WrightEagle 底层代码简介

柏爱俊

中国科学技术大学
计算机科学与技术学院
多智能体系统实验室

2011 年 8 月 11 日

主要内容

信息状态

- Client 实时地从 Server 处获得观察信息
- 观察信息过于原始，不便用于决策
- 经过处理和抽象后组织成“信息状态”（Information State）
- 决策直接使用“信息状态”

Parser & Observer

- Parser 通过解析 Server 发送过来的“字符串”，提取出感知信息 (Perception)
- Observer 维护从 Parser 获得的感知信息
- Parser 以一个单独的线程运行，Observer 是 Parser 线程和主线程的共享数据
- Parser.{h, cpp}, Observer.{h, cpp}

Parser & Observer

- Parser 通过解析 Server 发送过来的“字符串”，提取出感知信息 (Perception)
- Observer 维护从 Parser 获得的感知信息
- Parser 以一个单独的线程运行，Observer 是 Parser 线程和主线程的共享数据
- Parser.{h, cpp}, Observer.{h, cpp}

WorldState

- 把 Observer 提供的局部信息转换成全局信息
- 通过使用历史信息、Server 模型等方法提高精确度
- 维护了包括球、球员在内场上所有对象的“状态”(BallState & PlayerState)
- BallState.h, PlayerState.{h, cpp}, WorldState.{h, cpp}

WorldState

- 把 Observer 提供的局部信息转换成全局信息
- 通过使用历史信息、Server 模型等方法提高精确度
- 维护了包括球、球员在内场上所有对象的“状态”(BallState & PlayerState)
- BallState.h, PlayerState.{h, cpp}, WorldState.{h, cpp}

InfoState

- 将 WorldState 里的信息进一步计算和抽象得到 InfoState，包括：
  - 关于各种相对位置、角度等信息的 PositionInfo
  - 关于截球信息的 InterceptInfo
- InfoState.{h, cpp}, PositionInfo.{h, cpp}, InterceptInfo.{h, cpp}

InfoState

- 将 WorldState 里的信息进一步计算和抽象得到 InfoState，包括：
  - 关于各种相对位置、角度等信息的 PositionInfo
  - 关于截球信息的 InterceptInfo
- InfoState.{h, cpp}, PositionInfo.{h, cpp}, InterceptInfo.{h, cpp}

DecisionData

- 特指比 InfoState 更抽象的“信息状态”，包括：
  - 进攻和防守使用的 Strategy
  - 仅供防守使用的 Analyser
- 通过 WorldState、InfoState 和自身历史进行更新
- 可以维护了一些历史相关的信息
- DecisionData.{h, cpp}, Strategy.{h, cpp}, Analyser.{h, cpp}

DecisionData

- 特指比 InfoState 更抽象的“信息状态”，包括：
  - 进攻和防守使用的 Strategy
  - 仅供防守使用的 Analyser
- 通过 WorldState、InfoState 和自身历史进行更新
- 可以维护了一些历史相关的信息
- DecisionData.{h, cpp}, Strategy.{h, cpp}, Analyser.{h, cpp}

面向 Agent 的决策

- 所有行为的决策都是围绕 Agent 进行的
- 封装了所有“信息状态”，并提供了阵型系统、动作接口：
  - WorldState, InfoState, DecisionData
  - Formation
  - ActionEffector
- 具有派生能力，可以用于“反算”
- 全局存在唯一一个 Agent 代表球员自己
- Agent.{h, cpp}, Formation.{h, cpp}, ActionEffector.{h, cpp}, ...

面向 Agent 的决策

- 所有行为的决策都是围绕 Agent 进行的
- 封装了所有“信息状态”，并提供了阵型系统、动作接口：
  - WorldState, InfoState, DecisionData
  - Formation
  - ActionEffector
- 具有派生能力，可以用于“反算”
- 全局存在唯一一个 Agent 代表球员自己
- Agent.{h, cpp}, Formation.{h, cpp}, ActionEffector.{h, cpp}, ...

