蓝鹰仿真 2D 机器人足球若干关键技术实现

Implementation of Some Key Techniques in WrightEagle 2D Soccer Simulation

培 养 单 位 ： 计 算 机 科 学 与 技 术 系

专　　业 ： 计 算 机 科 学 与 技 术 专 业

学　　生 ： 柏 爱 俊、台 运 方

学　　号 ： PB05210411、PB05210055

指 导 教 师 ： 陈 小 平

二〇〇九年六月

致　　谢

本论文是在我们的导师陈小平教授的亲切关怀和悉心指导下完成的。他严肃的科学态度，严谨的治学精神，精益求精的工作作风，深深地感染和激励着我。从课题的选择到项目的最终完成，陈老师都始终给予我细心的指导和不懈的支持。在实验室这两年多来，陈教授不仅在学业上给我以精心指导，同时还在思想、生活上给我们以无微不至的关怀，在此谨向陈老师致以诚挚的谢意和崇高的敬意。

在此，我们还要感谢在一起愉快参加实验室工作的各位学长、同学，正是由于你们的帮助和支持，我们才能克服一个一个的困难和疑惑，直至本文的顺利完成。

目　　录

摘　　要

近年来，智能体及多智能体规划问题成为人工智能领域新的研究热点，且有着广泛的应用前景。MDP/POMDP 及其相关理论成为研究智能体及多智能体不确定性规划问题的标准模型，本文对此展开研究。

首先扼要介绍了智能体、多值能体系统和决策的基本概念和一般模型，然后重点介绍了 MDP 和 POMDP 的基本概念和求解方法，最后结合 RoboCup 仿真 2D 这个标准测试平台，分析了在这种接近现实世界应用的问题中，进行整体规划所需要处理的一些子问题的设计方法，并通过结合现有马尔可夫决策过程相关理论对这些问题进行建模及分析，给出该平台更一般的研究意义。

关键词：多智能体、马尔可夫决策过程、强化学习、机器人足球,POSG

Abstract

In recent years, agent and multi-agent planning problem have been new research hotspots in the field of Artificial Intelligence with a broad prospect.MDP/POMDP and its related theories become the standard model of solving agent and multi-agent planning problem in large scale uncertain environment which is mainly researched in this paper.

A brief description of the basic concepts and general model of agent and multi-agent planning problem is introduced in chapter 1. In chapter 2, the basic concepts and general model of MDP and POMDP is mainly introduced. And last based on Robocup simulation 2D which is regarded as a standard platform for research on multi-agent planning problem in large scale uncertain environment, by employing existing Markov decision theory to model these issues, more general research significance of this platfor is gived after some necessary explanation about the platform itself, addressing and analyzing the design methods on some sub-problem for the overall planning in such kinds of real world application.

Key Words: multi-agent, MDP, Reinforcement Learning, RoboCup,POSG

第 1 章　背景和理论基础

1.1　研究背景

1.1.1　智能体

在人工智能领域，智能体（agent），又叫主体，是指可以不断感知环境并作用于环境，最终完成一定目标的自主实体。

1.1.1.1　智能体结构

智能体可以被形式化定义成智能体函数（agent function）：

$$f:\ P^* \rightarrow\ A \tag{1-1}$$

P^* 是智能体可以感知到的一系列知觉对象（percepts），A 是智能体为作用于环境可能执行的动作序列（actions）。

1.1.1.2　智能体分类

根据智能行为的复杂程度，智能体可以被分成5类：

1. 简单反射式智能体
2. 基于模型的反射式智能体
3. 基于目标的智能体
4. 基于效用的智能体
5. 学习型智能体

简单反射式智能体 简单反射式智能体根据当前感知的信息来行动，智能体函数可以表示成若干*条件- 动作* 规则，即判断*条件*是否成立，如果成立，就执行相应的*动作*。

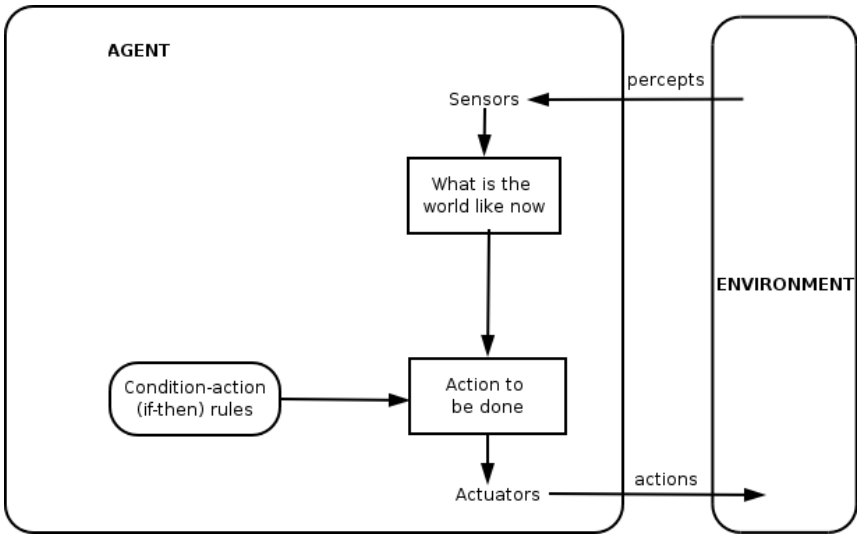


图 1.1　简单反射式智能体

基于模型的反射式智能体 基于模型的反射式智能体为环境维护一个内部状态，称为世界模型（world model），可以描述当前环境中不可感知部分的状态。这类智能体总是跟踪当前状态，并使用*状态- 动作*规则来选择应该执行的动作。

基于目标的智能体 基于目标的智能体可以识别目标状态和非目标状态，所以这允许智能体在多个可选动作中，选择一个能达到目标状态的动作来行动。

基于效用的智能体 基于效用的智能体不仅可以区分目标状态和非目标状态，还可以给每个状态一个评价值，评价值通过*效用函数（utility function）*给出。效用函数是状态到该状态效用值的一个映射。

学习型智能体 学习型智能体智能体可以在未知环境中行动，并且不断提高自己的行动能力，这通常表现为通过不断地学习，得到更好的效用函数。

1.1.1.3　智能体所处的环境

从智能体自身的观察出发，智能体所处的环境有以下一些特点：

完全可观察的 vs. 部分可观察的 如果智能体可以感知到所处环境完整、没有噪

音的信息，就称这个环境是完全可观察的，否则就是部分可观察的。完全可观察的环境一般只出现在一些软件系统中，多数研究的环境都是部分可观察的。

确定的 vs. 非确定的 确定的环境是指在当前状态下，智能体已经选定要执行的动作，那么动作执行后，环境将转移到唯一确定的下一个状态。如果用 $\mathcal{S}$ 表示状态集合，$\mathcal{A}$ 表示动作集合，$s \in \mathcal{S}$ 表示当前状态，$a \in \mathcal{A}$ 表示智能体执行的动作，$s' \in \mathcal{S}$ 表示下一个状态，定义状态转移函数（transfer function）$T(s, a, s')$ 表示智能体在状态 s 下，执行动作 a，转移到 s' 的概率，那么确定环境下状态转移函数表示成：

$$T(s, a, s') \rightarrow \{0, 1\} \tag{1-2}$$

并且：

$$\sum_{s' \in \mathcal{S}} T(s, a, s') = 1 \tag{1-3}$$

非确定的环境是指，动作执行后，状态按照一定概率的进行转移，状态转移函数表示成：

$$T(s, a, s') \rightarrow [0, 1] \tag{1-4}$$

并且：

$$\sum_{s' \in \mathcal{S}} T(s, a, s') = 1 \tag{1-5}$$

静态的 vs. 动态的 静态环境是指环境的状态转移仅依赖于智能体自身的动作，即在智能体决策过程中，环境状态是不会发生改变的。动态环境则相反。

离散的 vs. 连续的 环境是离散的还是连续的，取决于环境包含的状态数，离散环境包含有限个状态，而连续环境包含无限个状态。

1.1.1.4 智能体的分层结构

为了实现一些复杂的任务，智能体通常被设计成分层结构，包含很多子智能

体（sub-agent）。不同的子智能体处理不同的子任务，共同组成一个完整的智能系统。

1.1.2 多智能体系统

多智能体系统（multi-agent system, MAS）由若干相互作用的智能体组成，有以下几点基本特征：

自治性 每个智能体都是自主决策和行动的

局部性 每个智能体都只拥有环境的部分视图

分布性 每个智能体都是平等、独立的，不受其他智能体或系统控制

多智能体系统可以呈现出很复杂的行为，即使每个智能体的个体策略（individual strategies）很简单。

1.1.2.1 多智能体系统的主要研究内容

- 面向智能体的软件工程（agent-oriented software engineering）
- 信念、愿望和意图模型（BDI）
- 团体组织（agent-oriented software engineering）
- 通信（communication）
- 谈判（negotiation）
- 多智能体决策（multi-agent planning）
- 多智能体学习（multi-agent learning）

1.1.3 多智能体决策

多智能体决策（multi-agent planning）是指多智能体系统中的一些智能体一起为完成共同的任务时进行的决策行为。多智能体决策主要关注智能体之间的合作（coordination, cooperation）行为。

1.1.3.1 决策问题简介

决策问题又叫规划问题，是人工智能的基本问题，决策的目的是要研究如何从现有状态达到目标状态，在这个过程中采取什么样的手段和方法。决策的结果是给出一个能达到目标状态的行动序列（action sequence）或者给出每步选择行动的策略（policies/strategies）。

经典决策算法基于确定的环境模型，如 A^* 等。这类方法在实际应用时有很大的局限性。现实中的决策问题，环境经常是动态的、不确定的和部分可观察的，面对这类问题，研究者提出了基于马尔可夫模型的决策模型：马尔可夫决策过程（MDP）和部分可观察马尔可夫决策过程（POMDP）。

马尔可夫决策过程 尔可夫决策过程适用的系统有三大特点：一是马尔可夫性，又称无后效性，即系统的 $t+1$ 时刻的状态仅取决于 t 时刻的状态和智能体在 t 时刻选择的动作；二是状态转移可以有不确定性；三是智能体所处的每步状态可以被完全观察。马尔可夫决策过程是马尔可夫链（Markov chain）的扩展，如果智能体只能选择一个动作或每步状态可执行的动作是事先确定的，那么马尔可夫决策过程就退化为马尔可夫链。马尔可夫决策过程的求解方法主要有动态规划和强化学习。动态规划类的方法又主要分值迭代和策略迭代。

部分可观察马尔可夫决策过程 部分可观察马尔可夫决策过程适用于环境是部分可观察的情况。由于环境的部分可观察性，智能体不清楚自己到底处在怎样的状态，但是可以知道自己可能出于的一些状态和它们的分布，这就是信念状态（belief state）。引入信念状态，部分可观察马尔可夫决策过程退化为连续状态的马尔可夫决策过程，由于是连续状态的，所以求解仍然很困难。目前的求解算法主要倾向于直接求解部分可观察马尔可夫决策过程，主要有精确解法和近似解法。

1.1.3.2 多智能体决策的一般模型

MDP 和 POMDP 模型都认为环境中只有一个智能体，并把其他一切因素归因于客观环境。但一个环境中有多个智能体时，如果其他智能体的策略已知，那么同样可以把其他智能体都归于环境的一部分，仍可使用 MDP 和 POMDP 求

解。否则，其他智能体的选取何种策略决策也是需要考虑的，这就不能使用 MDP 和 POMDP 求解了。分布式马尔可夫决策过程（DEC-MDP）和分布式部分可观察马尔可夫决策过程（DEC-POMDP）是专门处理这类对智能体合作问题的。

在现实世界中，多智能体之间除了合作也可能存在对抗（competition），这类问题可以归为博弈。合作类问题中，多智能体有相同的收益评价，或者说共同目标是一致的；而博弈类问题中，多智能体之间的收益评价存在区别，甚至完全对立。部分可观察的随机博弈（POSG）便是进一步扩展的多智能体决策模型，可以处理这类带不确定性的博弈问题。

1.1.3.3 多智能体决策的一般过程

实际求解多智能体决策问题时，常常为了简化问题，把多智能体共同的任务细分成若干可并行处理的子任务，简称任务（task）。那么多智能体决策的一般过程为：

1. 任务分配（task allocation）
2. 决策前合作（coordination before planning）
3. 个体决策（individual planning）
4. 决策后合作（coordination after planning）
5. 决策执行（plan execution）

个体决策阶段要解决的是单智能体的决策问题，方便使用各种已有决策方法。

1.2 研究平台

仿真 2D 机器人足球是本文主要采用的研究平台。机器人足球比赛的最初想法由加拿大大不列颠哥伦比亚大学的 Alan Mackworth 教授于 1992 年正式提出。随后，Minoru Asada、Hiroaki Kitano、Yasuo Kuniyoshi 等学者创立了 RoboCup 机器人足球世界杯比赛。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会议（The 15th International Joint Conference on Artificial

Intelligence, 简称 IJCAI-97）上，机器人足球比赛被正式列为人工智能的一项挑战。至此，机器人足球比赛成为人工智能和机器人学的一个标准问题以及公共测试平台。

1.2.1 RoboCup 简介

RoboCup 的长期目标是：到2050 年一支完全类人的机器人足球队能够战胜当时的人类足球世界冠军队伍（Kitano, 1998）。从 1903 年莱特兄弟飞机上天到 1969 年阿波罗登月成功花了整整 66 年；从 1946 年首台通用电子计算机问世到 1997 年深蓝战胜当时的国际象棋世界冠军经历了 51 年，Robocup 组织也将自身长期目标的实现定为半个世纪左右的时间。事实上，机器人足球所涵盖的关键技术的深度及广度，意味着这一目标的实现将基本代表了机器人已可以真正走进人们的生活。回顾人类历史，第一第二次工业革命，人类分别进入蒸汽时代与电气时代，使人类从体力劳动中解放出来；第三次工业革命，计算机及网络技术的蓬勃发展,作为人类智能的延伸，人类进入信息技术时代。展望未来，人工智能技术的发展将使人类逐渐从脑力劳动中解放出来，而机器人则是所有这些技术的一个综合应用所在。做为 RoboCup 的长期目标，也做为一项极大的挑战，人类是否能够再次完成这一新的里程碑式的任务，将具有深远的意义。

就近期而言，RoboCup 通过其不同项目组为人工智能和机器人学提供了多个标准的测试平台，可用来检验信息自动化前沿最新研究成果，包括动态不确定的对抗环境下的多智能体合作、知识表示、实时推理、机器学习和策略获取等当前人工智能的热点问题以及自动控制、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。同时与影响范围最广的足球运动结合，容易受到公众的关注，利于促进了基础研究和实际应用的联系与转化。

1.2.2 仿真 2D 平台的特点

RoboCup 仿真 2D 机器人足球比赛提供了一个典型的研究大规模不确定环境下的多智能体合作及对抗问题的测试平台，研究的核心是多智能体决策理论。仿真 2D 涉及的决策问题有以下特点：

- 问题规模巨大

  - 战胜人类国际象棋冠军的“深蓝”所处的国际象棋问题的规模约为 10^{20}；围棋问题的规模约为 10^{200}
  - 而 RoboCup 2D 具有连续的状态及行动空间，按粗略离散，其决策问题的规模约为 10^{400} 以上
- 多智能体
  - 由于队友的存在，涉及合作
  - 由于对手的存在，涉及对抗
- 大量不确定因素
  - 环境部分可观察且存在噪音
  - 行动结果具有不确定性
  - 受限的通信模型
  - 对手模型未知
- 实时系统
  - 在每个极短的仿真周期内作出决策，考虑网络延迟的影响，每步决策时间只有几十毫秒

1.2.3 仿真 2D 平台简介

RoboCup 仿真 2D 机器人足球比赛是在标准计算机环境内进行的。比赛规则基本与国际足球联合会的比赛规则一致。比赛平台采用 Client/Server 结构，比赛时，参赛双的球队客户端（SoccerClient，以下简称‘client）通过网络连接到 RoboCup 委员会提供的标准比赛模拟器（SoccerServer，以下简称 server）上进行比赛，如图 1.2 所示。

