir2) \wedge a_i = moveTo(dir) : u(i,dir) \rangle$

$\langle p_7^{receiver}; moveToStartPos : 10 \rangle$

$\langle p_8^{interceptor}; intercept : 100 \rangle$

$\langle p_9^{passive}; moveToStartPos : 10 \rangle$

在某一时刻，场上协作图如图9.4.6



9.4.6 RoboCup中协作图应用的例子

假定场上形势有， $\neg IsPassBlocked(1,2,s)$ 和 $\neg IsPassBlocked(2,3,nw)$ 为真，可以写出三个球员的规则如下：

G1: $\langle p_1^{passer}; a_1 = passTo(2,s) \wedge a_2 = moveTo(s) : 50 \rangle$

$\langle p_2^{passer}; a_1 = dribble(n) : 30 \rangle$

$\langle p_3^{passer}; a_i = clearBall : 10 \rangle$

G2: $\langle p_7^{receiver}; a_2 = moveToStartPos : 10 \rangle$

G3: $\langle p_6^{receiver}; a_1 = passTo(2,dir) \wedge a_2 = moveTo(dir) \wedge a_3 = moveTo(nw) : 30 \rangle$

$\langle p_7^{receiver}; a_3 = moveToStartPos : 10 \rangle$

由前面的方法，可以先消去G1，则条件策略为：

$\langle p_1^{passer}; a_2 = moveTo(s) \wedge a_3 = moveTo(nw) : 80 \rangle$

$\langle p_1^{passer}; a_2 = moveTo(s) \wedge a_3 = \neg moveTo(nw) : 50 \rangle$

$\langle p_1^{passer}; a_2 = \neg moveTo(s) : 30 \rangle$

由此可以得到，最佳的策略为

$a_1 = passTo(2, s), a_2 = moveTo(s), a_3 = moveTo(nw)$

利用这样的方法，球员之间可以通过约定达成一致，Jelle的实验也证明这一方法相当有效。当然，对于同步球员行动，还可以采用通讯的方式，也会对于球队一致性的提高有一定帮助。

参考文献

[Peter Stone Thesis,1998] Peter Stone. *Layered Learning in Multi-Agent Systems* [D]. PhD thesis, Carnegie Mellon University, December 1998

[Reis & Lau, 2000] Luis Paulo Reis and Nuno Lau, *FC Portugal Team Description: RoboCup 2000 Simulation League Champion*, RoboCup-2000: Robot Soccer World Cup IV, Peter Stone, Tucker Balch and Gerhard Kraetzschmar editors, LNAI 2019, 29-40, Springer Verlag, Berlin, 2001

[S.Russell & P.Norvig, 2002] S. Russell & P. Norvig. *Artificial Intelligence: A modern approach* 北京:人民邮电出版社, 2002

[Jelle Kok,2003] Jelle R. Kok, Matthijs T. J. Spaan, and Nikos Vlassis. *Multi-robot decision making using coordination graphs*. In Proceedings of the 11th International Conference on Advanced Robotics, ICAR'03, pp. 1124–1129, Coimbra, Portugal, June 2003.

第十一章、机器人足球中的学习

11.1 决策树学习

决策树学习是一种基于符号的机器学习方法，也是一种有指导的学习方法。决策树是一个类似流程图的树结构，其中每个内部节点表示对某个属性的测试，每个分支代表属性的一个取值，每个叶子代表类别或决策结果。这样从根到叶的不同路径就代表的不同的决策规则。如图 11.1.1 是一棵判断大学生是否可以买电脑的决策树，内部节点用矩形表示，叶节点用椭圆表示。

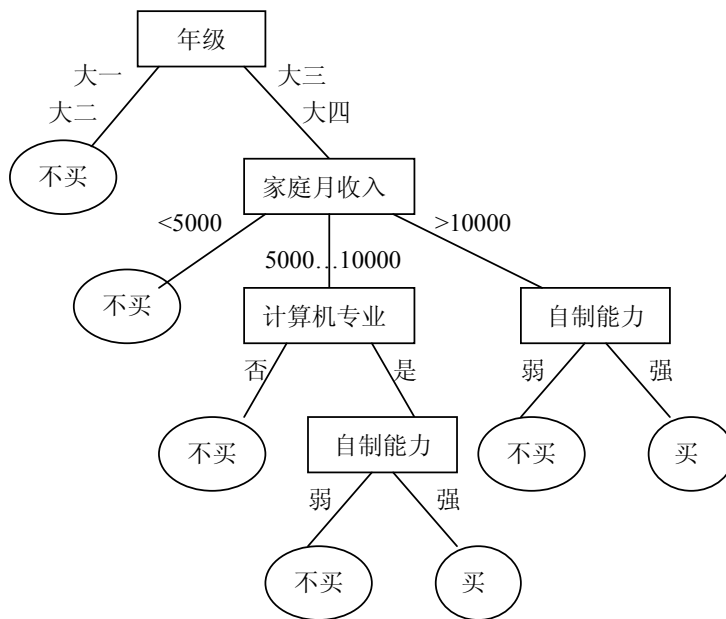


图 11.1.1: 大学生是否可以买电脑的决策树

11.1.1 决策树归纳

[最近的决策树看代码，还有看树的生成](#)

上面买电脑的决策树是由人手工给定的，对于很多复杂的实际问题，很难手工直接给出决策树。从大量的训练数据归纳构造出决策树的过程就是决策树学习。如表 11.1.1 给出了买电脑的部分训练数据，其中决策结果只有两类：买和不买。

表 11.1.1: 大学生是否可以买电脑的训练数据

编号	是否买电脑	年级	家庭月收入	自制能力	计算机专业
1	不买	二	8000	强	是

2	买	三	6000	强	是
3	不买	一	7500	弱	否
4	不买	三	3000	强	是
5	买	四	12000	强	否
6	不买	二	7000	弱	是

.....

决策树归纳的基本算法是贪心算法，自顶向下递归的构造决策树。一个著名的判定树归纳方法是 ID3 算法（Quinlan 1986），它试图生成对所有给定实例正确分类的最小决策树，即满足 Occam 1324 年提出的剃刀原理：少做可以，多则徒劳；如无必要，勿增其繁。这样，决策树中就有可能不出现实例中一些不必要的属性。图 11.1.2 给出了 ID3 算法，需要注意的是，如果要从上面表 11.1.1 中的实例归纳构造出图 11.1.1 那样的决策树，需要事先对实例中的连续属性进行处理，如把收入一项先分为三段离散值：小于 5000、5000 到 10000、10000 以上。ID3 的改进算法 C4.5（Quinlan 1996）则可以直接处理连续属性，在归纳过程中把它们分割成不同的区间。

输入： 训练实例集 S ，候选属性集 A

输出： 决策树

- 1 创建节点 n
- 2 如果 S 中的所有实例属于同一类，把 n 作为叶节点并以该类标记，返回节点 n
- 3 如果 A 是空集，把 n 作为叶节点并标记为 S 中出现最多的类别，返回节点 n
- 4 根据一定的标准从 A 中选择一个属性 a
- 5 把节点 n 标记为属性 a
- 6 对属性 a 的每个取值 v ，循环
 - 7 从 n 长出一个分支，标记为 v
 - 8 设 S_v 为 S 中属性 a 取值为 v 的实例集合
 - 9 如果 S_v 为空，则为分支 v 加一叶节点，并标记为 S 中出现最多的类别
 - 10 否则以参数 S_v 和 $A-\{a\}$ 递归调用本算法，把得到的决策树作为分支 v 的结果
- 11 返回以节点 n 为根的决策树

图 11.1.2：生成决策树的 ID3 算法

ID3 在所有可能树空间中实现了一个贪心算法，为当前树增加一个子树并继续搜索，但不进行回溯。这样就会非常高效，同时也不依赖于选择测试属性的标准。

11.1.2 属性选择

实例中的每个属性可以看成是为决策树提供的一定量的信息，ID3 选择属性的标准就是能为决策树增加多少信息，即属性的信息增益（Information Gain），具有最高信息增益的属性作为当前节点的测试属性。这样能确保得到一棵简单的决策树，使得决策过程中属性测试次数的期望值最小。

计算属性信息增益的数学基础是 Shannon（1948）的信息论，它形式化地定义了消息含有的信息量。给定一个消息空间 $X=\{x_1, x_2, \dots, x_k\}$ ，从 X 中获得一个消息 x 的不确定性（即信

息熵)是

$$H(x) = -\sum_{i=1}^k p_i \log_2 p_i$$

其中 p_i 是消息 x_i 的出现概率, 对数函数以 2 为底数, 因为消息用二进制编码。在已知条件 y 时得到一个消息 x 的不确定性用 $H(x|y)$ 表示。消息 y 能够减少 x 的不确定性, 这个减少的差值称为消息 y 提供的关于 x 的信息 $I_y(x)$, 即信息为不确定性(熵)的差:

$$I_y(x) = H(x) - H(x|y)$$

特别地, 当 $y=x$ 时, $I_x(x)=H(x) - H(x|x)=H(x)$, 称消息 x 包含的信息 $I(x)$ 的期望值在数值上等于熵 $H(x)$:

$$I(x) = -\sum_{i=1}^k p_i \log_2 p_i$$

可以把训练实例集 $S=\{C_1, C_2, \dots, C_n\}$ 看作一个消息空间, 其中 C_i 表示具有相同分类或决策结果为 c_i 的实例集。假定每个实例以相同的概率出现, 则决策树为 S 分类应包含的期望信息为

$$I(S) = -\sum_{i=1}^n p_i \log_2 p_i, \quad p_i = \frac{|C_i|}{|S|} \quad (11.1.1)$$

设属性 a 有 m 个不同取值 $\{v_1, v_2, \dots, v_m\}$, 根据属性 a 的取值可以把 S 分为子集 $\{S_1, S_2, \dots, S_m\}$, S_j 中实例的属性 a 的值为 v_j 。根据属性 a 进行分类所需信息的期望为

$$I(a) = \sum_{j=1}^m \frac{|S_j|}{S} I(S_j) \quad (11.1.2)$$

如果把属性 a 作为节点的测试属性, 则为决策树增加的信息就是总信息量减去属性 a 所需的信息, 即属性的信息增益为:

$$Gain(a) = I(S) - I(a) \quad (11.1.3)$$

以表 11.1.1 是否可以买电脑的数据为例, 假定只有前面六个训练实例, 且年级和收入已象图 11.1.1 那样离散化。数据中有两类决策结果: 买和不买, 对应分别有二个和四个实例。根据公式 (11.1.1) 可以得到决策树所需的期望信息为:

$$I(\text{买电脑}) = -\frac{2}{6} \log_2 \frac{2}{6} - \frac{4}{6} \log_2 \frac{4}{6} = 0.918$$

考虑属性年级, 根据它的两个取值: 大一大二和大三大四, 可以把实例分为两个子集, 同样根据公式 (11.1.1) 可以得到:

$$\begin{aligned} I(\text{大一大二}) &= -\frac{3}{3} \log_2 \frac{3}{3} = 0 \\ I(\text{大三大四}) &= -\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3} = 0.918 \end{aligned}$$

再根据公式 (11.1.2) 可以得到以年级分类所需的期望信息为

$$I(\text{年级}) = \frac{3}{6} I(\text{大一大二}) + \frac{3}{6} I(\text{大三大四}) = 0.459$$

则根据公式 (11.1.3) 属性年级的信息增益为

$$Gain(\text{年级}) = I(\text{买电脑}) - I(\text{年级}) = 0.918 - 0.459 = 0.459$$

类似可以得到其它几个属性的信息增益：

$$Gain(\text{收入}) = 0.378$$

$$Gain(\text{自制能力}) = 0.251$$

$$Gain(\text{专业}) = 0.044$$

所以在选择根节点时，年级的信息增益最大，ID3 就把年级作为根节点的测试属性。

11.1.3 算法扩展

ID3 算法根据已有实例产生的简单决策树，可以用来预测新实例的决策结果，在实际应用中性能良好，如预测国际象棋最后阶段不同走法的输赢等 (Quinlan 1983)。

然后 ID3 产生的决策树建立在好的训练实例集合上，如每个实例的属性都已赋值等。在真实的应用中，很可能出现下面的问题：

- (1) 出现两个或多个实例的各个属性值都相同但决策结果不同；
- (2) 一些实例的某些属性没有赋值，可能是很难获取这些值；
- (3) 一些属性取值是连续的；
- (4) 训练数据集太大；

这些问题的出现，就有了很多方法来加强和扩展 ID3 的基本决策树归纳，值得一提的是 C4.5 (Quinlan 1996)，采用了很多下面将要提到的归纳加强方法。

可以允许属性具有整个离散空间或连续值，对这种属性 a 的测试会产生两个分支： $a \leq v$ 和 $a > v$ 。 v 的确定是考虑该属性的信息增益，尽量选择使 a 的信息增益最大的分割点 v 。

如果属性值出现空缺，可以用该属性的最常见值替代。同时该属性的信息增益可以按照空缺比例减小。

对不同的实例可以赋不同的权值，用来反映它的重要程度。当权值调整时，就得到不同的决策树，然后再综合不同决策树的决策结果。

处理大数据量问题时，可以把数据划分成子集，对不同子集构造决策树，然后用其它子集的实例测试决策树的准确性，选择准确的决策树或对它们再进行一次综合决策。

11.2 神经网络学习

人工神经网络是基于连接的机器学习方法。人工神经元是对生物神经元的简化和模拟，它是神经网络的基本处理单元。连接主义者认为智能可以在大量简单的相互作用的部件（生物或人工神经元）组成的系统中涌现出来，通过学习和调整神经元之间的连接参数可以提高系统的智能。系统中的各个神经元同时相互独立地处理各自的输入，并行的求解问题。

为了建立一个神经网络，必须用一种编码方法把现实问题中的模式转化成网络中的数字量，神经元之间的连接是一个数字，输出也是对数字操作处理后得到的数字量。处理都是并行和分布式的，没有符号系统中的符号处理。使用神经网络一般使用训练的方法而不是直接的程序设计，这种方法吸引人的地方是：一个网络体系和训练算法经常能够抓住现实世界中的不变量，甚至以奇怪的吸引子方式，而这些都很难由人类或符号系统清楚直接的识别。

11.2.1 人工神经元模型

人工神经元是对生物神经元的简化和模拟，是神经网络的基本处理单元。每个神经元是一个多输入单输出的非线性元件，可以接收一组来自系统中其它神经元传来的输入信号，输出也可以送给多个神经元作为它们的输入。图 11.2.1 是神经元的结构模型，其中 $X = \{x_1, x_2, \dots, x_n\}$ 是输入向量； $W = \{w_1, w_2, \dots, w_n\}$ 是对输入的加权； Σ 表示为输入信号的累积； θ 是神经元的阈值，根据生物神经元的特性，当神经元获得的输入信号的累积效果超过阈值时，它就处于激发态，否则为抑制态； f 是激活函数或称传递函数，是对生物神经元激活方式的一个扩充，希望人工神经元对输入量有更一般的变换方式，使系统有更宽的使用面； y 是该神经元的输出，可以把它作为其它多个神经元的输入。

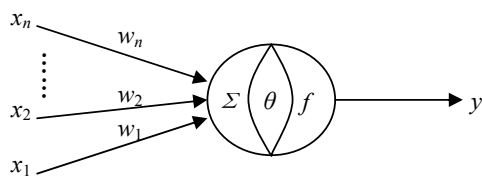


图 11.2.1：神经元模型

神经元的输入输出关系可描述为：

$$u = \sum_{j=1}^n w_j x_j - \theta, \quad y = f(u) \quad (11.2.1)$$

有时为了方便，常把 $-\theta$ 看成恒为 1 的输入 x_0 对应的权值 w_0 ，这时上面的公式变为：

$$u = \sum_{j=0}^n w_j x_j, \quad w_0 = -\theta, x_0 = 1$$

常用的传递函数有线性函数、阶跃函数、非线性斜面函数、S型函数等四种。

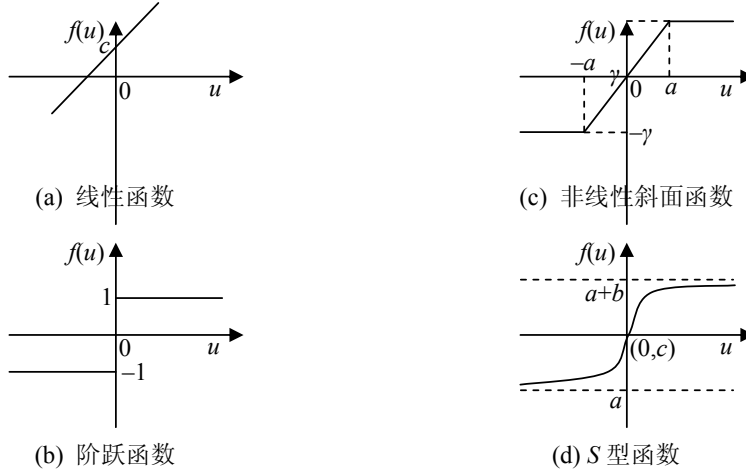


图 11.2.2: 常用传递函数

线性函数是基本的传递函数，它把神经元获得的输入进行适当的线性放大。它的一般形式是

$$f(u) = ku + c \quad (11.2.2)$$

其中 k 为方法系数， c 为位移，均为常数，图 11.2.2(a) 给出了线性函数的图像。

阶跃函数又叫阈值函数，通常有两种形式，分别如公式 (11.2.3) 和 (11.2.4) 所示，图 11.2.2(b) 给出了 (11.2.4) 的图像。

$$f(u) = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0 \end{cases} \quad (11.2.3)$$

$$f(u) = \begin{cases} 1 & u \geq 0 \\ -1 & u < 0 \end{cases} \quad (11.2.4)$$

非线性斜面函数是线性函数和阶跃函数的结合，实际上是一个分段线性函数，把线性函数限制在一个给定的范围 $[-\gamma, \gamma]$ 内：

$$f(u) = \begin{cases} \gamma & u \geq a \\ ku & |u| < a \\ -\gamma & u < -a \end{cases} \quad (11.2.5)$$

图 11.2.2(c) 给出了它的图像。

S型函数又叫压缩函数，是在给定区间内连续取值的单调可微分函数，通常用指数或正切等来表示：

$$f(u) = a + \frac{b}{1 + e^{-du}} \quad (11.2.6)$$

$$f(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}} \quad (11.2.7)$$

函数取值通常限定在 (0,1) 或 (-1,1) 内, 图 11.2.2(d) 给出了 (11.2.6) 的图像。

由多个神经元连接起来的神经网络的学习过程是调整输入权重的过程。神经网络获得一组输入后, 各个神经元并行独立的进行运算, 最终神经网络会得到一个输出, 用教师或其它方法对神经网络的输出进行评估, 根据评估结果来调整各个神经元的输入权重, 重复这样的过程直到评估满意。下面介绍几种神经网络的结构及它们的学习算法。

11.2.2 感知器

感知器是最早实现的人工神经网络, 最早由 McCulloch 和 Pitts (1943) 提出, 并由 Rosenblatt (1958) 设计了感知器学习算法。

简单的感知器就是如图 11.2.1 所示的一个人工神经元, 有是为了表达输入的方便, 在神经元前面加一个输入层, 该层内的节点只是把输入直接输出给后面的神经元, 不对输入进行任何操作, 如图 11.2.3(a) 所示。一个神经元的感知器还可以表示逻辑门, 如 $w_1 = w_2 = 1, \theta = 2$ 表示的与门和 $w_1 = w_2 = 1, \theta = 1$ 表示的或门等。

感知器也可以有多个神经元, 对应着该神经网络也有多个输出, 把这些神经元都称为输出层, 如图 11.2.3(b) 所示。不管是一个神经元还是多个神经元, 它们都只有输出层一层的神经元对所有输入进行操作, 称为单层感知器。

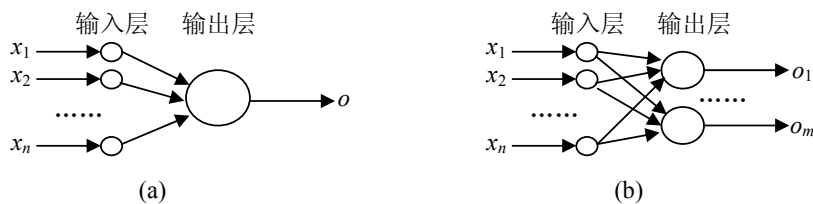


图 12.2.3: 单层感知器

单层感知器的学习是一种有指导的学习, 修改权重的计算公式是:

$$\Delta w_{ij} = \alpha(y_j - o_j)x_i \quad (11.2.8)$$

其中 α 是学习率, x_i 是输入, o_j 是第 j 个神经元的输出, y_j 是期望输出或称教师信号, Δw_{ij} 是第 j 个神经元的第 i 个输入权重的改变量。具体的单层感知器学习算法如图 12.2.4 所示。

- 1 初始化权矩阵 W , 其中 w_{0j} 表示第 j 个神经元的阈值 θ_j
- 2 设置最小误差 ε 和学习率 α
- 3 循环, 直到实际误差 E 满足 $E < \varepsilon$
 - 4 $E \leftarrow 0$
 - 5 对每个样本 (X, Y) , 循环

- 6 输入 $X = (x_0 = 1, x_1, x_2, \dots, x_n)$ ，通过感知器求输出 $O = F(XW)$
- 7 修改权矩阵 W ：对每个 $i (= 0, \dots, n)$ 和 $j (= 1, \dots, m)$

$$w_{ij} \leftarrow w_{ij} + \alpha (y_j - o_j) x_i$$
- 8 计算累积误差： $E \leftarrow E + \sum_{j=1}^m (y_j - o_j)^2$

图 11.2.4：单层感知器学习算法

单层感知器在开始得到很好的发展，但还是无法解决某些问题，如无法实现异或逻辑运算等，Minsky 和 Papert (1969) 证明了这样一个结论：单层感知器无法解决线性不可分问题，异或就是这样的一个问题。而异或运算是电子计算机最基本的运算之一，预示着单层感知器无法解决电子计算机可以解决的大量问题。