ActiveBehavior

- 完整的决策过程包括：规划阶段和执行阶段
- Behavior*Planner 负责规划，对应的 Behavior*Executer（如果有的话）负责执行
- 每个决策周期可以有多个行为进行规划，但最终只会选择一个行为去执行
- ActiveBehavior 是联系 Behavior*Planner 和 Behavior*Executer 的“桥梁”：
  - Behavior*Planner 规划的结果是 ActiveBehavior
  - ActiveBehavior 是可“执行”的，执行过程调用相应的 Behavior*Executer 进行
- ActiveBehavior 里面存储了行为执行时可能用到的所有数据，这些数据在行为规划时填充
- BehaviorBase.{h, cpp}

ActiveBehavior

- 完整的决策过程包括：规划阶段和执行阶段
- Behavior*Planner 负责规划，对应的 Behavior*Executer（如果有的话）负责执行
- 每个决策周期可以有多个行为进行规划，但最终只会选择一个行为去执行
- ActiveBehavior 是联系 Behavior*Planner 和 Behavior*Executer 的“桥梁”：
  - Behavior*Planner 规划的结果是 ActiveBehavior
  - ActiveBehavior 是可“执行”的，执行过程调用相应的 Behavior*Executer 进行
- ActiveBehavior 里面存储了行为执行时可能用到的所有数据，这些数据在行为规划时填充
- BehaviorBase.{h, cpp}

DecisionTree

- 行为决策的最上层
- 调用多个 Behavior*Planner 进行
- 选择出最优的 ActiveBehavior
- 执行最优 ActiveBehavior
- 设置历史 ActiveBehavior 信息，供行为保持用
- DecisionTree.{h, cpp}

DecisionTree

- 行为决策的最上层
- 调用多个 Behavior*Planner 进行
- 选择出最优的 ActiveBehavior
- 执行最优 ActiveBehavior
- 设置历史 ActiveBehavior 信息，供行为保持用
- DecisionTree.{h, cpp}

目录

- conf/ — player.conf、server.conf 等配置文件
- data/ — 离线计算的数据文件
- formations/ — 阵型文件
- src/ — C++ 源码
- Logfiles/ — 所有 Log 存在此处
- Debug/ — Debug 版本 Makefile
- Release/ — Release 版本 Makefile

文件

- `dbg`，`dd` — 动态调试工具
- `genlog`，`showlog` — SightLog 相关工具
- `memcheck` — 检查内存错误工具
- `initrc` — 以上工具共用脚本
- `dynamicdebug.txt` — 动态调试用
- `start.sh` — 球队上场脚本
- `Makefile` — Makefile

文件 (cont.)

- Types.{h, cpp} — 基本数据类型、宏定义
- Geometry.{h, cpp} — 几何相关计算
- Utilities.{h, cpp} — 功能函数、数据结构
- Dasher.{h, cpp} — 行动相关
- Kicker.{h, cpp} — 踢球相关
- Tackler.{h, cpp} — 铲球相关
- Behavior*.{h, cpp} — 各种行为规划、执行
- CommunicationSystem.{h, cpp} — 通信决策子系统
- VisualSystem.{h, cpp} — 视觉决策子系统
- ... — ...

Player 运行流程

- Client::RunNormal(···)
  - Client::SendOptionToServer(···)
  - Client::MainLoop(···)
    - Observer::WaitForNewInfo(···)
    - Player::Run(···)
    - Observer::SetCommandSend(···)
- Player::Run(···)
  - WorldModel::Update(···)
  - DecisionTree::Decision(···)
  - VisualSystem::Decision(···)
  - CommunicateSystem::Decision(···)

Player 运行流程

- Client::RunNormal(···)
  - Client::SendOptionToServer(···)
  - Client::MainLoop(···)
    - Observer::WaitForNewInfo(···)
    - Player::Run(···)
    - Observer::SetCommandSend(···)
- Player::Run(···)
  - WorldModel::Update(···)
  - DecisionTree::Decision(···)
  - VisualSystem::Decision(···)
  - CommunicateSystem::Decision(···)

