第 1 章 Robocup 3D 程序总体框架

1.1 总体框架

下面来介绍一个 Agent 程序的总体框架：

在RoboCup中，如何表示一个球员是十分重要的。一般研究人员都把球员设计成一个具有智能的Agent，要求Agent能够根据环境做出适当的动作，同时还应该考虑到单个Agent能够和其他Agent之间进行配合和协作。要构造一支强大的RoboCup球队，Agent的结构、Agent的个体智能以及多Agent之间如何进行合适的协作、协调是关键。

在设计Agent的时候，首先考虑的就是把Agent设计成一个分层结构，在不同的层次处理不同的信息，这样就能够进行连续决策。在具体设计的时候，设计3层结构，也就是3个大的模块：通信层及信息解析、动作层和高层决策层，如图1-1所示。

1．通信层及消息解析：它是整个结构的最底层，直接负责与SoccerServer进行通讯，从Server得到视觉信息、身体感知信息、比赛状态等信息，并且对这些信息进行解析并更新世界模型，同时负责向服务器传送智能体做出的动作指令，从而控制仿真世界。

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

3．高层决策层：是智能水平最高的，负责根据比赛情况进行实时动作评价并做出决策，负责进行球队整体策略。阵型的确定、攻防的转换、球员的跑位和球员之间的配合。基于场上形势的动态策略站位，基于场上形势的动作选择，射门还是传球，破坏还是转移，带球突破还是回传等待时机等这些问题都在高层决策层解决。根据世界模型分析预测可能出现的情况，从而产生动作。智能体每 20个周期只能收到一个描述场上状况的消息。然而，最高限度是每个周期都可以发送一条信息即完成一个动作。

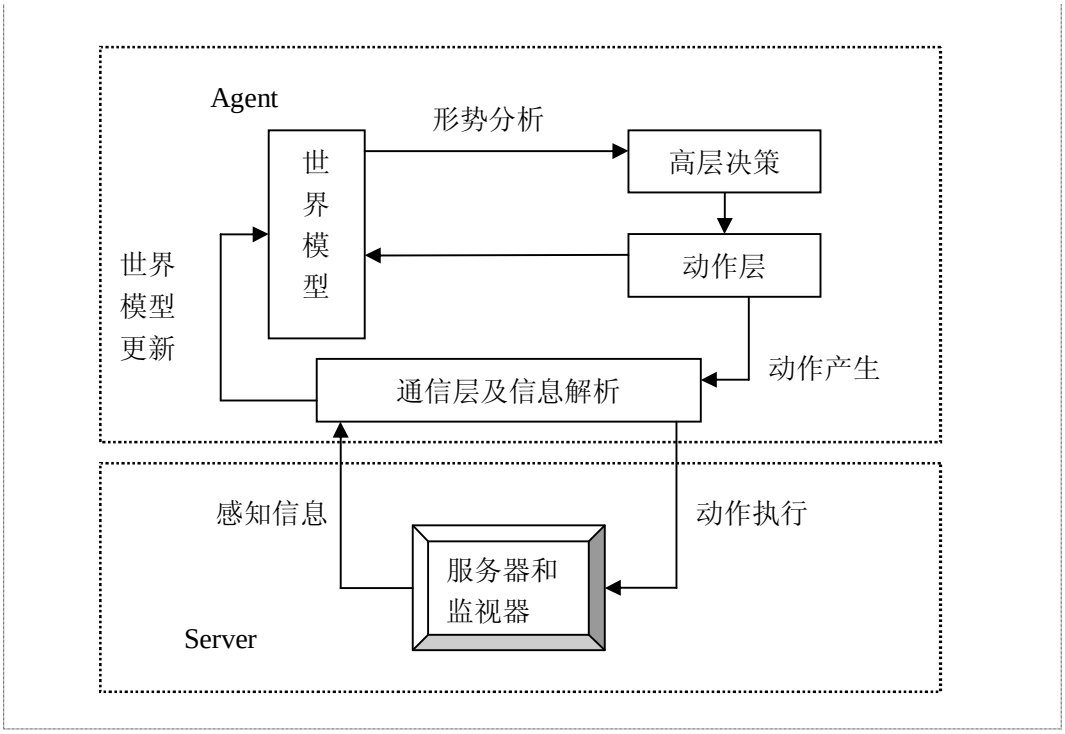


图 1-1 Agent 程序的总体框架

1.2 框架的各个层次

在已经分好的层次的基础上，对各层再细分成各功能模块。

1.底层通信层

通信层包括通信和信息解析两个方面。它是Agent 与SoccerServer 仿真环境的一个接口,主要处理的SoccerServer 与Agent 之间的信息交互。Agent 通过管道的方式与服务器通信,通过中间件SPADES 接收从Server 发送的信息,通过对信息的解析，转变成自己能够识别的字符串,然后送给中间建模层作为Agent 对当前环境的感知信息。

它主要是3 种方式来获得当前的环境信息: 视觉感知(Sense) 、比赛状态感知(GameState) 、智能体自身感知(Agent State) 。因此,在底层通信层,通信和信息解析两个模块共同完成对服务器信息的最终提取。

Communication 类主要完成3 个方面的功能:向管道写入决策后要执行的动作,包括时间通报(TimeNotify) ;从管道里读取各种服务器发送来的信息,包括空信息; 管道的重定向功能。其中,写和读管道是两个最关键的功能,分别以read 和write 两个管道下系统函数来完成。Agent的通信过程中的防错机制是建立在服务器规定的双方交换的信息格式的基础上的。