为了解决线性不可分问题，后来提出了多层感知器模型，是在输入层和输出层间加入一层或多层的隐层神经元，隐层中的神经元接收前一层的输出作为输入，并把运算结果作为下一层神经元的输入。反传(BP)网络是多层感知器中值得一提的网络结构，在很多领域都有很成功的应用。

11.2.3 反传网络

反传网络及学习算法由 Rumelhart 等 (1986) 提出，很快就在多个领域得到了成功的应用，对神经网络的第二次研究高潮起到了很大的作用。另外它的学习算法有很强的数学基础，有令人信服的权重修改方法。

反传网络结构符合多层感知器模型，有一层或多层的隐层神经元，同时要求相邻两层神经元间才有连接，同层神经元间和不相邻层的神经元间不能有连接，另外要求每个神经元的传递函数是处处可导的，一般用 S 型的传递函数。图 11.2.5 给出了反传网络的结构。

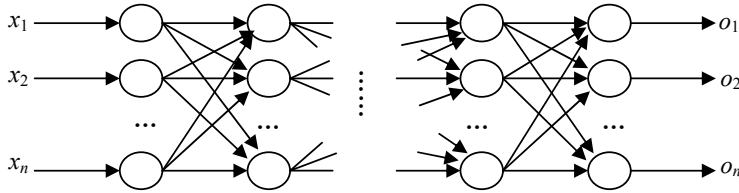


图 11.2.5：反传网络结构

反传网络的学习算法主要分两个阶段：先是从样本集中取出一个样本 (X_p, Y_p) ，将 X_p 输入网络，信息从输入层逐级的变换，最终得到网络的实际输出 O_p ；然后根据实际输出与理想输出 Y_p 的误差，逐级地从输出层到输入层方向用梯度下降的方法调整权重。因为调整权重是从后向前反向进行，且根据反向传播过来的误差来调节权重，所以才称为反传算法。

梯度下降的方法要求权重的改变满足下面的公式：

$$\Delta_p w_{ij} \propto -\frac{\partial E_p}{\partial w_{ij}} \quad (11.2.9)$$

其中 w_{ij} 是从节点 i 到节点 j 的权重， E_p 是样本 p 的误差，用下式来计算：

$$E_p = \frac{1}{2} \|Y_p - O_p\|^2 = \frac{1}{2} \sum_k (y_{pk} - o_{pk})^2 \quad (11.2.10)$$

下面以传递函数 $f(u) = \frac{1}{1+e^{-u}}$ 为例来介绍权重的修改方法，对传递函数求导可得

$f'(u) = f(u)(1-f(u))$ 。下面进行数学上的推导。

$$-\frac{\partial E_p}{\partial w_{ij}} = -\frac{\partial E_p}{\partial u_{pj}} \cdot \frac{\partial u_{pj}}{\partial w_{ij}}, \quad \frac{\partial u_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ji}} \left(\sum_i w_{ij} o_{pi} \right) = o_{pi},$$

$$\text{令 } \delta_{pj} = -\frac{\partial E_p}{\partial u_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial u_{pj}}, \quad \text{因为 } o_{pj} = f(u_{pj}), \quad \text{所以 } \frac{\partial o_{pj}}{\partial u_{pj}} = o_{pj}(1-o_{pj}),$$

$$\text{当 } j \text{ 为输出节点时, } \frac{\partial E_p}{\partial o_{pj}} = \frac{\partial}{\partial o_{pj}} \left(\frac{1}{2} \sum_j (y_{pj} - o_{pj})^2 \right) = -(y_{pj} - o_{pj}), \quad \text{所以}$$

$$\delta_{pj} = (y_{pj} - o_{pj}) o_{pj} (1 - o_{pj}), \quad \text{当 } j \text{ 不是输出节点时,}$$

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial u_{pk}} \cdot \frac{\partial u_{pk}}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial u_{pk}} \cdot \frac{\partial}{\partial o_{pj}} \left(\sum_j w_{jk} o_{pj} \right) = \sum_k \frac{\partial E_p}{\partial u_{pk}} \cdot w_{jk} = \sum_k \delta_{pk} w_{kj}.$$

所以权重的调整公式为：

$$\Delta_p w_{ij} = \eta \delta_{pj} o_{pi} = \begin{cases} \eta o_{pj} (1 - o_{pj}) (y_{pj} - o_{pj}) o_{pi} & j \text{ 是输出神经元} \\ \eta \left[o_{pj} (1 - o_{pj}) \sum_k \delta_{pk} w_{jk} \right] o_{pi} & j \text{ 不是输出神经元} \end{cases} \quad (11.2.11)$$

其中 η 是常系数， i, j, k 是有前向后相邻三层里的神经元。

11.3 强化学习

强化学习[Sutton & Barto 1998]是一种无指导的学习，它的主要思想是“与环境交互（Interaction with Environment）”和“试错（trial-and-error）”。这也是自然界中人类或动物学习的基本途径。当一个婴儿学习说话、学会走路时，并没有一个明确的老师，他的学习都是通过眼睛和其它的感知器官来观察不同的行动会有什么样的结果，观察什么样的行动可以得到自己满意的目标。先行动再观察是与环境交互的基本方式，在交互过程中，特别是开始，肯定会发现很多行动都无法完成目标，也就是在不断试错了。不仅婴儿是这样学习的，它会伴随我们一生，我们学习开车就是这样。学车人要时刻注意车子和环境的情况，要发现自己的不同行为到底会有什么样效果和影响。

11.3.1 强化学习模型

强化学习试图从交互过程中找到一些可以用计算机程序实现学习的方法，并不直接考虑人类或动物是怎么通过交互来学习的。同时强化学习也只考虑有具体目标的学习。为了得到可计算的学习方法，首先就要明确学习过程中的交互方式，图 11.3.1 给出了强化学习的标准模型。

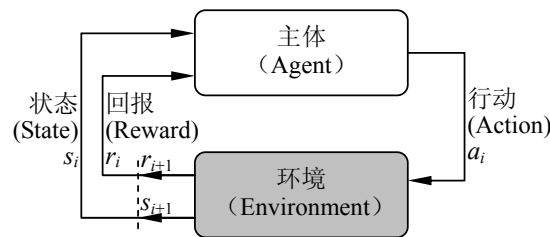
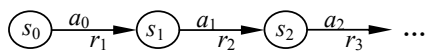


图 11.3.1：强化学习的标准模型

从模型中可以看到，主体与环境的交互接口包括行动（Action）、回报（Reward）和状态（State）。交互过程可以表述为如下模式：



这里的回报，和指导学习（Supervised Learning）中的教师是不同的。在指导学习中，教师要给出很多例子，告诉主体什么情况下，执行什么行动效果最好；在强化学习中，回报只告诉主体当前行动的执行效果，主体要在与环境交互过程中，不断测试每个行动的效果，在长时间的收集回报后，判断出每个行动的长远回报，完成主体的学习。

图 1 的强化学习模型事实上刻画了一类问题，只要一个问题可以描述成强化学习模型，那么解决这个问题所有方法，都称为强化学习方法。

11.3.2 几个关键概念

前面的模型为强化学习方法奠定了基础，但在模型里还隐藏了一些问题需要先讨论一下。这几个强化学习中的关键概念分别是延迟回报（Delayed Reward）、探索和利用的权衡（Tradeoff between Exploration and Exploitation）及与未知环境交互等。

1. 延迟回报

主体为了完成给定的任务，必须知道每个行动的长远回报，而不仅仅是即时回报。而长远回报必须经过一定时间的延迟之后才可以获得。与环境交互过程中， t 时刻的延迟回报 R_t 可以用下式表示：

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^T r_{t+T+1} \\ &= \sum_{k=0}^T \gamma^k r_{t+k+1} \end{aligned} \quad (11.3.1)$$

其中， $0 \leq \gamma < 1$ 是折扣率；对于有终任务（Episodic Tasks）， T 是从当前开始到完成任务所需要的时间，对于持续任务（Continual Tasks）， $T = \infty$ 。

如果环境的状态转换只与当前的状态和行动有关，与以前的状态和行动无关，即是一个马尔科夫过程（Markov Decision Process），就可以用下面 11.3.3 节介绍的马尔可夫过程来描述强化学习任务，从而可以方便的找到叠代的方法来逼近每个状态或行动的最佳长远回报。虽然很多问题不能用马尔科夫过程来描述，但以上的思想和方法仍然适用。

2. 权衡探索和利用

假定主体在与环境交互的学习过程中，发现某些动作的效果较好，那么主体在下次决策中该选择什么动作呢？一种考虑是充分利用现有的知识，仍然选择当前认为最好的动作。但这样做有一个缺点：也许还有更好的动作没有发现。反之，如果主体每次都测试新的动作，将导致有学习没进步，显然也不是我们愿意看到的。所以，主体应该在利用现有知识和探索新的效果更好的动作之间做出适当的权衡。

权衡的策略有很多，简单的策略是每次以概率 ϵ 随机选择动作，以概率 $1 - \epsilon$ 选择当前认为最好的动作，该方法称为 ϵ -贪心法。 ϵ -贪心法的缺点是在探索新的动作时，各个动作的选择概率是一样的，这样选择最坏行动的概率也很大。

还有一种常用的方法是以一定的概率选择各个动作，每个动作的选择概率和当前主体对它的评价有关，一般以下式作为行动选择的概率：

$$p(a | s) = \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}} \quad (11.3.2)$$

其中 $T > 0$ 是调控参数， T 越大，各个动作的选择概率越接近， $T \rightarrow 0$ ，则相当于贪心法； $Q(s,a)$ 是 11.3.3 节将要介绍的行动的长远评价。

以上两个选择动作的策略都只需要确定一个参数，一般来说， T 比 ϵ 更难确定一些。

3. 与未知环境交互

强化学习的很多问题，主体事先不知道环境是什么样的，只有在与环境的交互过程中才知道。如果出现下面的两种情况，还需要一定的方法来进行处理。

第一种情况是，环境的状态空间非常大，甚至是连续的，主体没有足够多的空间和时间来访问这些状态，这就要求使用一定的方法来泛化（Generalization）这些状态空间。有时候主体的行动空间可能也很大，或是连续的，也要有一定的泛化方法。

另一种情况是，主体只能感知环境的部分信息，这就需要用部分马尔科夫过程（Partially Observable MDP）来描述问题，并且需要主体记录历史信息，即以前观察到的环境信息和执行过的行动等。

11.3.3 马尔可夫过程

如果一个强化学习任务满足马尔可夫特性，任务的状态转换就是一个马尔可夫过程。马尔可夫过程是个四元组 $\langle S, A, T, R \rangle$ ，其中

S 是状态空间；

A 是行动空间，表明每个状态下主体可以执行的行动，状态 s 下可执行的行动记为 $A(s)$ ；

$T: S \times A \times A \rightarrow [0,1]$ 是状态转换模型， $T(s, a, s')$ 表示在状态 s 下执行行动 a 到达状态 s' 的概率，满足 $\sum_{s'} T(s, a, s') = 1$ ；

$R: S \times A \times A \rightarrow \mathbf{R}$ 是即时回报函数， $R(s, a, s')$ 表示环境对主体在状态 s 下执行行动 a 到达状态 s' 的实值即时评价。

从转换模型和即时回报函数可以看到，它们只与当前状态和行动有关，不需要历史信息，满足马尔可夫特性。

$\pi: S \times A \rightarrow [0,1]$ 是一个策略， $\pi(s, a)$ 表示在状态 s 下选择行动 $a \in A(s)$ 的概率。给定一个马尔可夫过程和策略 π ，可以用 $V^\pi(s)$ 表示从状态 s 开始，执行策略 π 后的长期期望回报，称为状态评价函数，可知

$$\begin{aligned}
 V^\pi(s) &= E_\pi \{R_t \mid s_t = s\} \\
 &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \\
 &= E_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s \right\} \\
 &= \sum_a \pi(s, a) \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s' \right\} \right] \\
 &= \sum_a \pi(s, a) \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]
 \end{aligned}$$

其中 E_π 表示满足策略 π 的期望。

同样，可以定义行动评价函数 $Q^\pi(s, a)$ ，并有

$$\begin{aligned} Q^\pi(s, a) &= E_\pi \{R_t \mid s_t = s, a_t = a\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\} \end{aligned}$$

求解一个强化学习任务，是希望找到一个最佳策略，首先就要比较两个策略的好坏，说策略 π 比 π' 好，要求对所有状态 s 有 $V^\pi(s) \geq V^{\pi'}(s)$ 。这样就可以如下构造一个策略 π^* ，它比其它所有策略都好：

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^\pi(s)$$

策略 π^* 称为最佳策略，对应的最佳状态评价记为 V^* 。同样最佳行动评价 Q^* 有

$$\begin{aligned} Q^*(s, a) &= \max_{\pi} Q^\pi(s, a) \\ &= E_{\pi^*} \{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \end{aligned}$$

最佳状态评价也可以由最佳行动评价来表示：

$$\begin{aligned} V^*(s) &= \max_a Q^*(s, a) \\ &= \max_a E \{R_t \mid s_t = s, a_t = a\} \\ &= \max_a E \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\} \quad (11.3.3) \\ &= \max_a E \{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \\ &= \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \end{aligned}$$

这样，

$$\begin{aligned} Q^*(s, a) &= E \{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\} \\ &= \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q^*(s', a')] \quad (11.3.4) \end{aligned}$$

上面两式称为 Bellman 最佳方程，有了这两个公式，就可以方便为状态或行动的长远评价设计叠代公式和相应的学习算法。

11.3.4 动态规划

动态规划（Dynamic Programming）是强化学习中最常用方法之一。由 (11.3.3) 式可以得到下面的迭代公式：

$$V_{t+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_t(s')] \quad (11.3.5)$$

当时间 t 趋于无穷时， $V_t(s)$ 趋于最佳状态评价 $V^*(s)$ ，实际操作中当状态评价的改变非常小时就停止迭代，得到对每个状态长远回报评价的估计。可以看出，动态规划需要事先知道状态转换概率 T 和即时回报的期望值 R ，即需要有环境的一个完整模型，可以离线进行学习。图 11.3.2 给出了具体的学习算法

-
- 1 初始化 V ，如任意状态 s 有 $V(s) \leftarrow 0$
 - 2 循环，直到 Δ 小于小正数 ε
 - 3 $\Delta \leftarrow 0$
 - 4 对每个状态 s ，循环
 - 5 $v \leftarrow V(s)$
 - 6 $V(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V(s')]$
 - 7 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
 - 8 输出一个确定的策略 $\pi(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V(s')]$
-

图 11.3.2 动态规划学习算法

基本的动态规划方法同等对待每个状态，它们循环迭代的次数是一样的，如果状态空间非常大，学习收敛的速度就会非常慢。异步动态规划是对基本算法的一个扩展，考虑在迭代时不是对每个状态依此地迭代，而是用一定的方法选择每次迭代的状态，比如评价值改变较大的状态优先进行迭代，这样在相同的时间内可以得到更好的策略。

动态规划还要求有完整的环境模型，很多问题事先很难给出环境的完整模型，比如机器人足球。实时动态规划对动态规划和异步动态规划的进一步扩展，在真实环境中不断测试得到环境模型。

11.3.5 蒙特卡罗法

蒙特卡罗法（Monte Carlo methods）也是强化学习中最常用的方法之一，主要用于有终任务。蒙特卡罗法不需要环境的模型，根据策略一步步的产生动作直到任务完成，然后对经过的状态或行动的评价进行统计更新。基本思想如图 11.3.3 所示。

-
- 1 用与当前长远回报评价有关的策略，不断产生动作直到一个任务完成；
 - 2 对任务中首次出现的每对 (s, a) ：
 - 3 计算出在本次任务中的长远回报 $R(s, a)$
 - 4 把 $R(s, a)$ 与以前各次任务中 (s, a) 的长远回报一起求平均，作为新的 $Q(s, a)$
 - 5 转 1
-

图 11.3.3 蒙特卡罗算法

11.3.6 时序差分

时序差分（Temporal Difference）也是强化学习的最常用方法之一。时序差分不需要环境的模型，每一步都根据即时回报，及时修改对长远回报的评价。当用真实环境中的行动测试代替环境模型时，从公式 (11.3.3) 和公式 (11.3.4) 可以得到如下迭代公式：

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (11.3.6)$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (11.3.7)$$

其中 α 是学习速率。如果 α 取值合适，在各种状态下，每个动作都被选中无限次，则 $V(s)$ 和 $Q(s, a)$ 分别以概率 1 收敛于最佳长远回报 $V^*(s)$ 和 $Q^*(s, a)$ 。

根据不同的修改方法，时序差分又分为很多不同的种类，Q 学习是最常用的方法之一，使用公式 (11.3.7) 对行动评价进行迭代，图 11.3.4 给出了具体的算法。

-
- 1 初始化 $Q(s, a)$
 - 2 对每个任务，循环
 - 3 设置初始状态 s
 - 4 对任务中的每一步，循环，直到 s 是目标状态
 - 5 根据策略和 Q 选择状态 s 下的行动 a （比如 ϵ -贪心策略）
 - 6 执行行动 a ，观察下一步状态 s' 和即时回报 r
 - 7 $Q(s, a) \leftarrow Q(s, a) + \alpha [r_{t+1} + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
 - 8 $s \leftarrow s'$
-

图 11.3.4 Q 学习

Q 学习迭代公式中使用了下个状态不同行动的最大评价，是策略无关的一种学习方法。也可以在迭代时使用下个状态的根据策略选择的实际行动的评价，称为策略相关的学习方法，Sarsa 是这样一种比较常用的学习方法之一，使用的迭代公式是：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (11.3.8)$$

时序差分的每一步都要对评价进行更新，如果不想每步都立即更新评价，而是过一定的时间步才更新评价，则可以引入一个参数 λ 来调整间隔步数，称为 λ -时序差分 (TD(λ))。如果 $\lambda=0$ ，则是一般的时序更新，如果 $\lambda=1$ ，则相当于蒙特卡罗法。图 11.3.5 给出了 Sarsa(λ) 学习方法，是比较常用的学习方法之一。

-
- 1 初始化 $Q(s, a)$
 - 2 对每个任务，循环
 - 3 设置初始状态 s
 - 4 根据策略和 Q 选择状态 s 下的行动 a （比如 ϵ -贪心策略）
 - 5 对任务中的每一步，循环，直到 s 是目标状态
 - 6 执行行动 a ，观察下一步状态 s' 和即时回报 r
 - 7 根据策略和 Q 选择状态 s' 下的行动 a'
 - 8 $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
 - 9 $e(s, a) \leftarrow e(s, a) + 1$
 - 10 对所有的状态行动对 (s, a)
 - 11 $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$
 - 12 $e(s, a) \leftarrow \gamma \lambda e(s, a)$
-

11.3.7 泛化

如果环境的状态空间非常大，甚至是连续的，如机器人足球，主体在进行强化学习时，没有足够多的空间和时间来访问这些状态，这就要求对状态空间进行泛化（Generalization），也称函数估计（Function Approximation）。泛化方法在其它领域已经进行了充分的研究和应用，如机器学习、人工神经网络、模式识别和曲线拟合等。理论上说，这些领域内使用的泛化方法都可以应用到强化学习中，只要把强化学习和这些泛化方法很好的结合在一起。

神经网络是比较常用的泛化方法之一，一般使用反传网络，网络的输入单元是环境的状态 s ，状态可以表示为向量的形式，网络输出是对该状态的评价 $V(s)$ ，对状态的评价都存储在网络的权重中。在强化学习的每一步迭代时，用式 (11.3.9) 可以得到输出单元的误差，再用体现梯队下降思想的反传算法修改网络权重。

$$Error = \sum_{s_t} (V(s_t) - (r_{t+1} + V(s_{t+1})))^2$$

常用的泛化方法还有下面几种线性方法。

Coarse 泛化是把一个多维连续空间转化为一组真假量，称为环境的二进制特征（Binary Feature）。图 11.3.6 是一个二维空间 Coarse 泛化的例子，每个圆圈（可以是别的形状）是一个真假量，如果当前状态点在某个圆圈内，则对应的真假量为真，否则为假。

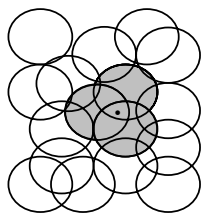


图 11.3.6: Coarse

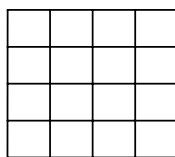


图 11.3.7: Tile 泛化

Tile 泛化是 Coarse 泛化的一个特殊形式，它的一组真假量可以全部覆盖状态空间，并且每次只有一个真假量为真。在二维情况下，可以用网格（可以是别的形状）来划分空间，每个格子是一个真假量，如图 11.3.7 所示。如果对一个多维连续空间只用一次 Tile 泛化，则等同于量化（Quantization）。为了使泛化更加精确，可以对状态空间进行多个 Tile 泛化，即用多组 Tile 真假量来泛化多维连续状态空间，如图 11.3.8 是二维情况下进行两个 Tile 泛化（可以采取两种不同的形状）。

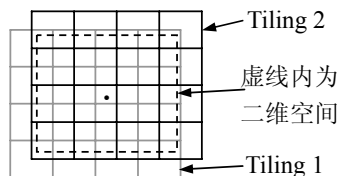


图 11.3.8: 多个 Tile 泛

WrightEagle

如果是一个高维空间，比如上百维，Tile 泛化就很难实现。但多数情况下，主体可能只会在其中很小的空间内活动。Kanerva 泛化从高维状态空间中，根据主体的实际活动情况，抽取出一些特征来表示状态，把这些特征都表示为二进制特征，比如用上面的 Tile 泛化。然后 Kanerva 随机产生一些原型状态（Prototype States），对当前主体的二进制特征状态，计算它与各个原型状态的海明距离（Hamming Distance），距离最小的就认为和原型状态是一个状态。

