FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

FC Portugal: Projecto e Implementação de uma Equipa de Futebol Robótico 3D

André Alçada Guimarães
José António Silva
Rui Alberto Felber Hilário
LEEC 2006

U.PORTO

LEEC Licenciatura em Engenharia Electrotécnica e de Computadores

Futebol Robótico 3D

André Alçada Guimarães
José António Oliveira Silva
Rui Alberto Felber Hilário

Universidade do Porto
Faculdade de Engenharia
FEUP

Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Electrotécnica e de Computadores
Rua Roberto Frias, s/n, 4200-465 Porto, Portugal

Julho de 2006

Financiamento

União Europeia


Governo da República Portuguesa
Ciência.Inovação 2010

Futebol Robótico 3D

André Alçada Guimarães

José António Oliveira Silva

Rui Alberto Felber Hilário

Alunos do 5º ano da Licenciatura em Engenharia Electrotécnica e de Computadores pela Faculdade de Engenharia da Universidade do Porto

Orientadores:

Luís Paulo Reis

Doutor em Engenharia Electrotécnica e de Computadores pela Faculdade de Engenharia da Universidade do Porto

Nuno Lau

Doutor em Engenharia Electrónica e de Telecomunicações pela Universidade de Aveiro

Trabalho realizado no âmbito do Projecto de Fim de curso na disciplina de PSTFC, do 2º semestre, do 5º ano, da Licenciatura em Eng. Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto, leccionada por João José Pinto Ferreira.

Julho de 2006

Financiamento FCT/POCI

União Europeia

FCT

Fundação para a Ciência e a Tecnologia

Governo da República Portuguesa

Ciência.Inovação 2010

Resumo

O objectivo fundamental do trabalho apresentado neste documento relaciona-se com o estudo de metodologias de coordenação entre múltiplos agentes autónomos computacionais em ambientes que implicam a execução cooperativa de tarefas, nomeadamente a liga de simulação 3D do RoboCup. Nesta liga é permitido a duas equipas de agentes autónomos e competitivos disputar um jogo de futebol simulado. O domínio providenciado é muito realista na medida em que inclui muitas complexidades do futebol real e dos sistemas robóticos, tais como: erros nos sensores e erros nos actuadores. Os desafios aqui colocados são muito superiores aos colocados noutros problemas típicos de inteligência artificial, como o xadrez. Aqui o mundo é dinâmico, parcialmente inacessível aos agentes, contínuo, não determinístico, multi-objectivo, parcialmente cooperativo e parcialmente adverso.

O trabalho realizado passou pelo desenvolvimento de algoritmos de decisão de passe e remate e o desenvolvimento de todos os algoritmos de comunicação assim como o próprio formato desta. Foi também modelizada a trajectória da bola para diversas combinações de chutos. O trabalho integrou-se num projecto mais vasto e que envolveu outros investigadores e que resultou na equipa FC Portugal 2006 que venceu as ligas de simulação 3D dos campeonatos Europeu e Mundial de futebol robótico.

As principais contribuições do trabalho aqui apresentadas foram: a criação de um sistema de comunicação que permite aos agentes transmitir o estado do mundo e os seus objectivos imediatos aos outros agentes; e a definição de mecanismos de decisão para a escolha da potência e ângulo a serem aplicados à bola por um agente de forma a esta atingir o ponto desejado do modo mais fiável, evitando os obstáculos.

Abstract

The main goal, described in this document, is the study of coordination between multiple autonomous computer agents in environments that need the execution of cooperative tasks, i.e., the 3D simulation league in RoboCup. In this league two teams of autonomous and competitive agents play a game of simulated soccer. This type of simulation is very realistic as it provides many of the complexities that exist in real soccer and also in robotic systems, like sensor and actuators errors. The challenges provided by this league are vastly superior to the ones provided by other typical artificial intelligence problems, like chess. Here the world is dynamic, partially inaccessible to the agents, continuous, non-deterministic, multi-goal, partially cooperative and partially adverse.