服务器规定,在每段信息的头部必须有4 字节(也就是一个长整型long 的长度) 作为信息长度校检位存放整条信息的长度。在通信过程中则利用

这一点将从管道中读取的信息的校检位中存放的数与整条信息长度相比较,如果不一致则认为信息发送有错,此信息被舍弃。同样的,在向服务器发送队列的之前,也要将信息的长度计算出来再加上4 个字节放在信息的首部,一道写入管道发送给服务器。服务器端在接收时也会做出相应的防错判断。在接收到正确的信息之后,程序会在内存开辟固定的地址,首地址取名为mbuffer ,并且会将首地址之后的4 字节地址返回。这样,其他模块就不需要考虑底层通信和防错机制的细节。这样做使得模块之间的层次变得清晰。各模块完成自己分内的事,不留‘后遗症’。

Msghandle 类实现的主要的功能是:将字符串解析成一个个有用的信息比如从中提取出坐标信息和球员号码之类的数据。鉴于其他球队的开发过程中,由于大量的使用了服务器自带的库函数来提取信息和建模而导致过分依赖服务器和与Monitor 有冲突情况,Agent完全抛开了服务器的自带的库函数,并且只用了少量的系统函数,从而提高了信息解析的效率和稳定性。

2.中层建模层

世界模型是Agent 用来描述周围环境信息的。球员Agent 应该能够知道球场上自己的速度和方向以及其他球员和球的速度和方向。显然,Agent 应该具有一个模块保存球场上各对象(22 个球员、球以及球场的一些标志的信息) ,所以,WorldModel 是必须的,通过它来保存这些对象的信息。世界模型中主要包含以下的信息:

(1) 环境的信息。环境信息包含Server 设置的各种参数,如球场长宽、仿真周期、球的最大速度、踢球的最大力量等等。

(2) 比赛信息。比赛信息主要包含世界模型对应的时间、比赛场地、队名、比赛模式以及双方的得失球等方面的信息。

(3) 场上对象的信息。对象信息主要包含球场上球员(自己和其他球员) 、球、地标(角旗及球门) 。Agent 得到的对象的信息一般是相对位置和相对方向,可以通过地标确定绝对量。WorldModel 类则是对球场、场上球员位置号码、球的位置、比赛状态、球员自身状态等世界模型进行建模。Geometry 负责建立笛卡尔基和极坐标点、线、圆等几何模型,Physics 主要负责诸如碰撞、加力、物体的速度等物理方面建模。这3个类的关系是十分紧密相互调用的。其他诸如Object 等类是作为球场数据的存储,将

各种数据按逻辑进行划分的功能。其目的是将数据统一封装在类中,便于更新和设置。

3．高层决策层

包括了动作的编辑及思考决策。分成了思考模块和动作模块。思考模块所完成的是根据战术要素、阵型要素、球员类型、对手模型、通讯模型等方面以及球场实时情况，敌我双方的态势而做出的判断，在这里仅仅编写了简单的决策。

整体的层次划分如图1-2所示：

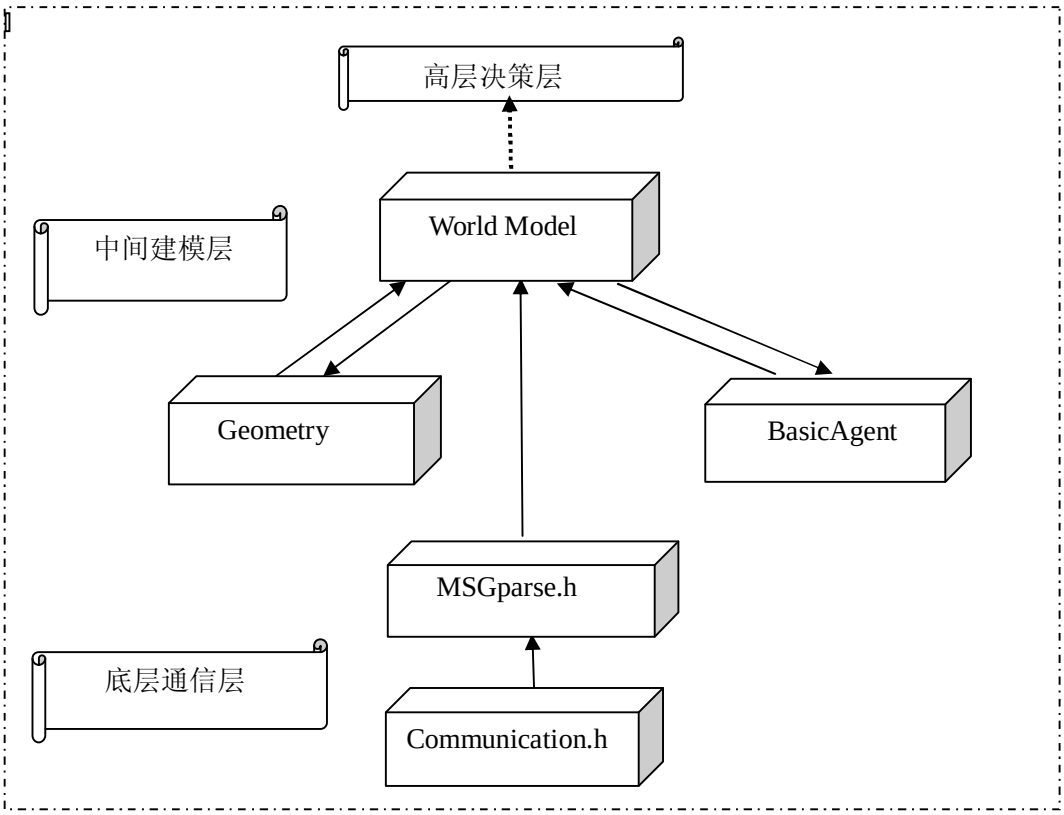


图 1-2 整体的层次划分

第 2 章 Robocup 3D 程序详细设计

2.1 世界模型

当一个良好的算法被设计后，同样需要完善的底层动作分解实现之，底层动作主要包括跑位、传球、截球、带球、断球、射门等个人技术。这些技术动作是最基本的，是实现高层的复杂策略所必不可少的，每个都是复合动作，实现都需要多个周期，由多个基本命令（dash，kick，turn）组成。