11.4 机器人足球中的应用

由于仿真机器人足球的复杂性，难以将机器学习的方法直接加以应用。人们一般先要把主体的决策分成不同的几个层次，在每个层次中又分成一些不同的任务，对应每个任务再使用相应的学习方法来完成学习任务。Peter Stone 率先提出了分层学习 (Layered Learning) 的想法，并且给出了四个原则[2]。结合人类足球的特点，可以把仿真机器人足球中主体的决策分为三个层次，分别是：个人技术决策、局部战术决策和全局战术决策。个人技术决策是最低层的决策，它的任务是一些个人必备的技术，如传球、带球、射门等，它们的决策结果是由基本动作组成的动作序列；局部战术决策是较高层次的决策，它的任务主要是几个队友之间的一些战术配合，如 2 过 1 等，决策结果是个人技术层的任务（即个人技术）组成的序列；全局战术决策是最高层的决策，涉及阵型、战术风格等，决策结果是战术配合组成的序列。

在全局战术层，目前还没有很好的学习方法，通常仍依靠手工编码。在个人技术层和局部战术层，机器学习使用的比较多，并且取得了值得重视的进展。下面简要介绍一些方法，细节请参考相应的参考文献。

个人技术是一个球员最基本的技能，如追球、拦球、传球、带球、射门、盯人等。这些个人技能的任务都比较简单。一个简单任务所需要的环境信息相对较少，球员的行动空间也大大减小。

Esses Wizards'2000 (Hu etc. 2000) 使用决策树的方法来评价传球点的好坏。Riedmiller etc. (2000) 使用实时动态规划学习球员的个人技术，采用神经网络作为连续状态空间的泛化。Arseneau etc. (2000) 用 Q 学习来学习如何带球和拦球，用神经网络来学习传球的评价。学习带球的方法是用三个参量表示环境状态：对方球门方向、对手距离（离散为三个区间）和对手方向（窄视野范围内离散为四个区间，其它离散为两个区间），用 5 个带球方向作为行动空间（假定球员对一个方向的带球已经学好）： -90° 、 -45° 、 0° 、 45° 和 90° 。这样不仅可以把球带向对方球门，还可以避让对手。如果对手把球得到，球员得到回报 $D - 0.01T$ ，其中 D 是到球门的距离， T 是总共花费的时间；带球中间每一步得到回报 $\Delta d - 0.01 \Delta t$ ，其中 Δd 是从上一状态到现在带球带过的距离， Δt 是从上一状态到现在经过的时间。传球学习用了三层反传网络，输入层有四个节点，分别是队友的距离方向和对手的距离方向，一个输出节点表示成功与否，15 个隐层节点。

如果个人技术已经做的比较好（通过学习或其它的方法），在学习几个队友之间的战术配合时，用半马尔科夫过程来描述问题，即直接把个人技术作为主体的行动空间，会大大简化战术的学习。

Stone & Veloso (1998) 提出了 TPOT-RL，结合了蒙特卡罗法和时序差分，来学习多个队友之间的传球和射门。Stone & Sutton (2001) 用 Sarsa(λ) 和 Tile 泛化来学习“3 对 2 控球 (Keepaway)”。Lind etc. (1999) 用和 Sarsa 类似的桶队列 (Bucket Brigade) 算法来学习门前战术（对方球门前的决策：射门、传球或带球）。除此之外，战术学习中使用比较多的方法仍然是 Q-Learning。Riedmiller etc. (1999) 用 Q-Learning 和第一节介绍过的神经网络泛化来学习球员如何选择个人技术，Hu etc. (1999) 用 Q-Learning 来学习“2 过 1”，Kostiadis & Hu (1999) 用 Q-Learning 来学习“2 过 1”和“2 过 2”，Peter & Sutton (2000) 用 Q-Learning 和 Coarse 泛化来学习“3 对 2 控球”，Kostiadis & Hu (2001) 用 Q-Learning 和 Kanerva 泛化来学习“3 对 2 控球”。

另外在决策之前，最好能充分了解对手情况，即要在线分析对手的行为。Drücker etc. (2002) 和 Riley & Veloso (2000) 使用决策树的方法来分析对手行为，可以有效地提高自己球

队的决策效果。

参考文献

[Arseneau etc. 2000]

Shawn Arseneau, Wei Sun, Changpeng Zhao, Jeremy R. Cooperstock: Inter-Layer Learning Towards Emergent Cooperative Behavior. *American Association for Artificial Intelligence* (AAAI 2000), Austin, Texas. July, 2000.

[Drücker etc. 2002]

Drücker, C., Hübner, S., Visser, U., Weland, H.-G.: "As time goes by" - Using time series based decision tree induction to analyze the behaviour of opponent players, In A. Birk & S. Coradeschi & S. Tadokoro (Eds.), *RoboCup 2001: Robot Soccer World Cup V* (Vol. 2377, pp. 325-330). Seattle, WA: Springer Verlag, 2002.

[Hu etc. 1999]

Huosheng Hu, Kostas Kostiadis, Zhengyu Liu. Coordination and Learning in a Team of Mobile Robots. *Proceedings IASTED Robotics & Applications Conference*, California, October, 1999.

[Hu etc. 2000]

H. Hu, K. Kostiadis, M. Hunter, N. Kalyviotis, M. Seabrook: Essex Wizards'2000 Team Description. 2000. <http://citeseer.ist.psu.edu/hu00essex.html>

[Kostiadis & Hu 1999]

Kostas Kostiadis, Huosheng Hu. Reinforcement Learning and Co-operation in a Simulated Multi-agent Systems. *Proceedings of IEEE/RJS IROS'99*, Korea, October, 1999.

[Kostiadis & Hu 2001]

Kostas Kostiadis, Huosheng Hu. KaBaGe-RL: Kanerva Based Generalization and Reinforcement Learning for Possession Football. *Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems (IROS 2001)*, Hawaii, October, 2001.

[Lind etc. 1999]

Jürgen Lind, Christoph G. Jung, Christian Gerber. Adaptively and Learning in Intelligent Real-Time Systems. *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[McCulloch & Pitts 1943]

W. S. McCulloch, W. Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:115-133, 1943.

[Quinlan 1986]

J. R. Quinlan: Introduction of Decision Trees. *Machine Learning*, 1(1):81-106, 1986.

[Quinlan 1996]

J.R. Quinlan: Bagging, Boosting and CN4.5. In *Proceedings of AAAI96*, Menlo Park, AAAI Press, 1996.

[Riedmiller etc. 1999]

Martin Riedmiller, Sebastian Buck, Sergio Dilger, Ralf Ehrmann, Artur Merke. Karlsruhe Brainstormers – Design Principles. *RoboCup-99 Team Descriptions*, Simulation League, Team Karlsruhe Brainstormers, 59-63, 1999.

[Riedmiller etc. 2000]

M. Riedmiller, A. Merke, D. Meier, A. Hoffmann, A. Sinner, O. Thate, R. Ehrmann. Karlsruhe Brainstormers – A Reinforcement Learning Approach to Robotic Soccer. *RoboCup-2000: Robot Soccer World Cup IV*. Springer, 2000.

[Riley & Veloso 2000]

Patrick Riley and Manuela Veloso. Towards behavior classification: A case study in robotic soccer. In *Proceeding of the Seventeenth National Conference on Artificial Intelligence*, Menlo Park, CA, 2000. AAAI Press.

[Rosenblatt 1958]

F. Rosenblatt: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65:386-408, 1958.

[Rumelhart etc. 1986]

D. E. Rumelhart, J. L. McClelland, and the PDP research group: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge, MIT Press, 1986.

[Shannon 1948]

C. Shannon: A Mathematical Theory of Communication. *Bell System Technical Journal*, 1948.

[Stone & Sutton 2000]

Peter Stone, Richard S. Sutton, Satinder Singh. Reinforcement Learning for 3 vs. 2 Keepaway. *RoboCup 2000: Robot Soccer World Cup IV*, 249-258, Springer, 2000

[Stone & Sutton 2001]

Peter Stone, Richard S. Sutton. Scaling Reinforcement Learning toward RoboCup Soccer. *Proceedings of the 18th International Conferences on Machine Learning*, 2001

[Stone & Veloso 1998]

Peter Stone, Manuela Veloso. Team-Partitioned, Opaque-Transition Reinforcement Learning. *RoboCup-98: Robot Soccer World Cup II*. Springer, 1998.

[Sutton & Barto 1998]

Richard S. Sutton, Andrew G. Barto: *Reinforcement Learning: An Introduction*. MIT Press, 1998.

第十一章 教练

本章将介绍 RoboCup 仿真足球队中的教练球员以及 RoboCup 仿真比赛中的教练比赛。第一节首先将简要介绍教练球员，教练球员所要解决的问题；在第二节中我们将详细介绍教练球员与 Server 之间交互；第三节将详细介绍 RoboCup 仿真组的教练比赛以及教练语言；第四节则给出一个简单的教练球员的结构框架。

11.1 教练简介

在 RoboCup 仿真机器人足球比赛当中，除了球员程序外，Soccer Server 还允许每个球队连入一个教练（Coach）程序。与普通队员不同，教练不出现在球场上进行比赛。相对于普通球员，教练球员拥有一定的特权来协助普通球员，例如教练可以获取准确的全局信息等等。通过这些特权，教练能够完成一些普通球员不能完成的工作。例如进行一些高层的分析工作并反馈给球员；对球员进行训练等等。

在 RoboCup 仿真组中，教练球员可以分为两种：在线教练和离线教练。其中在线教练可以正式比赛中连入，可以在比赛中给球员提供额外的信息和建议，以帮组球队进行一些战略上的调整。在线教练的功能包括：

- 可以获得无噪声的全局信息
- 可以在通讯受到一定限制的情况下和球员通讯

在比赛过程中，在线教练每次和队员发送自定义消息的长度只能有 128 个字符（由 server 参数 say_coach_msg_size 决定），如果采用标准教练语言通讯的话则可以放宽到 2013 个字符。为了防止在线教练利用通讯全局的控制球队，Server 对在线教练的通讯作了一定的限制。在标准教练语言规定了 4 种类型的消息：advice, define, info 和 meta 消息。在目前的比赛规则中，Server 的限制是：在 play-on 的模式下在线教练可以每 300 个周期发送上述类型的消息各一个（分别由 server 参数 clang_win_size, clang_advice_win, clang_define_win, clang_info_win 和 clang_meta_win 指定）；而且这些消息有 50 个周期的延迟（由 Server 参数 clang_mess_delay 指定）。当 play-mode 不是 'play-on' 时则没有这些限制。也就是说在非 'play-on' 的模式下，在线教练可以无延迟的发送大量消息。

值得注意的是，自定义格式的消息(freeform)只能是非 'play-on' 的模式下发送。而且在一场比赛中，在线教练只能发送 say_coach_cnt_max 条自定义格式的消息。当进入加时赛时，每 6000 个周期在线教练可以获得发送额外 say_coach_cnt_max 条 freeform 消息的能力。

而离线教练不能在比赛中使用，离线教练主要用于在开发设计球队时提供一些自动训练和调试的功能。例如可以利用离线教练来反复重现特定的场景，以此来实现球员技能的机器

学习过程。因此，和在线教练相比，离线教练拥有更多的特权，包括：

- 离线教练可以改变 Play-mode
- 离线教练可以把球和球员移动到任何位置并设置它们的方向和速度
- 离线教练可以不受限制的广播命令和信息，格式可以是用户自定义的

11.2 与 Soccer Server 交互

与普通球员一样，教练也使用 UDP 协议与 Soccer Server 相互通讯，并可以向 Soccer Server 发送指令以完成一定的任务。在缺省情况下，离线教练使用 UDP 端口 6001；在线教练使用 UDP 端口 6002。

11.2.1 通用指令

- (look)

向 Soccer Server 发送 look 指令将会得到球场上的全局视觉信息，包括球场上物体的位置、速度和角度信息。这些物体包括：左右球门中点、球以及在场上的所有球员。

可能得到的返回信息为：

- (ok look TIME (OBJ1 OBJDESC1) (OBJ2 OBJDESC2) ...)

其中 OBJi 可以是上面提到的任何物体，其标示符的命名请参照前面的章节所示。OBJDESCi 将以下面所示的格式给出：

- 对于球门中点： X Y
- 对于球： X Y DELTAX DELTAY
- 对于球员： X Y DELTAX DELTAY BODYANGLE NECKANGLE

值得注意的是，使用 look 指令只能得到当前周期的视觉信息，如果想每个周期都定期收到视觉消息，应该使用 (eye on) 指令。

- (eye MODE)

eye 指令用于指定教练的视觉模式。MODE 的取值为 on 或 off。如果(eye on)被发送，则 soccer server 将会在每个周期给教练发送格式为 (see_global ...) 的全局视觉消息；如果 (eye off) 被发送，则 server 将停止发送定期的视觉消息。

可能得到的返回消息为：

- (ok eye on)

(ok eye off)

收到这些消息表示命令执行成功

- (error illegal_mode)

MODE 的取值不是 on 或者 off

- (error illegal_command_form)

MODE 参数没有被指定

- (team_names)

该条命令将会使得 soccer server 返回在场上球队的队名。

可能得到的返回值为：

- (ok team_names [(team l TEAMNAME1) [(team r TEAMNAME2)])]

取决于连入 soccer server 的球队，server 将返回 0、1 或是全部球队的队名。其中第一个返回的队名是左边的球队。

11.2.2 在线教练受限使用的指令

- (init (version VERSION)) 用于离线教练
- (init TEAMNAME (version VERSION)) 用于在线教练

init 命令用于连接 Soccer Server，并在连接时指定协议的版本号。当在线教练连接时，TEAMNAME 必须是当前在球场上的某个球队的名字。离线教练在初始化时不需要向 Server 发送 init 指令，不过为了保证 Server 使用正确的协议，最好还是在初始化时向 Server 发送 init 指令。

可能得到的返回消息：

- (init ok)

表明离线教练初始化成功

- (init SIDE ok)

表明在线教练初始化成功，SIDE 为 ‘l’ 或是 ‘r’。

- (say MESSAGE)

当离线教练使用 say 指令时，Server 将向所有球员广播 MESSAGE，此时 MESSAGE 的格式是一个长度小于 say_coach_msg_size 的字符串，格式和普通球员通讯的格式相同，由字母和符号 “().+*/?<>_” 组成。当在线教练使用 say 指令时，MESSAGE 仅向自己的队员广播，（间隔，及限制）

可能得到的返回消息：

- (ok say)

say 指令执行成功

- (error illegal_command_form)

MESSAGE 不符合规定的格式。

- (change_player_type TEAM_NAME UNUM PLAYER_TYPE) 用于离线教练

- (change_player_type UNUM PLAYER_TYPE) 用于在线教练

该指令用于更改异构球员的类型。其中 UNUM 代表球员的类型 PLAYER_TYPE, PLAYER_TYPE 是一个 0 到 6 之间的数字, 0 代表缺省的球员类别。关于异构球员类型的详细信息请参考前面的章节。要注意的是, 对于在线教练而言, 一支球队中每种类型的球员不能超过 3 个, 而离线教练则没有这个限制。

Server 可能返回的信息:

- (warning no_team_found)
球队不存在
- (error illegal_command_form)
发送的消息格式不正确
- (warning no_such_player)
指定更换类型的球员不存在
- (ok change_player_type TEAM UNUM TYPE)
更换球员类型成功

下列返回消息只有在线教练连接才会出现。

- (warning cannot_sub_while_playon)
在 play-mode 为 'play_on' 时不能更换球员类型
- (warning no_subs_left)
在比赛开始后, coach 更换球员类型的次数超过了 3 次 (由 Server 参数 subs_max 指定)
- (warning max_of_that_type_on_field)
指定球员的类型不是缺省类型, 并且在场上已经超过了 3 个 (由 Server 参数 subs_max 指定)
- (warning cannot_change_goalie)
守门员的类型不能更改

当球员类型被更改后, 本队的队员将收到下列消息:

- (change_player_type UNUM TYPE)
对方的球员将收到:
- (change_player_type UNUM)

11.2.3 离线教练专用指令

- (change_mode PLAY_MODE)

该指令将球场上的 play-mode 改变为 PLAY_MODE。其中 PLAY_MODE 必须为

Soccer Server 所支持的 play-mode（请参照前面的章节）。值得注意的是 change_mode 指令仅仅更改比赛的 play-mode，并不会更改球的位置，例如将 play-mode 改为 kick-off 时，球并不会自动被摆放到开球位置。但在某些情况下，改变 play-mode 时 Soccer Server 会改变球员的位置，例如将 play-mode 改为 free-kick 时，在罚球半径内的对方球员会被 Server 自动移出。

Server 可能返回的信息为：

- (ok change_mode)
change_mode 指令执行成功
- (error illegal_mode)
指定的 play-mode 不正确
- (error illegal_command_form)
发送的消息格式不正确

● (move OBJECT X Y [VDIR [VELx VELy]])

该指令将移动指定的物体到指定的绝对坐标(X Y)。OBJECT 的命名格式请参照前面的章节。它既可以是球，也可以是球员。如果 VDIR 被指定，被移动球员的面向角将被设置为 VDIR。同样的，如果 VELx 和 VELy 被指定，物体的速度也会被置为相应的值。

Server 可能返回的信息为：

- (ok move)
move 指令被成功执行
- (error illegal_object_form)
OBJECT 的格式不正确
- (error illegal_command_form)
指定的消息格式不正确

● (check_ball)

要求 Soccer Server 检查球的位置状态。Server 可能返回的信息为：

- (ok check_ball TIME BALLPOSITION)

其中返回的位置状态 BALLPOSITION 被定义为以下几种：

- in_field
球在界内
- goal_l
球在左边半场的球门内

- goal_r

球在右边半场的球门内

- out_of_field

球在界外

要注意的是当 Server 返回 “goal_l” 和 “goal_r” 时并不意味着球越过了球门线。

- (start)

该指令将使 Server 开始比赛，即将 play-mode 切换为 ‘kick_off_l’。执行 start 指令等效于按下了 monitor 的 kick_off 按钮。要指出的是当离线教练没有通过 init 命令初始化时，第一条来自于该教练的任何指令都将使 Soccer Server 开始比赛。

Server 可能返回的信息为：

- (ok start)

指令成功执行

- (recover)

recover 指令将球员的耐力值 (stamina)、恢复值 (recovery)、力量值 (effort) 和听力容量 (hear capacity) 恢复到初始状态。Server 可能返回的信息为：

- (ok recover)

指令成功执行

- (ear MODE)

该指令打开或是关闭离线教练的听觉消息。MODE 必须是 on 或者 off。如果 (ear on) 被发送，则 Server 将会把所有的听觉信息发送给离线教练。如果 (ear off) 被发送，则 Server 将会停止向离线教练发送任何听觉消息。Server 可能返回的信息为：

- (ok ear on)

- (ok ear off)

ear 指令成功执行

- (error illegal_mode)

MODE 不是 on 或是 off

- (error illegal_command_form)

发送的指令格式不正确

11.2.4 Server 消息

除了上节中提到的返回信息外，Soccer Server 还将发送一些别的消息给在线教练和离线教练。如果教练连接时，init 指令中指定的协议版本大于 7.0 时，则在 init 指令之后 Server

将会发送下列的参数消息（[参照章节 xxx](#)）。

- (server_param ...) 一次
- (player_param ...) 一次
- (player_type ...) 每种球员类型一条

如果教练打开了视觉消息开关（eye on），则在每个周期的开始，它将定期收到全局视觉消息：

(see_global TIME (OBJ1 OBJDESC1) (OBJ2 OBJDESC2) ...)