Player 决策流程

- DecisionTree::Decision(···)
  - DecisionTree::Search(···)
    - Bheavior∗Planner::Plan(···)
    - Bheavior∗Planner::Plan(···)
    - ···
  - ActiveBehavior::Execute(···)
    - Behavior∗Executer::Execute(···)
- Bheavior∗Planner::Plan(···)
  - Bheavior∗Planner::Plan(···)
  - Bheavior∗Planner::Plan(···)
  - ···

Player 决策流程

- DecisionTree::Decision(···)
  - DecisionTree::Search(···)
    - Bheavior∗Planner::Plan(···)
    - Bheavior∗Planner::Plan(···)
    - ···
  - ActiveBehavior::Execute(···)
    - Behavior∗Executer::Execute(···)
- Bheavior∗Planner::Plan(···)
  - Bheavior∗Planner::Plan(···)
  - Bheavior∗Planner::Plan(···)
  - ···

PlayerParam & ServerParam

- 继承于 ParamEngine，维护所有“参数”信息
- 通过 Server 发来的信息、配置文件和命令行参数来更新
- ParamEngine.{h, cpp}, PlayerParam.{h, cpp}, ServerParam.{h, cpp}

PlayerParam & ServerParam

- 继承于 ParamEngine，维护所有“参数”信息
- 通过 Server 发来的信息、配置文件和命令行参数来更新
- ParamEngine.{h, cpp}, PlayerParam.{h, cpp}, ServerParam.{h, cpp}

Dasher

- GetBall(···) — 在指定周期或以“最快”方式截球
- GoToPoint(···) — 按指定方式“最快”跑动到某一目标点
- CycleNeedToPoint(···) — 计算跑动某一点所需的整数周期
- RealCycleNeedToPoint(···) — 计算跑动某一点所需的实数周期
- Dasher.{h, cpp}

Dasher

- GetBall(···) — 在指定周期或以“最快”方式截球
- GoToPoint(···) — 按指定方式“最快”跑动到某一目标点
- CycleNeedToPoint(···) — 计算跑动某一点所需的整数周期
- RealCycleNeedToPoint(···) — 计算跑动某一点所需的实数周期
- Dasher.{h, cpp}

Kicker

- KickBall(···) — 按指定方式踢球
- GetMaxSpeed(···) — 得到指定方向出球能球得到的最大速率
- GetStopBallAction(···) — 得到停球动作
- GetAccelerateBallAction(···) — 将球加速到指定的目标速度
- GetKickBallToAngleAction(···) — 将球踢向指定方向
- Kicker.{h, cpp}

Kicker

- KickBall(···) — 按指定方式踢球
- GetMaxSpeed(···) — 得到指定方向出球能球得到的最大速率
- GetStopBallAction(···) — 得到停球动作
- GetAccelerateBallAction(···) — 将球加速到指定的目标速度
- GetKickBallToAngleAction(···) — 将球踢向指定方向
- Kicker.{h, cpp}

Tackler

- TackleStopBall(···) — 铲停球
- CanTackleStopBall(···) — 是否铲停球
- TackleToDir(···) — 将球铲到指定方向
- CanTackleToDir(···) — 是否可以将球铲到指定方向
- GetBallVelAfterTackleToDir(···) — 得到将球铲到指定方向后的出球速度
- Tackler.{h, cpp}

Tackler

- TackleStopBall(···) — 铲停球
- CanTackleStopBall(···) — 是否铲停球
- TackleToDir(···) — 将球铲到指定方向
- CanTackleToDir(···) — 是否可以将球铲到指定方向
- GetBallVelAfterTackleToDir(···) — 得到将球铲到指定方向后的出球速度
- Tackler.{h, cpp}

VisualSystem

- RaiseBall(···) — 按指定权值注意球
- RaisePlayer(···) — 按指定权值注意球员
- SetForceSeeBall(···) — 设置强制看球 (要能看到才会去看)
- SetForceSeePlayer(···) — 设置强制看某人
- SetCritical(···) — 设置是否强制使用窄视角
- ForbidDecision(···) — 禁止视觉决策
- SetCanTurn(···) — 设置视觉决策时是否考虑转身动作
- ChangeViewWidth(···) — 高层决定改变视角
- VisualSystem.{h, cpp}