底层动作的实现可采用几何的方法对运动模型进行解析计算，也可采用神经网络的方法进行场景训练，使用监督学习的方式，通过对多层前馈神经元网络的训练，实现如带球，传球，射门等基本技术。

在底层的通信和信息的解析完毕后，首先要做的一件事就是更新 Agent 的世界模型，这个函数便是上边 **Pasre**，下面我们将详细介绍有关世界模型的一些问题。

世界模型（WorldModel）是Agent 用来描述周围环境信息的。Agent 应该能够知道球场上自己以及其他球员和球的速度和方向。显然，Agent 应该具有一个模块用于保存球场上各对象的信息，所以，World Model 是必须的。

参照UvA2D的底层，把World Model划分为4个部分： Basic、Update、Predict和HighLevel，分别负责信息的存储、更新、对象状态的预测和高层的算法，这里的高层并非是高层决策。

相互关系如图2-1所示：

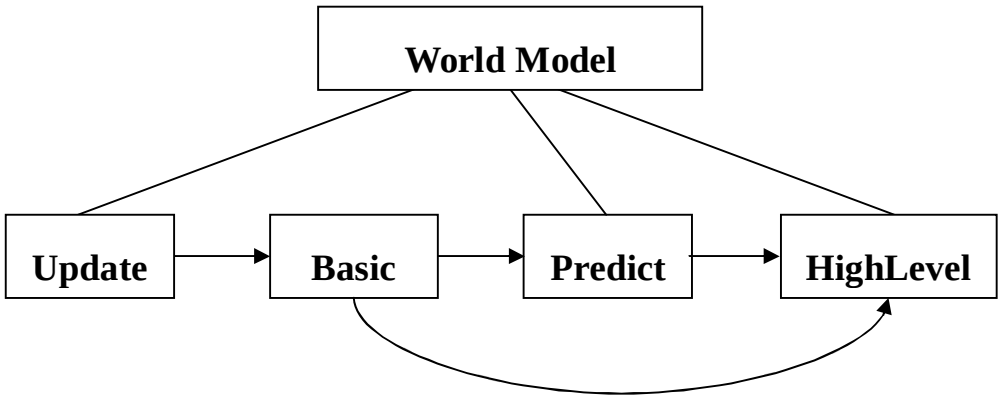


图 2-1 世界模型各个部分之间的关系

Basic: 主要用于实现信息的存储。这些信息将会被 World Model 中的 Predict、HighLevel 以及其他类如 BasicAgent 和 Decision 使用。

Update: 消息解析模块对服务器发送来的消息进行解析，然后更新存放在 Basic 中的信息。该部分负责在基本信息更新之后，计算球员及球的位置和速度，并把信息存储到相应的数据结构中。

Predict: 主要完成关于 Agent 自身、其他球员和球状态的预测。

HighLevel: 在前两者基础之上的一些基本函数，如判断Agent自身是否离球最近、是否在某个区域内，截球点的计算等。

2.2 Agent 的自定位算法

Agent每个周期能接收到1个视觉消息。时间通报只是一个带有时间周期的空消息，不含视觉信息，目的是让Agent能返回一个动作。然而由于Agent的视角和所处位置的影响，Agent虽然能看到球场上所有的球员、球场标志和球，但存在一定的误差，特别当Agent倒地后误差更大。所有的极坐标都是这些对象相对于Agent的极坐标。可以利用Agent与球场标志的相对极坐标计算出Agent的位置。进而可以算出其它可见对象的位置。使用适当的算法后可以获得Agent、球和其他球员的速度。可以说Agent自定位是世界模型维护的基础。

由于每个球场标志的绝对坐标都是已知的，可以分别利用这 8 个标志分成 4 组分别利用余弦定理计算出 Agent 的绝对坐标，再将计算出的 4 个绝对坐标累加求平均[3]。Agent 相对于球场标志的极坐标是存放在一个 1 行 8 列的数组中的。实际的算法中如何区分这 8 个相对极坐标是否可用？程序的消息解析模块在对消息进行解析并更新 World Model 时，写入相对极坐标的同时也记录下它的有效时间周期。因此可以通过判断相对极坐标对应的时间周期是否与当前的视觉消息的时间周期相等，来决定是否可以用于自定位算法。

2.3 agent 关节运动参数

当高层决策层接收到 agent 的状态，并通过决策决定 agent 的行动后，

将会以 agent 能够识别的命令参数发送给 agent。而这些命令就是 agent 的行动参数。而这些行动参数主要就是机器人的 14 个关节的运动情况。可以说一个足球机器人 3D 仿真软件的关键之处就在于行动参数的设定。比如机器人奔跑时如何设定各个关节的移动速度和角度才能使机器人能最快最平稳的向球跑去，机器人摔倒后怎么样调节各个关节才能最迅速的从地上爬起来。3D 机器人的行动参数设计得好可以使机器人的行动快速而准确，从而在比赛中能够很容易的控制住球从而赢得比赛。