其中 OBJi 为物体的名字；OBJDESCi 为该物体的状态信息，其格式如下所示：

- 对于球门中点：X Y
- 对于球：X Y DELTAX DELTAY
- 对于球员：X Y DELTAX DELTAY BODYANGLE NECKANGLE

如果教练打开了听觉消息开关（ear on），则它将收到所有的听觉消息——包括裁判和球员所发出的所有消息。其格式如下：

- (hear TIME referee MESSAGE)
- (hear TIME (p “TEAMNAME” NUM) “MESSAGE”)

其中前一种格式为裁判所发出的消息，而后一种为普通球员所发出的消息。

11.3 在线教练比赛以及标准教练语言

如何通过外部的观察来对多智能主体系统进行分析，建模、评测并用来指导主体 / 团队的行为是 RoboCup 比赛的重要挑战之一，也是多主体系统研究工作中的重要问题之一。RoboCup 仿真足球赛的在线教练比赛正是为了促进相关领域的研究而设立的。其关心的重点领域包括：

- 单主体 / 多主体团队的建模
- 单主体以及多主体的意图、行为和计划识别
- 多主体团队的性能分析以及评测
- 多主体团队合作问题诊断以及自适应

11.3.1 在线教练比赛

在线教练是 RoboCup 仿真比赛中拥有一定特权的特殊球员。与普通球员不同，在线教练可以从 Soccer Server 中接收到无噪声的全局信息。因此在线教练非常适合用于促进相关的研究。从 2001 年起，RoboCup 组织正式设立了在线教练比赛。

由于在线教练教练比赛的详细规则每年都在调整,因此读者请参照当年组委会所发布的正式规则为准,在这里就不再赘叙了。通常来说,教练比赛的方式大致如下:首先组委会由志愿者提供的若干个支持标准教练语言的球队中各抽取几个球员,然后组成一支供参赛教练指挥的标准球队。然后再从当年参加 RoboCup 仿真组比赛的球队中挑出若干支作为标准对手。比赛将通过参赛教练指挥的球队与标准对手球队比赛的结果来进行计分评比。得分最高的教练获胜。

11.3.2 标准教练语言总览

为了让不同的教练和球队能够相互理解,RoboCup 组委会开发了标准教练语言。其设计思想是通过建立一个统一的、具有清晰语义的语言,使用该语言相互通讯的球队和教练在相互交流时不会产生歧义和误解。通过标准教练语言,一个在线教练能够指挥由其他研究小组所开发的球队。下面将详细介绍标准教练语言。

标准教练语言提供了五种基本消息: Info, Advice, Define, Meta 和 Freedom。

Info 消息:

Info 消息用于传递教练认为球员应该知道的消息,也就是教练用来通知球员的消息。例如,在线教练可以利用 info 消息告诉自己的球员对方的 10 号经常在某个位置出现等。Info 消息可以用来承载对手(单个球员、一组或是整个球队)或是己方球队的信息。其格式如下:

(info TOKEN1, TOKEN2, ... , TOKENn)

Advice 消息:

Advice 消息用于传递教练对于球员的指令。这些指令可以针对某一个球员,也可以某一个小组,甚至可以是整个球队。其格式如下:

(advice TOKEN1, TOKEN2, ... , TOKENn)

Define 消息:

Define 消息用于预定义区域、指令、条件和行为的名称,以便在其后中调用。通过这种方式,教练可以在有限的长度中承载更多的信息;增加可读性。其格式如下:

(define DEFINE_TOKEN1, DEFINE_TOKEN2, ... , DEFINE_TOKENn)

DEFINE_TOKEN 有四种类型:

- (definer name REGION)
- (definec name CONDITION)
- (defined name DIRECTIVE)
- (definea name ACTION)

其中名字的长度被限制在 40 个字符之内。

Meta 消息:

Meta 消息用于传递教练和球员之间通讯时的元消息。例如已被发送的消息的数量，标准语言的版本信息等等。这些消息可以用来支持调试功能和提高兼容性。其格式如下：

(meta META_TOKEN1, META_TOKEN2, ... , META_TOKENn)

在目前的标准教练语言版本当中仅有一种形式的 META_TOKEN,

- (version X) X 为教练使用的语言版本

Freeform 消息:

标准教练语言同时也支持用户自定义的消息，这是通过 Freeform 消息来实现的。自定义的消息长度必须小于 say_coach_msg_size，而且只能在 play-mode 不是 'play_on' 的时候发送。这些消息可以包括英文字母表的所有字符以及().+*/?<>_ 其格式如下：

(freeform "STRING")

要注意的是引号不能省略。

11.3.3 标准教练语言语法

下面是标准教练语言的 BNF 范式:

<COACH_MESSAGE> → <INFO_MESS>

| <ADVICE_MESS>

| <META_MESS>

| <DEFINE_MESS>

| <FREEFORM_MESS>

<INFO_MESS> → (info <TOKEN_LIST>)

<ADVICE_MESS> → (advice <TOKEN_LIST>)

<TOKEN_LIST> → <TOKEN_LIST> <TOKEN> | <TOKEN>

<TOKEN> → (<TIME> <CONDITION> <DIRECTIVE_LIST>) | (clear)

<CONDITION> → (true) | (false)

| (ppos <TEAM> <UNUM_SET> <INT1> <INT2> <REGION>)

//其中 INT1, INT2 构成了满足条件的队员的人数区间 [INT1, INT2]

| (bpos <REGION>)

| (bowner <TEAM> <UNUM_SET>)

| (playm <PLAY_MODE>)

| (and <CONDITION_LIST>)
 | (or <CONDITION_LIST>)
 | (not <CONDITION>)
 | "STRING"

<CONDITION_LIST> → <CONDITION_LIST> <CONDITION>

<DIRECTIVE_LIST> → <DIRECTIVE_LIST> <DIRECTIVE> | <DIRECTIVE>

<DIRECTIVE> → (do <TEAM> <UNUM_SET> <ACTION>) |

| (dont <TEAM> <UNUM_SET> <ACTION>)

| "STRING"

<ACTION> → (pos <REGION>)

| (home <REGION>)

| (bto <REGION> <BMOVE_SET>)

| (bto <UNUM_SET>)

| (mark <UNUM_SET>)

| (markl <UNUM_SET>)

| (markl <REGION>)

| (oline <REGION>)

| (htype <HET_TYPE>)

| "STRING"

<PLAY_MODE> → bko | time_over | play_on | ko_our | ko_opp | ki_out

| ki_opp | fk_our | fk_opp | ck_our | ck_opp | gk_our

| gk_opp | gc_our | gc_opp | ag_our | ag_opp

<TIME> → [int]

<HET_TYPE> → [int] // 其中 0 表示基本类型 -1 表示清除

<TEAM> → our | opp

<UNUM> → [int(0-11)]

<UNUM_SET> → { <UNUM_LIST> }

<UNUM_LIST> → <UNUM_LIST> <UNUM> | e // 在这里 e 表示空集

<BMOVE_SET> → { <BMOVE_LIST> }

<BMOVE_LIST> → <BMOVE_LIST> <BMOVE_TOKEN> | <BMOVE_TOKEN>

<BMOVE_TOKEN> → p | d | c | s

//分别表示传球，带球，解围，射门。要注意的是指定多个动作时之间不需要
 空格


```

<REGION> → <POINT>
        | (null)
        | (quad <POINT> <POINT> <POINT> <POINT>)
        | (arc <POINT> [real] [real] [real] [real])
        | (reg <REGION_LIST>)
        | "STRING"

<REGION_LIST> → <REGION_LIST> <REGION> | <REGION>

<POINT> → (pt [real] [real])
        | (pt [real] [real] <POINT>)
        | (pt ball)
        | (pt <TEAM> <U→)

<META_MESS> → (meta <META_TOKEN_LIST>)
<META_TOKEN_LIST> → <META_TOKEN_LIST> <META_TOKEN> |
<META_TOKEN>
<META_TOKEN> → (ver [int])

<DEFINE_MESS> → (define <DEFINE_TOKEN_LIST>)
<DEFINE_TOKEN_LIST> → <DEFINE_TOKEN_LIST> <DEFINE_TOKEN>
        | <DEFINE_TOKEN>
<DEFINE_TOKEN> → <CONDITION_DEFINE>
        | <DIRECTIVE_DEFINE>
        | <REGION_DEFINE>
        | <ACTION_DEFINE>
<CONDITION_DEFINE> → (definec "[string]" <CONDITION>)
<DIRECTIVE_DEFINE> → (defined "[string]" <DIRECTIVE>)
<REGION_DEFINE> → (definer "[string]" <REGION>)
<ACTION_DEFINE> → (definea "[string]" <ACTION>)

<FREEFORM_MESS> (freeform "[string]")

```

11.3.4 标准教练语言语义解释

下面将详细解释标准教练语言中各个符号的含义。

在 Info 和 Advice 消息中，除了(clear)以外，每一个 TOKEN 都指定了一个反应式计划。

当用在 Info 消息当中时, TOKEN 将作为教练需要通知给队员的信息,既可以是关于对手的,也可以是关于己方球队计划和行为;当用在 Advice 消息中时, TOKEN 则是球队所需要执行的计划。在所有的 TOKEN 当中, (clear)比较特殊。它的作用是通知队员丢弃以前接收到的所有 TOKEN。这相当于把所有仍生效的 TOKEN 的 TIME 值直接设为 0。

每个 TOKEN 中都包括一个时间, 触发条件和一组动作列表, 其格式为:

(<TIME> <CONDITION> <DIRECTIVE_LIST>)

其中 TIME 为该 TOKEN 的生命周期, 通过 TIME 教练可以指定经过若干个周期以后该 TOKEN 应该被废弃。要注意的是, TOKEN 的生命周期从教练发送该 TOKEN 算起, 而不是队员接收到算起。

通过这种方式, 教练可以很容易的通过 ADVICE 消息实现对球员的指挥。因为每个 TOKEN 都可以看成是一个 if-then 规则:

```
if CONDITION
then { DIRECTIVE1, DIRECTIVE2 ... }
```

同样的, 球员也可以很容易的实现教练的支持。只要在程序中建立一个规则集存放教练所发送过来的 TOKEN, 然后和当前的世界状态比较, 查看是否有满足条件的 TOKEN, 执行相应的操作即可。值得注意的是, 如果在某一个周期同时有多个 TOKEN 满足条件, 则球员可以根据实际情况自行选择一个合适的 TOKEN 来执行。

CONDITION:

在 TOKEN 中的条件是由预定义的原子条件加上逻辑操作符复合连接而得到的。其基本的原子条件包括:

- (true)
永远为真
- (false)
永远为假
- (ppos TEAM UNUM_SET INT₁ INT₂ REGION)
当给定的一组球员在指定区域内的人数大于 INT₁, 小于 INT₂ 内时为真。其中 TEAM 为指定的球队, 取值为"our"或是"opp"; UNUM_SET 为可以指定的球员的集合; INT₁ 为满足条件人数的下界, INT₂ 为上界; REGION 为指定的区域, REGION 可以是事先预定义好的名字, 也可以是临时指定的。
- (bpos REGION)
当球在给定区域内时为真
- (bowner TEAM UNUM_SET)

当给定的一组球员中某一个球员控球时为真

- (playm PLAY_MODE)

当球场上的 play-mode 为 PLAY_MODE 时为真。请参照前面的语法定义查看相关的 play-mode 的缩写。

在 TOKEN 的条件定义中，逻辑操作符的格式为

- (and CONDITION₁ CONDITION₂ ... CONDITION_n)
- (or CONDITION₁ CONDITION₂ ... CONDITION_n)
- (not CONDITION)

例如，当要表示“对方 3 号球员或是 4 号球员在区域 X 控球”时，则用标准教练语言可表示为：

(and (ppos opp {3 4} X) (bowner opp {3 4}))

DIRECTIVES:

Directives (指令)为教练对球员的指示说明，也就是说当条件满足时，指定球员应该执行的动作 ACTION。ACTION 的类型和含义请参见下面的内容。Directives 包括两种形式：

- (do TEAM UNUM_SET ACTION)
指定的球员必须执行 ACTION
- (dont TEAM UNUM_SET ACTION)
指定的球员不能执行 ACTION

在 INFO 消息中，Directives (指令)用来通知球员，当条件满足时，给定球队的某些球员会有哪些行为。而当在 ADVICE 消息中时则作为对球员的指令，此时 TEAM 取值为”opp”则没有任何意义。

ACTION:

Action 表示了球员应该执行的动作，包括：

- (pos REGION)
前往区域 REGION
- (home REGION)
指定球员的缺省位置。这条指令通常用于指定球队的阵形。
- (bto REGION BALLMOVE_SET)
将球用指定的方式踢往给定区域。其中 BALLMOVE_SET 指定了踢球的方式，它的取值范围为’p’，’d’，’c’和’s’。’p’代表传球(pass)，’d’代表带球(dribble)，’c’代表解围(clear)，’s’代表射门(shoot)。多个踢球方式可以同时指定，此时各字符之间不需

要空格。例如”pd”，”s”或者”pdc”等。当多个踢球方式被同时指定时，具体采用哪一种方式可以由球员根据具体情况自行决定。

- (bto UNUM_SET)

传球给指定集合中的某个队友，具体哪个由球员自行决定。

- (mark UNUM_SET)

盯住指定集合中的某个对方球员。

- (markl REGION)

防住当前球的位置到指定区域之间的传球路线。

- (markl UNUM_SET)

防住当前球的位置到指定集合中某个对方球员之间的传球路线。

- (oline REGION)

指定造越位战术的区域

- (htype TYPE)

该指令用于通知球员的异构类型。对于异构球员的描述请参照前面的章节。

REGION:

标准教练语言中，区域可以包括下面几种：

- 点， 点的类型包括：

- (pt X Y)

绝对坐标点(X, Y)。其中 X , Y 为实数

- (pt X Y POINT)

指定的坐标点为 (X, Y) + POINT

- (pt ball)

指定坐标点的位置为当前球的位置

- (pt TEAM UNUM)

指定坐标点的位置为当前给定球员的位置

- (quad POINT₁ POINT₂ POINT₃ POINT₄)

定义一个四边形，其中该四边形的四个顶点分别为 POINT₁，POINT₂，POINT₃ 和 POINT₄

- (arc POINT RADIUS_SMALL RADIUS_LARGE ANGLE_BEGIN ANGLE_SPAN)

定义扇形。其中 POINT 指定了该扇形的圆心；RADIUS_SMALL 指定了该扇形的内环半径；RADIUS_LARGE 为外环半径；ANGLE_BEGIN 为起始边的角度；ANGLE_SPAN 则指定了该扇区的张角。利用特殊的参数，我们可以利用 arc 来定义圆形区域等。例如：

(arc (pt 0 0) 0 r 0 360) 表示以 (0, 0) 为圆心，r 为半径的圆。

- (null)
表示一个空区域
- (reg REG₁ REG₂ ... REG_n)
所定义的区域为给定区域的并集

UNUM SET

UNUM SET 用于定义球员的集合。在这个集合当中，球员号先后次序并无关系。如果集合中存在 0，则该集合表示全队球员。UNUM SET 的具体格式为：

{ NUM₁ NUM₂...NUM_n }

11.4 教练程序示例

这一节中将给出一个简单的 Coach 示例程序。该示例程序利用标准 C++实现，运行在 linux 环境下，实现了一个教练的基本框架。读者可以通过这个基本框架了解教练的基本原理，也可以在此之上继续开发自己的教练程序。

在很多情况下，我们可以通过读取比赛的 log 来对特定的球队进行建模，而不必获取该球队的运行代码。因此该示例程序可以设置成为三种类型：在线教练，离线教练和一个可以读 log 进行分析的教练。

从结构上说，教练示例程序可以分为以下几个主要部分：

- 世界模型，这个部分是教练对于球场环境的认知，包括每个周期球的位置，速度；球场上各个球员的位置，速度，朝向角度等等。
- 通讯模块，该部分负责与 Server 通讯，并且负责解析 Server 传递的消息，并根据视觉消息和听觉消息中的信息更新世界模型。
- 主循环，该部分为教练的主要执行部分。每个执行周期都将被调用直至程序结束。因此所有主要的分析程序都放在主循环中

从执行流程上看，教练示例程序设置了两个线程：通讯进程和主循环进程。其中通讯线程负责处理运行通讯模块，主循环负责运行主循环模块。每次主循环线程执行一次分析处理过程时，该线程将进入休眠并等待激活信号；通讯线程则一直阻塞于与 Server 通讯的 UDP Socket，每接收到一条消息，则调用解析器处理该消息。当接收到视觉消息时，通讯线程将发送信号激活主循环线程进行新一轮的分析。

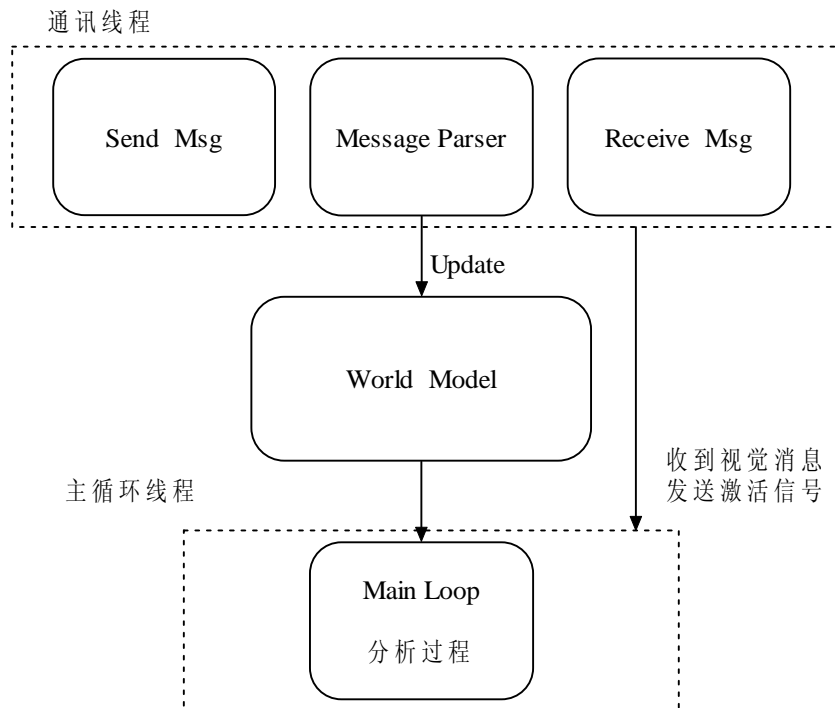


图 11.1 教练示例程序结构图

值得注意的是，当教练示例程序设置为读 log 进行分析的模式时，程序并不需要去连接一个 Server。这时世界模型当中每一个周期的信息都不是通过接收 Server 的消息得到，而是从 log 中读出。因此这时通讯线程并不会真正的创建，与此对应，主循环线程的激活信号将利用一个系统定时器定时产生。

在示例程序的实现当中，这三种类型的教练都各自封装成一个类，通过一个公共的抽象基类来进行操作。从功能上看，离线教练涵盖了在线教练的所有基本功能，因此离线教练类从在线教练类中继承而来。详细的继承关系图如下所示：

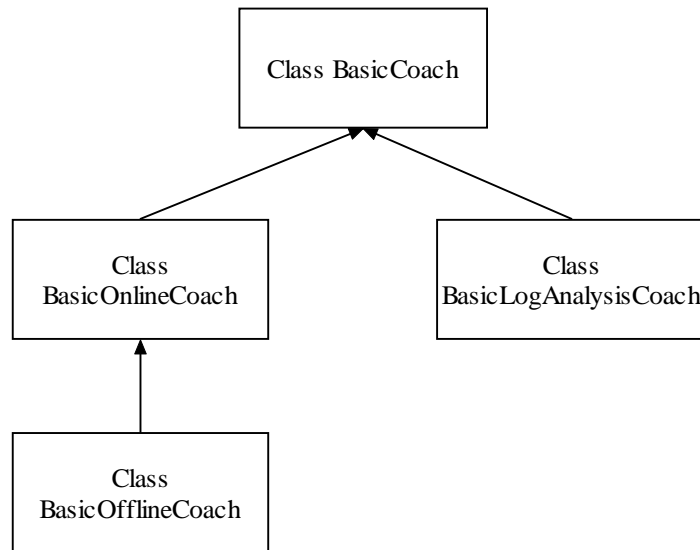


图 11.2 类继承关系图

详细的定义由下面的代码所示：

```

class BasicCoach {
public:
    BasicCoach():m_parseFactory(),m_log(), m_coach_type(INVALID)
        { m_wm = new WorldModel; }
    virtual ~BasicCoach()
        { delete m_wm; }

    virtual int InitCoach(const char *teamName, const char *addr, int port , const char
*conf_file = NULL) = 0;
    virtual int MainLoop() = 0;
    virtual int MsgLoop() = 0;

protected:
    virtual MsgParser*  GetParser(int type)
        { return m_parseFactory.GetParser(type, m_wm); }
    virtual int ParseConfFile(const char *conf_file) = 0;

protected:
    WorldModel *      m_wm;
    ParserFactory     m_parseFactory;
    Logger            m_log;    // log recorder
    int               m_coach_type; // indicate what the coach type is
};

class BasicOnlineCoach : public BasicCoach {
public:

```

WrightEagle

```
BasicOnlineCoach() : BasicCoach(), m_bServerAlive(false), m_bEye(false)
    { m_coach_type = ONLINE;}

virtual ~BasicOnlineCoach() { }

virtual int InitCoach(const char *teamName,const char *addr, int port, const char
*conf_file = NULL);
virtual int SendMsg(const char *msg);
virtual int Say(const char *msg);
virtual int PreLoop();
virtual int MainLoop();
virtual int Thinking();
virtual int MsgLoop();

protected:
    virtual int    SendInitMsg(const char* teamName);
    virtual int    ParseConfFile(const char *conf_file);

    virtual int ChangePlayerType(int unum, int type);
    virtual int Look();
    virtual int Eye(bool mode);  // mode == true --> send (eye on)
    virtual int TeamNames();

protected:
    UDPSocket      m_sock;
    pthread_mutex_t m_mutex;
    pthread_cond_t  m_cond;    // used as a signal to awake main thread in
mainloop
    bool           m_bServerAlive;
    bool           m_bEye;     // eye mode
};

class BasicOfflineCoach : public BasicOnlineCoach {
public:
    BasicOfflineCoach() : BasicOnlineCoach(), m_bEar(false)
        { m_coach_type= OFFLINE; }
    ~BasicOfflineCoach(){ }

    virtual int InitCoach(const char *teamName,const char *addr, int port, const char
*conf_file = NULL);
    virtual int Thinking();  // put periodic analysis code here

protected:
    int  SendInitMsg(const char* teamName);
```



```

virtual int ParseConfFile(const char *conf_file)
    { return 0; }

virtual int  ChangePlayerType(int unum, int type, side_t s);
virtual int  ChangePlayerMode(playmode_t mode);
virtual int  CheckBall();
virtual int  StartGame();
virtual int  Recover();
virtual int  Ear(bool mode);      // mode = true --> send (ear on)


virtual int  MoveObj(int unum, side_t s, double posx, double posy, double ang,
double velx, double vely);
virtual int  MoveObj(int unum, side_t s, double posx, double posy, double ang =
INVALID_VAL);


protected:
    bool      m_bEar;
};


class BasicLogAnalysisCoach : public BasicCoach {
public:
    BasicLogAnalysisCoach() : BasicCoach()
        { m_coach_type= LOGPARSER; }
    ~BasicLogAnalysisCoach(){}


    virtual int MsgLoop()
        { return 0; }


    virtual int InitCoach(const char *teamName,const char *addr, int port, const char
*conf_file = NULL);


    virtual int MainLoop();
    virtual int Thinking();


protected:
    virtual int ParseConfFile(const char *conf_file);
    std::string  m_ourTeamName;
};

```

代码说明：

在抽象基类 BasicCoach 中，MainLoop 和 MsgLoop 分别是示例程序的两个主要函数。其中 MainLoop 为主循环；MsgLoop 为消息接收和处理接口，将会在通讯线程中被调用。另外 BasicCoach 的世界模型是通过类 WorldModel 来实现的。类 WorldModel 将负责存储所