In this context, pass and shoot decision algorithms were created as well as the support for communication between agents. Several functions were created to describe the ball trajectory resulting of several kick power and angle combinations. This work was joined with the work previously developed resulting in the agent FC Portugal 2006 that won the 3D simulation league in the European and World Championships of robotic soccer.

The main contributions of this work are the creation of a communication system that allow our agents to describe the world state to their fellow agents and the creation of decision mechanisms of the angle and power of the kick applied on the ball by an agent allowing it to reach the desired goal in a reliable way, avoiding all the obstacles.

Aos Orientadores

Agradecimentos

Não podia faltar aqui um agradecimento aos nossos orientadores Luís Paulo Reis e Nuno Lau, pela ajuda e por nos terem incutido um espírito de vencedores.

Vai também uma nota de agradecimento ao LIACC e a FEUP por disponibilizar as instalações e equipamentos.

Não podemos deixar de agradecer ao PRODEP pela bolsa POCI atribuída para a realização deste trabalho.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

1. Introdução

1.1 Enquadramento

Este trabalho enquadra-se no âmbito do Projecto de Fim de curso na disciplina de Projecto, Seminário ou Trabalho do Fim de Curso. As áreas abordadas no trabalho são essencialmente a Inteligência Artificial, Agentes Autónomos, Decisão e Comunicação Inteligente.

1.2 Motivação

Este trabalho resulta da importância crescente das metodologias de coordenação em sistemas multi-agente. A importância do RoboCup como domínio de teste e desenvolvimento de algoritmos a ser aplicados no domínio da Inteligência Artificial constitui outra das motivações deste trabalho.

1.3 Objectivos

A iniciativa RoboCup é um projecto de investigação e educação internacional com o objectivo de promover a investigação em Inteligência Artificial (Distribuída) e Robótica Inteligente. O projecto baseia-se na utilização de um problema standard – o futebol robótico – onde um elevado conjunto de tecnologias são necessárias para ser capaz de construir uma equipa de Robôs reais ou virtuais que seja capaz de participar num desafio de futebol, jogado de acordo com um determinado conjunto de regras pré-especificado.

Este projecto permitiu trabalhar directamente nas equipas de futebol robótico do LIACC-FEUP e em conjunto com os investigadores envolvidos no projecto no LIACC e na FEUP.

O objectivo principal deste trabalho é desenvolver novas capacidades básicas para a equipa FCPortugal 3D, refinar o modelo de actualização do estado do mundo, integrar um novo

motor de decisão individual e colectiva e desenvolver um novo modelo de comunicação estratégica para o Futebol Robótico em 3D.

Destaque-se que, até ao início deste projecto, a FEUP conquistara já 16 troféus internacionais no RoboCup, incluindo dois títulos de campeão do mundo e três de campeão da Europa. No âmbito deste projecto a equipa participou no campeonato da Europa (Holanda) e no campeonato do Mundo de Futebol Robótico (Alemanha).

1.4 Estrutura do Relatório

Este trabalho encontra-se estruturado em 8 capítulos dos quais, o primeiro é composto por esta introdução ao trabalho.

No segundo capítulo é apresentado o conceito de RoboCup e são descritas as diferentes modalidades que o constitui.

No terceiro capítulo é dado destaque à equipa FCPortugal, focando a sua estrutura e dinâmica.

No quarto capítulo apresenta-se a descrição do problema focando o futebol robótico simulado, destacando as diferentes modalidades e fazendo uma breve comparação com o anterior problema *standard* em Inteligência Artificial.

A implementação do trabalho, o kick e a comunicação, seguem-se nos dois capítulos seguintes.

Nos dois últimos capítulos é feita uma análise dos resultados obtidos e as conclusões gerais deste trabalho, e são apresentadas algumas perspectivas de desenvolvimentos futuros.