VisualSystem

- RaiseBall(···) — 按指定权值注意球
- RaisePlayer(···) — 按指定权值注意球员
- SetForceSeeBall(···) — 设置强制看球 (要能看到才会去看)
- SetForceSeePlayer(···) — 设置强制看某人
- SetCritical(···) — 设置是否强制使用窄视角
- ForbidDecision(···) — 禁止视觉决策
- SetCanTurn(···) — 设置视觉决策时是否考虑转身动作
- ChangeViewWidth(···) — 高层决定改变视角
- VisualSystem.{h, cpp}

CommunicateSystem

- SendBallStatus(···) — 广播球状态信息
- SendTeammateStatus(···) — 广播队友状态信息
- SendOpponentStatus(···) — 广播对手状态信息
- ParseReceivedTeammateMsg(···) — 解析听到的队友“喊话”
- CommunicateSystem.{h, cpp}

CommunicateSystem

- SendBallStatus(···) — 广播球状态信息
- SendTeammateStatus(···) — 广播队友状态信息
- SendOpponentStatus(···) — 广播对手状态信息
- ParseReceivedTeammateMsg(···) — 解析听到的队友“喊话”
- CommunicateSystem.{h, cpp}

DynamicDebug

- 正常比赛时，Client 是实时运行的，无法直接使用 gdb 调试
- 让 Client 在运行时记录所有从 Server 处收到的“字符串”到文件 — MsgLog
- 通过读取 MsgLog，模拟从 Server 收到“字符串”，让 Client 按“步进”方式运行
- 从而可以使用 gdb 进行调试，这就是动态调试 (Dynamic Debug)
- sudo apt-get install gdb cgdb

DynamicDebug

- 正常比赛时，Client 是实时运行的，无法直接使用 gdb 调试
- 让 Client 在运行时记录所有从 Server 处收到的“字符串”到文件 — MsgLog
- 通过读取 MsgLog，模拟从 Server 收到“字符串”，让 Client 按“步进”方式运行
- 从而可以使用 gdb 进行调试，这就是动态调试 (Dynamic Debug)
- sudo apt-get install gdb cgdb

DynamicDebug 操作流程

1. 修改 `initrc` 里面的 BINARY 名称
2. 修改 `conf/player.conf` 里面的 `team_name` 名称
3. 打开 `save_server_message` 选项
4. 跑比赛（MsgLog 记录在 `Logfiles/`目录内）
5. `./dd` 球员号码，按照“步进”方式运行 Client
6. `./dbg`，使用 gdb attach 到 Client 进程上，进行调试

Logger

记录 Client 运行期间的 Log

- TextLogger — 以文本形式记录 Log
- SightLogger — 以 rcg 格式记录 Log
- `./genlog` 球员号码 — 根据 MsgLog 运行 Client 记录 Log
- `./showlog` — 使用 rcsslogplayer 查看 SightLog

Bazaar

分布式版本管理软件，常用命令：

- bzr init
- bzr add FILE...
- bzr commit -m "COMMENT"
- bzr log
- bzr revert -r REVISION_NO
- bzr branch FROM_LOCATION TO_LOCATION
- bzr merge LOCATION
- sudo apt-get install bzr

Bazaar

分布式版本管理软件，常用命令：

- bzr init
- bzr add FILE...
- bzr commit -m "COMMENT"
- bzr log
- bzr revert -r REVISION_NO
- bzr branch FROM_LOCATION TO_LOCATION
- bzr merge LOCATION
- sudo apt-get install bzr

参考资料

- WrightEagle 2D, *WrightEagleBASE* 底层代码介绍, http://ai.ustc.edu.cn/en/robocup/2D/materials/10/Introduction-to-WrightEagleBASE.pdf
- WrightEagle 2D, *WrightEagleBASE-3.0.0*, http://ai.ustc.edu.cn/en/robocup/2D/releases/WrightEagleBASE-3.0.0.tar.gz