有的球场上的信息（每个周期的数据都将保存），同时提供了一系列的接口函数来操作这些数据。类 **BasicCoach** 通过指针 **m_wm** 来操作类 **WorldModel**。

另外由于在线教练，离线教练和读 **log** 的教练所用的接受的消息都不一样，因此在示例程序中采用了一个 **ParserFactory** 来根据设置的教练类型自动获取正确的消息解析器。除此之外，在示例程序中还提供了一些方便的操作接口，例如提供了用于 **Debug** 教练的日志记录接口(类 **Logger**)，教练配置文件读取接口等等。

详细的代码请参见本书附带的代码包。

11.5 教练应用实例

在 **RoboCup** 仿真球队当中，教练球员有很多实际应用。常见的用途包括：

- 自动训练球队：

在设计 **RoboCup** 仿真机器人球队的时候，很多情况下要使用机器学习的算法。这时就需要构建训练数据，设计特定的训练场景，然后反复训练；有的时候还需要给球队一些反馈。由于教练可以获得球场上的准确信息，同时利用离线教练的场景设置功能，我们可以的反复不断产生训练场景并且可以根据球队的表现给出相应的反馈信息，以帮助机器学习算法的实现。

- 调试球队

利用离线教练的场景设置功能，我们还可以方便的支持球队的调试工作。例如可以利用教练设计一个球队性能评测工具，可以自动的记录在哪些场景下球队的性能不够理想。然后可以从 **log** 中读取这些场景的信息并记录下来，在以后调试的时候就可以再现这些场景，找到球队的问题所在。通过这种方式就可以让机器自动的进行一些测试和调试工作，减轻设计者的负担。

- 离线建模分析

在比赛之前，如果可以事先知道对手的一些特点的话无疑可以帮助球队进行一些战术上的调整。利用教练就可以很方便的做到这一点。我们可以通过读取对手历史比赛纪录的 **log**，然后对这些 **log** 进行分析，从中可以找出比赛对手的一些特点，这样就可以有针对性的进行一些战术调整了。

- 在线指挥球队

教练还可以用于在线比赛中协助球队更好的进行比赛。由于教练可以比球员获取更精确的信息，同时也没有实时决策的限制。因此教练可以使用一些复杂的算法，进行一些复杂的长周期的决策。这有助于教练更好的判断和分析比赛的形势。因此可以利用在线教练可以有效的帮助球队自动适应对手，进行场上的动态战略调整。

仿真机器人足球 3D 比赛

12.1 3D 比赛介绍

12.1.1 研究概况

3D 仿真机器人足球是 2003 年新兴起的一项机器人足球竞赛,是当今人工智能领域里一个极富挑战性的高技术密集型项目,并于 2004 年 7 月首次加入 Robocup 比赛中。3D 技术不同于 2D 技术,它采用新的比赛平台,增加了高空球部分,能更真实地模拟现实生活中的比赛环境。目前,3D 技术尚处于初级研究阶段,但是其发展相当快,并且逐渐取代 2D 技术,成为研究的热点。

12.1.2 比赛规则 (2005 年)

3D 比赛环境目前采用 Linux 系统,仿真平台包括 rcssserver3D 和 SPADES (System for Parallel Agent Discrete Agent Simulation) 以及多种库。其规则如下:

1. 球员部分:

a. 任何球员 (agent) 不得破坏和跟踪 SPADES 的运行,否则视为非法球员。也就是说,所有的调用必须是动态的,除非你的球员以单线程 (进程) 运行,否则不可以静态连接来执行。

b. 所有球员之间的通讯只能通过 SPADES/rcssserver3D。

c. 每一位球员将以 <agent type external> 的格式向 SPADES 球员数据库提供入口如果一个队被怀疑违反上述规则,组委会将有权请求查看源码。

2. 比赛结构:

a. 所有的比赛将采用人工裁判。

b. 比赛分上下两个半场,每半场 5 分钟。在特殊情况下会有加时。

c. 在比赛期间,可以修改原来程序或者提交新的程序,如果出现风险,自己承担。

d. 比赛得分规则:胜一局 3 分,平局各得一分。如果有球队违规,则以 0:3 或者以本轮中两队最大的进球差 (大于 3 球) 退出本场比赛。

e. 参看 agentdb.xml 例子提供相似的球员数据库文件包,放在根目录下

(例如: /home/[teamdir]/agentdb.xml)

f. 球员数据库文件将会被你所在组中不同的用户使用,所以你的文件必须设置读写控制。

g. 如果你的程序不能运行,组委会不会修复。

3. 比赛程序:

a. 比赛由三轮组成。

b. 第一轮由组委会根据上一年的比赛结果选定种子队,随机分散到不同的组中。

c. 第一轮将有 6 个组,每组 5 支队。采用循环赛制,每组积分高的前两名进入第二轮 (如果平局,参看相应处理)。

d. 第二轮将有 2 个组，每组 6 支队。同样采用循环赛制，每组积分最高的进入半决赛（如果平局，参看相应处理）。

e. 最后一轮是决赛，每一场必须有胜者。如果没有，就进行三个加时比赛，采用金球规则（谁先进球谁胜利）。如果还没有胜者，将人工投币决定胜负。

4. 平局规则：

在前两轮中，平局队伍数 $n \geq 2$ ，将采用下述规则：

a. 得分情况

b. 相互比赛结果：

如果 $n=2$ ，有利于两队相互间比赛的胜者。

如果 $n>2$ ，有利于同其它队比赛得分最高者。

c. 本轮的净胜球。

d. 平局队伍之间的进球差。

e. 如果 ($n>2$ ，本轮所有的进球分数（包括平局队伍之间的进球分数）。

f. 本轮所有得分。

g. 如果 $n=2$ ，将结合金球规则进行三个加时的比赛决定胜者。如果 $n>2$ ，平局队伍将以 3~6 数字进行随机统一编号。编号后的队伍以除以 3 的标准选出分组，每组将结合金球规则进行三个加时的比赛决定胜者。如果采用这种办法仍然不能判断胜负，将采用人工投币决定。

5. 犯规

通常，踢球时自动由 3D soccer server 检测犯规情况。有时，一些犯规仅能由人工裁判检测出来，这些人工裁判被授予对犯规队的自由踢球的权力。一些需要人工裁判检测的犯规如下：

a. 一方队员围着球以致对方球员不能接触到球。

b. 一方的许多队员挡住球使之不能对方球员被踢进。

c. 一方球员故意挡住对方球员不让其前进。

d. 其它一些故意妨碍公平竞赛的事项也会在请示犯规委员会后被判犯规。

6. 有效比赛

a. 球员连接失败

在一些情况下，球员由于自身的一些原因不能启动比赛，将采用一下程序：

(1) 球员不能在比赛 30s 前连接到仿真器，比赛将重启三次。

(2) 如果不能解决问题，在征得比赛另一方同意后，可以在 2 分钟内小范围修改源代码或者二进制码。

(3) 如果修改后仍不能运行，比赛将被取消，该方将以 0:3 的比分失败。否则比赛顺利进行。

b. 服务器连接失败

如果在比赛中，服务器失败，比赛不会偏袒任何一方，人工裁判将就比赛延长的问题请示组委会。如果组委会决定停止比赛，本场比赛将重新启动。

7. 远程参赛

一般说明：远程参赛只能用于一些极端情况。组委会没办法为远程参赛队伍就启动过程找问题，因此，远程参赛队伍必须确保第三方（例如：组委会）能简单又稳定地在一个有可能不同于自己编译的平台上运行其代码。如果不能顺利运行，将取消比赛资格。

12.2 3D 机器人足球平台介绍

3D 机器人足球仿真系统使用的是 rcssserver3D（以下简称 Server）和 SPADES（System for Parallel Agent Discrete Agent Simulation）组成的一个系统。由于这个系统分成了这两个层次来处理，所以仿真的过程就比以前复杂一些，再加上增加了第三维，以及这个系统更接近现实的物理模型，使得我们必须对整个系统由一个很好的认识才能够正确无误的设计我们的 agent 出来，因此必须对系统的各个部分以及它们之间的联系有一个很好的认识。

SPADES 是一个可以用来做各种多智能体（Multi-Agent）仿真的平台，主要负责的是一般仿真平台中都有的比较底层的模拟和通讯工作。对于仿真部分，它是以事件驱动的方式进行的，即：接受 agent 的动作请求事件→根据请求改变仿真世界→将仿真世界的信息发送给 agent。对于通讯部分，它主要负责 Server 与 agent 之间的通讯，同时也负责监视 agent 的（思考）计算所消耗的时间。

Server 的基础和核心是 SPADES。Server 的所有代码都是实现 SPADES 提供的接口，相当于是 SPADES 的一个插件。Server 的运行是靠 SPADES 调用这些接口来完成，所以要了解 server 就是要了解这些接口的功能。

下面是 SPADES 与 server 以及 agent 之间的关系示意图：

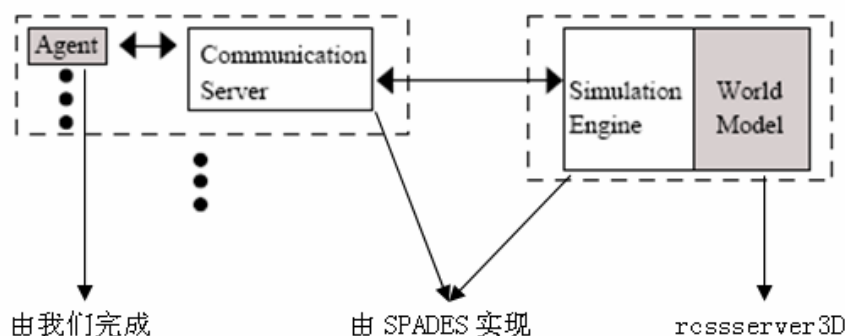


图 12.1 SPADES、Rcssserver3D 和 agent 的关系示意图

作为 agent，我们要完成的工作是：从 commserver 接受信息，然后处理这些信息，并作出决策得到要执行的动作，然后把动作消息通过 commserver 发送出去。这些消息都是字符串。

下面我们分部分详细的了解一下整个系统是如何运作的，有什么特点，这些特点可以给我们的 agent 带来什么样的影响等。我们按照系统从底层到上层的顺序来看，即 SPADES → Server → Agent。

12.2.1 SPADES

12.2.1.2 SPADES 概述

SPADES 是为 agent-based 分布式仿真而设计的一个中间部件。它主要是为了方便以 agent 的思考为研究对象的人工智能研究而开发，从而使得做这样一个仿真简单方便，高效而且可以重复的再现一个特定的场景。

SPADES 中的 agent 被定义为一个从仿真中获取信息，通过一定的计算做出（行为）反应得计算实体。SPADES 为我们提供了如下的功能：

提供对现实中信息获取、思考以及动作执行的延迟时间的模拟的功能。其中思考延迟是通过监视 agent 思考（计算）所占用的时间来实现的。

允许 agent 运行在用网络连接的多台机器上，所有的网络通讯以及与之相过的一些问题都有 SPADES 完成，不论 agent 分布在几台机器上都不用做任何改动。

仿真的结果不会因为网络延迟，机器的负载而有任何影响，这些只会影响到仿真的速度。

SPADES 对于 agent 没有任何结构上的要求，agent 可以用任何可以使用管道的编程语言编写。

Agent 的动作不需要同步。

SPADES 还提供了各种 log 的记录功能，对程序运行过程中的一些错误、运行的状态等都可以做相应的记录。

上面介绍过整个 SPADES 系统是由四部分组成的：agent、CommunicationServer、SimulationEngine、WorldModel，其中 SimulationEngine 是 SPADES 的核心部分，它负责着所有的仿真事件的处理以及所有网络通讯的协调。而 commserver 则是与 agent 密切相关的。每一台要运行 agent 的机器上必须有 commserver 运行。它负责着所有的与 agent 之间的通讯（通过 unix 管道），同时还负责着监视 agent 的计算时间从而得到 agent 的思考延迟。它与 SimulationEngine 之间则是通过 TCP/IP 连接来相互通讯。WorldModel 在我们所使用的这个仿真系统中就对应着 Server，它的主要功能是建立 3D 机器人足球仿真环境的模型，它必须给 SPADES 提供两个基本的功能：把仿真环境的状态模拟到任何一个将来的时刻，还要能模拟事件的发生（即根据事件的发生相应的改变受事件影响仿真环境的状态所发生的改变）。在机器人足球这个系统中，这两个功能重要就是建立足球场上的物理环境模型以及处理足球比赛中所发生的事件（包括 agent 的感知和动作的定义）。下面关于 Server 的部分将具体的介绍 Rcserver3D 是如何来实现这些的。最后的 agent 就是我们要实现的部分。

SPADES 仿真既是连续的又是离散的。所谓连续的就是指系统按照一定的时间片不断的向前模拟，改变环境的状态。而离散是指系统同时还处理一些可以发生在任何时刻的时间，并根据时间的结果改变环境状态。这两种方式分别是对 WorldModel 和 Agent 而言的。对于 WorldModel 来说，SPADES 就提供了连续的改变环境状态的方式，这样对于模拟一些连续的过程特别是像物理运动这样的过程是非常好的。对于 Agent 来说则是离散的，这样就使得 agent 只需要对一些离散的时间作处理而不需要在固定时间做出动作，同时 agent 也可以随时发出一个动作请求，从而能在几乎连续的时间上作出动作反应。

12.2.1.2 思考周期

SPADES agent 的动作一般应该是 Sense-Think-Act 循环的模式，我们可以把一个循环看成是 agent 的一个思考周期。下面是思考周期的一个示意图。

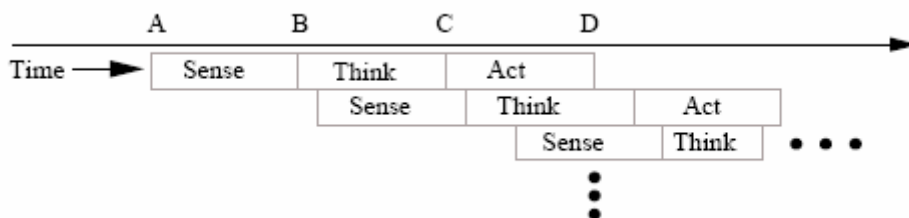


图 12.2 Agent 思考周期示意图

Agent 总是按照下面的步骤完成计算：

1. 先等待接受一个感知信息

2. 然后根据感知信息计算，决定一组动作，并把它们发送到 commserver
3. 最后发送一个思考完毕的消息。

SPADES 严格限制 agent 只能对在收到感知信息后做出决策以及发动作，所以系统提供了一个特殊的动作——请求 time notify。Time notify 实际上是一个空的感知信息，它的作用就是使 agent 能够在任何一个需要的时间发出动作，即使此时没有普通的感知信息。

Sense-Think-Act 循环中的一个重要的概念是延迟时间。这个时间是用来描述过程中各个阶段完成所需要的时间，例如，在现实中，感知需要用传感器来采集信息，有时可能还需要一些处理才能输出，所以从开始采集数据到输出数据就一定有一个延迟，同样的，思考，执行动作等都需要一定的时间，这些就是整个循环中的延迟时间。如上面的图中所示，A、B、C、D 是四个时间点，A 代表了感知信息的开始采集的时刻，B 代表感知信息输出和思考开始的时刻，C 代表思考结束和开始发动作的时刻，D 就是经过延迟后动作生效的时刻。在这四个点中间就分别是三个延迟，A 和 B 之间的是感知延迟，B 和 C 之间的是思考时间，C 和 D 之间的就是动作执行延迟。另外一点，两个思考周期（即 Sense-Think-Act 循环）之间是不允许有重叠的，即一个思考周期只可能是在前一个周期结束之后的某一个时间开始。

在这里，思考延迟就是 agent 做出决策并发出动作所需要的时间，这个时间由 commserver 通过监视 agent 的运行时间来获得。当收到思考完成的消息（Done thinking message）时，commserver 计算 Linux Kernel 提供的 agent 所占用的 CPU 时间计算动作延迟，并把延迟换算成仿真时间（spades time）发送给 simulation engine，用来模拟 agent 的思考延迟。在同一个思考时间段内发出的动作最后都将打上相同的时间戳。

12.2.1.3 事件

上面提到过，SPADES 被设计为支持在多台机器上共同运行一个仿真。其中一台机器在中心，负责协调、通讯、以及运行 simulation engine 和 worldmodel，而在其他的机器上则运行着 agents，这些 agents 必须由各台机器上的 commserver 启动，并由 commserver 来监视 agent。

对于 agent 来说，仿真过程实际上是对一系列的事件的处理的过程，这里的事件就是仿真的最基本的对象。Worldmodel 通过将仿真时间增加到一个特定的时刻，并实现这段时间内应该发生的事件（即对仿真的环境的状态作相应的改变）。

SPADES 提供了一些基本的事件类型，任何其他的事件都应该是从这些类型中派生出来的。Fixed agent 事件是一类特殊的事件，它被用来是多个 agent 的行动多一些并行性。Fixed agent 事件有如下的特点：

- * 它们不依赖于当前的仿真状态
- * 它们只影响单个的 agent，可以使通过发送一个消息给那个 agent
- * 感知事件和 time notify 事件都是 fixed agent 事件

Fixed agent 事件是唯一可以让 agent 开始一个思考周期的事件，但是它们不是一定要开始一个思考周期

- * 将 fixed agent 事件同其他的事件区分开就可以提出保证事件执行的正确性的条件：
- * 所有非 fixed agent 事件都必须按时间顺序执行
- * 所有发送感知信息给 agent 的事件都是 fixed agent 事件
- * 所有关于某一个特定 agent 的 fixed agent 事件必须按时间顺序执行

有了这些限制，SPADES 就能够保证事件执行的正确性。下面的图给出了各种类型的时间的关系：

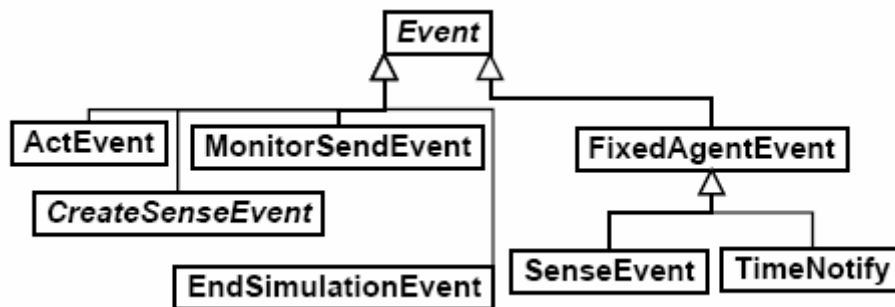


图 12.2 SPADES 事件层次图

Event 是事件的最基本的类型，所有的其他事件类型都有它派生出来。它定义了事件的一些基本属性，如事件发生的时间，事件的顺序（优先级）等。

ActEvent 当 agent 发出动作消息时，就会有这种消息产生。它是由 Worldmodel 的 *parseAct* 方法在收到 agent 的动作消息的时候生成的。它保存了触发这个事件的 agent 的信息 (id)。

CreateSenseEvent 负责产生 agent 的感知信息。它同样保存了 agent 的 id，这个感知信息最后将发送给这个 agent。

EndSimulationEvent 这个事件的执行将开始发出结束消息，通知所有 agent 仿真结束

FixedAgentEvent 前面已经介绍过这种类型的事件了，这里主要介绍它的两个重要的子类型：*SenseEvent* 和 *TimeNotify*：

SenseEvent 所有发送给 agent 的感知信息都必须通过这个事件来完成，它的执行就会产生一个将要发送给 agent 的感知信息。它由一个叫做 *ThinkingType* 的变量，给出了这个感知信息的类型，可能的类型有如下一些：

TT_Invalid 无效的

TT_Regular 一个正常感知信息，它将开始一个思考周期，开始计算思考时间

TT_Untimed 同上，但不计思考时间

TT_Not 一个不能开始思考周期的感知信息，或称为通知。Agent 不能在收到此消息后进入思考周期，处理这个消息所消耗的计算时间将被累计到下一个感知信息开始的思考周期中

SenseEvent 中有两个时间，分别是它产生的时间和 agent 可以开始思考的时间。*SenseEvent* 中还包含着发送给 agent 的感知数据，这些数据的结构可以由 Worldmodel 任意的定义了我。

TimeNotify 当 agent 请求一个 time notify 是就会产生这样一个事件。

12.2.1.4 Agent 接口

SPADES 提供了多种类型的 agent，由于对于我们来说主要是用其中的一种——外部 agent，因此我们集中的看一下这个类型的 agent。

外部 agent 在运行时单独作为一个进程，与 server 不在同一个进程，因此需要与 commserver 通讯，这个通讯是通过 linux 管道来完成。Agent 还要向 commserver 提供如下必要的信息以保证 commserver 能正确地启动 agent：

- (1) Agent 描述符
- (2) 用来给 agent 计时的模块
- (3) Agent 的工作目录
- (4) Agent 的可执行代码位置以及执行时需要的参数

从 Agent 角度来看：agent 进程由 commserver 按照 agent 提供的初始信息初始化，然后将从 commserver 接收 server 发来的初始化信息，接着当 agent 的所有初始化工作结束之后，就要回复一个初始化结束的消息给 server。这种机制允许 agent 在刚开始的时候可以做任意长时间的初始化工作（不能超过最大限制）。之后 agent 就进入循环等待从 commserver 接收各种各样的消息，处理之。如前面说的，这些信息分为两类，一类是开始一个思考周期的事件，另一类则不行。Sensation 和 time notify 是两个可以开始思考周期的事件，agent 可以发出动作，在完成思考和动作的发送之后必须发送一个思考完毕的消息给 commserver，以通知它 agent 已经完成思考。其他的感知信息都是非思考周期感知信息，agent 不能对它们作出动作反应，只能做一般的处理，并且这些处理时间都将算在下一个思考周期中。

SPADES 提供给 Agent 的基本动作主要有两种，一个是 act 消息，它的具体格式完全由 Worldmodel 定义，具体格式在后面介绍 Rcserver3D 的时候再作说明。另一个动作就是请求 time notify，使得 agent 能在预定的时间收到一个 time notify 消息。需要注意的是在每一个思考周期的结束，必须发送思考完毕的消息。