Capítulo 2

2. RoboCup

Com o objectivo de dinamizar a evolução da Inteligência Artificial (IA), em particular dos Sistemas Autónomos Multi-agentes, a primeira edição do RoboCup decorreu em 1997 em Nagoya no Japão, depois de se ter organizado um pré-RoboCup em Osaka para identificar possíveis problemas e dificuldades na organização deste tipo de eventos a nível mundial. Esta iniciativa foi levada a cabo pelo Dr. Hiroaki Kitano (Tóquio), um investigador na área da IA que se tornou presidente e fundador da RoboCup Federation.

2.1 Liga RoboCup

Os desenvolvimentos no campo da IA avançam lado a lado com a evolução dos computadores que, ao longo do tempo foram fazendo com que se começassem a encarar essas máquinas como inteligentes, alterando mesmo o nosso conceito de inteligência e aproximando os conceitos 'máquina', tradicionalmente não inteligente, da 'inteligência', capacidade antes consignada exclusivamente ao homem. Neste sentido, devemos então fornecer à máquina uma avalanche de dados, teorias formais de 'bom senso', de crenças, de um universo simbólico superior, segundo a qual a máquina seria capaz de raciocinar utilizando conceitos complexos, e de perceber o seu meio envolvente.

A existência de um ambiente dinâmico e de ser jogado contra um adversário cujo objectivo é exactamente o oposto, levou com que o futebol fosse inicialmente a grande motivação do RoboCup, não só pela popularidade em quase todo o mundo, mas por apresentar desafios científicos bastantes importantes, isto porque é um jogo colectivo com problemas individuais (identificação e localização de objectos relevantes e de si próprio, dribles, remates, etc.), e ao mesmo tempo, problemas colectivos (estratégia, passes, etc.).

A seguir passamos a descrever cada uma das competições existentes na Liga RoboCup.

2.2 Competições existentes

Actualmente o RoboCup está dividido em quatro competições sendo cada uma delas composto por várias ligas. Destas competições é de salientar, para além do futebol robótico em si, outros desafios associados a este. Destacam-se o RoboCup Rescue, o RoboCup Júnior e mais recentemente o RoboCup@home.

2.2.1 RoboCupSoccer

No RoboCupSoccer estão englobadas as competições de futebol robótico, nas suas variadas formas. Ligas robóticas (utilizando robôs pequenos, médios, cães e humanóides) e as ligas de simulação.

2.2.1.1 Middle size league

Nesta liga, duas equipas jogam com quatro robôs móveis autónomos com rodas, com alturas que variam entre os 50cm e 90cm. Toda a informação relativa ao ambiente é obtida pelos sensores e a comunicação (se existir) é feita via *wireless*. Para orientar os robôs são colocados objectos diferenciados por cores no terreno de jogo.

Tirando a introdução e remoção de robôs do terreno de jogo e recolocação da bola em campo, não é permitida qualquer intervenção humana durante o jogo.

2.2.1.2 Small size league

Aqui os robôs são menores. Todo o processamento é efectuado através de um computador central, baseado em informação (posição dos jogadores, posição da bola) vinda de uma câmara de vídeo colocada por cima do campo de jogo. A intervenção humana resume-se à introdução e remoção dos robôs no campo e a comunicação entre os robôs é feita usando comunicação wireless.

2.2.1.3 Four-legged league

Nesta liga, como o nome indica, são usados robôs de quatro patas. Cada equipa é constituída por 4 robôs da Sony, o cão robótico AIBO. Apresentam sensores incorporados e comunicam via *wireless*.

2.2.1.4 Simulation leagues

Neste trabalho um destaque especial é dado à liga de simulação. Nesta, ao contrário das descritas anteriormente, não existem robôs físicos. Existe sim um simulador que fornece aos agentes todos os dados que seriam obtidos na realidade por sensores. O SoccerServer providencia um domínio realístico, no sentido em que inclui muitas complexidades do futebol real (erros, actuadores de energia limitada, etc.).