典型 client 的程序流程和 server 的程序流程如图 1.3 所示。

server 发给 client 的信息包括感知信息、视觉信息和听觉信息。由于每个球员都有一定的视野范围，所以视觉信息是局部的，只包括视野内物体的相对位置、速度，并且附加了随机误差；听觉信息包含了某个队友或己方教练上周期说

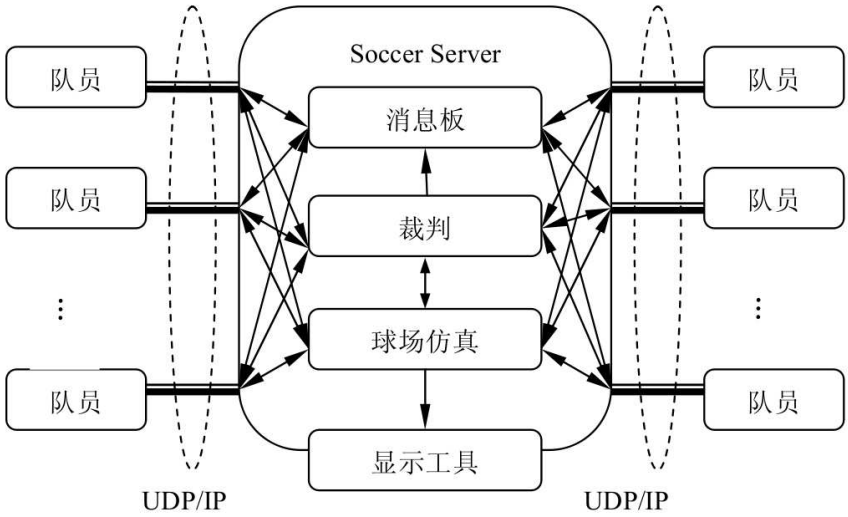
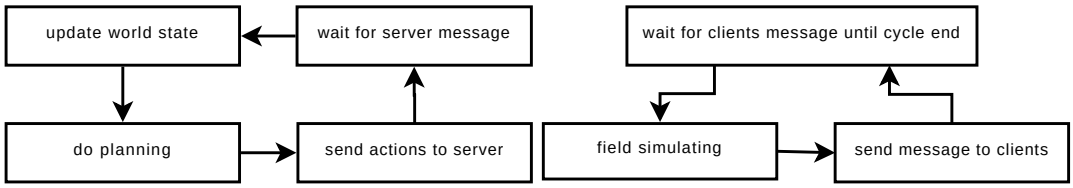


图 1.2　仿真 2D 机器人足球比赛平台



(a) 典型 client 的程序流程　　(b) server 的程序流程

图 1.3　平台流程

的话，server 规定球员间一次通信只能发送长为 10 个字节的信息，并且裁判信息也是通过听觉的形式发送给球员的；感知信息主要包体力、速度等球员自身的信息。

client 发给 server 的信息是决策后选择的动作信息。server 为球员提供的原子动作主要有：

- say(*message*) 喊话给其他队员
- turn(*angle*) 转身 *angle* 大小
- dash(*power, angle*) 加速，沿相对 *angle* 产生大小为 $f(power, angle)$ 的加速度

- kick(*power, angle*) 踢球，沿相对 *angle* 产生大小为 g(*power, angle*) 的加速度
- turn_neck(*angle*) 调整脖子角度
- change_view(*view_mode*) 调整视角宽度[①]
- catch(*angle*) 守门员可以沿相对 *angle* 方向扑球

其中 turn，kick，dash 是互斥命令，即每周期只能发送这些命令中的一个。

1.3 MDP理论

MDP 为大量规划问题提供了形式上的基础。MDP 的优势是可以处理状态转移的不确定性，并且可以把环境中主体以外因素造成的改变都归于环境的不确定性，从而简化了问题。

1.3.1 MDP 的基本框架

形式上，MDP 是一个四元组：$\langle \mathcal{S}, \mathcal{A}, T, R \rangle$，$\mathcal{S}$ 是环境中所有状态的集合，又称为状态空间（state space），$\mathcal{A}$ 是主体可以执行的动作集合，成为行动空间（action space），$\mathcal{T}$ 描述了状态转移模型，$\mathcal{R}$ 是回报模型。

状态空间 $\mathcal{S}$，是环境中所有状态的集合，在某一时刻 t，系统仅处于某一确定的状态 $s_t \in \mathcal{S}$。

在时刻 t，主体选择一个动作 $a_t \in \mathcal{A}$ 并执行这个动作（并不是所有状态下都可以选择所有的动作）。

转移模型 T 给出了状态被动作改变的具体方式。形式上，T 是一个函数，$T\colon \mathcal{S} \times \mathcal{A} \to P(\mathcal{S}; \mathcal{S}, \mathcal{A})$，从状态、动作映射到状态上的概率分布。我们把状态 s 下执行动作 a，转移到状态 s' 的概率记为 $P(s'|s, a)$。

回报模型 R 给出了一次状态转移下的立即回报，这是一个函数$\mathcal{S}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{A} \to \mathbf{R}$。那么状态 s 下选择动作 a 的立即回报为：

$$R(s, a) = \sum_{s' \in \mathcal{S}} P(s'|s, a) \cdot R(s, a, s') \tag{1-6}$$

① 脖子宽度越宽，视觉延迟越大

转移模型和回报模型都符合马尔可夫性，即：

$$P(s_{t+1}|s_0, a_0, s_1, a_1, \ldots, s_t, a_t) = P(s_{t+1}|s_t, a_t) \tag{1-7}$$

和

$$R(s_t, a_t|s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \ldots, s_0, a_0) = R(s_t, a_t) \tag{1-8}$$

MDP 问题的解称为策略（policy），这是一个从状态到动作的映射，给出了当前状态下应该执行的动作。主体完成任务执行的动作数，称为视界（horizon）。如果一个策略里面，每步执行的最优动作跟这步所处的阶段数无关，那么这个策略被称为是稳定的，否则就是非稳定的。有限视界的 MDP 问题一般具有非稳定策略，表示成一系列的动作映射 $\pi_t(s)$，其中 $t = 0, 1, \ldots, h$；而无限视界的 MDP 问题总是具有稳定策略，表示成 $\pi(s)$。

1.3.2 求解 MDP 问题

求解的目的是找到使长期期望回报（expected cumulative reward）最大化的策略。对于有限视界的 MDP 问题，长期期望回报为：

$$E\left[\sum_{t=0}^{h} R_t\right] \tag{1-9}$$

其中 R_t 是阶段 t 时获得的立即回报：

$$R_t = R(s_t, \pi_t(s_t)) \tag{1-10}$$

对于无限视界的 MDP 问题，引入折扣因子 $\lambda \in (0, 1)$ 得到带折扣的长期期望回报（expected cumulative discounted reward）：

$$E\left[\sum_{t=0}^{\infty} \lambda^t R_t\right] \tag{1-11}$$

定义值函数（value function）$V^\pi(s)$: $\mathcal{S} \to \mathbf{R}$ 为采用策略 π 时在状态 s 下的长期期望回报：

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \lambda^t R_t\right] \tag{1-12}$$

可以递归表示成：

$$V^{\pi}(s) = R(s, \pi(s)) + \lambda \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V_{\pi}(s') \tag{1-13}$$

上述定义给出了计算策略 π 的值函数的方法，同时也可以通过值函数计算得到相应的策略。为此，定义行动值函数 $Q^{\pi}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbf{R}$ 为在状态 s 下执行行动 a 的期望回报：

$$Q^{\pi}(s, a) = R(s, a) + \lambda \sum_{s' \in \mathcal{S}} P(s, a, s') V^{\pi}(s') \tag{1-14}$$

那么，状态 s 下的最优动作应为：

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a) \tag{1-15}$$

即：

$$\pi(s) = \arg\max_{a \in \mathcal{A}} \left[R(s, a) + \lambda \sum_{s' \in \mathcal{S}} P(s, a, s') V^{\pi}(s') \right] \tag{1-16}$$

同时有：

$$V^{\pi}(s) = \max_{a \in \mathcal{A}} \left[R(s, a) + \lambda \sum_{s' \in \mathcal{S}} P(s, a, s') V^{\pi}(s') \right] \tag{1-17}$$

公式 1-17 被称为 Bellman 迭代式，这是求解 MDP 问题的理论基础。

若对于所有状态 s，有 $V^{\pi}(s) \geq V^{\pi'}(s)$，则称 $V^{\pi} \geq V^{\pi'}$。使 V^{π} 达到最大的策略 π 为最优策略，用 π^* 表示，π^* 的值函数记为 V^*。

常见的求解算法主要分值迭代和策略迭代两类，值迭代算法的思想是用 V 通过迭代逐步逼进 V^*，记 $V_1, V_2, \ldots, V_k, \ldots$ 为逼进序列。迭代过程按照 Bellman 迭代式进行，容易用动态规划方法实现，见算法 1

从算法 1可以看出，动态规划需要事先知道状态转移模型 T 和回报回报 R，即需要一个环境的完整模型，很多问题事先很难给出环境的完整模型（如机器人足球，博弈问题），这时可以用强化学习方法。强化学习的主要思想是“与环境交互”和“试错”。让主体在环境中不停的决策，选择一个动作用于环境，环境接受该动作后状态发生变化，同时产生一个立即回报反馈给主体，主体根据立即

```
1  foreach s do
2      V_1[s] ← 0;
3      t ← 0;
4  repeat
5      t = t + 1;
6      foreach s ∈ S do
7          foreach a ∈ A do
8              Q_t[s,a] ← R(s,a) + λ Σ_{s'∈S} P(s,a,s') V_{t-1}[s'];
9          V_t[s] ← max_a Q_t[s,a];
10 until |V_t[s] − V_{t-1}[s]| < ε for all s ∈ S;
```

1 Dynamic Programming

回报序列不断修正状态的值函数，最终逼近最优策略的值函数，完成学习过程。常用算法如下：

蒙特卡罗方法 蒙特卡罗方法把交互过程中得到的即时回报的平均作为值函数的估计。蒙特卡罗方法仅适用于有终任务，当一个任务完成后，计算出此任务状态序列的每个状态 s 的长期回报 $R(s,\ a)$，回报函数可以由环境给出也可以编制到程序内部，用多次执行任务的平均值更新值函数。伪码见算法 2。 其中$visits[s]$ 记录了状态 s 在整个学习过程中被访问的次数，利用它实现对评价估计求平均。

λ-时序差分 时序差分（*Temporal Difference*）结合了蒙特卡罗方法和动态规划的思想，和蒙特卡罗方法一样不需要完整的环境模型；同时也像动态规划一样，随时更新评价估计，而不需要等任务完成。一个简单的时序差分学习按下式进行迭代：

$$V(s_t)\ \leftarrow\ V(s_t)\ +\ \alpha\,[r_{t+1}\ +\ \gamma V(s_{t+1})\ -\ V(s_t)] \tag{1-18}$$

在更新评价时，不是让它直接等于即时回报和后续评价之和，而是引入学习率 α 逐步调整，可以设定为一个 0 到 1 之间的常数，也可以如下：

$$\alpha\ =\ \frac{1}{1\ +\ visits[s_t]} \tag{1-19}$$

```
1  foreach s do
2      V[s] ← 0;
3      visits[s] ← 0;
4  repeat
5      s ← Initial( );
6      repeat
7          (S, A) ← GenerateEpisode(s, π);
8          foreach (s_t, a_t) ∈ (S, A) do
9              R ← Σ_{i≥0} γ^i r(s_{t+i+1}, a_{t+i+1});
10
11             R ← (V[s_t] × visits[s_t] + R) / (visits[s_t] + 1);
12
13             V[s_t] ← R;
14             visits[s_t] ← visits[s_t] + 1;
15     until s = terminal ;
16 until end ;
```

2 Monte Carlo

在学习过程中根据状态 s 的访问次数增多而不断减小。

每步行动产生的影响不仅是当前状态，也可能影响以后的状态，虽然迭代公式考虑了这种影响，把即时回报和后续评价作为评价更新的依据，但这种更新对回报函数大多数为零的问题显得较慢。如果每次行动之后不仅对当前状态更新，也对之前的很多步更新，则可以较快得完成学习，为此引入一个和回溯有关的参数 λ 来调整这种影响，得到λ-时序差分（*TD*(λ)）。

记时刻 t 的评价更新差值为：

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{1-20}$$

把迭代公式改写为:

$$V(s) \leftarrow V(s) + \alpha\delta_t e_t(s) \tag{1-21}$$

上式的 $e_t(s)$ 称为回溯权重，离当前状态 s_t 越远的状态 s，$e_t(s)$ 应越小。$e(s)$ 在每步行动后根据参数 λ 进行衰减，一种衰减方式为：

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) + 1 & s = s_t \\ \gamma\lambda e_{t-1}(s) & s \neq s_t \end{cases} \tag{1-22}$$

算法 3给出了 λ 时序差分算法。

```
1  foreach s do
2      V[s] ← 0;
3      e[s] ← 0;
4  repeat
5      s ← Initial( );
6      S ← ∅;
7      repeat
8          a ← π(s);
9          Execute(a);
10         r ← Reward(s, a);
11         s′ ← Observe( );
12         S ← S ∪ s;
13         δ ← r + γV[s′] − V[s];
14         e[s] ← e[s] + 1;
15         foreach s ∈ S do
16             V[s] ← V[s] + αδe[s];
17             e[s] ← γλe[s];
18         s ← s′;
19     until s = terminal ;
20 until end ;
```

3 λ Temporal Difference

1.4 POMDP理论

MDP 可以处理状态转移的不确定性，但却要求状态是确定的，所以不能适用于部分可观察环境，为此引入观察和观察模型，得到 POMDP 框架。

1.4.1 POMDP 的基本框架

POMDP 是一个六元组：$\langle \mathcal{S}, \mathcal{O}, \mathcal{A}, T, O, R \rangle$，其中 $\mathcal{S}$，$\mathcal{A}$，T，R 跟 MDP 中的定义一样。$\mathcal{O}$ 是主体所有可能接受到的观察的集合，称为观察空间。O 定义了观察模型，$O : \mathcal{A} \times \mathcal{S} \to P(\mathcal{O}|\mathcal{A}, \mathcal{S})$，是从状态、动作到观察上概率分布的映射。$P(o|a, s')$ 表示执行完动作 $a \in \mathcal{A}$ 转移到状态 $s \in \mathcal{S}$，获得观察 $o \in \mathcal{O}$ 的概率。

注意到回报函数 R 也可能跟观察有关，但仍然可以写成 $R(s, a)$ 的形式：

$$R(s,a) = \sum_{s' \in \mathcal{S}} \sum_{o \in \mathcal{O}} P(s'|s,a) \cdot P(o|a,s') \cdot R(s,a,s',o) \tag{1-23}$$

由于在 POMDP 中，主体不能知道自己所处的真实状态，所以 POMDP 的策略不再是从状态到动作的映射。相反，在时刻 t，主体必需根据此前的观察历史 $\langle (o_0, a_0), (o_1, a_1), \ldots, (o_t, a_t) \rangle$ 选择自己的最佳动作。当视界很大时，维护这样一个历史会占用很大资源，不过可以证明维护信念状态（belief state）是等效的。

信念状态 b 是状态上的概率分布，$b(s)$ 即为当前处于状态 s 下的概率。POMDP 问题具有一个初始信念状态，描述了初始状态的概率分布，以后每个阶段利用观察和行动更新这个信念状态，更新过程如下：

$$b_a^o(s') = \frac{P(o|a,s') \sum_{s \in \mathcal{S}} P(s'|s,a) b(s)}{P(o|a,b)} \tag{1-24}$$

其中

$$P(o|a,b) = \sum_{s' \in \mathcal{S}} P(o|a,s') \sum_{s \in \mathcal{S}} P(s'|s,a) b(s) \tag{1-25}$$

这样，POMDP 里策略就是从信念状态到动作的映射。

1.4.2 求解 POMDP 问题

我们已经看到信念状态可以完全表示观察历史，并且策略就是从信念状态到动作的映射，那么现在的问题是如何求解得到最优策略。类似地，定义策略 π 下，信念状态 b 的值函数为：

$$V^{\pi}(b) = R(b,s) + \lambda \sum_{o \in \mathcal{O}} P(o|a,b) V^{\pi}(b_a^o) \tag{1-26}$$