这里有 SPADES 的两个参数与 agent 的思考密切相关：max_agentq_trailing_time 和 max_timeout_trailing_time。这两个参数是与 agent 的思考计时有关的。在上面所说的 agent 的思考过程中，有可能 agent 对一个感知信息的处理时间过长，与下一个感知信息重叠，则这个感知信息就会转变成一个通知信息，如果是 time notify 消息也同样会变成一个通知消息。这里衡量 agent 思考超时的标准就是 max_agentq_trailing_time 和 max_timeout_trailing_time，他们分别对应着感知信息和 time notify 消息。

另外如果设置允许发送 agent 思考时间消息（Think Time Message），agent 就会收到这种消息报告上个思考周期 agent 所用的思考时间。

总结上面的这些 agent 的接口，agent 与 SPADES 之间的接口可以详细的说明如下：

Agent 与 SPADES 之间的通讯都是间接的通过 commserver 进行的，而 agent 与 commserver 之间的通讯则是使用 linux 管道进行。Agent 所接收和发送的消息都应该是长度前缀的字符串，这种字符串的特点是每个消息的前 4 个字节是用来保存这个消息的长度的，它是按网络字节流顺序来发送，并且这个长度不包括前 4 个字节。紧跟在前 4 个字节之后的数据就是这个消息的正文，消息正文的第一个字节是用来说明此消息的类型的。

下面按照 agent 的输入和输出分类说明 agent 接收和发送的消息的格式。

(1) Input (agent 可能接收到的) 消息格式：

- **Stime time data**

这是一个发送给 agent 的感知信息，它将开始一个思考周期。

第一个 time 是这个感知信息产生的时间，第二个时间是 agent 受到这个信息的时间，如果 SPADES 中的 send_agent_send_time 被设置为关，则第一个时间总是-1。Data 是由 Worldmodel 生成的任意格式的字符串数据，给出了 Worldmodel 由定义的感知的具体内容（内容由消息产生时的 Worldmodel 中的环境的状态决定）。Agent 对此消息可以回复动作消息，然后必须发送一个思考完毕的消息（Done Thinking message）。

- **Itime time data**

这个是一个通知消息（inform message）。它的意义和上面的 S 消息一样，只是它不能开始一个思考周期，也就是说在收到这个消息之后 agent 不能回复任何动作。它的产生原因如前面所述，可能是由于 agent 的思考时间过长而错过了上一个感知信息导致的，所以在收到这个消息时就应该注意是否 agent 的计算时间过长了。

- **Ttime**

这是一个 **time notify**，一个空的感知信息，它虽然没有给 **agent** 提供任何感知信息，但是它开始了一个思考周期，**agent** 可以在收到这个消息之后回复动作。它的产生只可能是 **agent** 在之前特意申请了这样一个 **time notify**。**time** 参数就是 **agent** 在之前申请的时候要求收到这个 **time notify** 消息的时间。注意这个时间与这个消息被 **agent** 收到的时刻没有关系。

- **Otime**

这是一个过期的 **time notify** 消息，它不能开始一个思考周期，所以 **agent** 也不能对他回复任何动作。

- **X**

这是一个结束消息，告诉 **agent** 应该结束程序了。在收到这个消息之后不会有任何消息会发送给 **agent**，**agent** 也不能回复任何东西。

- **Ddata**

在仿真的最开始 **agent** 会收到这个初始化消息，在正常启动的情况下 **data** 是空的，**agent** 在初始化结束后应该回复一个初始化结束的消息（**initialization done message**）。

- **Ktime**

这个消息告诉 **agent** 上一个思考周期的思考用了多长时间。只有 **SPADES** 的 **send_agent_think_times** 参数设置为 **on** 的时候才会有这个消息发送给 **agent**。

- **Etoken**

如果 **agent** 发出错误格式的消息或者发生其他错误，会有错误报告消息发送给 **agent**。这个就是给 **agent** 的错误报告消息，**token** 的不同值给出了错误的类型：

no_token_on_line **agent** 发送了一个空消息给 **commsserver**

act_when_not_thinking **agent** 在思考周期之外发送了动作

rtn_when_not_thinking **agent** 在思考周期之外请求 **time notify**

bad_time_in_rtn 请求 **time notify** 时的时间格式错误

done_thinking_when_not_thinking **agent** 在思考周期之外发出 **done thinking message**

bad_token **agent** 发送的消息的第一个字符不是合法的字符

init_done_when_not_init **agent** 在非初始化阶段发送了初始化结束的消息

(2) Output (**agent** 的合法的输出) 消息格式：

- **Adata**

这是 **agent** 发出的动作的消息。**data** 是由 **Worldmodel** 任意定义的格式，同样使用长度前缀的字符串，所以对长度没有限制。

- **Rtime**

请求 **time notify** 时发出的动作。**time** 是 **agent** 要求收到 **time notify** 消息的时间，这个时间是仿真时间（**SPADES time**），是一个整数。

- **D**

思考完毕消息（**done thinking message**）。**Agent** 在每一个思考周期的最后都必须发

送这个消息，包括有 time notify 开始的思考周期。

- **X**

退出消息，告诉系统这个 agent 将要退出。这个消息既可以在思考周期中发出，也可以是在非思考周期中发。如果是在思考周期中发送，则这个消息同时也有 ‘D’ 消息的作用，之后 agent 就不会再收到任何其他的信息。

- **I**

在初始化完毕时回复，说明 agent 初始化结束的消息。

到这里我们已经基本了解了 SPADES 中的一些基本概念，SPADES 是如何工作的以及 SPADES 的 agent 应该如何与 SPADES 交互。Agent 必须按照 SPADES 规定的方法与 SPADES 通讯，因此需要一个专门的模块来完成这个通讯工作。同时，在 agent 的思考、动作等方面也要符合 SPADES 的要求，这样的 agent 才能在 SPADES 的基础上正确运行。在接下来的一节我们就要来看一下 Server 方面了。

12.2.2 Rcserver3D

Rcserver3D 实际上是实现了 SPADES 中的 Worldmodel。从上面对 SPADES 的分析我们已经可以了解一点 Server 的信息了，这些信息反应了整个 3D 仿真的运行基础，而关于 soccer simulation 3D 的具体信息就要从 Server 中来获得了。

先看一下 Server 的整体结构：

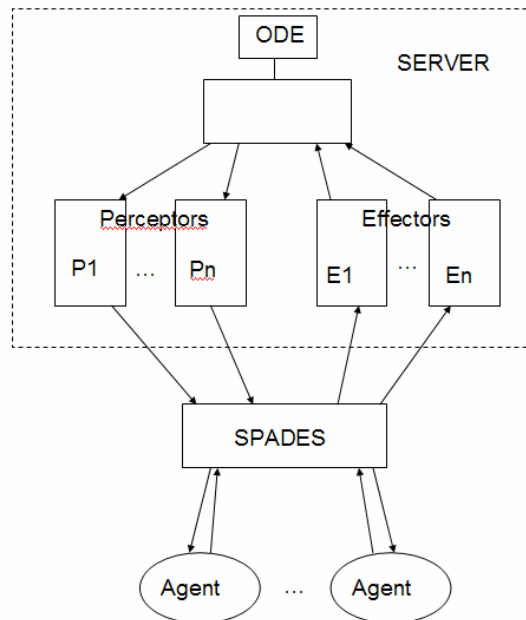


图 12.4 Server 的整体结构

我们来看一下 Server 中与 SPADES 密切相关的一个问题——Worldmodel 与 agent 的接口：Worldmodel 与 agent 之间的交互是通过事件来完成。从 Worldmodel 的角度来看，在如下图的 agent 的 Sense-Think-Act 周期中，会有图中所示的事件发生。这个 Sense-Think-Act 周期是由 CreateSenseEvent 事件的执行开始，它会产生一个 SenseEvent 事件，在经过感知延迟之后，SenseEvent 事件被执行，感知信息就在此时发送给 agent，同时开始一个思考周期；

agent 在经过思考之后发出一组动作，这组动作将根据 agent 思考所用的计算时间打上时间戳并发送到 Worldmodel，这个时候 Worldmodel 使用 `parseAct` 方法处理这个动作消息，并将它转换成一个 `ActEvent` 事件，然后经过一段时间的动作延迟，这个 `ActEvent` 事件就被执行，这次 Sense-Think-Act 循环也就结束了。

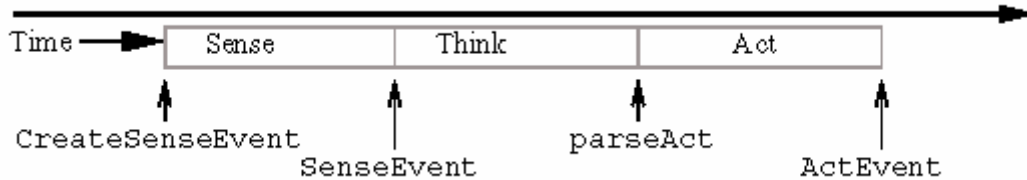


图 12.5 Sense-Think-Act 示意图

Server 的作用就是定义这些事件的具体内容，当然上面这些事件只是 Server 要实现的事件中的与 agent 相关的一部分，还有更多事件时处理其他 soccer 比赛中出现的各种事件的。Server 对所有的这些事件的定义和处理，就构成了 soccer 比赛的基本模型。

主要的模型是球场的几何模型、物理模型，比赛的规则模型，在有一些就是管理 monitor 和 agent 的一些模块。其中球场的几何和物理模型主要有一个叫做 ODE 的库来实现。ODE 的全称是 open dynamics engine,即开放动力学引擎，它被设计来做一些对真实物理环境的模拟，对各种几何体、复合连接体以及表面材质相关的碰撞、摩擦等都能处理，因此在这里是用这个库就可以使我们的仿真比赛具有更加真实的物理环境，同时也增加了 agent 控制的难度。Server 在收集了各种场上的有关信息（几何和物理方面的）后就会转交给 ODE 来处理，一般是计算一定的时间间隔后物理世界的状态，然后将 ODE 计算返回的结果保存，这样就模拟了随时间的推移，整个仿真中的物理运动。由于了解这一部分的模型对 agent 的运动的控制至关重要，因此我们对此作了详细的研究和分析，通过理论推测和实验验证的办法来了解这个物理模型，详细情况参见其它文档。Server 定义的一些基本的环境说明如下：

比赛环境相关的一些参数：

场地：标准 FIFA 足球场大小，100 到 110 米长，64 到 75 米宽（随机），球门宽度 7.32 米，目前由于 agent 的高度很矮而且不能跳，所以球门高度为 0.5 米，而不是标准 FIFA 球门的 2.44m；球场重力加速度为 9.81m/s²

足球：直径 0.222 米，重量在 0.41 到 0.45 千克之间。

球员：目前的球员是用球体来表示的，直径为 0.44m，重量 75kg 所有的球员的能力都是相同的。

每个球员是一个模拟的机器人，Server 给每个球员提供了各种功能部件以实现球员与 Server 之间的交互。下面我们就来看一下这些接口设备，即下面将要介绍的 Effectors 和 Perceptors（操纵装置和感知装置）：

Create Effector

当一个球员最初连接到 Server 的时候，只是一个基本的 SPADES 的 Agent，并没有代表任何的实体，这个时候的 Agent 能做的唯一的一件事就是调用 Create Effector 这个部件，通过它就可以给这个 Agent 申请一个具体的“身体”，这个“身体”对于不同的球员可以是不同类型的，这样就可以模拟不同的机器人类型。目前只有一种固定类型的机器人，因此 Create Effector 不接受任何参数而直接使用默认的机器人类型。

用法和举例：(create)

Init Effector

提供对球员的初始化功能。在这里将对其它的一些 Effectors 和 Perceptors 进行初始化，在进行这些初始化以前，这些部件都是不可用的。这个初始化过程中还负责着队员的所属球队名和球员号码的设定。

用法： `(init (unum <number>) (teamname <string>))`

使用举例： `(init (unum 7) (teamname RoboLog))`

Beam Effector

这个部件的效果跟 2d 机器人足球仿真 server 中的 move 命令类似，是用来把球员放置到一个指定的位置。它的目的在于在一些特殊的情况下把球员迅速移动到指定的位置，目前主要是用来在比赛前将球员在球场上放好。

用法： `(beam <x> <y> <z>)`

使用举例： `(beam -6.6 0 0)`

Drive Effector

这是球员的一个主要的功能部件之一，它是一个类似万向轮的驱动装置，可以对机器人施加任何方向的力，这样机器人就可以获得任何方向的加速度，并且可以轻微的跳起来一点。如果机器人离开了地面，则不能获得任何方向的力，即在空中的运动状态是不能由机器人自己主动的改变的。同时机器人也不可能在具有很大速度的时候突然停下来，但是可以通过施加反向的力来减速。如果不改变施加的力，机器人将一直受到前面施加的力；而当停止施加力（即施加的力为 0）时，机器人仍然会前进一段距离，但最终会由于地面的阻力而停止下来。使用 DriveEffector 将消耗电池，你可以通过 AgentStatePerceptor 获得电池的电量，如果没有电了，DriveEffector 就不会工作了。施加的力是以直角坐标系坐标(x,y,z)的形式设置的，力的大小最大值为 100。DriveEffector 施加的力是由一些误差的，可能达到每个轴 2% 的误差，误差的分布为均值为 0.0 的正态分布。

用法： `(drive <x> <y> <z>)`

使用举例： `(drive 20.0 50.0 0.0)`

Kick Effector

和 DriveEffector 一样，这是球员的另外一个重要的功能部件，它使球员具有踢球的能力。如果没有这个部件，球员也是可以通过撞球来让球移动，但是那样就无法利用增加的第 3 维的特性了。提供 KickEffector 就是为了提供把球踢到空中的功能。KickEffector 实际是在短时间内给球一个很大的加速度，使得球沿着球员和球的中心的连线飞出去。KickEffector 只有在进入它的作用范围（即踢球距离）之内的时候才会有效果。KickEffector 接受 2 个参数，一个是踢球的高度角，高度角被限制在 0 到 50 度角之间；另一个是踢球的力量的大小，它是一个百分比数值，表示用最大力量的 x% 踢球。KickEffector 不能控制踢球的水平方向，必须调整球员到适当的位置来控制踢球的水平方向。跟 DriveEffector 一样，当球员离开地面的时候 KickEffector 也不再有效果。

KickEffector 的误差参数:

水平角度误差: 均值为 0.0 方差为 $\sigma = 0.02$ 的正态分布

高度角误差: 均值为 0.0 方差在 0 或 50 度附近时为 $\sigma = 0.9$, 在中间时 $\sigma = 4.5$ 的正态分布

力量误差: 均值为 0.0 方差为 $\sigma = 0.4$ 的正态分布

用法: (kick <angle> <power>)

使用举例: (kick 20.0 80.0)

Vision Perceptor

我们的机器人通过 VisionPerceptor 可以获得 360 度全方位的视觉信息。VisionPerceptor 给我们发送所看到的场上的物体的列表, 这些物体包括其它球员、球以及场上的地标。目前场上有 8 个地标, 分别是球场的四个角以及四个门柱。在发送给我们的视觉信息中, 每一个物体都有如下几项: 离自己的距离, 水平平面的角度 (0 度指向对方球门), 高度角 (0 度表示水平)。VisionPerceptor 不发送物体的速度。所有的视觉信息是以模拟的装在球员中心的摄像机的视角给出的。

视觉的误差参数: 每次比赛摄像机的位置会有一个固定的误差 (-0.005m 到 0.005m), 这个误差在整个比赛中保持不变。动态误差都是均值为 0.0 的正态误差。距离误差的方差为 $\sigma = 0.0965$, 水平角度的误差方差为 $\sigma = 0.1225$, 高度角误差方差为 $\sigma = 0.1480$

视觉信息格式:

```
(Vision
  (<Type>
    (team <teamname>)
    (id <id>)
    (pol <distance> <horizontal angle> <latitudal angle>)
  )
)
```

Type 的可能值及相应的 id 值:

- 'Flag' 可能的 id 值为 '1_l', '2_l', '1_r', '2_r'

- 'Goal' 可能的 id 值为 '1_l', '2_l', '1_r', '2_r'

- 'Player' id 值为球员的队员号

视觉信息举例:

```
(Vision (Flag (id 1_l) (pol 54.3137 -148.083 -0.152227)) (Flag (id 2_l) (pol 59.4273 141.046 -0.131907)) (Flag (id 1_r) (pol 61.9718 -27.4136 -0.123048)) (Flag (id 2_r) (pol 66.4986 34.3644 -0.108964)) (Goal (id 1_l) (pol 46.1688 179.18 -0.193898)) (Goal (id 2_l) (pol 46.8624 170.182 -0.189786)) (Goal (id 1_r) (pol 54.9749 0.874504 -0.149385)) (Goal (id 2_r) (pol 55.5585 8.45381 -0.146933)) (Ball (pol 6.2928 45.0858 -0.94987)) (Player (team robolog) (id 1) (pol 7.3364337 5.86774)))
```

GameStatePerceptor

它将告诉球员比赛现在的状态。比赛开始时的第一个 GameState 信息将告诉球员一些比赛变量，如球的重量，场地大小等。

信息格式: (GameState (<Name> <Value>) ...)

Name 的可能值:

‘time’给出当前的时间，浮点数，单位秒。

‘playmode’用字符串给出当前的 playmode。

可能的 playmode 有:

```
"BeforeKickOff", "KickOff_Left", "KickOff_Right", "PlayOn",
"KickIn_Left", "KickIn_Right", "corner_kick_left", "corner_kick_right", "goal_kick_left", "goal_kick_right", "offside_left", "offside_right", "GameOver", "Goal_Left", "Goal_Right", "free_kick_left", "free_kick_right", "unknown"
```

视觉信息举例: (GameState (time 0) (playmode BeforeKickOff))

AgentState Perceptor

给出机器人当前的状态，即电池电量和温度信息格式:

```
(AgentState
  (battery <battery level in percent>)
  (temp <temperature in degree>)
)
```

举例: (AgentState (battery 100) (temp 23))

以上这些 Effectors 和 Perceptors 就是 Server 给球员提供的接口，通过这些接口，就可以完成球员与 Server 之间的交互了。

综合以上所有的信息，我们了解了整个系统的基本工作原理和相关的接口。接下来就可以看一下我们的球员应该怎么样来实现了。

12.3 WrightEagle3D Agent 程序设计

12.3.1 WrightEagle3D Agent 整体结构设计

通过上面的分析我们可以知道，要设计一个 3D 仿真系统的 Agent，必须做好两部分的工作，即与 SPADES 和 Server 都要有很好的交互。由于 Server 是建立在 SPADES 的基础之上的，因此一个良好的球员首先应该是一个符合 SPADES 要求的 Agent，然后才是与 Server 交互，即按照 Server 的要求处理它发动给我们的消息并发出 Server 允许的合法动作。这样 Agent 如图所示就应该分为两个层次，一个负责处理 SPADES 层上的事情，另一个负责处理 Server 层的事情。

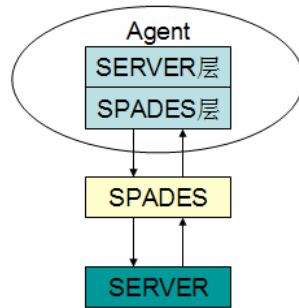


图 12.6 Agent 分层结构

SPADES 层的主要功能是完成 Agent 与 SPADES 之间的交互，使 Agent 是一个符合 SPADES 要求的 Agent。其主要工作包括：

按照 SPADES 的要求对 Agent 进行初始化

负责 Agent 与 SPADES 直接的通讯

处理 SPADES 发送给 Agent 的异常信息

SPADES 层将底层的初始化及通讯都封装起来，使上层不必关心与 SPADES 交互的一些细节，也不需要关心如何利用 SPADES 来实现与 Server 的通讯，而 SERVER 层在有了 SPADES 层的支持之后，就可以专心的处理 Server 的各种消息了。

SPADES 层首先要能够接受 SPADES 发送给 Agent 的消息，这就要求一个能够处理 SPADES 的 communication server 通过管道发送过来的消息的功能模块，这里我们用 commserver 来完成，commserver 同时还肩负着发送动作请求给 SPADES 的任务；接下来就是对收到的消息的处理模块，它要能够解析 SPADES 发送过来的消息，并做出必要的处理，保存相关的数据留给上层 SERVER 层使用。SPADES 不仅仅要能够处理这些消息，还要给 SERVER 层提供方便的接口让 SERVER 层能够得到当前的这些消息，并且；要能够发送 SERVER 层请求发送的消息。这里面有一个特殊的动作就是前面提到过的 RequestTimeNotify 动作，它要发送给的消息中包括一个时间，这个时间是 SPADES 中的 simtime，是以仿真周期为单位的一个时间，而 SERVER 层请求的时间则是按 Server 的时间给出的，单位是秒。这两者是不同的，中间需要有一个转换机制，因此需要有一个时间管理器，否则容易出现混乱。