Cada agente é um processo separado e são no total 11 de cada equipa. Actualmente existem três ligas de simulação: 2D, 3D e coach.

2.2.2 RoboCupRescue

A busca e salvamento em cenários de catástrofe são uma das áreas onde se prevê grande intervenção de sistemas artificiais no futuro. Apareceu na edição de 2001 uma nova competição integrada no RoboCup, o RoboCupRescue. O objectivo principal do projecto é promover a pesquisa e desenvolvimento de soluções para esta área, o que leva a uma crescente melhoria de soluções de coordenação de equipas multi-agente. Sendo estes agentes diferentes uns dos outros, constituindo por isso equipas de agentes heterogéneos, soluções robóticas de busca e salvamento, etc. No futuro, muita da tecnologia desenvolvida para o RoboCup irá certamente integrar as equipas reais de busca e salvamento.

Dentro desta liga podemos destacar a liga robótica e a liga de simulação:

2.2.2.1 Rescue Robot league

O objectivo é procurar as vítimas e construir um mapa do ambiente, num edifício parcialmente destruído onde existem vítimas que esperam ajuda exterior, com zonas de diferentes níveis de dificuldade. Os robôs podem ser autónomos ou tele-operados

2.2.2.2 Rescue Simulation league

Nesta liga tenta-se criar um cenário mais realista possível em que o cenário de catástrofe é mais generalizada. Os agentes desenvolvidos pelas equipas irão actuar neste ambiente. Três conjuntos de agentes (bombeiros, polícias e ambulâncias, com as respectivas centrais) irão fazer tudo para salvar o maior número possível de vítimas e evitar que os incêndios existentes se propaguem e consumam toda a cidade, minimizando os estragos na cidade. Este ambiente obriga as equipas a pesquisar e apresentar soluções em diversas áreas (planeamento multi-agente, coordenação de agentes heterogéneos, planeamento robusto, etc.). A equipa FCPortugal conquistou o primeiro lugar, este ano, no campeonato europeu realizado em Eiendhoven nesta modalidade.

2.2.3 RoboCupJunior

Para que os mais jovens estudantes dos ensinos primário e secundário também possam participar e estimular o interesse deste grupo nas áreas da robótica e da IA, foi criado este grupo de competições. Procura-se também promover as capacidades dos participantes em trabalhar e resolver problemas para atingir um objectivo comum, fornecendo aos jovens diversos desafios.

2.2.4 RoboCup@Home

RoboCup@Home é uma liga nova dentro das competições de RoboCup que focaliza em aplicações do mundo real e na interacção homem-máquina com robôs autónomos. O alvo é promover o desenvolvimento das aplicações robóticas úteis que possam ajudar a seres humanos na vida diária.

2.2.5 RoboCup na FEUP

A equipa da FEUP, 5DPO, é campeã nacional na categoria de robôs pequenos e vice-campeã na categoria de robôs médios, competição que foi realizada no 6º Festival Nacional de Robótica 2006 em Guimarães.

A nível internacional, a 5DPO é actualmente vice-campeã mundial de robôs pequenos no campeonato mundial de Futebol Robótico - RoboCup 2006 - realizado em Bremen na Alemanha. A equipa conquistou também dois campeonatos europeus e um terceiro lugar no mundial. Nos robôs médios o 5DPO obteve dois terceiros lugares em campeonatos europeus.

No próximo capítulo é dado destaque à equipa FCPortugal, que se sagrou este ano campeã europeia, no Europeu realizado em Eindhoven e campeã mundial na edição do RoboCup realizada em Bremen na Alemanha, na modalidade de simulação 3D. Fazemos uma breve descrição das diferenças face à versão 2D e uma pequena descrição de cada um dos módulos do FCPagent e do código central da equipa.