其中 $R(b,s) = \sum_{s \in \mathcal{S}} R(s,a)b(s)$，$P(o|a,b)$ 如式 1-25 所定义。

所以 POMDP 的 Bellman 迭代式为：

$$V^{*}(b) = \max_{a \in \mathcal{A}} \left[R(b,s) + \lambda \sum_{o \in \mathcal{O}} P(o|a,b) V^{*}(b_a^o) \right] \tag{1-27}$$

这使利用动态规划方法求解 POMDP 问题称为可能，但由于信念状态空间是连续的，实际求解还是有难度的，这里就不介绍了。

1.5 POSG理论

简单的来说POSG问题可以看作一共六元组[?]，$< I, S, A_i, O_i, P, R_i >$.其中

- I代表一个有限的智能体索引序号
- S代表有限的状态集合
- A_i代表一个对智能体i有限的行动集合，$\vec{A} = \times_{i \in I} A_i$为全部智能体行动的集合,可用向量$\vec{a} =< a_1, \ldots, a_n >$,表示一个联合行动。
- O_i是一个对智能体i有效的观察集合。$\vec{O} = \times_{i \in I} O_i$为全部智能体联合观察的集合，可用向量$\vec{o} =< o_1, \ldots, o_n >$,表示一个联合观察。
- P 是一个马尔可夫状态转移及观察的概率分布函数，可用$P(s', \vec{o}|s, \vec{a})$表示在状态$s$执行联合动作$\vec{a}$后进入状态$s'$并获得联合观察$\vec{o}$的概率。
- $R_i : S \times \vec{A} = \mathcal{R}$为智能体$i$的收益函数。

POSG的问题模型与DEC-POMDP非常接近，所不同的是在收益函数方面，POSG的各个智能体可以有不同的收益函数。

第 2 章　Server模型及WE2009结构框架

RoboCup 仿真 2D 比赛为研究大规模不确定环境下的多主体决策问题提供了标准测试平台。然而，由于规模上的原因，在该平台上进行整体规划的设计离不开对某些子问题的处理。本章将配合对 2D 平台的一些必要的说明，针对一些典型的子问题进行介绍。

2.1　基本介绍

主体，也就是参与比赛的球员程序的基本结构如图 2.1 所示，主要包括三部分内容：服务器接口、信息状态模块、决策模块。

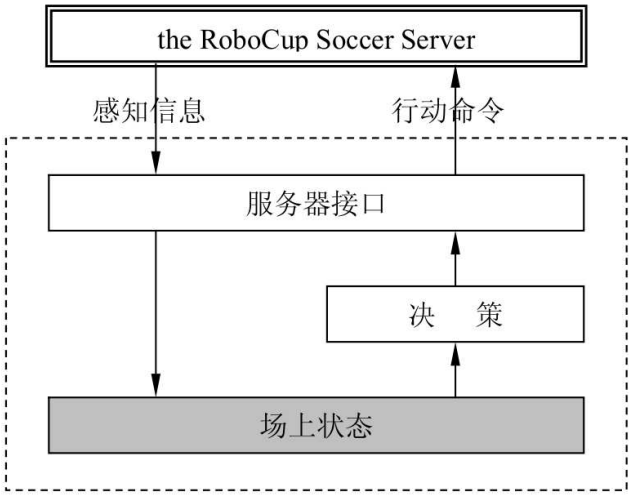


图 2.1　主体总体结构

服务器接口 服务器接口是主体与环境交互的桥梁，一方面它接收来自 server 的感知信息，并加以解释，转化成信息状态模块可以理解的信息结构，另一方面它负责把决策模块产生的动作命令发送给 server

信息状态模块 能供决策模块决策使用的信息统称信息状态，主要包括世界状态、截球信息、策略信息等，其中截球信息、策略信息是由世界状态经过一些计算得到的较高级的信息。信息状态模块主要负责信息状态的维护和更新

决策模块 决策模块是主体的核心，它使用信息状态模块提供的信息进行决策，最终产生 server 接受的动作命令，通过服务器接口发送给 server

下面重点介绍一下信息状态模块和决策模块。

2.1.1 信息状态模块

信息状态模块的详细结构如图 2.2 所示。 Parser 将从 server 直接接收

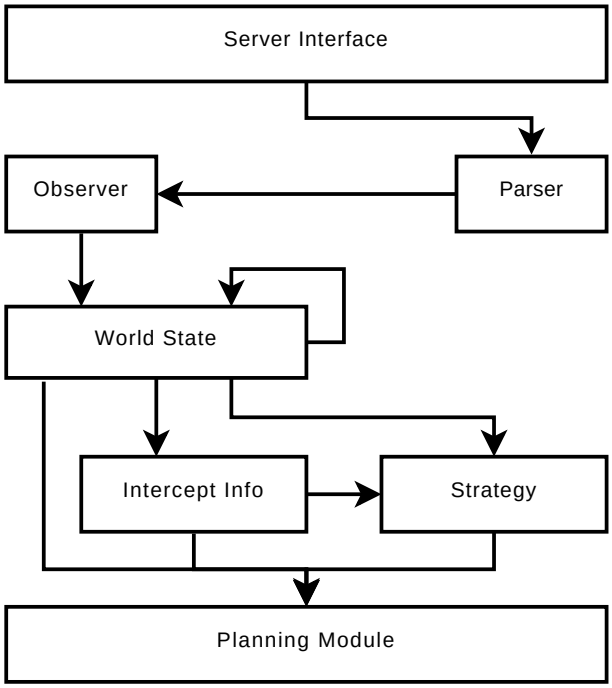


图 2.2 信息状态模块的详细结构

来的字符串信息解析成包括视觉信息、感觉信息等在内的感知对象交给 Observer，WorldState 使用从 Observer 得到的感知对象并结合历史世界状态完成更新，InterceptInfo 是从 WorldState 里面挖掘得到的截球信息，Strategy

存储了最快截到球的队员、球自由运动剩余的周期数等方便使用的策略信息。WorldState、InterceptInfo、Strategy 一并作为信息状态交给决策模块决策使用。

2.1.2 决策模块

采用分层的思想，决策主要分为三个层次：战略决策、战术决策和个人技术决策。

2.1.2.1 战略决策

战略层是在决策系统模块中的最高层。在这层中自主体决定自己是哪一角色，所处的阵形，判断出自己处于哪个模式，是进攻，还是防守，要执行什么任务，或配合完成什么合作协议，明确当前要达到的战略目标。战略决策是三层决策中最高层的决策，也正因为它的决策高度，它所考虑的信息是非常之多的，也是非常复杂的，在目前除了根据人类踢足球的常识应用分析的方法来实现外，还没有其他更好的办法。我们使用的最为成功的一个方法就是一种能基于状态的搜索，这也是要基于一定的足球知识来得到的。

2.1.2.2 战术决策

当战略层确定了所处的行为模式后，战术层来具体贯彻执行。通过一些细节分析后，本层制订出为了完成这一特定的目标，所应该采取的行为策略。包括跑位、造越位、抢球、拦截、带球、传球、射门和盯人等等。在实际中我们都为每个行为量身定做了一套模型，这套模型中普遍的思想是利用试错搜索返回的错误信息结合具体经验的启发式搜索。

2.1.2.3 个人技术决策

个人技术决策是决策系统的最低层，用以实现自主体的个人技能(如踢球、跑动等)。在每个周期，最终都是由这一层发出本周期所要执行的基本动作命令。这一层更多关注的平台本身的特点和高层的良好交互，被抽象成很多优化问题，适合使用 MDP 或 POMDP 求解。

2.2 server内部模型

每周期开始结束前，server根据收集到球员发送过来的控制命令，更新平台内部场景，并在下周期开始前根据各个球员的视觉等条件，分别发送相应的信息。球员则根据server每周期发送过来的场景构建整个场景模型。以下分别介绍server的动力学模型，误差模型，以及视觉模型等。

2.2.1 server的动力学模型

在server中，物体的运动时通过一个简单的逐步仿真进行的。每个仿真周期，server都通过如下公式计算：

$$(u_x^{t+1}, u_y^{t+1}) = (v_x^t, v_y^t) + (a_x^t, a_y^t) + (r_x^t, r_y^t) + (w_x^t, w_y^t)$$

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

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

$$(a_x^{t+1}, a_y^{t+1}) = (0, 0)$$

其中(u_x, u_y)代表server内部更新位置使用的速度，(v_x, v_y)代表实际观察到的速度，(a_x, a_y)代表加速度，(r_x, r_y)代表物体移动中受到的噪音影响，(w_x, w_y)代表模拟中风向等影响,(p_x, p_y)代表物体位置。Decay代表速度每周期的衰减，对象若不同，Decay的值也不同。每周期server会根据对象的动作和受力等更新对象的加速度，有此得到对象速度和位置的改变。

2.2.2 server的误差模型

简单的说，server的误差模型分为三类，一类是执行动作的误差，一类是在模拟中加的误差，另一类是发送给人的时候加的误差。而第三者往往占据了主要成分。第一类误差在底层的计算中考虑较少，因此以下就分别介绍最后两类误差。

2.2.2.1 模拟中的误差

由前面所述的动力学模型可见，模拟中分别加上了(r_x, r_y)的噪音误差和(w_x, w_y)的风向影响误差。

对于噪音误差，server随机产生的(r_x, r_y)的值是模取0到速度(v_x, v_y)的模乘以误差因子randp之间取随机数，方向取$-180°$到$180°$之间的随机值。目前server（13.2.1）球的randp因子是0.05，球员的randp因子是0.1。噪音误差是模拟过程中的主要误差，对精确计算对象位置等影响很大。

对于风向误差，server内部随机产生的风向$(windvector_x, windvector_y)$和随机因子wind_rand,server根据速度的模，风向和随机因子按照一定模型产生一个随机量(w_x, w_y).但是对于当前server，由于wind_rand始终为零，当前server模拟中(w_x, w_y)实际上为$(0, 0)$.因此暂时对server模拟没有影响。

2.2.2.2 发送过程中的误差

发送过程中的误差在server所加的误差中占据了主要成分，但是发送过程中的误差却是server模拟现实环境不可缺少的一部分。发送过程的误差集中体现在视觉上，总的原则是让距自己距离近处的物体误差相对较小，距自己距离远处的物体误差相对较大。以下以位置为例具体介绍server在发送过程中所加的误差。

对于位置的计算一般是通过视觉。server发送视觉中的物体位置信息是分别发送其距离自己的距离dist和相对对自己观察方向角度dir。视觉中的物体由，场内的标志，线，球，球员。目前自己位置的计算时根据场内标志相对自己的位置计算出来，因为标志时固定的。球和球员的位置时根据自己的位置以及视觉中它们相对自己的位置计算出来的。 server在发送时给dist和dir加误差使用的算法时一样的。如下所示：

```
1  double Quantize(const double & v , const double & q)
2  {
3          return rint(v/q) * q;
4  }
5  double calcQuantDist(const double & dist , const double& qstep)
6  {
7          return Quantize(std::exp(Quantize(std::log(dist + EPS)
               , qstep)) , 0.1);
8  }
9  int Rad2IDegRound(const double &a)
10 {
11         return static_cast<int>(rint(a * 180 / M_PI));
12 }
13 int calcDegDir(const double &a)
14 {
15         return Rad2IDegRound(a);
16 }
```

由如上代码可见，dir发送过来的一定是整数，由于采用四舍五入，其最大误差为0.5度。对于dist，其加的误差却随着qstep的不同而不同。对于视觉中物体根据qstep可以分为两类。一类时比较精确的，例如场内标志等，当前server（13.2.1）qstep参数是0.01.一类时比较模糊的，例如球和球员，当前server（13.2.1）的qstep参数时0.1。qstep不同得到的误差模型差别很大。以下分别以真实值为横坐标，加误差后的值为纵坐标，比较qstep为0.01和0.1的不同。

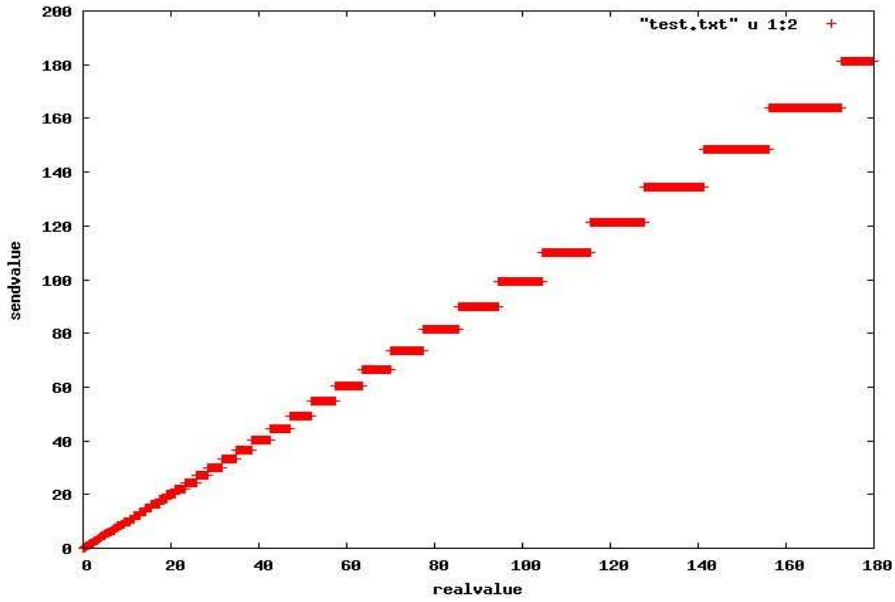


图 2.3　qstep为0.1时误差图

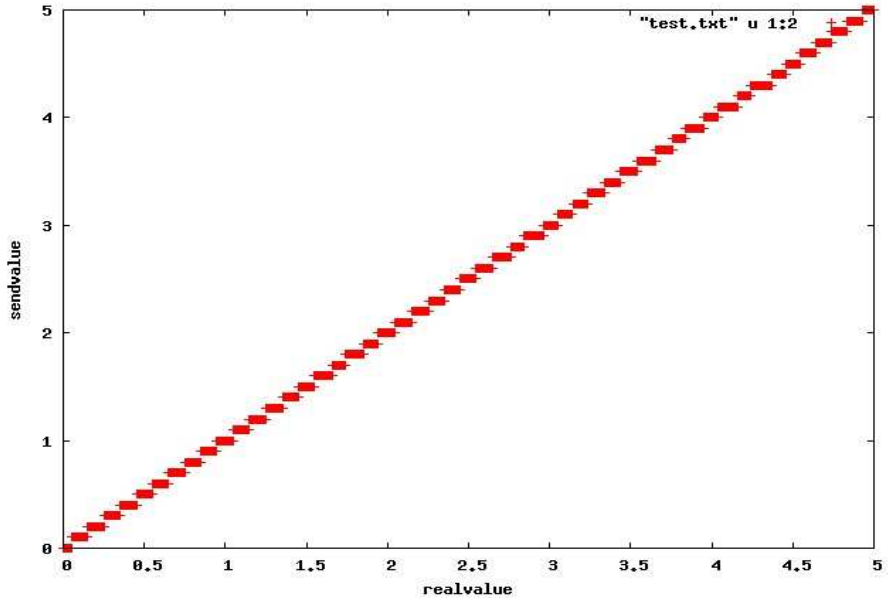


图 2.4　qstep为0.01时误差图

可见，server发送过来的dist时一系列离散的值，每一个值对应一个区间，但是经过计算几乎全部区间的平均值并不是此发送的值，因此可以用此平均值代替发送值。这可以从理论上提高计算精确度。

2.2.3 球员视觉模型

球员通过视觉获得视野中的物体信息，每个一段时间，视觉感知信息由server自动发送到球员处。视觉中的信息全部是相对自身的信息，不能直接获得视觉中物体全局位置等信息。