而机器人的全身关节表如下表 2-1 所示

序号	名称	Effector / Perceptor	参数1	参数2
1	left shoulder	lae1_2	>0：整条胳膊向前	>0：整条胳膊向左
		laj1_2	<0：整条胳膊向后	<0：整条胳膊向右
2	left upper arm	lae3	>0：从上俯视逆时针旋转	无
		laj3	<0：从上俯视顺时针旋转	
3	left lower arm	lae4	>0：小臂向前摆动	无
		laj4	<0：小臂向后摆动	
4	right shoulder	rae1_2	>0：整条胳膊向后	>0：整条胳膊向右
		raj1_2	<0：整条胳膊向前	<0：整条胳膊向左
5	right upper arm	rae3	>0：从上俯视逆时针旋转	无
		raj3	<0：从上俯视顺时针旋转	

表 2-1 ROBOT 详细关节表

下面是设计的机器人在执行最简单的走向球的的动作的程序代码：

```
if(timer.GetTime() > 100 && timer.GetTime() <= 105)    //to left leg
{
    cout<<"step 2    "<<timer.GetTime ()<<endl;
    AdjustHJ(JID_LLEG_1, 0, 0.1,1);  AdjustHJ(JID_RLEG_1, 0, 0.1,1);
    AdjustHJ(JID_LLEG_2, -0, 0.15,1);AdjustHJ(JID_RLEG_2, 0, 0.1,1);
    AdjustHJ(JID_LLEG_3,35,0.15,1);
AdjustHJ(JID_RLEG_3,35,0.15,1);
    AdjustHJ(JID_LLEG_4,      -70,      0.15,1);AdjustHJ(JID_RLEG_4,
```

```
-70,0.15,1);
        AdjustHJ(JID_LLEG_5, 35, 0.15,1); AdjustHJ(JID_RLEG_5, 35,
0.15,1);
        AdjustHJ(JID_LLEG_6, -0, 0.15,1);    AdjustHJ(JID_RLEG_6,    0,
0.1,1);
    }
    if(timer.GetTime() > 105 && timer.GetTime() <=107)     //right leg up
    {   cout<<"step 3   "<<timer.GetTime ()<<endl;
        AdjustHJ(JID_LLEG_1,                                          0,
speed,1);AdjustHJ(JID_RLEG_1,0,speed,1);
        AdjustHJ(JID_LLEG_2,   1,   speed,1);AdjustHJ(JID_RLEG_2,   0,
speed,1);
        AdjustHJ(JID_LLEG_3,35,speed,1);AdjustHJ(JID_RLEG_3,        45,
speed,1);
      AdjustHJ(JID_LLEG_4,-70,speed,1);
AdjustHJ(JID_RLEG_4,-100,speed,1);
        AdjustHJ(JID_LLEG_5,                                          35,
speed,1);AdjustHJ(JID_RLEG_5,50,speed,1);
        AdjustHJ(JID_LLEG_6,                                          -0,
speed,1);AdjustHJ(JID_RLEG_6,0,speed,1);
    }
    if(timer.GetTime() >107 && timer.GetTime() <=109)   // right leg
    {cout<<"step 4   "<<timer.GetTime ()<<endl;
    AdjustHJ(JID_LLEG_1, 0, speed,1);AdjustHJ(JID_RLEG_1,0,speed,1);
    AdjustHJ(JID_LLEG_2, 0, speed,1);AdjustHJ(JID_RLEG_2,0,speed,1);
    AdjustHJ(JID_LLEG_3,35,speed,1);AdjustHJ(JID_RLEG_3,55,speed,1);
    AdjustHJ(JID_LLEG_4,-70,speed,1);AdjustHJ(JID_RLEG_4,-100,speed,1
);
    AdjustHJ(JID_LLEG_5, 36, speed,1);  AdjustHJ(JID_RLEG_5,        50,
speed,1);
    AdjustHJ(JID_LLEG_6,                0,                speed,1);
```

```
    AdjustHJ(JID_RLEG_6,0,speed,1);
    }
 if(timer.GetTime() >109&& timer.GetTime() <=111)              //left leg
    {cout<<"step 5    "<<timer.GetTime ()<<endl;
    AdjustHJ(JID_LLEG_1, 0, speed,1);    AdjustHJ(JID_RLEG_1,0        ,
speed,1);
    AdjustHJ(JID_LLEG_2, 0, speed,1);    AdjustHJ(JID_RLEG_2,         0,
speed,1);
    AdjustHJ(JID_LLEG_3, 35, speed,1);   AdjustHJ(JID_RLEG_3,        55,
speed,1);
    AdjustHJ(JID_LLEG_4, -70, speed,1); AdjustHJ(JID_RLEG_4,        -40,
speed,1);
    AdjustHJ(JID_LLEG_5, 38, speed,1);   AdjustHJ(JID_RLEG_5,        10,
speed,1);
    AdjustHJ(JID_LLEG_6,                 0,                  speed,1);
    AdjustHJ(JID_RLEG_6,0,speed,1);
    }
    if(timer.GetTime() >111&& timer.GetTime() <= 113)              // left
leg up
    {cout<<"step 6    "<<timer.GetTime ()<<endl;
    AdjustHJ(JID_LLEG_1,                 0,                  speed,1);
    AdjustHJ(JID_RLEG_1,0,speed,1);
    AdjustHJ(JID_LLEG_2,                 0,                  speed,1);
    AdjustHJ(JID_RLEG_2,-1,speed,1);
    AdjustHJ(JID_LLEG_3,45                                   ,speed,1);
    AdjustHJ(JID_RLEG_3,35,speed,1);
    AdjustHJ(JID_LLEG_4,  -100,  speed,1);AdjustHJ(JID_RLEG_4,  -70,
speed,1);
    AdjustHJ(JID_LLEG_5, 50, speed,1);   AdjustHJ(JID_RLEG_5,        35,
speed,1);
    AdjustHJ(JID_LLEG_6, 0, speed,1);    AdjustHJ(JID_RLEG_6,         0,
```

```
speed,1);
    }
    if(timer.GetTime() >113 && timer.GetTime() <= 115)          // left leg
    {cout<<"step 7    "<<timer.GetTime ()<<endl;
    AdjustHJ(JID_LLEG_1, 0, speed,1);    AdjustHJ(JID_RLEG_1,-1,
speed,1);
    AdjustHJ(JID_LLEG_2, 0, speed,1);    AdjustHJ(JID_RLEG_2,        0,
speed,1);
    AdjustHJ(JID_LLEG_3, 55, speed,1);   AdjustHJ(JID_RLEG_3,       35,
speed,1);
    AdjustHJ(JID_LLEG_4,   -100,   speed,1);AdjustHJ(JID_RLEG_4,   -70,
speed,1);
    AdjustHJ(JID_LLEG_5, 50, speed,1);   AdjustHJ(JID_RLEG_5,       36,
speed,1);
    AdjustHJ(JID_LLEG_6, 0, speed,1);    AdjustHJ(JID_RLEG_6,        0,
speed,1);
    }
    if(timer.GetTime() >115    && timer.GetTime() <= 117)           // left
leg
    {    cout<<"step 8    "<<timer.GetTime ()<<endl;
    AdjustHJ(JID_LLEG_1, 0, speed,1);    AdjustHJ(JID_RLEG_1,        0,
speed,1);
    AdjustHJ(JID_LLEG_2, -0, speed,1);   AdjustHJ(JID_RLEG_2,        0,
speed,1);
    AdjustHJ(JID_LLEG_3,55 ,speed,1);    AdjustHJ(JID_RLEG_3,       35,
speed,1);
    AdjustHJ(JID_LLEG_4, -40, speed,1);  AdjustHJ(JID_RLEG_4,       -70,
speed,1);
    AdjustHJ(JID_LLEG_5, 10, speed,1);   AdjustHJ(JID_RLEG_5,       38,
speed,1);
    AdjustHJ(JID_LLEG_6, -0, speed,1);   AdjustHJ(JID_RLEG_6,        0,
```

```
speed,1);
    }
    if(timer.GetTime() <=117)return false;
    timer.SetTimer(105);
    return true;
 }
```