Capítulo 3

3. FCPortugal

FCPortugal é uma equipa de simulação robótica que se tornou campeã europeia este ano, na modalidade de simulação 3D, em Eindhoven e campeã mundial em Bremen – RoboCup 2006. A equipa deu os primeiros passos no ano 2000 e foi desenvolvida por Luís Paulo Reis, Universidade do Porto, juntamente com Nuno Lau e Luís Seabra Lopes da Universidade de Aveiro, vencendo em Junho desse mesmo ano em Amesterdão, o Campeonato Europeu de RoboSoccer e dois meses depois o campeonato mundial, marcando 86 e 94 golos respectivamente sem sofrer nenhum.

3.1 RoboCup3D Vs RoboCup2D

Em 2004 nasceu a liga de simulação 3D, exactamente com os mesmos objectivos da 2D. Esta nova competição adiciona uma terceira dimensão ao jogo procurando-o tornar mais interessante e realístico. A transição do 2D para o 3D introduziu novos desafios na simulação de Sistemas Multi-Agentes como *"like searching of huge state space"*, raciocínio espacial, etc. O servidor 3D imita o ambiente do futebol real considerando a física envolvida nele. Ao contrário da simulação 2D, a simulação 3D é praticamente continua e o servidor 3D usa "SPADES middle-ware agent" para remover alguns dos inconvenientes do servidor 2D, como a flutuação do desempenho da equipa devido à máquina e à carga da rede. Para a avaliação do agente, segue também o *"thinking time"* do agente.

3.2 FCPortugal Agent

O FCPortugal Agent 3D (FCPagent) é composto por 6 módulos – Figura 1.

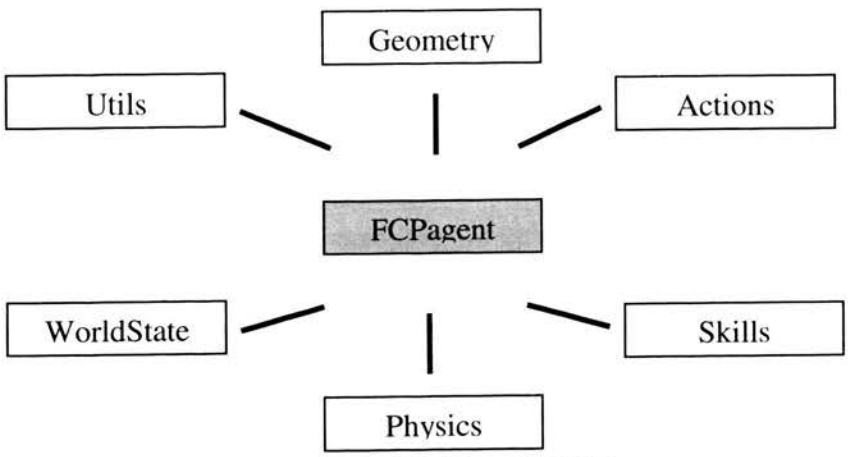

Figura 1: Módulos do FCPagent

3.2.1 WorldState

É de certeza o mais complexo – Figura 2. Apresenta todas as informações acerca do estado do mundo que o agente precisa para decidir qual acção a tomar. Existem três tipos de informações que o WorldState precisa: informações acerca dos objectos (jogadores, marcadores e da bola), informação das condições do jogo (comprimento do campo, da baliza, etc.) e o estado do jogo (o resultado, o tempo, etc.).