根据上面给出的 SPADES 层的处理过程，给出这个层的结构如下图所示：

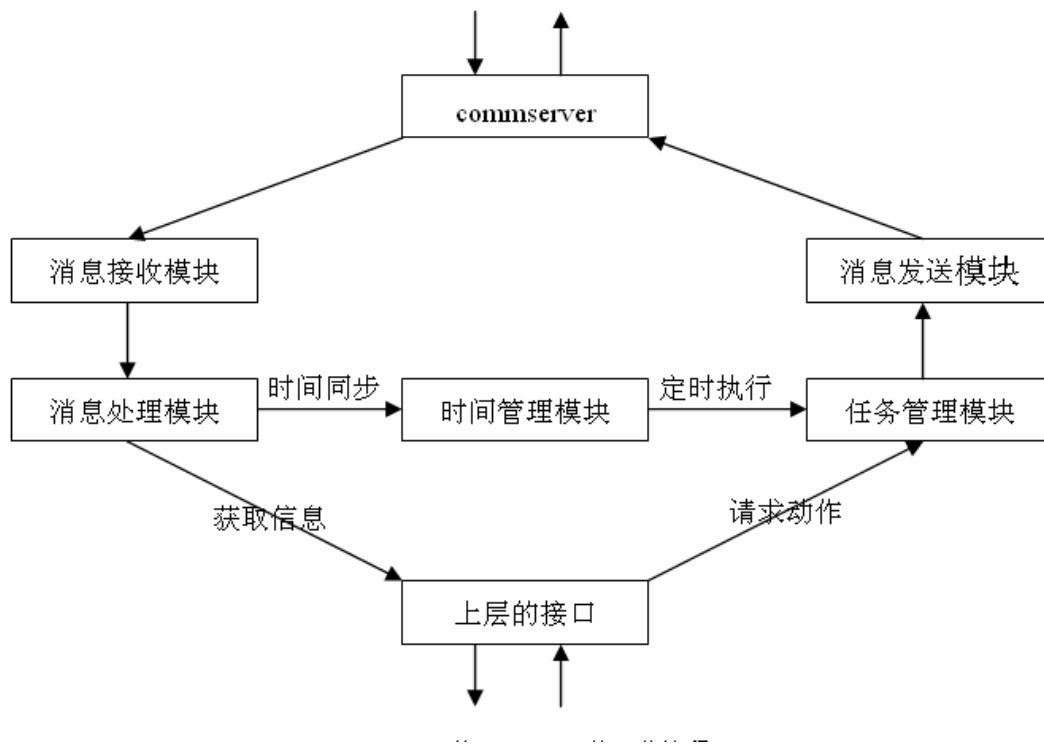


图 12.7 Agent 的 SPADES 层的工作流程图

SPADES 层的一般工作流程为：在仿真的最初几个周期内完成作为 SPADES 的 agent 应该做的初始化工作，接着进入数据接受和处理的循环，同时接受由 SERVER 层发送动作的请求，并在要求的时间发送出去。在每次对接受信息的处理工作中，SPADES 层要负责对其内部的时间作同步更新，任务管理模块则要保证 SERVER 层请求的动作在准确的时刻发送出去。其中 commserver 这个模块负责通过管道接收和发送长度前缀的字符串消息。通过这一层的处理，就将每个消息中 SPADES 定义的第一个字符以及某些消息中附带的 simtime 等信息都作了处理，并把每个消息中 SPADES 定义的 data 数据保存起来留给 SERVER 层来解析和处理。

接下来，SERVER 层的工作就是从处理 SPADES 层接收的消息中的 data 数据，并按照 Server 对这些消息的定义来处理，并做出决策，然后组装成由 Server 定义的动作消息，通过 SPADES 层发送出去。

首先是消息的处理，在这里主要是感知信息。如前所述，这个感知信息是由三部分组成的，它们分别由 VisionPerceptor、GameStatePerceptor 和 AgentStatePerceptor 产生，我们要分别从这几个部分中解析解析出相应的信息，为 agent 的决策提供基本的数据。在这里最重要的是视觉信息，由于视觉信息中有噪声，所以为了给 agent 提供准确的信息，我们在解析之后立即对这些有噪声的信息进行处理，尽量减少噪声。其他的信息就可以直接解析出来即可。

在这个解析的工作中需要一个数据区来保存结果数据，这些数据都是 agent 决策过程中要用到的，因此需要要把它们暂时保存起来，我们把这个数据区命名为 Worldmodel。Worldmodel 中的有一些数据是在初始化的时候就固定下来，而另一些信息比如视觉信息则是每次都需要更新的。

在解析完毕后，agent 就要根据这些信息作出决策，规划出一组动作来执行，这一块是整个 agent 的核心部分，相当于人的大脑，一般是作为一个独立的模块，结构可以多种多样，其主要功能简单的说就是利用上面的这些信息，为了达到一定的目的而决策出一组动作来完

成目的。思考模块的具体结构在这里就不详细介绍了。在 soccer sim 中，首先要能够控制 agent 自如的运动，因此在这里可以有一层专门用来完成各种基本的动作的控制，然后由更高层次的决策来决定用什么动作。

最后要做的事情就是把动作发送给 Server 执行，这里就要按照 Server 定义的通讯规则，发送字符串形式的命令来传送这些动作，为了简单，同时也为了避免低级错误，可以由专门的模块负责这些动作请求字符串的封装动作。

这样，SERVER 层的大致结构就出来了，就是如下图所示的结构：

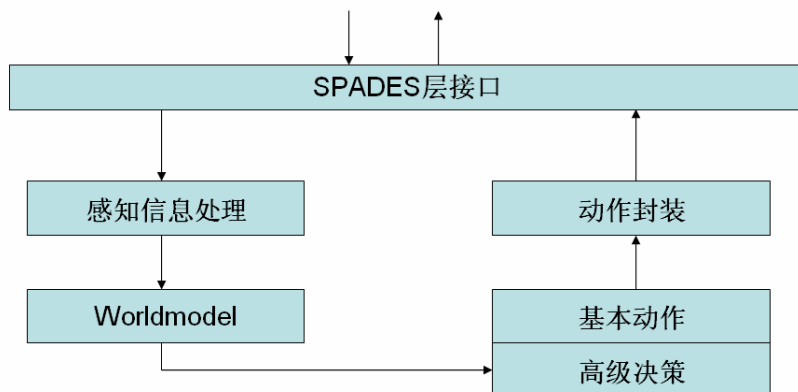


图 12.8 Agent 的 server 层结构图

感知信息处理模块负责这从 Server 发送给 agent 的感知信息（字符串）中提取出 agent 所需要的各种信息，其主要功能实质上是字符串解析。解析出的内容包括视觉信息、比赛状态、agent 状态三个方面，其中比赛状态、agent 状态可以不经处理直接保存下来，而视觉信息由于加入了噪声，根据需要可对这些信息作滤波处理之后再保存，也可能需要把收到的原始信息保存起来，这个要根据决策的需要而定。如果要保持这个模块的通用性则可以两种信息都保存，并提供两种接口分别访问原始数据和处理过后的数据。

Worldmodel 模块实际上是一个保存各种信息的数据存储区，主要是用来保存感知信息处理模块解析和处理之后的各种数据，以供 agent 思考的时候使用。

动作封装模块把思考决定的动作按照 Server 规定的命令形式组装成命令字符串，并把组装好的字符串交给 SPADES 层发送出去。

Agent 思考模块其实应该是整个 agent 最复杂的部分，应该作为独立的一层，再这里为了简化层次结构，将这一层合并到 SERVER 层中，用一个模块来代替它。它的任务是通过计算或者逻辑推理来指导 agent 做出一系列的动作，最终完成某一个特定的目标。在这里我们针对 soccer simulation 这个具体的目标给出了两个基本的层次，分别用来完成基本的动作控制和高级的思考决策。

12.3.2 3D 仿真机器人的视觉

在新的 3D 仿真机器人足球比赛中，机器人的视觉已经完全拓展到三维空间了，机器人所看到的实体都处在一个三维的空间中，而且视觉信息信息也存在着很大的随机误差，因此 3D 机器人的视觉研究又产生了很多新的问题，带来了新的挑战。在这一小节中，我们将介

绍 3D 机器人的视觉信息模型，视觉模型，以及机器人将如何处理这些视觉信息。

12.3.2.1 server 的视觉信息模型

我们知道一个用摄像头定位的真实机器人的定位方法：拍摄定标图像，通过分析图像来得到与定标的距离矢量，而定标的坐标矢量一般是固定的，把两个矢量相加就得到了机器人的位置。

在 3D 仿真系统中，Agent 定位的原理与上面差不多，方法更简单。在 server 中没有仿真一个摄像头，而是直接给出了距离矢量，这就不需要经过图像分析了，但为了与现实中视觉相似，server 对这个视觉矢量加入了误差。server 对视觉信息的模拟方法如下：在 server 中维护着 agent 的一个准确位置 X （笛卡尔坐标系），用定标的位置矢量 Y 减去自己的位置矢量，就得到自己到定标的距离矢量 D ，对这个矢量的三个分量上加上均匀分布的误差，然后将矢量 D 转换成球坐标矢量 L ，再对 L 的三个分量分别加上不同方差的正态随机误差。这样 Agent 的视觉就有了很大的随机误差，由于误差有正态分布，而不再是具有门限了，因此出现很大的视觉偏差就变得可能。这一定程度上模仿了人的视觉，人看到东西也不能精确的确定距离，而是估计一个距离出来，这个距离存在着随机误差。下面用数学推导来描述上面的视觉信息模型：

笛卡儿坐标与球坐标的相互转换。

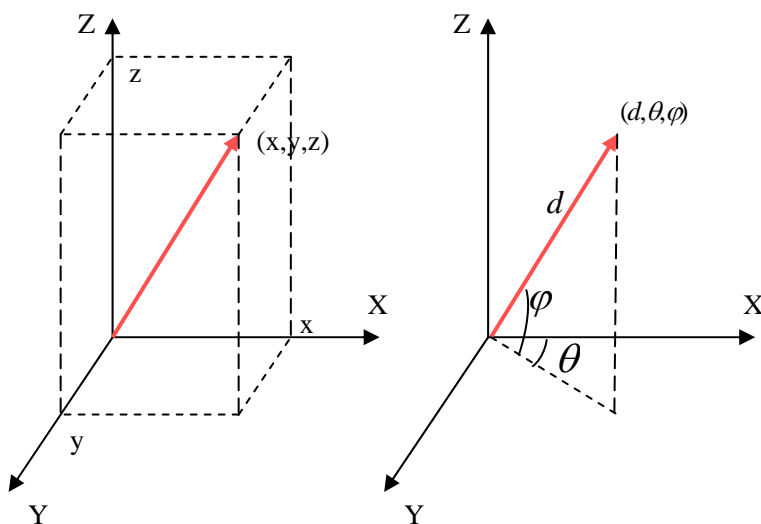


图 12.9 笛卡儿坐标系与球坐标系

这两个坐标系可以相互转换，从笛卡儿坐标系转换到球坐标系的公式如下：

$$d = \sqrt{x^2 + y^2 + z^2}$$

$$\varphi = \arctan\left(\frac{y}{x}\right)$$

$$\theta = 90^\circ - \arccos\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right)$$

而从球坐标到笛卡儿坐标的转换公式为：

$$x = d * \cos \varphi * \cos \theta$$

$$y = d * \cos \varphi * \sin \theta$$

$$z = d * \sin \varphi$$

在 Server 中，两个坐标是保存在一个变量中的，这个变量的类型为 ObjectData，Server 还提供了函数实现上面的坐标转换。

视觉信息的误差模型

设 X 是 Server 的一个视觉信息，Server 对 X 的处理如下：

e_x, e_y, e_z 是在区间 $[-\text{cal_error_abs}, \text{cal_error_abs}]$ 均匀分布的变量

首先令 $X = X + (e_x, e_y, e_z)$,

将 X 转换为球坐标 $X \rightarrow \bar{X}$,

设：

$$\Delta d \sim N[0, \text{sigma_dist}]$$

$$\Delta \varphi \sim N[0, \text{sigma_phi}]$$

$$\Delta \theta \sim N[0, \text{sigma_theta}]$$

$$\text{令 } \bar{X} = \bar{X} + (\Delta d, \Delta \varphi, \Delta \theta)$$

最后得到的 \bar{X} 就是要发送给 Agent 的矢量。

12.3.2.2 Agent 的视觉建模

众所周知，在人的视觉系统中是需要一个参照系的，通常人都会以自己为参照，来确定其他物体的位置。但在 Robocup3D 中，所有的 Agent 及其他实体都为了统一建模共用同一个参照系，这个参照系是以球场中心为坐标原点，从己方球门指向对方球门为 x 轴的右手系笛卡儿坐标系，所有其他物体的位置都是用一个笛卡儿坐标表示，也就是到球场中心的矢量坐标。这样，所有的实体的位置具有统一的格式，计算时不需要通过坐标变换来统一数据。但是，Agent 从 server 收到的视觉信息是以 Agent 自己为参考点的球坐标矢量，为了计算出位置，要先把 Server 发过的视觉信息转换为直角坐标，让后在加上 Agent 自己的坐标矢量。这就是一个简单的定位方法，这样计算出来的位置会有很大的误差。下面将介绍一种更好的方法来处理视觉信息。

12.3.2.3 Agent 的视觉信息处理及定位计算

上面介绍了 server 的视觉信息模型以及 Agent 对整个世界是怎么建模的，由于 server 的视觉信息加上了随机误差，因此 agent 对收到的视觉信息不能直接利用，而需要先进行一些处理。我们知道，Agent 从 server 收的视觉是一组向量，每个向量对应着一个实体到 Agent 的距离矢量，其中有些实体的坐标是确定的，包括球场的四个角标，球门的四个门柱，我们称之为定标。首先，Agent 利用这些定标来实现自己的定位：假设在 n 时刻，Agent 收到的

来自一个定标的矢量序列为： X_0, X_1, \dots, X_n ；如果我们只用 X_n 来定位，会存在较大的误差，但我们把以前的矢量序列以某种方法累加起来定位，则有可能消除随机的误差。这种方法就是卡尔曼方法，实际上把矢量序列合起来并不容易，还需要考虑到每步的动作更新，这些都将在下节的卡尔曼滤波中介绍。

12.3.3 卡尔曼滤波

12.3.3.1 卡尔曼滤波理论简介

卡尔曼滤波理论是由卡尔曼 (R.E.Kalman) 于 1960 年首次提出的，是一种线性最小方差估计。卡尔曼滤波具有如下特点：

(1) 算法是递推的，且使用状态空间法在时域内设计滤波器。所以卡尔曼滤波适用于对多维随机过程的估计。

(2) 采用动力学方程即状态方程描述被估计量的动态变化规律，被估计量的动态统计信息由激励白噪声的统计信息和动力学方程确定。由于激励白噪声是平稳过程，动力学方程已知，所以被估计量既可以是平稳的，也可以是非平稳的，即卡尔曼滤波也适用于非平稳过程。

(3) 卡尔曼滤波具有连续型和离散型两类算法，离散型算法可直接在数字计算机上实现。正由于上述特点，卡尔曼滤波理论一经提出立即受到了工程应用的重视，阿波罗登月飞行和 C-5A 飞机导航系统的设计是早期应用中的最成功者。目前，卡尔曼滤波理论作为一种最重要的最优估计理论被广泛应用于各种领域。其中，最典型的应用就是机器人视觉处理及定位，我们在计算机仿真的条件下，要用离散的卡尔曼滤波器。

12.3.3.2 离散型卡尔曼滤波基本方程

设 t_k 时刻的被估计状态 X_k 受控制输入 U_{k-1} 和系统噪声序列 W_{k-1} 驱动，驱动机理由下述状态方程描述

$$X_k = A_{k,k-1} X_{k-1} + B_{k-1} U_{k-1} + \Gamma_{k-1} W_{k-1}$$

对 X_k 的量测满足线性关系，量测方程为

$$Z_k = H_k X_k + V_k$$

式中： $A_{k,k-1}$ 为 t_{k-1} 时刻至 t_k 时刻的一步转移矩阵； B_{k-1} 为控制驱动阵； Γ_{k-1} 为系统噪声驱动阵； H_k 为量测阵； V_k 为量测噪声序列； W_{k-1} 为系统激励噪声序列。

同时， W_k 和 V_k 满足

$$\left. \begin{aligned} E[W_k] &= 0, \text{Cov}[W_k, W_j] = E[W_k W_j^T] = Q_k \delta_{kj} \\ E[V_k] &= 0, \text{Cov}[V_k, V_j] = E[V_k V_j^T] = R_k \delta_{kj} \\ \text{Cov}[W_k, V_j] &= E[W_k V_j^T] = 0 \end{aligned} \right\}$$

上式中： Q_k 为系统噪声序列的方差阵，假设为非负定阵； R_k 为量测噪声序列的方差阵，

假设为正定阵。即 W_k 和 V_k 服从下面的分布：

$$P(W) \sim N(0, Q)$$

$$P(V) \sim N(0, R)$$

我们用 \hat{x}_k^- 表示由前一步的值经过一步状态转移后的估计值，用 \hat{x}_k 表示在第 K 步视觉下的估计值。我们可以定义两个误差：先验误差和后验误差

$$e_k^- \equiv x_k - \hat{x}_k^-$$

$$e_k \equiv x_k - \hat{x}_k$$

则先验误差的协方差为：

$$P_k^- = E[e_k^- e_k^{-T}],$$

而后验误差的协方差为：

$$P_k = E[e_k e_k^T],$$

\hat{x}_k^- 可由前一步状态 \hat{x}_{k-1} 用一个状态转移得到，这个过程的控制量为 u_k ，这一步叫做动作更新。

$$\hat{x}_k^- = A_{k-1} \hat{x}_{k-1} + B u_k$$

而 \hat{x}_k 可由 \hat{x}_k^- 和视觉到 \hat{x}_k^- 的差的线性组合求得。

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H_k \hat{x}_k^-)$$

上面的 K_k 叫做增益阵， K_k 的选择对滤波是很重要的，一般要求 K_k 使的上面的后验误差协方差最小。将上式代入协方差公式，可求出 K_k 的表达式。当然，不同的要求推出的 K_k 是不同的，下面是一种常见的 K_k 形式：

$$K_k = \frac{P_k^- H_k^T}{H_k P_k^- H_k^T + R_k}$$

从上式可以看出当测量误差方差 R_k 趋近于零矩阵时，增益 K_k 会使得估计值更偏向于视觉。

$$\lim_{R_k \rightarrow 0} K_k = H_k^{-1}$$

而当先验误差协方差趋于 0 时，增益 K_k 会使得估计值更偏向于动作更新。

$$\lim_{P_k^- \rightarrow 0} K_k = 0$$

12.3.3.3 卡尔曼滤波器在 Agent 定位中的应用:

首先介绍一下整个球场的坐标系。整个球场是一个以中心为原点的笛卡尔坐标系，从自己球门到对方球门的线为 x 轴，对方球门为 x 正方向。在 3D 模型中,Agent 有全局视觉，它能收到 Server 送来的 8 个定标相对自己的位置，这 8 个定标分别是：球场的四个角，以及四个门柱。它们的绝对位置 (x_i, y_i, z_i) 在 Server 初始化的时候确定，在前面介绍过，球场长宽都是随机设定的，因此 8 个定标的绝对位置也是随机的。每个定标的视觉用 $(r_i, \varphi_i, \theta_i)$ 表示，这是个球坐标的矢量， r_i 表示距离， φ_i 表示水平偏角， θ_i 表示垂直偏角。则自己的位置可有下面公式确定：

$$(x_i + r_i \cos \theta_i \cos \varphi_i, y_i + r_i \cos \theta_i \sin \varphi_i, z_i + r_i \sin \theta_i)$$

由于 Server 是一个不确定环境，为了表示这种不确定性，Server 对每个视觉分量都加了误差，这个误差包括两部分：截断误差和随机正态分布的误差。因此每次实际发给 Agent 的视觉值是不准的，用公式表示为： $(r' + \Delta r, \varphi' + \Delta \varphi, \theta' + \Delta \theta)$ ，其中 Δr ， $\Delta \varphi$ ， $\Delta \theta$ 服从下面的分布：

$$\Delta r \sim N[0, Q_r]$$

$$\Delta \varphi \sim N[0, Q_\varphi]$$

$$\Delta \theta \sim N[0, Q_\theta]$$

Q_r ， Q_φ ， Q_θ 是由 Server 确定的。

我们可以用一个卡尔曼滤波器来消除上面的随机误差。我们对 8 个定标分别用 8 个滤波器来定位，然后用 8 个结果来确定最后的定位结果。理论上讲，8 个滤波器的结果应该越来越接近，但由于视觉还存在截断误差，并且每个滤波器的收敛时间是不一样的，因此 8 个滤波器的结果可能有点发散。

由于随机噪声是加在球坐标上的，而人的状态是用直角坐标表示的，所以不能用线性的卡尔曼滤波器，而要用一种非线性的滤波器。非线性滤波的基本原理不变，也是由动作更新和视觉更新两步组成，但是视觉更新和动作更新是在不同的坐标系下完成，从球坐标到直角坐标之间转换需要一个雅可比矩阵。人的动作更新矩阵也是雅可比矩阵。

设人的状态用 $X_k = (x_k, y_k, z_k, vx_k, vy_k, vz_k)$ 表示，从周期 k 到周期 k+1 的控制量为 $U_k = (px_k, py_k, pz_k)$ ，动作更新函数为 $f(X_k, U_k)$ ，设 $A_{k,k-1}$ 为一步转移矩阵：

$$A_{k,k-1} = \frac{\partial f(X_k, U_k)}{\partial X_k}$$

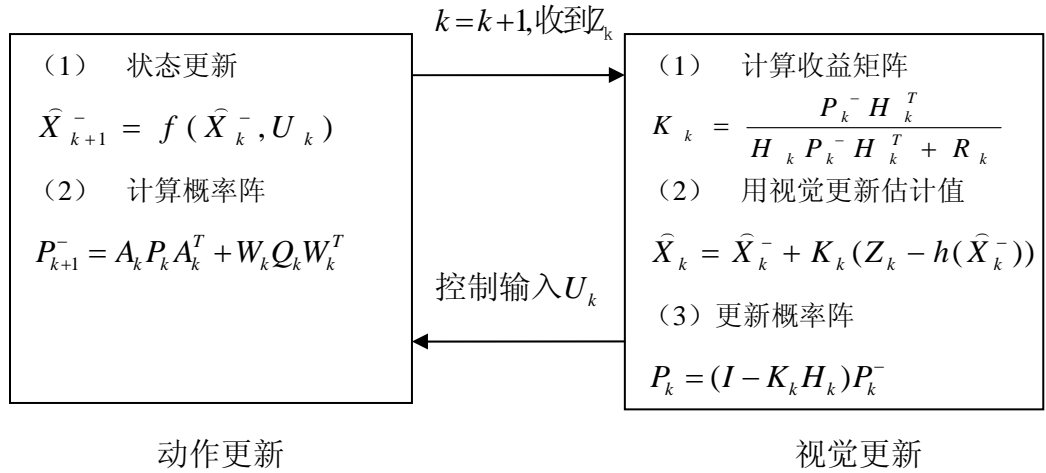
直角坐标到球坐标的视觉转换函数为：

$$\begin{pmatrix} r \\ \varphi \\ \theta \end{pmatrix} = h(X) = \begin{pmatrix} \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \\ \arctan\left(\frac{y_i - y}{x_i - x}\right) \frac{360^0}{2\pi} \\ 90^0 - \arccos\left(\frac{z_i - z}{\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2}}\right) \frac{360^0}{2\pi} \end{pmatrix}$$