控制视觉到来时间frequence主要由视野范围width，视觉质量quality决定.视觉范围有三种，narrow，normal和wide。当前server（13.2.1）对应角度为60，120，180.视觉质量为high和low两种。其关系如下

$$frequence = step \times widthfactor \times qualityfactor$$

$step$是默认视觉到来周期，当前server下为2,$widthfactor$与世界范围width有关，narrow时widthfactor为0.5,normal时widthfactor为1,wide时为2;$qualityfactor$与视觉质量quality有关，为high时为1，为low时为0.5。在WE2009中一般不采用low视觉。

另外，球员有一个以visible_distance(大约3米）为半径的邻域，物体在此范围内即使不在视角内，球员也可以看到物体大概类型和位置，但不知道物体的确切类型（如球员号码）等。另外server内规定球员获得视觉信息的完整性由物体的距离所决定，为此由四个距离unum_far_length , unum_too_far_length,team_far_length ,team_too_far_length大致规划了视觉的完整性.下图2.5具体表现了视觉的完整性：

假定a到f表征为球员，对于球员a，由于其在邻域，因此可见，但是只能知道他为球员或球或标志，不能知道他号码，球队等其他信息;对于球员b和g由于其位于视角外也不在邻域中，因此其不可见。对于球员c可以得到其球队名称和号码。对于球员d可以识别队名，但是仅由50%几率可以识别号码。对于球员e，只有25%几率可以识别队名;对于球员f，虽然也在视野中但是只是匿名球员。

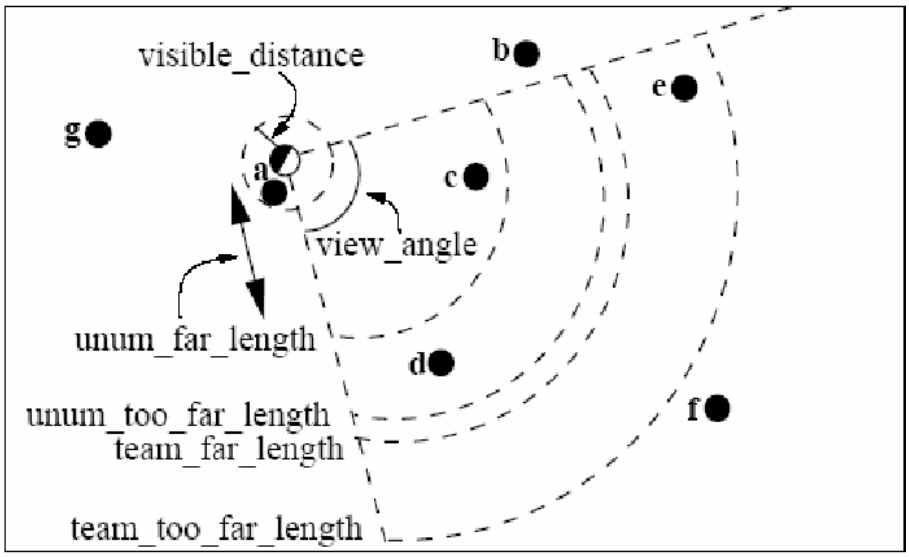


图 2.5　球员视觉范围

2.2.4　球员听觉模型

由于仿真听觉带宽拥挤且不可靠，每个球员都有一个有限的听力能力，最大值为server参数HEAR_MAX.球员每收到一条信息，听力能力下降hear_decay，每周期听力能力上升hear_inc,当听力能力小于hear_decay 时球员便不可听到信息。当前server大概一个球员最多两周期才能收到一条听觉信息。

如果一周期由多个听觉信息到达，球员只能接受到其中随机一个，其他会被server丢弃，因此听觉信息时不可靠的;球员的通讯范围也是有限的，一个消息只能到达距离发送消息的人小于audio_cut_dist的球员。例外的是教练的消息可以不受听觉能力和距离的限制，总能使所有球员都听到。

2.2.5　身体感知模型

身体感知是指每周期server发送球员的自身状态，类似机器人上传感器。感知信息主要有，自己的视角，自己的体力和体力容量，自己速度，自己脖子角度，以及从比赛开始各个动作的执行次数之和。

根据视角可以判断当前周期有无视觉到来，自己的脖子角度与全局视觉角度可计算出身体角度。值得注意的是自己脖子角度所加的误差时按照如下算法的：

```
int Red2IDeg(const double& a)
{
        return static_cast<int>(a * 180 / M_PI);
}
```

比如，如果返回角度时1.0，其代表1.0到1.99可能角度，最大误差接近1度,如果用1.5度代替1度进行计算，这样就把最大误差降到0.5度了。

2.3 WE2009新球队结构

WE2009新球队是参照原球队的基础上，按照POSG（Partial Observable Stochastic Games）模型对仿真2D问题建模并规划出新球队结构，按照新结构重写所有球队代码，并最终去用此新球队参加Robocup世界杯比赛。

2.3.1 根据POSG为仿真2D问题建模

可用将将POSG的六个要素分别对应仿真2D问题相应的部分。

- 智能体索引I:对应场上22名球员和2个教练;
- 状态S:由当前时间，比分，所有动态物体的位置，速度信息，以及个球员的身体姿态等组合而成，对应了一个能完整描述问题的世界状态。
- 行动A_i:由server指定的原子动作构成，对应了每个离散周期球员可用发送给server的指令。例如dash，turn，kick等。
- 观察O_i：对应server没周期发送给球员的视觉，听觉，感觉信息。
- 转移函数P:对应了描述过程变化规律的信息，主要体现server模拟中所遵照的动力学定律及server中人为制订的比赛规则等。
- 收益函数R_i:智能体i执行动作的立即收益，特点是球队内部一致，球队间相反，涵盖了合作与对抗两类问题的特点。

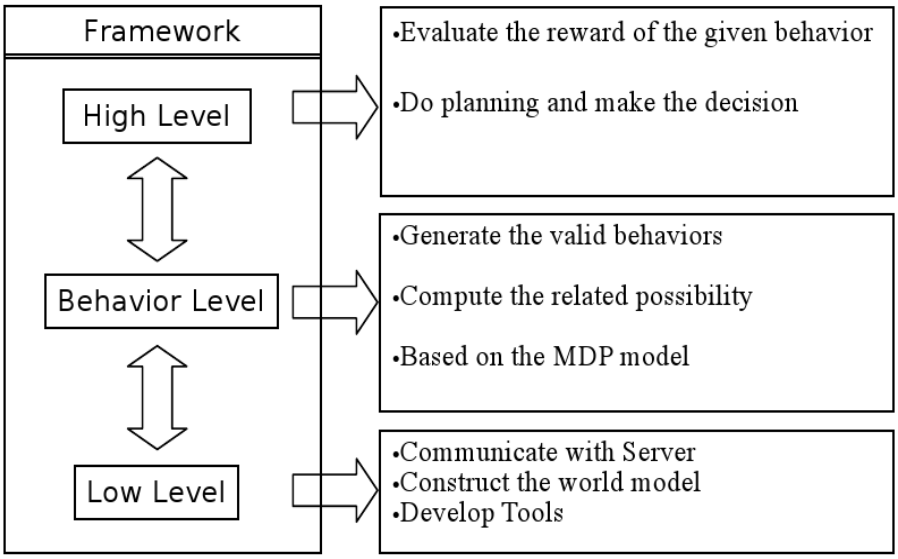


图 2.6　问题初步架构

2.3.2　根据POSG初步规划球队的架构

通过POSG模型，我们可以更好的了解仿真2D问题，对应求解此问题，规模是主要问题。最基本的做法是采用分层的思想，将问题框架分为三层处理。

最底层负责与server通讯并分析由server发送过来的信息从而构建世界模型，世界模型以状态表示。中间层也就是行为层，负责在给定状态队给定的球员生成各种可能的行为，如跑位，传球，射门等，¿输出的行为是一系列有效行为的集合，可能的结果状态，相关的概率等。高层负责对行为曾进行长期的评价及长期的规划等，并最终做出决策。初步结构如图2.6。

2.3.3　WE2009结构简介

按照上一节规划三层结构，球队可以同样分为三层（以普通球员为例，教练分层不明显）。

- 底层：Paser类，Observer类，WorldState类，WorldModel类，Communicate-System类的一部分。

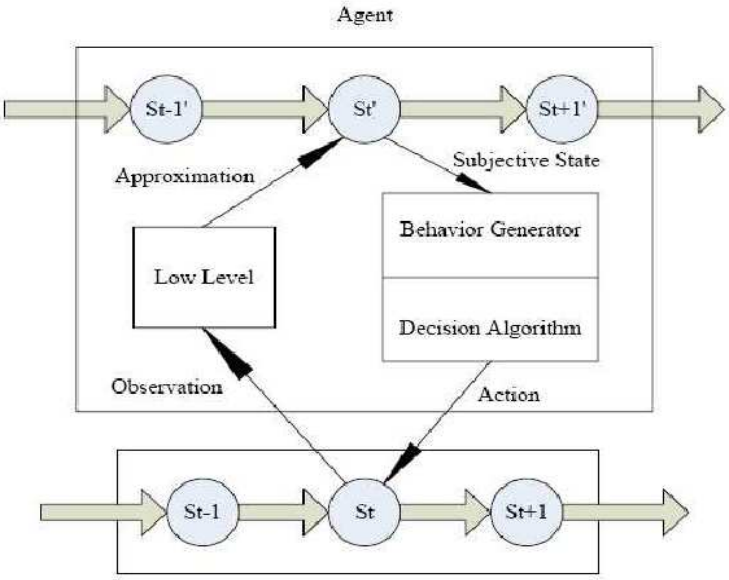


图 2.7　WE2009架构

- 行为层：Behaivior开头的所有类，InfoState类，Strategy类等等一些为Behavior辅助的类。
- 高层：DecisionTree类及DecistionData类，VisualSystem类，CommunicateSystem类的一部分。

如图2.7,每周期开始server发送过来的信息通过Paser类及CommunicateSystem类的一部分解析后，存入Observer类中，再根据Observer类的内容，构建当前环境状态WorldState，己方视角的WorldState和对方视角的WorldState共同构成了世界模型WorldModel。再根据WorldModel中信息更新行为层使用的信息InfoState，Strategy等等，并通过Behavior开头的类生成一系列有效的动作集合，可能的结果状态，及相关的概率。最后有DecisionTree类完成主要动作评价及选择，VisualSystem类完成视觉相关动作的评价及选择，CommunicateSystem类完成听觉动作的相关评价及选择。

第 3 章　环境更新设计

本章将详细介绍仿真2D环境中更新细节。本章将分以下几个部分进行。

- 环境更新大体框架
- 环境更新各部分细节

3.1　WE2009中部分可观察环境的框架

由球员的视觉模型可以明显的看出，球员对环境只能观察一部分，并不能全局观察，但是在比赛中却需要每周期都对全局环境了解，因此必须建立根据局部信息估算全局信息的模型，也就是完成对仿真2D中部分可观察环境的建模。

3.1.1　对问题的初步分析

由server模型可以看出，得到场景下一周期物体信息由三种途径，通过上一周期模拟，通过听觉信息得知，通过视觉信息或感知信息获得。而每周期都可以根据是否由听觉和视觉等将物体分为可知物体¿和不可知物体。对于可知物体，有如上三种获得信息途径。对于不可知物体，只能通过上周期信息按照server的内部模型模拟。而且以上三种方式都可以根据server模型计算其最大误差范围。因此初步确定的更新模型如下：

- 对空间中所有对象的全部信息估算到下一周期，并计算其最大误差范围。
- 通过听觉信息计算可知物体的信息，并计算其最大误差范围，再于相应的物体的信息最大误差范围比较，选择误差范围大的覆盖。
- 通过视觉和感知信息计算可知物体的信息，并计算其最大误差范围，再与相应的物体的信息最大误差范围比较，选择误差范围大的覆盖。

3.1.2　WE2009环境建模框架

目前设计的底层更新的框架如下:

- 1.根据听觉更正上周期信息.
- 2.场内所有信息根据server内部模型模拟至本周期.
- 3.根据听觉更正当前周期的信息.
- 4.根据感知信息,更新自身信息.
- 5.判断是否有视觉信息,如果没有转至11
- 6.更新自身位置,身体角度等与视觉有关信息.
- 7.更新可知编号的球员的信息.
- 8.更新不可知编号的球员的信息.
- 9.更新球相关的信息.
- 10.如果视觉周期不是当前周期,按server内部模型模拟到当前周期.
- 11.根据感知信息中的碰撞信息修正.
- 12.评价环境中各信息的置信度.
- 13.计算与动作相关的信息.

对于最大误差范围计算，由于信息过多，若对所有的信息均计算最大误差范围，工作量大而且没有必要。买目前在更新中只对球和自己的位置和速度信息设置了最大误差范围比较。因为在实际环境建模¿中，这两部分信息要求的准确度最高，而且使用效果也最好。

通过更新框架可以看到，环境更新可以分为一下几个部分介绍。

- 模拟更新部分
- 听觉更新部分
- 自身信息更新部分
- 球员信息更新部分

- 球信息更新部分
- 视觉延迟处理部分
- 碰撞处理部分
- 置信度评价部分
- 特殊模式处理部分
- 动作相关信息更新部分

3.2 模拟更新部分

server内部总是按照一定的模型进行模拟，在已知上周期信息的条件下，如果按照server模型进行模拟的话可以大致得到下周期的信息。影响模拟准确度的条件主要是两个，一是server模拟的误差，二是球员行动的不确定性。对于server模拟的误差主要是动作执行的误差和模拟过程中的噪音误差，球员行动的不确定，使得有些信息的模拟完全不准确。模拟更新主要分为三个方面.

3.2.1 自身信息的模拟更新

自身信息模拟更新相比较其他两部分而言，模拟的准确度要高的很多，原因主要是对于自身的模拟不存在行动的不确定性，每周期自己行动都可以通过上周期的决策历史信息得到，动作的执行与否也可以通过 sense 信息得到。影响自身信息模拟的准确性主要是server模拟的误差，表现在一方面是自己执行动作时有误差，比如球员想踢出1.7的球速，如果他的kick_rand为0.15的话那么他踢出的球速就可能是1.55到1.85（估计值，真实可能不一样）,另一方面是server模拟的误差，也就是上一章提到的噪音误差。

更新主要分两个方面，在有动作时根据动作更新，在无动作时根据 server 模型模拟到下一周期。动作模型太多，在此就不作介绍，server 模拟的公式可见上一章关于 server 的动力学模型的介绍。对于自身信息，计算最大误差范围eps主要针对位置，因为自身速度是通过 sense 信息发送过来，精度很高，而且每周期都有，因此没有必要使用最大误差范围提高精度。对于位置的eps，分两种情况，

在碰撞时由于内部模型很复杂把eps设置成很大的数以强制更新，在不碰撞时主要根据 server 的噪音模型计算，公式如下：

$$eps_{t+1} = eps_t + (pos_{t+1} - pos_t).Mod() * player\_rand$$

.其中pos_{t+1}表示模拟后的位置，pos_t表示模拟前的位置。

3.2.2 球的模拟更新

球相对于自身信息而言没有其准确，原因是球会受到球员施加动作，而这些动作除了自身对球的动作以外，其他动作不可知。这就造成了其他球员踢球时，球的信息模拟不准确。然而对于自己踢球动作可以按照踢球模型获得球下周期的位置和速度。同时，即使在无动作时，球模拟也受到 server 误差的影响，同样也受噪音误差影响。

对于球的更新也分两个方面，一是自己踢球时根据动作信息更新球的位置和速度，一是自己无动作时按照 server 的动力学模型模拟。踢球的模型暂不介绍，可以在 server 的手册中获得，server 的模拟公式仍然参照上一章关于 server 的动力学模型介绍。对于球来说，主要对其位置和速度计算最大误差eps，由踢球模型可以得到，踢球的模拟最大不准确度为$2 \times kick_rand$,踢球时，对于位置，$epspos_{t+1} = epspos_t + (pos_{t+1} - pos_t + 2 * kick\_rand).Mod() \times ball\_rand$,对于速度，$epsvel_{t+1} = epsvel_t + (epspos_{t+1} - epspos_t) \times ball_decay$在没有踢球时，eps公式也如上，只不过需要将$kick_rand$设为0.