2.4 agent 进攻路径规划

在足球机器人比赛中，机器人能否有效地抢点目标进攻很大程度上决定着比赛的成败，目前这方面的研究一般集中在以实现最短路径为主的规划上，在实时性要求不高的环境下是可以接受的，但由于比赛情形瞬息万变，基于最短路径的抢点进攻规划在实际应用上效果并不理想。我们提出一种基于曲线的路径规划方法，具有时间最短的特点，并通过仿真实验验证了该方法的可行性和有效性。

1 路径最短的抢点进攻规划分析

图2-6所示为以往研究中机器人在对方球门前抢点进攻路径规划，根据球的运动方向，可以预测出机器人和球的大概相遇范围，作为进攻抢点位置。抢点进攻路径走法是机器人先原地转一定角度，然后再走直线进攻，从而实现路径最短规划。但由于该走法经历了两次从零加速过程(转角度和直线)，若考虑到机器人的物理特性、启动时间、惯性等，从零开始加速需要较长的时间周期，所以该路径所需时间并不是最短的，不能很好地满足比赛的要求。

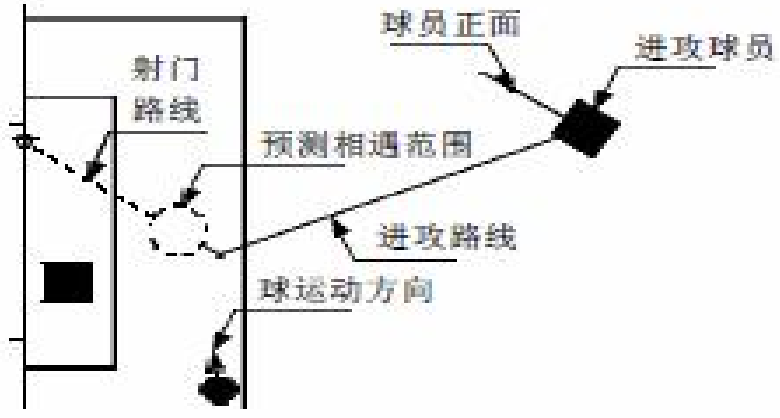


图 2-6 机器人的距离最短抢点进攻路径

2 理想的进攻路径

对图4-6的抢点进攻路径做出修改后，如图4-7所示。机器人移动过程中改变方向，抢点到与球相遇位置，其移动方向与球门成一夹角，避开对方守门员将球

踢入球门。另外，考虑到机器人要以尽可能快的速度移动，则所经路径应尽量短而平滑，同时在平滑过程中还需要满足两个条件：①路径起始端点必须与机器人0角度线相切；②路径终点必须与射门路径相切。图4-8为拟合后的机器人理想进攻路径，为一平滑曲线。