H_k 为视觉转换矩阵：

$$H_k = \frac{\partial h(X)}{\partial X}$$

上面的 (x_i, y_i, z_i) 表示第 i 个定标的直角坐标，而 (x_k, y_k, z_k) 表示第 k 周期的运动更新值。由上可知， H_k 矩阵是变化的。收益矩阵 K_k 和状态概率阵 P_k 也是变化的，其中 P_k 一个周期要更新两次，为示区别我们用 P_k^- 表示动作更新后的概率阵，用 P_k 表示视觉更新后的概率阵。 Q_k 为系统噪声序列的方差阵， R_k 为量测噪声序列的方差阵，这两个矩阵都是常量矩阵，他的值可以从 Server 设定的误差参数中推导出来。下面用一个图来表示整个滤波系统的时序关系图。



上面的时序关系图描述了卡尔曼滤波器启动后的动态过程，还有一个重要的问题就是初始状态的设定。卡尔曼滤波算法是最终收敛的，这就是说我们任意设定一个初值，在很长一段时间后结果仍会收敛，但是好的初值会很快收敛，因此我们要选择一个较小误差的初值。至于怎么选取初值，有很多方法可用，比如收集几个周期的值，然后估计出一个初值来，这样也许启动较慢，你也可以只用一个周期的值。另外还有一个问题就是滤波器的重置，当视觉或动作受到很大的干扰时，会导致结果出现很大的偏差，这时更好的选择可能是重新启动滤波器。

12.3.4 WrightEagle3D Agent 动作设计

12.4.1 基本动作设计

Rcssserver3D 向我们提供了两个动作——跑和踢。其中的“跑”是向身体某个方向施加驱动力，“踢”是从身体与球的连心线方向将球以指定高度角踢出。当然，让高层决策使用这两个动作进行比赛是极为不便的。WrightEagle3D 将这两个动作进行封装、组合为六个基本动作：

首先是几个跑动的动作：

尽快到达某点 一般使用最大驱动力进行身体控制，是队员进行战术跑位的基本动作。

以指定速度跑向某点 不必使用最大驱动力，可以减少电量的浪费。用于非紧急情况的跑动。

快速追球跑 由于仿真针环境下，球员只能在距离球小于 7cm 的情况下，才可将球踢向身体正前方。所以，实现高效、带有目的性的助跑动作，是将球准确提出的关键。这个动作要充分考虑到球与球员的当前状态，并根据球踢出的方向进行一定的轨迹规划。

然后是几个踢球的动作，它们都是当身体处于有效踢球范围时执行的动作：

踢出地面球 2D 足球比赛中的经典动作，将球以指定的速度踢至某点，精度较高，但可能在途中被对方队员拦截。是精确传球，带球的必备武器。

踢出高空球 这个动作负责踢出高空球，将球送往指定的落点，是个相对比较安全的动作，不用担心球在空中会被拦截，而且球被踢出的距离会相对远一些，但落点可能会有不小的偏差。该动作在无法找到地面传球线路时相当有用，也可以用于长传与大脚解围。

射门 这个动作与前面两个动作有所不同，它所要做到的是尽一切可能，把球射向对方球门，并保证球到达球门时具有尽可能大的速度。通常，将球以较小的高度角踢出，这样踢出速度快、方向性好，可以取得很好的射门效果。另外，在某些情况下还可以使用高空球将球挑过门将与防守队员，完成一次极为精彩的射门。

完成这几个基本的动作后，我们所面对的，已经不是只能通过字串通信进行动作的简易程序了，Agent 已经懂得了怎样跑、怎样踢，接着可以进行复杂一些的动作设计了。

12.4.2 组合动作设计

大家也许注意到基本动作仍旧是简单的跑和踢，要想完成一个连贯的动作，我们必须进行合理的规划，然后组合基本动作，完成某种技术动作。WrightEagle3D 设计了八个组和动作。

首先是持球动作，也就是当某队员被拦截模型判定为控球队员时的可能动作，每个动作都是三个环节：规划+跑动+踢球。

带球 如果仔细观察一场 3D 仿真足球比赛会发现，所有球员的运动能力实在有限，由于视觉误差，跑动经常会发生错误，如果绕球摆动或许能获得较高的踢球精度，但可能痛失一次进攻机会。所以，带球将发挥它的独特功效，向前小距离踢出，一方面可以获得继续调整的机会，一方面也会干扰对方的决策与控制，从而保持自己的进攻优势。但也要提到的是，总结实际经验，频繁的带球会导致进攻速度缓慢，在总体局势上对己方的防守带来巨大的压力，而且由于双方身体能力相同，是不可能仅仅依靠带球摆脱对方防守队员的。

传球 这个动作的任务是将球传给指定人，或将球传向某个区域。它需要对传球点进行规划，决定是使用高空球，还是地面球进行传递，是所有使用的最多的持球动作，如果传球能够迅速、准确，那么己方在比赛中将占有相当大的优势。

射门 这个射门动作比起基本动作中的射门会复杂很多，它需要综合考虑对方门将的站

位，自己与球门的位置关系，是否强行射门等，经过计算评估后确定将球射向球门的哪一点。

然后是非持球动作：

拦截 主要用途是，利用对方踢球之前会减速进行调整，尽全力从对方脚下抢球，完成一个防守动作。

自由跑位 当队员距离球很远时，为保持整体上的战术布局采取的跑位。

战术跑位 用于完成接球队员的主动跑位、前锋队员的主动切入进区等战术性动作，跑位点要综合考虑整体局势和持球队员的决策。

盯人防守 分析比赛会发现，所有的队员的运动能力完全相同，而且实现瞬间的加速减速也是不可能的，所以人盯人防守成为最有效的防守手段。在执行这个动作时，先执行防守队员的分配算法，使得每个队员在视觉不完全准确地情况下也可以不重复、不遗漏的确定自己的盯防目标。之后，只需跟在对方的身后，阻止对方的进攻线路或进入拦截动作。

最后，还有一个专门属于守门员的动作：

守门 Rcserver3D 中的守门员与一般球员完全相同，并没有特别的接球动作，所以守门员的设计与现实会有很大区别。分析比赛并总结经验，3D 仿真平台下的守门员，如果不具备果断的出击与强力的抢断就形同虚设。球员的速度非常缓慢，是根本无法在对几米内的射门进行拦截的。因此，这里需要更富有激情的门将，更多的出击不仅可以增加扑救的概率，而且可以干扰对方调整，使其无法提出高质量的射门。

12.4 Rcserver3D 安装与运行指南

12.4.1 准备工作

目前 3D 仿真平台 Rcserver3D 的最新稳定版本为 0.3 release，其工作环境为 Linux，它是运行在 SPADES 这个中间部件的基础之上的，因此在安装 rcserver3D 之前必须安装 SPADES。目前由于 Rcserver3D 和 SPADES 都还没有任何的安装包发布，因此只能从源代码编译安装。它们的源码包都可以在 SourceForge 上面下载到。

SPADES 和 Rcserver3D 的编译安装中我们会需要一些库的支持，在安装之前最好把这些库配置、安装好，否则在后面的配置中还是会被迫中断请求安装这些库。具体需要的库的列表如下：

缺省安装所需要的库：

ZLIB 1.1.4
 Ruby-1.8.0 (or above)
 Expat-1.95.7 (or above)
 Boost 1.30.2 (or **older** versions)
 J2sdk
 OpenGL, GLUT
 ODE 0.039 (or above)

完全安装额外需要的库：

SDL 1.2
 FreeType 2.1.2

这些库的具体配置与所使用的 Linux 平台有关，在不同的 Linux 系统中的配置稍微有一些差别，在这里就不多说了。准备好这些之后，接下来我们就可以安装 SPADES 了。

12.4.2 SPADES 的安装

安装 SPADES 的最简单的方法就是使用缺省安装，即不做任何特殊的配置，完全按照默认的参数来编译。在这种情况下，你只需要在 SPADES 源代码的根目录下输入

```
./configure
make
make install
```

就可以了。只要你上面的库都正确的安装好了，现在就只需要等待编译完成并 install 之后就 ok 了。

如果你需要对 SPADES 的安装作一些调整，就需要在 ./configure 的时候加入参数来使编译符合你的要求。下面就简单介绍一下这些参数：

--enable-debug（默认关闭）当这个选项被打开时，action logging 功能将被开启（注：error logging 总是开启的）。这时将记录 SPADES 的 WorldModel 所作的任何动作。关闭这个选项可以减小 SPADES 可执行程序的大小，并且提高运行速度。

--with-perfctr=DIR 要使用 perfctr timer 需要使用 perfctr 系统，它是对 Linux kernel 的一个补丁和一个用来访问 perfctr timer 的用户库。如果这个用户库不在你的系统得标准 include path 中，就需要用这个参数来指定它的位置。

如果不指定这个参数 configure 将自动检测 perfctr 是否可用，然后自动的允许/禁止 perfctr timer。如果指定了这个参数，但 configure 无法在指定位置找到 perfctr，configure 将会终止。

没有 perfctr timer 的情况下 SPADES 同样可以运行，可以使用其他的 timer，但是会对 SPADES 的工作效果又一定的影响。

--with-jiffies（默认打开）决定是否 build jiffies timer。这个 timer 使用 /proc 文件系统。如果由于某些原因 SPADES 无法使用它，你可以关闭这个选项。SPADES 没有这个 timer 同样可以工作，但是如果既没有 perfctr timer 又没有 jiffies timer，SPADES 就不能够计算 agent 所用的计算时间了。

--with-fork-dynlib-loc=PATHTO LIB 为了拦截从动态库中对 fork 及其他相关函数的调用，SPADES 需要知道定义 fork 函数的动态库的位置。SPADES 同时认为 clone 函数也定义在同一个库中。

12.4.3 Rcserver3D 的安装

Rcserver3D 有两种安装方式，缺省安装和完全安装。缺省安装是 Robocup2005 比赛时 3D 仿真所使用的平台，而完全安装提供了一个叫做 kerosin 的额外的库，它为 agent 提供了更好的视觉效果，特别是对于连接体的 agent。

和安装 SPADES 一样，安装 Rcserver3D 的最简单的方法就是使用缺省安装，在 Rcserver3D 源代码的根目录下输入

```
./configure
make
make install
```

就可以了。只要你上面的所说的东西都正确安装好了，现在就只需要等待编译完成并 install 之后就 ok 了。

要使用完全安装，你只要在 configure 的时候加上参数 `--enable-kerosin[=yes|no]` 即可。

在 configure 的时候同样可以加一些参数来调整安装的内容和方式，常用参数主要是用来指定一些库的安装位置的，如果 configure 找不到需要的库就会停止，并且显示相应的提示信息，这些信息会教你如何指定库的位置的。

12.4.4 仿真系统的运行

12.4.4.1 运行所涉及的文件及对它们的作用的解释

系统安装完成之后，有下列的程序被安装在默认位置 `/usr/local/bin/`（或用户指定位置）：

rcssserver3D 3D 仿真的 server 主程序，主要负责完成各种事件的仿真的计算，并处理与 agent 和 monitor 之间的通讯。它是建立在 SPADES 的基础之上的，所以作为它的 agent 也必须是符合 SPADES 要求的 agent。

rcssmonitor3D-lite 缺省安装所安装的 monitor，可以连接 server 观看比赛，也可以用来播放一场比赛的录像（播放 monitor.log 文件）。

rcssmonitor3D-kerosin 完全安装所安装的 monitor，本文编写时还不可用。

rcsoccersim3D 用来启动 server 的脚本文件，同时会启动 monitor。

agenttest 随 rcssserver3d 发布的测试 agent，是了解 agent 的很好的切入点

commserver SPADES 的 communication server 的可执行文件，它可以用来在网络运行的时候从远端机器上启动 agent 并帮助完成 agent 与 Rcssserver3D 的通讯。注意：如果要使用网络运行，远端机器上的 agent 必须由 commserver 来启动。

show_tttimes.pl 这是 SPADES 提供的一个 perl 脚本，它能将 agent think time 文件从机器格式转换成更方便我们理解的格式。详细情况请参考 SPADES manual 4.7.1。

* 单机运行需要的配置文件：

rcssserver3d.rb Rcssserver3D 的配置文件，默认位置在 `/home/ACCOUNT_NAME/.rcssserver3d/rcssserver3d.rb`，这个文件中包含了大部分的 Rcssserver3D 运行时的参数，包括对于球场、球员以及球的一些设置，同时它还包含了 Rcssserver3D 启动时的一些参数设定。下面关于 Rcssserver3D 的运行方法的讲述中将涉及到对这个文件的修改。

Agentdb.xml Agent 的启动配置文件。这个文件是提供给 SPADES，用来给出 Agent 的可执行代码的位置以及 Agent 运行环境的设定。在 Rcssserver3D 的运行中，这个文件的默认路径在 `/usr/local/share/rcssserver3d/` 目录下。

* 网络运行需要的额外的配置文件：

commserver.conf 在网络运行模式中用来启动远端 Agent 的 commserver 程序需要的配置文件。其默认位置在 `/usr/local/share/spades/commserver.conf`，一般使用的时候不需要对它作改动，如果有特殊的需要可以参照 SPADES mannual 来进行相应的设置。

12.4.4.2 基本运行方式：单机运行、网络运行、启动 agent、启动 monitor

两种运行方式中 rcssserver3D 的启动方法都是执行 rcssserver3D，它的启动需要用到两个配置文件 rcssserver3D.rb 和 agentdb.xml，可以用参数 `--agent_db_fn AGENT_DB_FILE` 来指定 agentdb.xml 文件的位置，如果不指定将使用上面所说的默认的配置文件。

既然都使用一个程序启动，怎么会有两种运行方式呢？关键就在于 rcssserver3d.rb 文件。

在这个文件中可以配置 rcssserver3D 的运行方式，还有允许上场的球队、球员数目。下面就是这个配置文件中的相关部分：

```
# game Setup

# start the integrated comm server
Spades.RunIntegratedCommserver = true

# the number of commServers to wait for before the simulation is
# initially unpaused
Spades.CommServersWanted = 1

# queue agents for startup
spadesServer.queueAgents('foo', 11)
spadesServer.queueAgents('bar', 11)
```

(1) **单机运行：**当 Spades.RunIntegratedCommserver 设为 true 时表示使用单机方式运行 rcssserver3D，这时 rcssserver3D 启动的同时也要负责启动 agent，所以只要 rcssserver3d.rb 文件和 agentdb.xml 文件配置正确，启动 rcssserver3D 之后就不用任何其它操作，仿真就开始运行了。注意：在这种情况下 Spades.CommServersWanted 必须设成 1。

(2) **网络运行：**当 Spades.RunIntegratedCommserver 设为 false 时表示使用网络方式运行，这时 rcssserver3D 启动后就进入等待状态，直到有远端的 commserver 连接上，然后由这些远端 commserver 启动远端的 agent。在这种情况下，参数 Spades.CommServersWanted 的值就要根据实际要使用的远端机器的数目来决定，Rcssserver3D 将一直等待直到设定的数目的 commserver 连接上。网络运行方式中需要注意的一个问题是，作为 server 的机器上的 agentdb.xml 文件必须包含所有远端机器的 agentdb.xml 的内容。

(3) **Agent 的启动：**agent 的启动必须通过 rcssserver3D，在上面配置文件的例子中 spadesServer.queueAgents 函数就告诉了 rcssserver3D 我们要启动的 agent 是哪些。其中'foo'和'bar'是 agent 的类型，它们必须是在 agentdb.xml 中定义过的，如'bar'的定义可以为：

```
<agent_type_external name="bar">
  <inputfd>3</inputfd>
  <outputfd>4</outputfd>
  <timer>jiffies 2000</timer>
  <working_dir>../agenttest</working_dir>
  <exec_line>agenttest --teamname foo</exec_line>
</agent_type_external>
```

在这些配置中，我们需要注意的是<agent_type_external name="bar">和<exec_line>，一个是定义新的类型的名字，就是在 rcssserver3d.rb 中要使用的名字。而<exec_line>则是指定 agent 的可执行码的位置以及启动时需要的命令行参数，把这两处修改好一个 agent 类型就基本定义好了。

在单机运行方式下，只要 rcssserver3d.rb 和 agentdb.xml 都配置正确，rcssserver3D 就会自动启动 agent。

在网络运行方式下，client 要从远端连接上 server 必须通过 SPADES 的 commserver 程序，因此 client 机器上面可以不安装 rcssserver3D，但是一定要安装 SPADES。server 端和 client 端都要进行 agentdb.xml 的配置，client 端只要配置本机将会运行的 agent 类型就可以了，而 server 端则需要有所有将要启动的 agent 类型的定义。

Client 端在配置好 agentdb.xml 之后，就可以用下面的命令来启动 commserver：

```
commserver --file /usr/local/share/spades/commserver.conf --agent_db_fn
/AGENT_DB_PATH/agentdb.xml --engine_host HOST_IP
```

如果 commserver 正确的连接上了 rcssserver3D, 就会在 rcssserver3D 的控制下启动 agent 的。

最后说明一下, 'foo'和'bar'是 agent 的类型, 而不是一个队的队名, 队名是在 agent 与 rcssserver3D 的交互过程中由 agent 告诉 rcssserver3D 的。在一场比赛中, 只允许又两个队的队员上场, 这是用队名来区分的, 而不是 agent 的类型。因此一次比赛可以启动 2 个以上的 agent 类型, 只要这些 agent 的队名只有两个。当然, 每个队的队员不能超过 11 个。

(4) **Monitor 的启动:** 单机方式下, 在 rcssserver3D 开始运行后运行 rcssmonitor3D-lite 就可以看到比赛的情况。在上面 rcssserver3D 的运行配置都完成之后, 也可以用脚本 rcsoccersim3D 来启动, 它将自动启动 rcssserver3D, 然后启动 rcssmonitor3D-lite。在网络运行方式下, 可以用下面的命令来启动 monitor 连接到 server:

```
rcssmonitor3D-lite --server HOST_IP
```

12.4.4.3 Monitor 中的基本操作

```
q          | quit the monitor
p          | pause the simulation/logplayer
r          | unpause the simulation/logplayer
n          | toggle display of uniform numbers
2          | toggle display of two dimensional overview
?          | display keybindings
```

CAMERA MOVEMENT

```
c          | toggle automatic (ball centered) camera
w/s        | move camera forward/back (zoomlike)
a/d        | move camera left/right
+/-        | move camera up/down
Arrow keys | move camera
```

SIMULATION

```
k          | kick off (start the game)
b          | drop the ball at its current position
[SPACE]    | toggle kick off side (left-right-random)
```

LOGPLAYER

```
m          | toggle single step mode for logplayer
>          | move one step forward
```

12.4.4.4 常见问题

Q: 错误提示: could not find agent type xxx

A: 这个错误通常是由于 agent 的类型错误导致的，首先确定你启动 rcssserver3D 的时候使用的是哪一个 agentdb.xml 文件，如果你没有指定，系统会在当前目录找，如果找不到则会使用默认位置（/usr/local/share/rcssserver3d/agentdb.xml）来代替。如果这个没有问题，在确定你在 rcssserver3d.rb 文件中所使用的 agent 类型与 agentdb.xml 文件中定义的类型名是否一致。在网络运行方式尤其容易出现这个问题，主要是 server 端和 client 端的 agentdb.xml 不一致。注意上面的这几点，这个问题一般就会解决。

Q: 错误提示：bad file descriptor

A: 这个错误一般说明 agent 方面出了问题，经常是 agent 出现段错误之类的情况导致 agent 被系统 kill 了，从而 agent 所持有的 file descriptor 被强行释放。

* 其他

如果你遇到其它问题，可以到 3D 仿真的 mail list 上面去讨论你的问题：
sserver-three-d@lists.sourceforge.net

参考文献：

- [1] SPADES manual, Patrick Riley, pfr+@cs.cmu.edu, December 7, 2003
- [2] TEXT_INSTEAD_OF_A_MANUAL.txt in rcssserver3d release
- [3] Rules 3D Simulation League RoboCup 2005, Osaka, January 11, 2005

仿真机器人足球术语表：

英文	中文	备注
RoboCup	机器人足球世界杯及学术会议	The Robot World Cup Soccer Games and Conferences
The RoboCup Federation	RoboCup 联盟	
Simulator	仿真	
Soccer Server	机器人足球仿真比赛平台	
Soccer Client	仿真球员	
Soccer Monitor		
logplayer	比赛录像播放器	
agent	智能体？	
multi-agent	多主体	
Hidden state	隐藏状态？	Each agent has only a partial world view at any given moment
Single-channel	单通道	
Low-bandwidth	窄带宽	
Teamwork	团队合作	
Opponent modeling	对手建模	
Off-line	离线	
On-line	在线	
Perception	感知	
Action	行为	
Sensor infomation	感知信息	
Opaque-Transition	不透明转换？	
Locker-room agreements	更衣室协定	
PTS	周期团队同步	Periodic team synchronization
TPOT-RL		Team-Partitioned, Opaque-Transition Reinforcement Learning
SBSP	基于形势的战术跑位	Situation Based Strategic Positioning
Formation	阵型	

WrightEagle