3.2.3 球员的模拟更新

相对于球和自身信息来说，球员信息是最不准确的，因为球员是自主的，每周期都可按照自己的意愿运动，如果按照 server 内部模型模拟，因为受到行为不确定的影响，很难保证模拟的准确性。相比之下，server的误差影响要小的多。

对于球员来说，因为模拟信息不准确度很高，使用最大误差范围估计的话也就没有意义，因为没办法正确估计，或者说估计的值基本都是一样的。对于球员的模拟并不是完全按照 server 动力学模型模拟，具体体现在如果上周期看到球员速度，那么估计球员下周期也会按照这个速度运行，认为速度不衰减。之后的话就按照server模型模拟。

3.3 听觉更新部分

听觉信息信息是上周期其他球员发送，当前周期接受到的。听觉信息可以按照信息的时间分为两类，一类是上一周期的信息，一类是本周期的信息。两类信息作用完全不同，需要分开更新。前一类听觉信息主要用来广播位置，而后一种主要是一些配合决策。

3.3.1 听觉信息为上周期信息

听觉信息为上周期信息更新是 UpdateFromAudio 函数，更新的内容主要分为两个部分，一类是 coach 发送过来的信息，一类队友发送过来的信息。

对于 coach 发送过来的信息，目前更新的主要是 coach 统计的对手类型信息和 coach 统计的体力信息等，这些信息球员本身不可获得但又非常重要，只能通过可精确获得全局信息的 coach 统计得到。

对于队友发送过来的信息，也可以分为两部分，一类是球的信息，另一类是球员的信息。对于球的信息，前面提到的使用最大误差范围进行选择更新在这里并不适用。因为对球的最大误差估计不一定准确，特别是当队员踢球的时候。此时的更新的条件是当球位置完全是猜测，或者置信度小于一定值，或者说话人距离球位置比自己近的时候采取覆盖。eps的计算大多是调试出来的值。对于球员，更新的条件与球类似。

3.3.2 听觉信息为本周期信息

听觉是本周期信息，一般是用于策略配合上，例如，球员A要传球给B，那么A在踢出前一周期喊传球，并说出球踢出的速度和位置，也就是下一周期位置。球员B收到消息时便正好是当前周期的信息。便用来更新底层。此时传出的信息主要是发送者自身信息或者球的信息。另外对球的信息更新采取的是完全覆盖，因为此时传送的信息一般相当准确，其eps也基本上采用的经验值。

3.4 自身信息更新部分

自身信息更新可以分为两部分，一类是根据感知信息更新，一类是根据视觉信息更新。以下具体介绍更新过程。

3.4.1 感知信息更新

前面说过球员从 server 获得信息可以通过视觉，听觉，感知获得。感知信息获得的全部是自己的信息，在此全部更新。感知信息解析出来的数据大多可以直接应用，并不需要做过多的处理，值得注意的是前面在介绍误差模型时提到的对于$neckdir$，由于 server 内部采用的是取整的加误差方法，具体表现是1.0对应1.0 到1.99,如果用1.5代替1.0进行计算的可以将原来的误差缩小一半。这是旧球队所没有的。

3.4.2 视觉信息更新

自身信息中，适用视觉更新的主要是自己的位置，自己的身体方向。至于自己的速度则在感知信息中获得。

自己身体角度计算根据全局脖子角度和相对脖子角度（$neckdir$)确定身体角度。全局脖子角度（$neckGlobalAngle$)，是通过视觉中球场边线获得的。计算公式如下：

$$neckGlobalAngle = lineGlobalAngle - lineRelativeAngle$$

边线的全局角度是已知的，server发送过来的是边线角度（$lineRelativeAngle$),由上一章$neckGlobalAngle$的误差模型获得其误差为0.5度。由$neckGlobalAngle$-和$neckdir$便可以得到身体角度（$bodydir$).

自己位置的计算主要是根据角标计算，从视觉中所有的角标选出最近的一个（由server模型可得，越近越精确）。然后根据公式

$$playerPos = flagPos - polar2vector(flagDist, neckGlobalAngle + flagDir)$$

计算。其中，$flagPos$是已知的角标的全局位置，$flagDist$，$flagDir$是 server 发送过来的角标距离自己的 dist 和 dir （见上一章 server 视觉模型）。通过上一章介绍的server误差模型可以看出，server 发送过来的 dist 是离散的值，对应的真实值是一个区间。由计算结果显示发送过来的 dist 大多并不是区间的平均值，用平均值代替dist进行计算的话理论上可以减小误差。因此计算中的$flagDist$均是替换过的。假设区间长度为$2 * delta_d$, 则$flagDist$对应的真实长度区间是$[flagDist - delta_d, flagDist + delta_d]$,另外，由 server 误差模型可

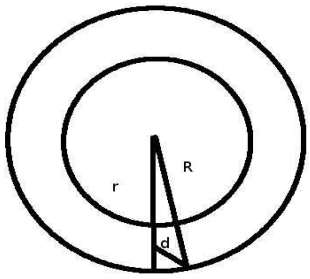


图 3.1　位置最大误差模型

得$flagDist$误差为0.5.另外$neckGlobalAngle$是全局位置，也可由 server 模型得到其误差为0.5。因此可以得到计算后的值对应的真实值的误差范围如图3.1.

如图其中 R 代表$flagDist + delta_d$，r 代表$flagDist$,d 是包含此范围的最小半径，因此可以用来作自己位置的最大误差范围 eps 。误差角度为$neckGlobalAngle$和$flagDir$误差角度之和为θ。可以通过如下公式计算

$$eps = \sqrt{r \times r + R \times R + 2 \times R \times r \cos\theta}$$

3.5　球员更新部分

在上一章介绍视觉模型时，就可以见到，视觉中球员有可以同时知道队名和号码的，有只知道队名的，有匿名的。因此可以根据如此大致将球员的更新分为两类，一类是完全已知队名和号码的，一类是不完全已知队名和号码的。

3.5.1 已知队名和号码的球员的更新

对已知队名和号码的球员，其更新的量有位置，速度，身体角度，脖子角度，手指向，铲球与否等。

对于位置的更新，按照如下公式解析 server 信息进行

$$playerPos = selfPos + polar2vector(playerDist, neckGlobalAngle + playerDir)$$

其中，$playerPos$代表球员的位置，$selfPos$代表自己的位置，$playerDist$代表 server 发送过来视觉中球员相对自己的距离，$playerDir$代表 server 发送过来球员相对脖子的角度。$neckGlobalAngle$代表脖子的全局角度。

对于速度的更新，按照如下公式解析 server 发送过来的视觉信息。

$$relPos = polar2vector(1.0, playerDir)$$

$$relVel.x = distChg * relPos.x - (dirChg * PI/180 * playerDist * relPos.y)$$

$$relVel.y = distChg * relPos.y + (dirChg * PI/180.0 * playerDist * relPos.x)$$

$$playerVel = selfVel + relVel.rotate(neckGlobalAngle)$$

其中，$playerDist$,$playerDir$是 server 发送过来的相对于自己的距离和角度。$distChg$是观察到的距离变化量，$dirChg$是观察到的角度变化量。速度在视角内且距离自己20米内才能看到，因此存在看到球位置却看不到速度的情况。例如邻域内或者视角内距离过远。如此只能根据位置相减获得球员的速度信息。公式如下：

$$playerVel_{t+1} = playerPos_{t+1} - playerPos_t$$

值得注意的是$playerPos$是根据自己的位置算出来的，其中自然附加了计算自己位置的误差，为了消除这部分误差，使用本周期的速度代替自己位置相减则可以很大程度上减小球员苏的的误差。优化后的公式如下：

$$playerVel_{t+1} = playerPos_{t+1} - playerPos_t - (selfPos_{t+1} - selfPos_t) + selfVel_{t+1}$$

至于球员的身体角度等信息，更新都非常简单，在此就不多做介绍。

3.5.2 不完全已知队名和号码的球员的更新

对不完全已知队名和号码的球员更新是首先采用身份预测算法猜测出各个球员的真实号码等，然后进行更新。更新的公式前面已经具体介绍过了，这里详细介绍身份预测算法，这是底层更新的最难点之一。

用集合P代表待匹配的老球员，集合O代表待匹配的未知球员，身份预测算法如下：

- 1.对所有的p_i, o_j，当满足$radius_max(p_i) > distance(p_i, o_j)$,将$p_i$与$o_j$相互加入对方的匹配列表中。
- 2.对所有未知球员o_j, 当o_j的匹配列表只有一个元素时，把该元素对应的老球员p_j与o_j匹配，从O剔出o_j,并通过pj查询到其他可能与其匹配的未知球员，将p_i从它们的匹配列表中删除。
- 3.重复步骤2,直到集合O中元素的新一轮匹配未能识别任何球员，进入步骤4.
- 4.按照$distant(p_i, o_j)$最小的原则，由贪心算法给出对剩余所有未知球员的匹配。

使用如上算法，经过一些预处理分析和根据 server 特性的优化，匹配基本上可以达到100%正确。

3.6 球信息更新

球信息属于整个底层中要求准确度最高的信息之一，如何更准确的更新球的信息是底层更新的难点之一.球更新的信息主要是球位置和速度。以下分别介绍球位置和速度的更新。

3.6.1 球位置更新

对于球位置，对 server 发送过来的信息处理的公式如下：

$$ballPos = selfPos + polar2vector(ballDist, neckGlobalAngle + ballDir)$$

其中，$ballDist$,$ballDir$是 server 发送过来的球相对自己的距离和角度。对球位置所做的减小误差的方式主要有两种，一种可称为在特殊情况下的修正，一种是根据最大误差范围修正。

在特殊情况下的修正主要有，对方开球时球与自己的距离不可能小于一个规定值$freekickBuffer$,以及当自己与球碰撞时，根据 server 模型可以得到此时人与球的距离恰好是球的大小和球员的大小之和。

通过最大误差范围eps修正，是针对球位置修正的主要部分。计算球视觉最大误差范围模型参见计算自身位置使用的模型。

使用最大误差修正最麻烦的地方是之前模拟球位置计算的最大误差范围实际上有可能会因为其他球员踢球等导致不正确。此时如果不用当前计算值可能导致信息不正确。因此在使用最大误差范围前必须判断其是否正确，判断主要根据的是看到的位置是否在替换前的可能区域里，如果不在的话说明存储的最大误差范围不正确，应直接用当前计算值更新。这样可以避免大部分情况。另外需要注意的是由于近身球速特别重要，修正不针对近身球位置，因为以后要用到位置相减估计速度,如果修正则会导致近身球速不准。

使用最大误差修正主要体现在三个方面。

- 看到位置在原先误差范围内时，根据当前值和以前值的最大误差范围大小决定使用谁更新。
- 看到位置不在原先误差范围内且连续两周期看到位置时，根据移动最小原则，将球移动到朝向最新位置的最大误差的边界上。
- 看到位置在原先误差范围内，且最大误差范围相同时，原先值与最新值求平均值。

3.6.2 球速更新

有视觉时球速获得有两种途径，一种时根据 server 发送过来的$distChg$等量更新，另一种是根据球本周期和上周期位置相减更新。为了更准确的获得球速，球速的大体更新结构是先通过视觉更新，再通过位置相减修正。

球速根据$distChg$等信息更新，对 server 发送过来的信息处理公式如下：

$$relPos = polar2vector(1.0, ballDir)$$

$$relVel.x = distChg * relPos.x - (dirChg * PI/180 * ballDist * relPos.y)$$

$$relVel.y = distChg * relPos.y + (dirChg * PI/180.0 * ballDist * relPos.x)$$

$$ballVel = selfVel + relVel.rotate(neckGlobalAngle)$$

由此可以计算的到视觉速度。参考 server 对速度所加的误差，计算最大误差范围公式如下：

$$mod.x = 0.02 * ballDist + fabs(distChg)/ballDist * distEps + 0.02 * distEps$$

$$mod.y = 0.1*PI/180*ballDist+fabs(dirChg)*PI/180*dEps+0.1*PI*dEps/180$$

$$eps = mod.Mod()$$

其中$dEps$是$ballDist$对应的真实值的区间半径。计算最大误差范围后，仍然采用最大误差范围判断是用以前的值还是用当前计算值。

通过位置相减与球员计算位置类似，最大不同是球员在看不到球速的时候才采用，对于球即使看到球速也要用位置相减来修正。修正采用的是最大误差范围判断。位置相减公式如下：

$$ballVel_{t+1} = ballPos_{t+1} - ballPos_t - (selfPos_{t+1} - selfPos_t) + selfVel_{t+1}$$

由于消去了自身位置的误差，此时eps计算只是本周期$ballDist$的误差半径和上周期$ballDist$的误差半径之和。在近身使用位置相减修正速度时条件很苛刻，在此就不做详细叙述。更新时仍然根据最大误差范围判断是否采用之前的还是当前计算值。

3.7 视觉延迟处理部分

通讯采用的是UDP/IP协议，不能保证通讯稳定性，另外也可能因出现球队计算超时等同步问题，server 发送过来的消息有可能已经延迟一个周期的。为了不浪费视觉信息，必须尽可能的利用起来。

准确的来说，视觉延迟处理部分并不是单独的一个部分，而是穿插在视觉更新中间的，视觉更新中有一个象征更新视觉时间的量$mSightDelay$,当它为0时表示更新本周期视觉，为1表示更新上周期视觉。如果更新不是本周期视觉，再通过模拟函数$EstimateToNow$将信息更新到当前周期。

另外值得注意的是当$mSightDelay$不为0时，不采用使用最大范围误差进行更新.因为开始更新就直接将信息模拟到最新，此时底层存储的时当前周期的模拟信息，视觉更新时，如果不是当前周期的话显然不能比较它们之间的最大误差范围。

3.8 碰撞更新部分

碰撞更新主要针对自己与球或球员碰撞，且当前没有视觉时对模拟的信息所做修正。server 内部在发生碰撞时对碰撞的物体除了强制移动外，还将碰撞双方的速度分别乘以-0.1 ,以表示因碰撞速度减慢。强制移动因为无法估计其方向，在没有视觉时无法修正，因此在此仅针对速度做相应的修正。

碰撞主要分为三类，一类是门柱碰撞，一类是与球碰撞，一类是与球员碰撞。自己的速度每周期都通过感知信息发送过来，碰撞一般不针对自己的信息修正。碰撞信息修正过程如下：

- 与球碰撞,将球速乘以-0.1,
- 与球员碰撞，先计算距离自己最近的球员，将其速度乘以-0.1。

3.9 置信度评估部分

置信度评估部分主要是部分不精确的底层信息做的特殊处理，处理包括三方面，一是预测球应该在视野中，但是不在视觉信息的处理;一类是预测某个球员应该在视野中，但不在视觉信息中;最后是过多周期没有看到球员处理。

3.9.1 球信息不准的处理

底层中球应该在视野中但是却不在视觉信息中，此时应该修正球的信息。修正主要由两种方式，一种是在球有可能跑出边界时将球移到边界，另外在球不可

能跑出视觉范围将球置信度设为0，且把球信息初始化，认为球完全不可信。计算球是否可跑出视觉范围过程如下：首先计算跑出去的最近的点，获得最少跑出去需要的距离$distant$,然后通过如下公式计算球跑出最少需要的时间：

$$cycle = log(1 - distant * (1 - ballDecay)/max\_speed)/log(ballDecay)$$

然后再与球未看到的周期数比较，如较大，说明球不可能跑出去，可能出现server 强制移动球等情况，将球置信度设为0,并初始化球位置和速度。如果小于，则将球移动到如上计算的最近的点。

3.9.2 球员信息不准确的处理

对于球员应该在视野中却不在视觉信息中的情况，修正与球类似，但是处理方式不同。对于球员的处理更复杂一些，首先是计算一系列球员在不同方向出去的相对最近的点，通过如下公式计算各点最少需要的周期。

$$cycle = distant/max\_speed;$$

因为球员可以保持最大速度移动，而球不可，故公式与球不同。如果所有点都不可出去，则忘记此球员（置信度设为0,位置和速度初始化），如果仅有一个，则将球员移动到此点，如果有多个，则根据对手阵形，球门位置等信息选择最可能的点。