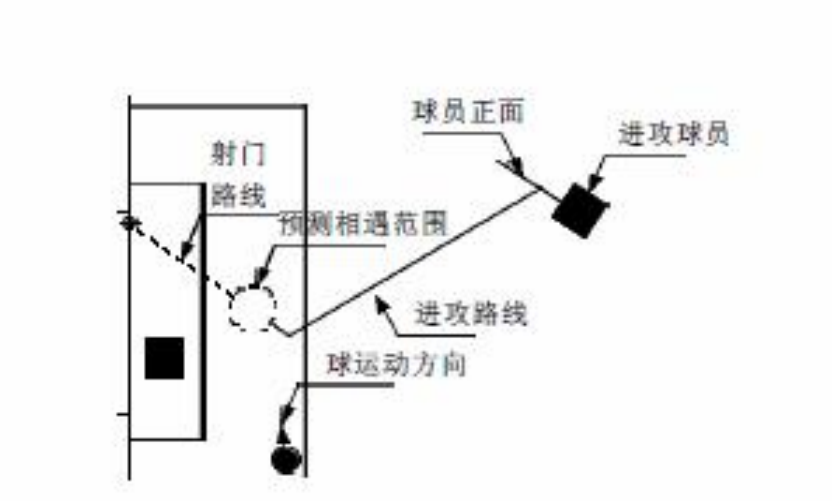


图 4-7 修改后的进攻路径

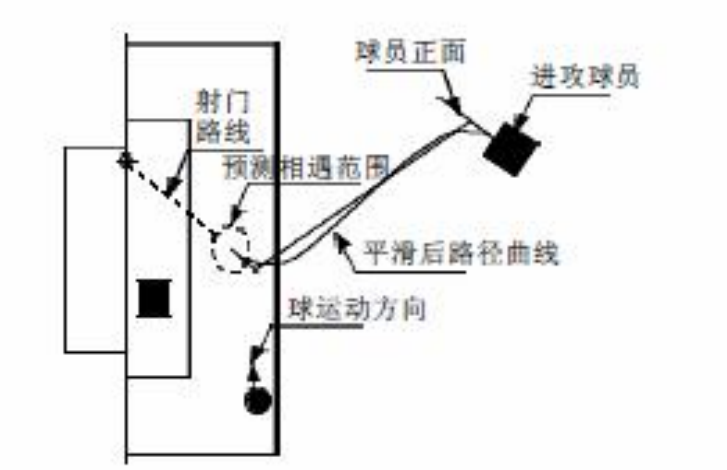


图 4-8 拟合后的机器人进攻路径

3 时间最短的抢点进攻路径

图4-9描述了用不同的控制点所产生的曲线图，其中交叉圆圈为控制点。在生成曲线时应遵循机器人时间最短的抢点进攻路径。根据经验值，在比赛中当射球点与机器人较远时D的理想取值范围是机器人边长的0.9~1.3倍。当射球点与机器人较近时D取值范围是机器人边长的0.5~0.8倍。把曲线三等分，取距离机器人的2/3 分点、机器人位置和机器人位置0度角线为切线作圆周，根据圆周设置机器人左右轮速度，如图4-10(a)所示。在比赛中，由于系统每1/n秒刷新一次(n 值

与比赛系统有关，例如在仿真平台下n = 60)，所以每个周期机器人只是走了虚线框内的一段，如图4-10(b)所示。这样机器人所走的路径就逼近曲线，从而实现图4-8中拟合后的机器人进攻路径。

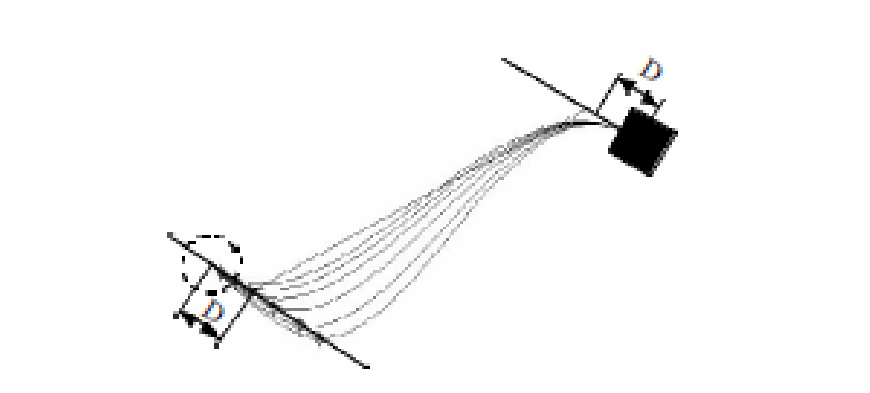


图 4-9 沿 Bezier 曲线的进攻路径

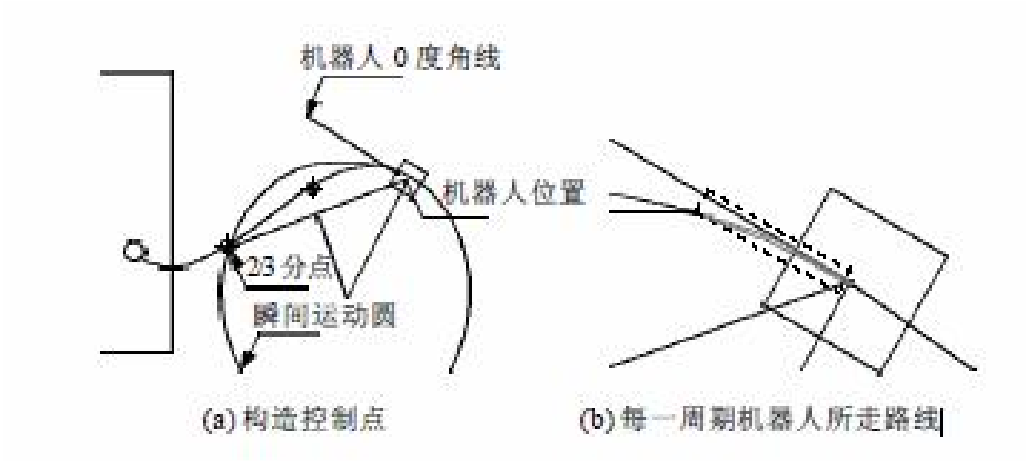


图 4-10 构造控制点以及机器人所走路线

第 3 章 Robocup 3D 程序具体实现

3.1 程序的具体运行流程

为完成程序的整体运行，所用的到所有类有 Communication 类，Msgparse 类，Worldmodel 类，Basicagent 类，Decision 类，Geometry 类和 Poweragent 类。

下面介绍各类的用途：

Communication 类：用于底层通信，与 Server 连接，负责每个周期接受服务器的消息，并向服务器发送 Agent 新的信息。

Msgparse 类：用于对接受到的消息进行解析，分清哪些消息分别属于什么信息，提供处理从长字符串中提取数字和处理极坐标等的函数入口，以便在需要时随时调用。

下图是整个程序的在各个类函数上的具体实现和运行流程：

图 3-1 （下接图 3-2）

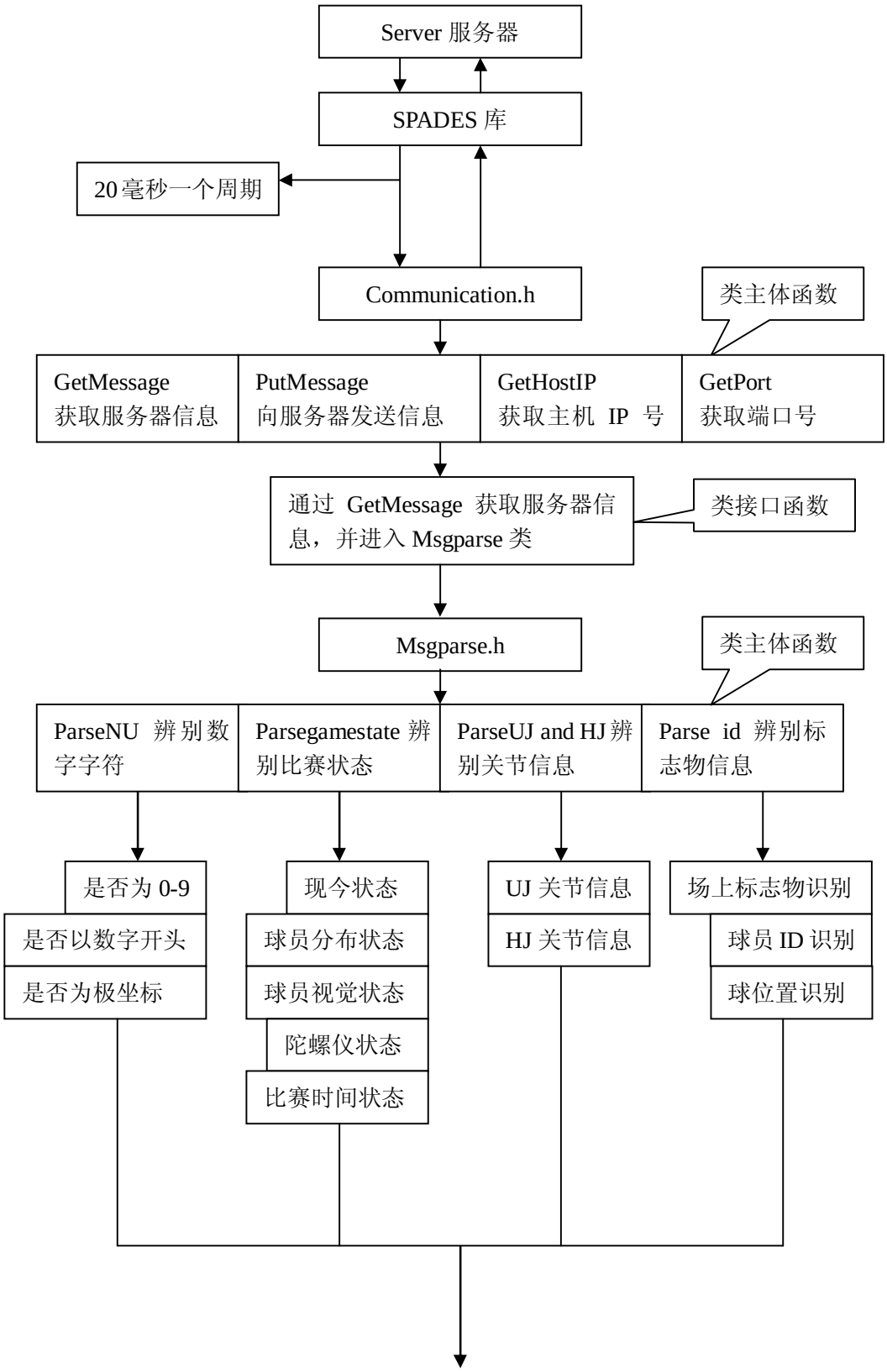
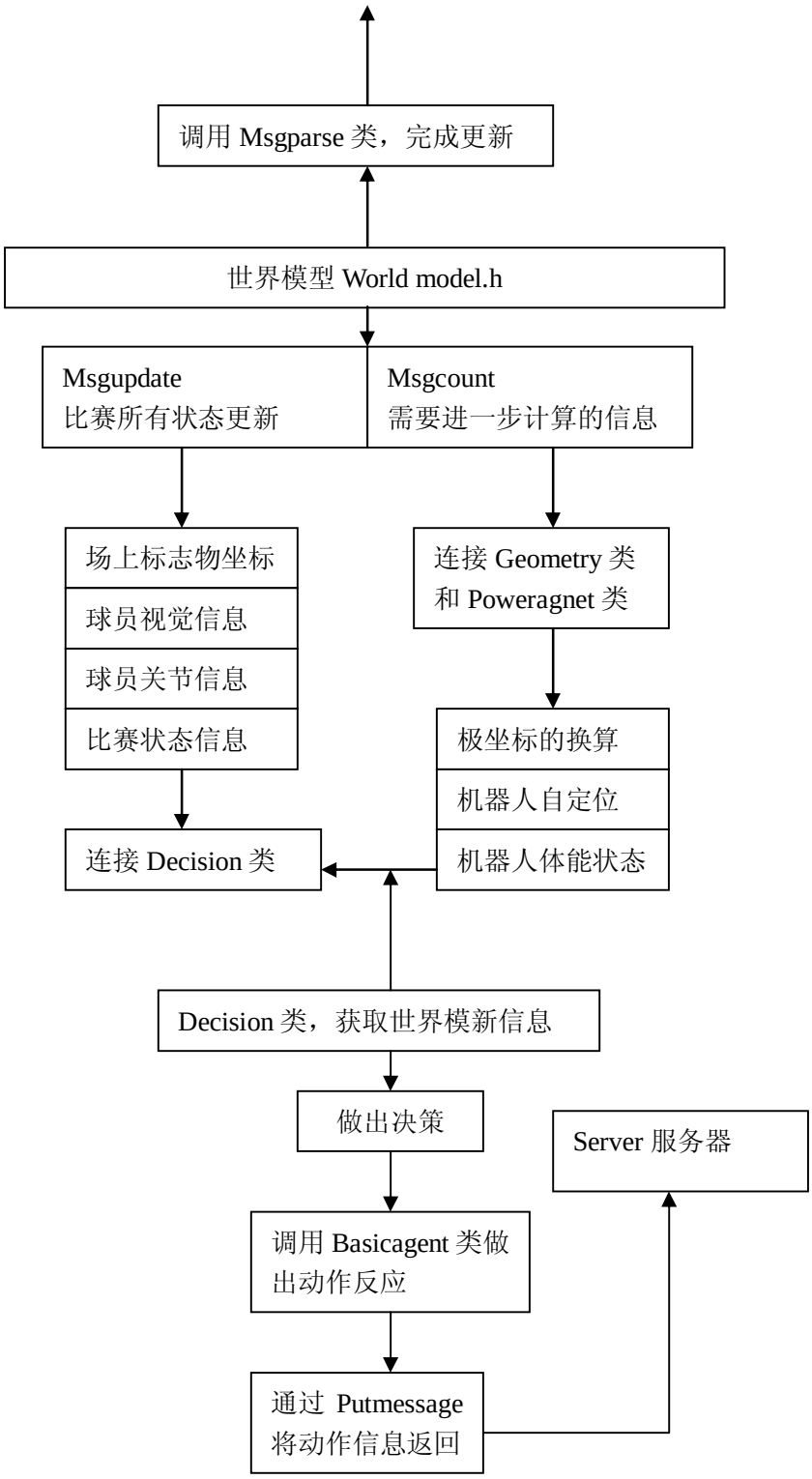