3.9.3 多个周期没看到球员

当多个周期没有看到球员时，暂时是100个，就认为球员可能出现故障了，将其生存标志设为false。认为出现死球员的情况。

3.10 特殊模式处理部分

server 内部模拟比赛时根据情况不同设置不同的比赛状态，比如发角球，发门球等，在这种特殊的比赛模式下，很多信息都是确定的，比如球的位置和速度。特殊模式处理一般是在特殊模式开始时将确定的信息强制更新，但一般不能一直强制更新，否则会出bug。

3.11 动作相关信息更新部分

动作相关信息主要是指可踢，可铲，可catch。此部分根据 server 内部kick，tackle，catch模型更新各个球员是否可踢，可catch，铲球概率如何等等。

第 4 章　个人技术层设计

4.1　个人技术层简介

个人技术决策是决策系统的最低层，用以实现自主体的个人技能(如踢球、跑动等)。在每个周期，最终都是由这一层发出本周期所要执行的基本动作命令。这一层更多关注的平台本身的特点和高层的良好交互，被抽象成很多优化问题，适合使用 MDP 或 POMDP 求解。

在WE2009新球队的行为层设计上，大多是按照WE2008球队的思想，按照新的结构重新编写，也有针对以前动作重新设计的。其中最重要的是基于MDP模型重新设计了GoToPoint行为。

4.2　基于MDP的GoToPoint设计

GoToPoint 动作属于个人技术决策，提供了使球员快速跑到某点的功能，输入为球员的当前状态和要跑的点，输出为球员应该执行的原子动作，包括转身和加速。下面首先介绍这两个原子动作模型。

4.2.1　转身模型

球员可以用原子动作 turn 改变自己的身体方向，turn 命令格式为：turn (*moment*)，其中 $moment \in [-180°, 180°]$，turn 以后，身体方向按下式更新：

$$\begin{aligned} eff\_moment &= \frac{moment}{1.0+inertia\_moment \cdot speed} \\ body\_dir &= body\_dir + (1.0 + r) * eff\_moment \end{aligned} \tag{4-1}$$

其中，$speed$ 为球员速度的大小，$inertia\_moment$ 是一个常数，r 为 server 附加的随机噪音。这个 turn 模型，使球员在速度较大时转身比较困难，即一次转身的最大角度小于 $moment$，符合实际情况。

4.2.2 加速模型

加速模型提供了球员基本的加速能力，传统的 dash 命令格式为: dash (*power*)，其中$power \in [-100, 100]$，$power > 0$ 时沿身体方向加速，$power < 0$ 时沿身体方向的反方向加速。最新的 server 支持多向 dash，dash 命令的格式为：dash (*power dir*)，考虑到近阶段要兼容老的球队，目前 dash 方向只提供身体方向、身体方向的反方向和两个身体侧方向，所以 $dir \in \{-90°, 0°, 90°, 180°\}$。考虑到人沿不同方向加速时效率不一样，表现为沿身体方向加速最容易、身体反方向加速较困难、身体侧方向加速最困难，server 引入 dir_rate(*dir*) 来刻划这种区别，

$$\mathrm{dir\_rate}(dir) = \begin{cases} 1.0 & dir = 0° \\ 0.5 & dir = -180° \\ 0.25 & dir \in \{-90°, 90°\} \end{cases} \tag{4-2}$$

dash 产生的加速度公式为：

$$acc = power \times \mathrm{dir\_rate}(dir) \times \mathrm{dash\_power\_rate} \times effort \tag{4-3}$$

其中，$\mathrm{dash\_power\_rate}$ 对于球员而言是一个常数，$effort$ 跟球员的体力有关，体力充沛时（即体力数值上大于一个 server 给定的阈值），也可以视为常数。加速度方向沿 $body\_dir + dir$ 方向，其中 $body\_dir$ 为球员的身体方向。dash 以后，球员位置、速度的更新按下式进行：

$$\begin{aligned} \vec{u}_{t+1} &= \vec{v}_t + \vec{a}_t + \vec{r}_t \\ \vec{p}_{t+1} &= \vec{p}_t + \vec{u}_{t+1} \\ \vec{v}_{t+1} &= \vec{u}_{t+1} \cdot \mathrm{decay} \\ \vec{a}_{t+1} &= 0 \end{aligned} \tag{4-4}$$

其中 $\vec{a}_t$、$\vec{v}_t$、$\vec{p}_t$ 分别表示球员在 t 时刻的加速度、速度、位置，decay 是速度衰减因子，对球员而言是为常数，$\vec{r}_t$ 为 server 附加的随机噪音。

4.2.3 使用 MDP 建模求解

为了简化问题，首先通过平移和旋转，把一般 GoToPoint 问题转化成目标点为原点、球员初始位置在负 x 轴上的 GoToOrigin 问题。server 给出的运动模型满足马尔可夫性，很直接想到使用 MDP 建模 GoToOrigin 问题并求解，为此给出这个 MDP 问题的基本定义如下：

状态空间 使用一个五维向量 $(p_x, p_y, v_x, v_y, \alpha)$ 表示状态，其中 $(p_x, p_y) = \vec{p}$，$(v_x, v_y) = \vec{v}$，α 为身体角度。由于球员跑到某一点并不要求精确跑到该点，所以目标状态取为原点附近半径为球员可踢半径的一个圆。这里状态空间是连续的，下面会专门介绍。

动作空间 主体可以选择的动作为 turn(*angle*) 和 dash(*power, dir*)，$angle \in [-180, 180] \cap angle \neq 0°$ 按照 $2°$ 进行离散，$power \in [-100, 100] \wedge power \neq 0$ 按照 5 进行离散，$dir \in \{-90°, 0°, 90°, 180°\}$。

转移函数 这里为了简化问题，认为状态的转移是确定，严格按照不加噪音的 server 模型进行转移。

回报函数 在目标状态执行任何动作回报为 0，其他状态回报为 -1，折扣因子取 1.0，则一定策略下，某一状态长期回报的相反数就是从该状态到目标状态所要经历的周期数。

由于状态空间是连续的，不能枚举出所有状态，所以不能采用一般查表的方法表示值函数，考虑到人工神经网络具有较好的范化能力，这里使用一个反向传播网络表示值函数。网络的结构为 $6 \times 6 \times 6 \times 1$，六个输入分别为：$(f(p_x), g(p_y), h(v_x), h(v_y), \cos(\alpha), \sin(\alpha))$。其中 f, g, h 是线性函数，目的是把各自的输入在其定义域内规范化到 $[-1, 1]$ 内；身体方向 α 被表示成两个输入 $(\cos(\alpha), \sin(\alpha))$，是为了使相邻的角度能表示成相近的输入。网络的输出为对应输入状态的值函数。

因为无法遍历所有状态，不能有效的使用基于动态规划的值迭代算法来进行求解，而是采用了强化学习方法。

4.2.3.1 学习过程

学习时，通过一个内建的模拟器，让主体在模拟器环境里学习。学习的目的是保证距离目标点 45 范围内 GoToOrigin 策略最优，45 这个值取自实际比赛的经验值，因为实际比赛中很少有出现要规划一个大于 45 的 GoToOrigin 动作序列，对于大于 45 的问题可以采取分段的 GoToOrigin 来逼近最优策略。模拟器模拟的场地是一个长为 52，宽为 5 的矩形，注意到这个场地比实际比赛场地要小，这是考虑到实际的最优策略不可能会使主体超出这个矩形，凡是超出这个矩形的行动序列肯定不是最优策略，都将被舍弃，不会被用来更新值函数（网络）。

主体在环境里采用 ϵ-贪心算法进行动作选择，即以 ϵ 的概率随机选取一个动作，以 $1-\epsilon$ 按照 $\pi(s)$ 选取动作，这样做的目的是为了防止值函数陷入局部最优，而无法得到进一步修正，实现时 $\epsilon = 0.0025$。

每个学习周期内，首先把主体随机赋值一个随机状态，主体从随机状态开始进行动作选择，模拟器模拟出主体下一个状态，不断重复此过程，直到主体成功到达目标区域或失败（包括出界和形成环）。对于成功的样例，最后一个成功状态的值函数为 0，从后往前回溯值函数依次递减，如此得到一系列网络的训练样本，使用批处理的训练方式训练网络；对于失败的样例，则不做处理。

学习过程中发现，即使一次批处理训练网络时误差之和很小（小于 0.001），但实际测试仍然会有失败的情况，所以不好使用网络的误差小于某个阈值来判断网络是否已经收敛，而是使用了基于测试判断方法，即每 1000 个学习周期后，中断学习并随机测试 1000 次，如果这 1000 次测试中，都没有出现失败的情况，就认为网络已经收敛了，这个方法虽然也不能保证网络已经收敛了，但由于测试强度很大，所以通过测试的网络实际使用效果还是不错的。

4.2.3.2 实验结果

图 4.1，4.2，4.3 是固定 $v_x = 0$，$v_y = 0$，$\alpha = 0$ 的值函数视图，图 4.4, 4.5 是固定 $x = -30$，$y = 0$，$\alpha = 0$ 的值函数视图。

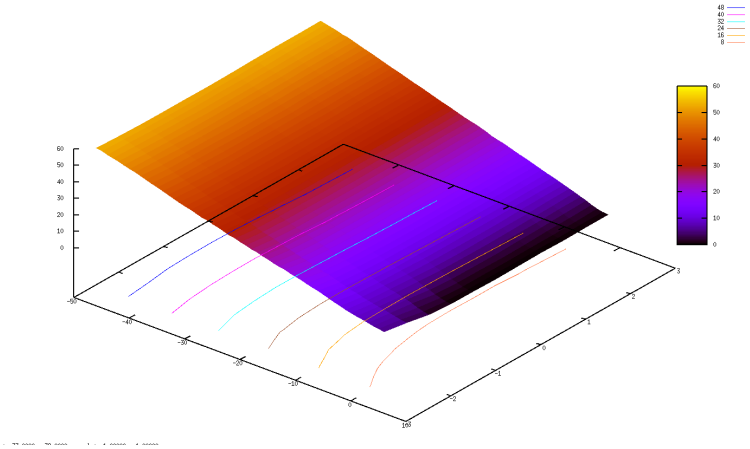


图 4.1　值函数 $V(x, y, 0, 0, 0)$ 视图

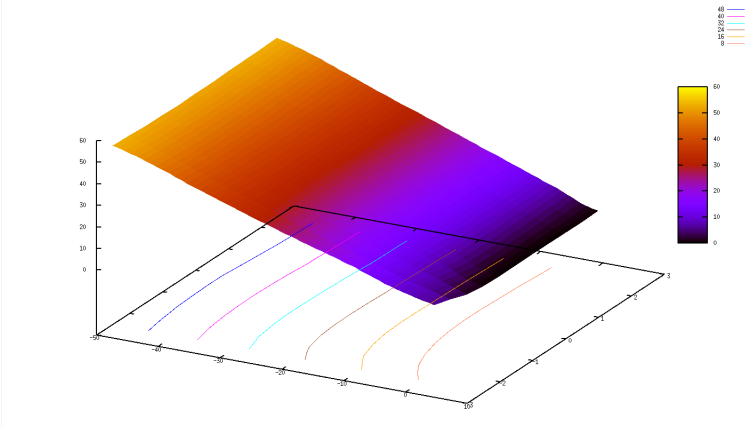


图 4.2　值函数 $V(x, y, 0, 0, 0)$ 视图

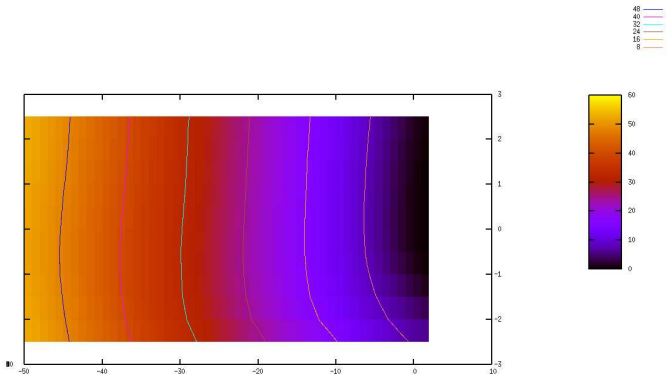


图 4.3　值函数 $V(x, y, 0, 0, 0)$ 视图

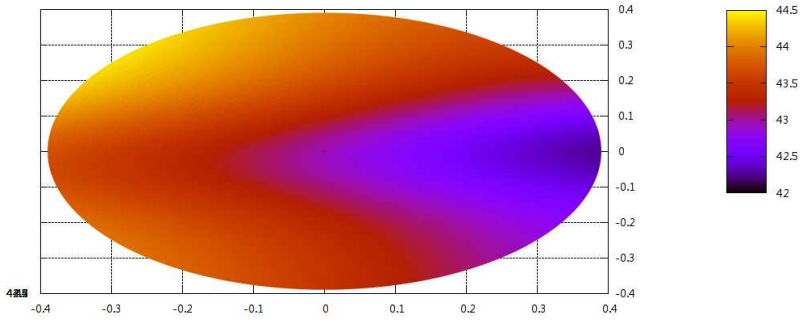


图 4.4　值函数 $V(-30, 0, v_x, v_y, 0)$ 视图

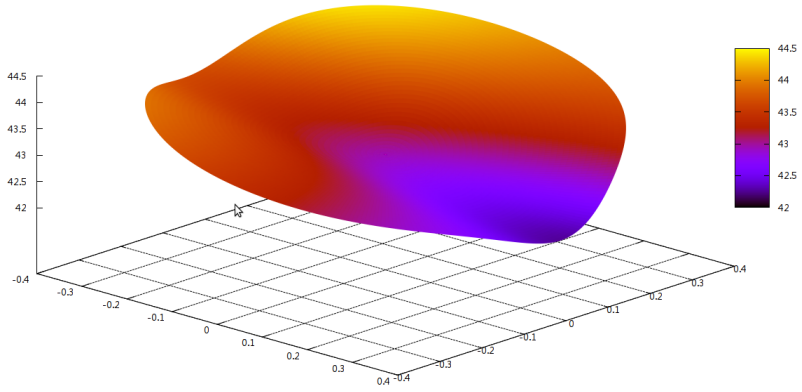


图 4.5　值函数 $V(-30, 0, v_x, v_y, 0)$ 视图

第 5 章　行为层设计

5.1　行为层简介

球员具有多种行为，包括传球、带球、截球、射门、盯防、封堵、跑位等，这些行为的具体决策位于行为层。每个行为负责产生自己行为的最佳动作，再由决策层仲裁选择行为。

这里主要介绍一下截球行为的设计。

5.2　截球行为的设计

截球行为属于多主体决策中的个体决策，在球员决策结构里面，位于战术决策层。截球行为的主要目的是判断当前周期是是否应该自己去截球，并且如果要去截球应该在未来哪个周期截到球最好。截球行为的核心是计算场上每个人的截球周期，这也是其他战术决策层行为最关心的信息状态，所以特别进行了研究。

5.2.1　求解截球周期

如图 5.1 即为一个典型的截球场景，球以一定的初速度向前运动，图中一连串灰色的点标记了球将要出现的位置，$\vec{p_b}^t$ 表示球在 t 周期后的位置，$t=0$ 时即为当前位置，求解截球周期就是要找出一些时间点 t 使得 $\mathrm{player\_time}(\vec{p_b}^t) \leq t$（这称为可截条件），其中 $\mathrm{player\_time}(\vec{p})$ 为球员从当前位置最快跑到点 $\vec{p}$ 所要花的时间。

简单的求解方法就是在一定范围内遍历 t，依次判断每个点是否满足可截条件，从而得到截球解的区间，这被称为模拟法，但由于 player_time 的计算比较耗时（因为要充分考虑转身、体力等影响因素，没有解析表示），所以模拟法求解会花费很多时间。实际使用时往往是与解析法相结合，解析法使用一个可解析表示的球员理想情况下（不考虑转身、体力等因素）的跑动时间函数 $\mathrm{player\_time}'$，满足 $\mathrm{player\_time}'(\vec{p}) \leq \mathrm{player\_time}(\vec{p})$，结合球的运动模型，可以求出一个扩大了的截球区间，再用模拟法缩减这个区间，从而节省了计算时间，但不影响解的质