图 3-2 （上接图 3-1）



第 4 章 主要工作

根据组委会的要求，以及比赛的规定。我们做了合理的计划，逐步完成。主要的研究工作有：

1.熟悉了Robocup 3D的基本情况。

Robocup 3D即机器人足球世界杯。它是国际上一项为提高相关领域的教育和研究水平而举行的大型比赛和学术活动，是当前人工智能和机器人领域的研究热点之一。它经过了长时期的发展，规模也在不断地扩大，并从Robocup 2D发展到Robocup 3D。我将两者进行了详细的对比，我了解到2D仿真并不是真正的物理建模过程，它所维护的只是一些虚拟的运动模型。而3D 仿真试图模拟物体实际运动的状态，增加比赛的真实感，而不像2D一样，类似FIFA一样的电脑游戏。

2.了解Robocup 3D的开发环境及服务器的安装。

因为Robocup 3D官方比赛平台是在Linux—SUSE系统下运行的，所以我们对Linux—SUSE系统的基本知识以及基本操作进行了学习并掌握了服务器的安装配置问题。

3.对Robocup 3D的程序体系进行了详细设计。

首先我们对Robocup 3D程序的总体框架进行了设计。在具体设计的时候，设计3层结构，也就是3个大的模块：通信层及信息解析、动作层和高层决策层。其次，在已经分好的层次的基础上，对各层再细分成各功能模块：底层通信层、中层建模层以及高层决策层。最后，将Robocup 3D程序进行了详细的设计，包括世界模型、坐标系的研究，并设计出Agent的自定位算法以及关节运动参数。

4.将设计好的Robocup 3D程序运行仿真。

在做好上述工作的基础上，将所设计的程序运行仿真，得到了预期的效果。