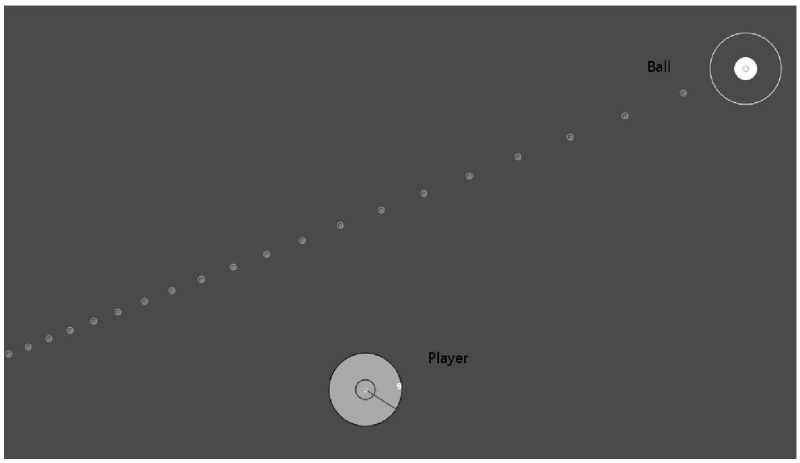


图 5.1　典型的截球场景

量。下面重点介绍解析法。

首先以球运动方向为 x 轴正方向，球的初始位置为原点，建立坐标系（如图 5.2），此坐标系中球员的初始位置为 (x_0, y_0)，球速大小为 v_0，球运动的衰减因子为 α，球员运动最大速度为 v_p，球员的可踢范围为 ka，球员没有被看到的周期差为 cd。

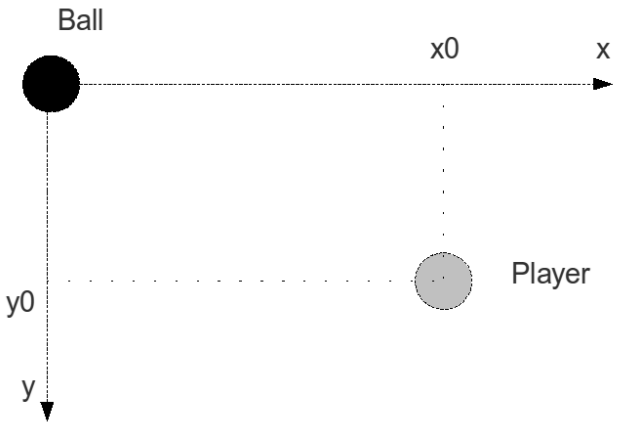
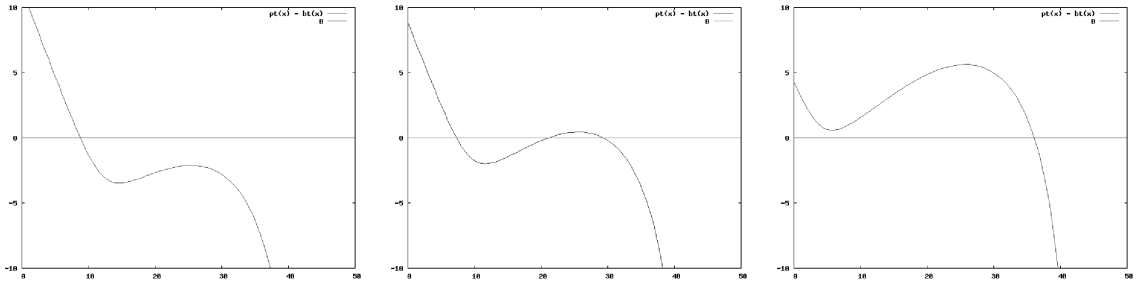


图 5.2　坐标转换以后的截球问题



(a) $x_0 = 4, y_0 = 3.6$ (b) $x_0 = 9, y_0 = 3.6$ (c) $x_0 = 12, y_0 = 3.6$

图 5.3 几组不同的 (x_0, y_0) 对应的 $f(x)$ 图像

根据球的运动模型，周期 t 时，球速大小为 $v_0\alpha^t$，球运动的距离为 $\frac{v_0(1-\alpha^t)}{1-\alpha}$，取 x 轴上一点 x，球运动到该点所花时间为：

$$bt(x) = \frac{\log(1 - \frac{x(1-\alpha)}{v_0})}{\log(\alpha)} \tag{5-1}$$

理想情况下球员跑到该点所花时间为：

$$pt(x) = \frac{s(x) - ka}{vp} - cd^{①} \tag{5-2}$$

其中，

$$s(x) = \sqrt{(x - x_0)^2 + {y_0}^2} \tag{5-3}$$

截球条件为：

$$pt(x) \leq bt(x) \tag{5-4}$$

定义函数 $f(x) = pt(x) - bt(x)$，则截球区间的边界点满足 $f(x) = 0$，可以通过牛顿迭代法求解这个方程，但要注意迭代初值的选取。下面首先通过实际的例子，研究一下解的形式。根据实际情况，取 $v_0 = 2.54, ka = 1.0, v_p = 1.0, cd = 0$，分别选取几组不同的 (x_0, y_0)，画出 $f(x)$ 图像如图 5.3。可见，解的形式为 $[x_1, \infty)$ 或 $[x_1, x_2], [x_3, \infty)$。

如果确定 $(x_0 = 4.0, y_0 = 3.6)$，而让 v_0 变化，得到图 5.4。

① 这里假设此球员在没有被看到的周期内一直加速截球，所以最后时间要减一个 cd。

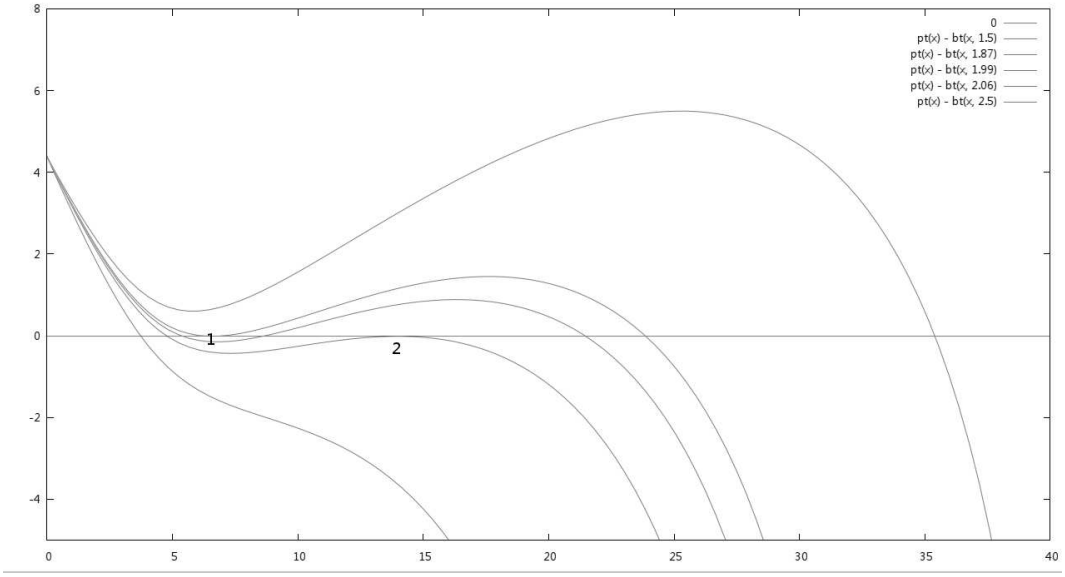


图 5.4　确定 $(x_0 = 4.0, y_0 = 3.6)$，而让 v_0 变化

观察发现，当 v 取某些特定值时，$f(x)$ 会与 x 轴相切，图中点 1 为内切点（x_1），点 2 为外切点（x_2），内切点对应的球速记为 v_1，外切点对应的球速记为 v_2。当 $v_0 \leq v_2$，球运动的早期就可截；当 $v_2 < v_0 < v_1$ 时，存在两个截球区间；当 $v_0 \geq v_1$ 时，球运动的后期才可截。事实上，通过牛顿迭代法求解 $f(x) = 0$ 时，初值的选取就跟 v_0, v_1, v_2 的大小关系有关：当 $v_0 \leq v_2$，可以选取初值为 x_0，求解得到区间的左边界；当 $v_2 < v_0 < v_1$ 时，分别选取初值为 $x_0, x_1 + \epsilon, x_\infty - \epsilon$，求解得到区间的边界；当 $v_0 \geq v_1$ 时，选取初值 $x_\infty - \epsilon$，求解得到区间左边界。其中 $\epsilon > 0$ 为一个小的实数，$x_\infty = \frac{v_0}{1-\alpha}$ 是球可以运动的最远距离。

所以现在问题的关键是求出内切点和外切点，切点满足方程 $f(x) = 0 \wedge f'(x) = 0$，联立消去 v_0 得到方程：

$$g(x) = 1 - \alpha^{pt(x)} \cdot (1 - x(x - x_0)\log(\alpha)/(s(x)v_p)) = 0 \tag{5-5}$$

容易通过牛顿迭代法求出 $g(x) = 0$ 的两个解，分别为内切点和外切点，从而可以进一步求解 $f(x) = 0$，得到截球问题的解析解。解析求得的区间大于实际区间，在区间边界使用模拟法缩减区间，从而可得实际截球区间。

第 6 章　决策层设计

6.1　决策层概要介绍

决策层主要针对行为层规划的动作评估，选择最好的动作执行。WE2009新的决策层分为三个部分：

普通行为决策 位于DecisionTree中，主要是dash，turn，kick等不可同时执行的动作选择。

视觉行为决策 位于VisualSystem中，主要是对可与dash等同时执行的turn_neck的决策。

通讯决策 位于CommunicateSystem中：主要是对球员之间的通讯做决策。

在决策层三类决策中，普通行为决策和通讯决策较简单，原因主要是在底层生成动作时已经包含了对各个动作的评估。WE2009新球队的另一个重要的创新之处是基于POMDP重新设计了视觉行为决策。

6.2　基于POMDP的视觉系统决策设计

视觉决策的目的是为了能较好的掌握场上物体（包括球和球员）最新的状态。

6.2.1　视觉模型

视觉相关的动作有 change_view 和 turn_neck，下面分别介绍这两个动作。

change_view change_view 命令用来改变视觉宽度，球员一共有三种视觉宽度：narraw，normal 和 wide，三种宽度对应的视觉延迟（就是上次收到视觉到收到新视觉的周期差）不一样，分别为 1，2 和 3。

turn_neck turn_neck 命令改变球员的脖子角度，便于观察不同的方向，球员脖子角度在 $[-90°, 90°]$ 范围内。

6.2.2 视觉信念状态

在视觉决策系统里面，场上的物体（球和球员）被抽象成视觉对象，视觉对象主要包含三部分数据：关注频率 $freq$，$freq$ 越小的物体越应该受到关注，关注频率由各个战术行为根据当前周期下的具体情况提出；视觉延迟 $cycle\_delay$，为这个物体没有被看见的周期数；活动范围，维护了这个物体在场上每点出现的概率，实现时，把场地离散成了一些方格，所以实际维护了在每个方格内出现的概率。视觉信念状态就定义成场上每个视觉对象活动范围的联合分布。

6.2.3 信念状态的更新

视觉决策系统的核心就是维护信念状态。信念状态的更新主要是每个视觉对象活动范围的更新，其更新原则如下：如果周期开始时看到了这个物体，那么这个物体就以概率 1 出现在一个确定的格子内；否则假设其活动范围按照随机游动模型进行扩散，事实上得到一个高斯分布。下面给出典型信念状态的例子。

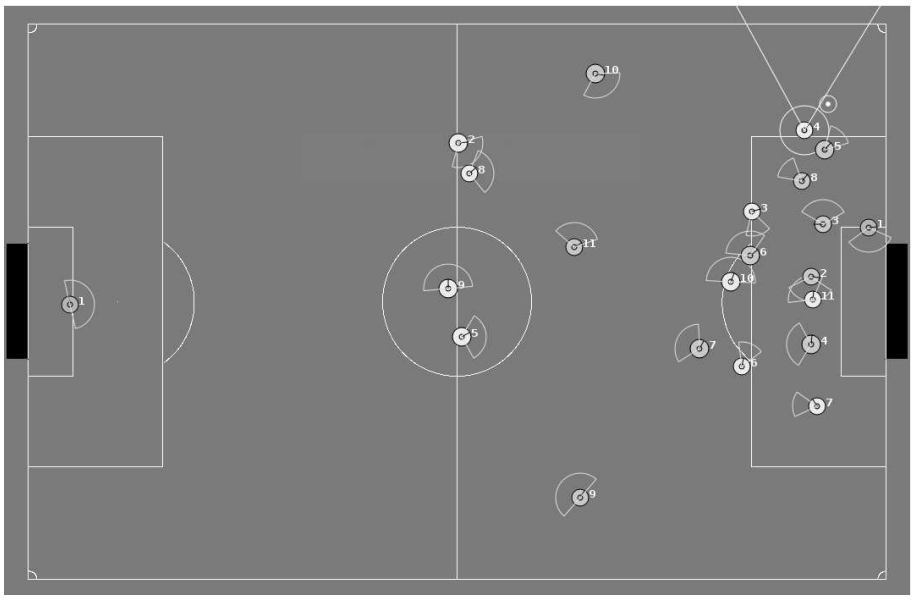


图 6.1　真实世界模型

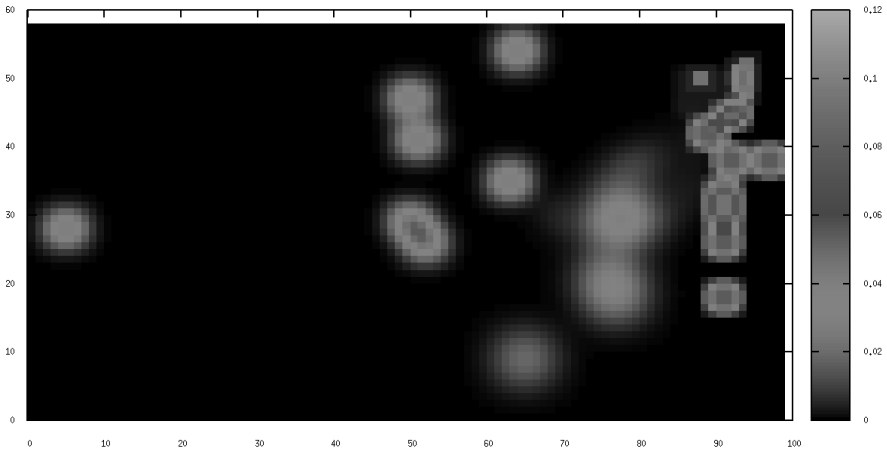


图 6.2　4号球员的信念状态

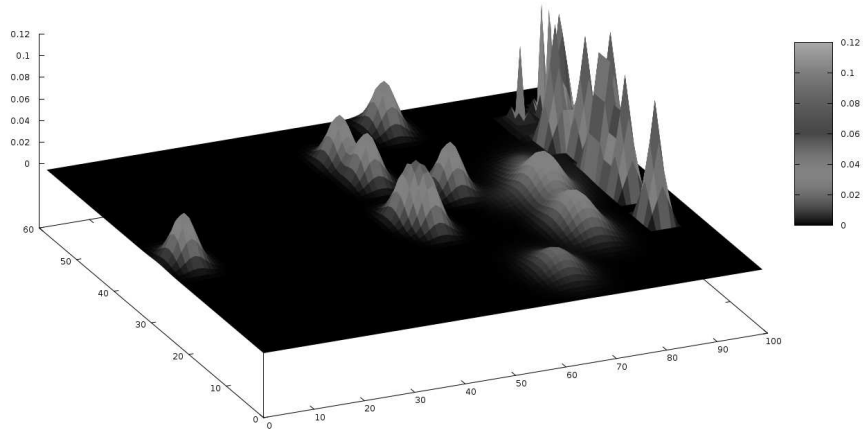


图 6.3　4号球员的信念状态（3D视图）

6.2.4　视觉决策的立即回报

视觉决策的立即回报是一个评价系统，首先定义每个视觉对象 o 完全被看到的回报为：

$$full\_reward(o) = 1 - \max(1 - (cycle\_delay(o)/freq(o)), 0) \qquad (6\text{-}1)$$

视觉动作被抽象成一个二元组：$(view\_width, neck\_dir)$，用 a 表示，则在视觉动作 a 下，视觉对象 o 的实际回报为：

$$reward(o,a) = full\_reward(o) \times \frac{\sum_{g\in\mathcal{G}} visible(g,a)\cdot appear(o,g)}{sight\_in\_delay(a)} \tag{6-2}$$

其中：$\mathcal{G}$ 表示场地上所有格子的集合；函数 $visible(g,a) = 1$，如果动作 a 下可以看到格子 g，否则等于 0；$appear(o,g)$ 为对象 o 在格子 g 出现的概率；$sight\_in\_delay(a)$ 为视觉动作 a 下，新视觉到来前没有视觉的周期数。

这样，信念状态 b 下，执行视觉动作 a 的立即回报即为：

$$R(b,a) = \sum_{o\in\mathcal{O}} reward(o,a) \tag{6-3}$$

其中，$\mathcal{O}$ 为场上所有应考虑的视觉对象的集合。

6.2.5 视觉决策的动作选择

由于视觉对象的关注频率信息只能通过外部调用获得，无法在视觉决策里面做出较好的预测，考虑视界大于 1 的视觉决策困难比较大，所以目前视觉决策系统考虑的视界为 1。这种情况下，$Q(b,a) = R(b,a)$，所以视觉决策的动作选择按下式进行：

$$\pi(b) = \arg\max_{a\in\mathcal{A}} R(b,a) \tag{6-4}$$

由于某些特殊情况下，战术层行为决策需要强制看某一物体，这时就不能只使用关注频率来突出这一物体的重要性了，为此通过强制把需要强制看的物体的得分设置成一个较高的值来解决这一问题，设置公式如下：

$$reward[o] \leftarrow important(o) * (1 + (cycle\_delay(o) + 1)/freq(o)) \tag{6-5}$$

其中，$important(o)$ 是为了反应不同物体的重要性，以便处理存在多个强制要看的物体而产生竞争的问题：

$$important(o) = \begin{cases} 50 & o\ is\ ball \\ 10 & otherwise \end{cases} \tag{6-6}$$

6.2.5.1 实验结果

编程实现时，场地被均匀分成了 100×60 个小方格，每个格子内维护了每个物体在该格出现的概率。如果对物体在某个格子内出现概率的下限不设限制，则一个长时间看不到的物体就会扩散成一个很大的范围，导致每次扩散时花费较多的时间，目前选择 0.005 作为格子内物体出现概率的下限，即对扩散后出现小于 0.005 情况的格子不进行扩散。这个概率下限是可以调整的，使用概率下限为 0.005 的版本的球队跟 为1.000 的版本的球队自动测试 100 场，结果见表 6.1。

Version	*Games*	*Goals*	*Points*	*AvgGoals*	*AvgPoints*	*Win*	*Draw*	*Lost*
0.005	100	338	199	3.38	1.99	60	19	21
1.000	100	231	82	2.31	0.82	21	19	60

表 6.1 自动比赛测试结果

事实上，概率下限为 1.0 的版本的球队没有维护完整的信念状态，只是简单地的直接使用每个物体上次看到的位置进行视觉决策，这跟以前球队老的视觉决策系统是等价的。测试结果标明，基于 POMDP 的视觉决策系统对球队整体性能的改进效果是很明显的。

第 7 章　WE2009新球队测试测试结果

针对WE2009新球队的测试工作分为两部分，一部分是底层信息精确度测试，一是行为层和决策层测试。

针对底层的测试，可以采用底层信息和真实值相差的统计结果比较。针对决策层的测试只能根各个强队打比赛，比较比赛结果。

7.1　底层信息测试

在写完底层后，针对整个底层的所有信息写了一份《WE2009底层测试报告》，在统计上基本上所有量的精确度达到了WE2008球队的水平，随后又针对底层作了相应的改进，使得在某些信息的更新上¿获得了很好的效果。由于篇幅限制，本节仅介绍的是一些改进取的明显效果的信息。

7.1.1　自身信息的测试

自身信息主要时自己的位置，脖子角度，和身体角度取得了比较好的效果。

7.1.1.1　自身位置

测试针对有视觉情况下测试的，主要针对位置与真实值的x坐标差距，y坐标差距，与真实值差距的模，三个量的平均值和方差比较。测试结果如下表：

对于平均值:

平均值	个数	x分量差	y分量差	模差
WE2008	13182	0.1184	0.1125	0.1815
WE2009	11629	0.0736	0.0753	0.1176

对于标准差:

标准差	个数	x分量差	y分量差	模差
WE2008	13182	0.1188	0.1029	0.1357
WE2009	11629	0.0770	0.0771	0.0971

由统计结果可见位置精确度有了很大的提高。

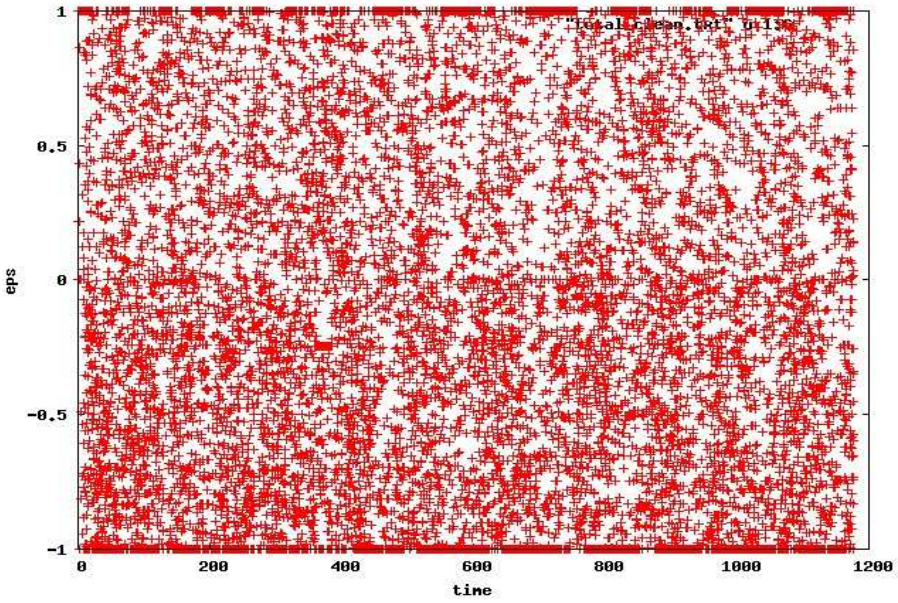


图 7.1　WE2008自身角度

7.1.1.2　脖子角度测试

第二章介绍 server 误差模型时已经指出，server 对脖子角度加误差时直接取整，脖子角度为1.0度时对应的区间为1.0度到1.9999度，如果用区间的平均值1.5度代替1.0度进行计算，误差会缩小一半。测试结果如图7.1 ,以及图7.2：

7.1.1.3　身体角度测试

对身体角度,有视觉时，改进效果很明显。之所以取得这样效果主要时脖子角度改进的效果。测试结果如图7.3 ,图7.4:

7.1.2　球信息测试

影响球信息改进主要还是使用最大误差范围，但是效果没有对自身位置改进的效果好，分析原因主要是因为球会因为别的球员施加动作而导致最大误差范围计算不准，这影响了改进效果。

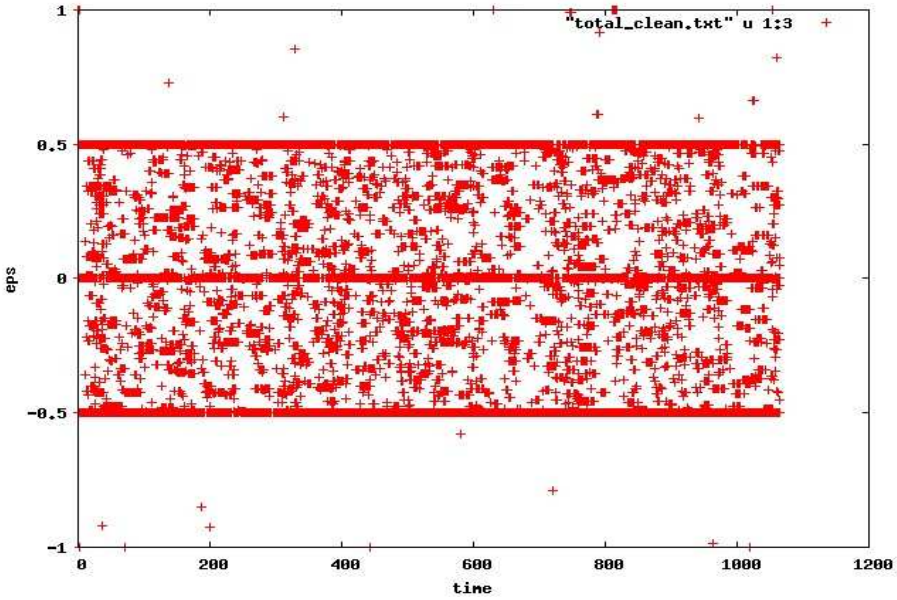


图 7.2　WE2009的自身角度

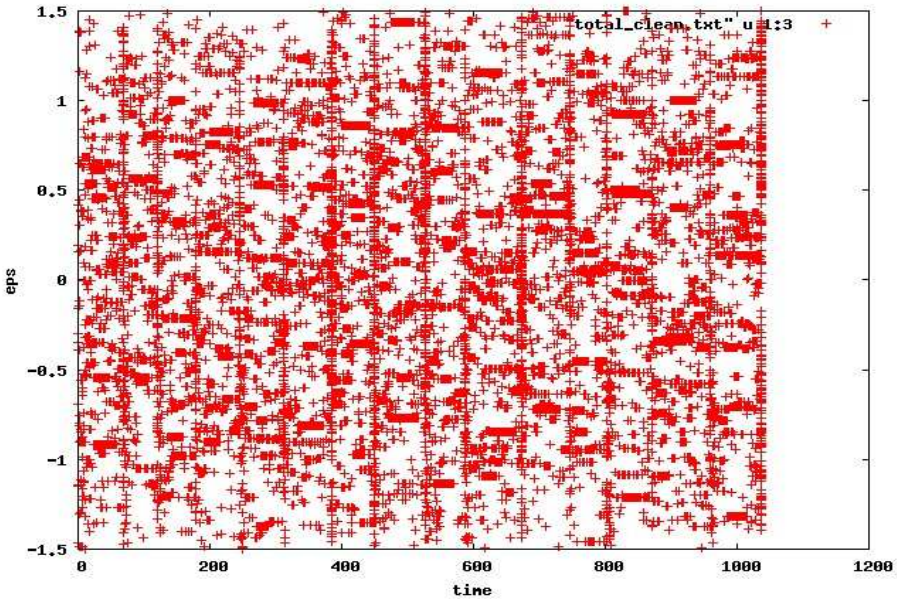


图 7.3　WE2008身体角度

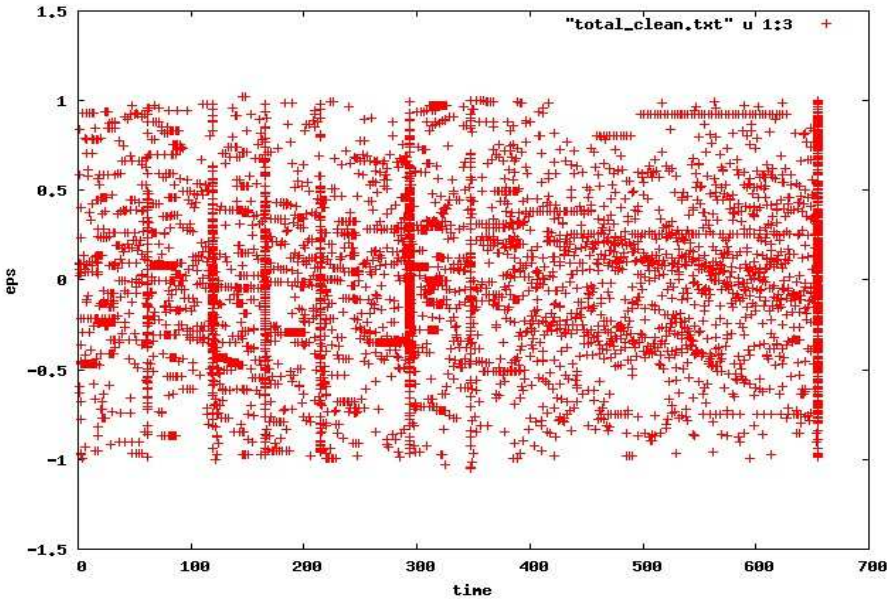


图 7.4　WE2009身体角度

7.1.2.1　球位置测试

改进主要针对有视觉时信息，当有视觉时.测试结果如下表：

对于平均值:

平均值	个数	x分量差	y分量差	模差
WE2008	7775	0.6033	0.4123	0.8121
WE2009	8223	0.5329	0.3933	0.7365

对于标准差:

标准差	个数	x分量差	y分量差	模差
WE2008	7775	0.6268	0.3569	0.6283
WE2009	8223	0.5658	0.3610	0.5888

可见，球位置只有非常小的改进。

7.1.2.2　球速测试

球速测试结果如下表。

对于平均值:

平均值	个数	x分量差	y分量差	模差
WE2008	7775	0.2582	0.1466	0.3286
WE2009	7508	0.1472	0.0631	0.1702

对于标准差:

标准差	个数	x分量差	y分量差	模差
WE2008	7775	0.4923	0.2597	0.5385
WE2009	7508	0.2866	0.1131	0.3026

可见球速改进非常明显，误差降低了接近一半。

7.2 行为和决策层测试

决策层测试主要是与各强队比赛，观察比赛成绩确定效果. 以下是分别与helios（世界杯第三）和desertealge（全国赛第一）的测试结果。

7.2.1 与helios测试结果

比赛测试结果如下：

球队	总场数	胜场数	负场数	平场数	平均进球数	平均失球数
WE2008	84	41	26	17	1.8	1.5
WE2009	100	83	8	9	3.7	1.3

可见与helios测试结果，胜率提高到了80%，进球数比以前平均每场提高2个。

7.2.2 与deserteagle测试结果

比赛测试结果如下：

球队	总场数	胜场数	负场数	平场数	平均进球数	平均失球数
WE2008	84	25	36	23	0.8	1.1
WE2009	100	61	23	16	1.8	0.9

可见与deserteagle比赛胜率提高到了60%,平均每场进球比以前提高了1个。

参考文献

[1] Nils J. Nilsson: *Artifficial Intelligence: A New Synthesis*,
China Machine Press, Beijing, 2000

[2] Tom M. Mitchell: *Machine Learning*,
China Machine Press, Beijing, 2003

[3] Richard S. Sutton and Andrew G. Barto: *Reinforcement Learning: An Introduction*,
MIT Press, Cambridge, MA, 1998

[4] Mance E. Harmon and Stephanie S. Harmon: *Reinforcement Learning: A Tutorial*,
Wright-Patterson AFB, OH, 45433

[5] Gerhard Weiss: *Multiagent System: A Modern Approach To Distributed Artificial Intelligence*,
The MIT Press Cambridge, Massachusetts London, England

[6] Mathijs de Weerdt, Adriaan ter Mors, and Cees Witteveen: *Multi-agent Planning: An introduction to planning and coordination*,
Dept. of Software Technology, Delft University of Technology

[7] Leslie Pack Kaelbling: *Planning and acting in partially observable stochastic domains*,
AJU, 1998

[8] 宋志伟: *基于逻辑马尔可夫决策过程的关系强化学习研究*,
中国科学技术大学博士学位论文

[9] 范长杰: *基于马尔可夫决策理论的规划问题的研究*,
中国科学技术大学博士学位论文

[10] 吴锋：*多主体系统中的对手建模和策略协作*,
中国科学技术大学学士学位论文

[11] 宋志伟，陈小平：*仿真机器人足球中的强化学习*,
机器人，25(7):761-766, S, 2003