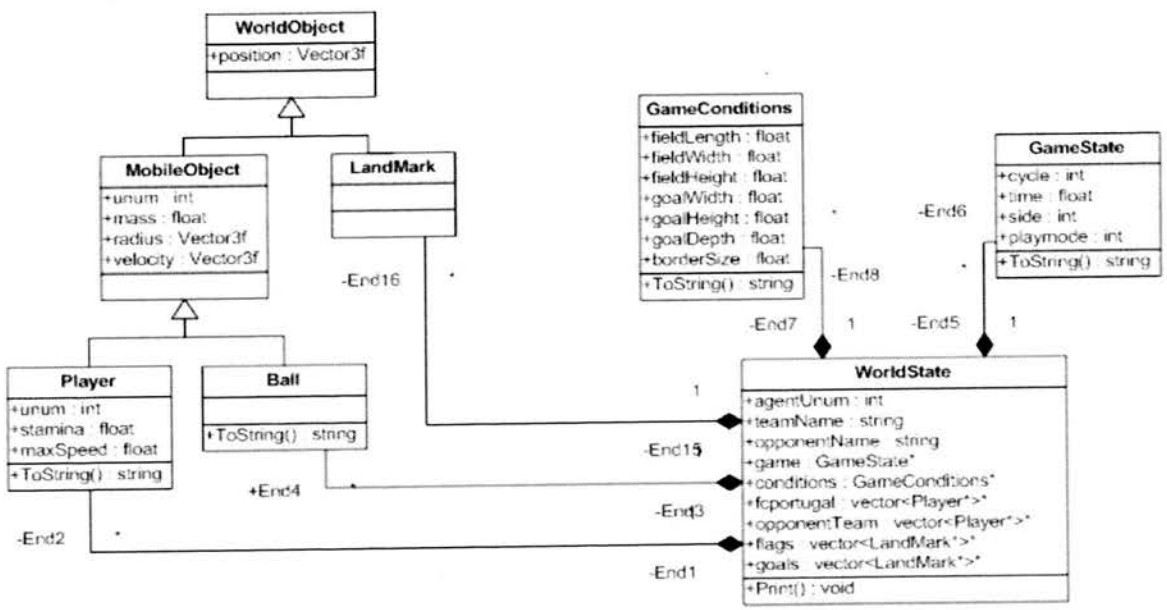

Figura 2: Arquitectura do WorldState

3.2.2 Physics

Para produzir interacções físicas entre os corpos precisamos deste pacote. Podemos estimar a velocidade dos objectos, a força aplicada num dado corpo e a aceleração.

3.2.3 Geometry

Aqui temos implementado duas classes: o Vector3f e o Vector. Ambos disponibilizam métodos para a manipulação e cálculos usando vectores 2D e 3D.

3.2.4 Skills

Disponibiliza as acções de mais baixo nível que um agente pode efectuar – Figura 3. Chutar a bola, rodar o corpo, interceptar a bola e fazer dribles são alguns dos exemplos. A cada um está associado um método *Execute( )*, que permite executar a acção.

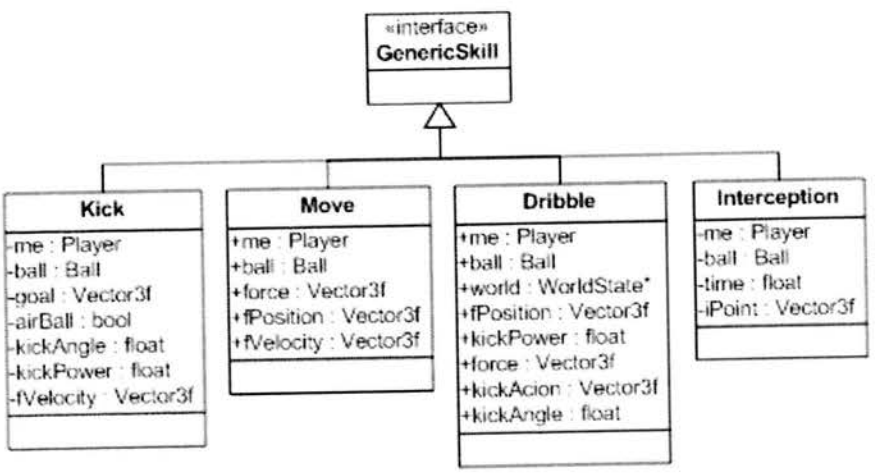

Figura 3: Arquitectura do Skills

3.2.5 Actions

Actions – Figura 4 - é um conjunto de skills que em conjunto produzem um comportamento de mais alto nível. Como exemplo, temos um remate à baliza, fazer um passe longo, etc.

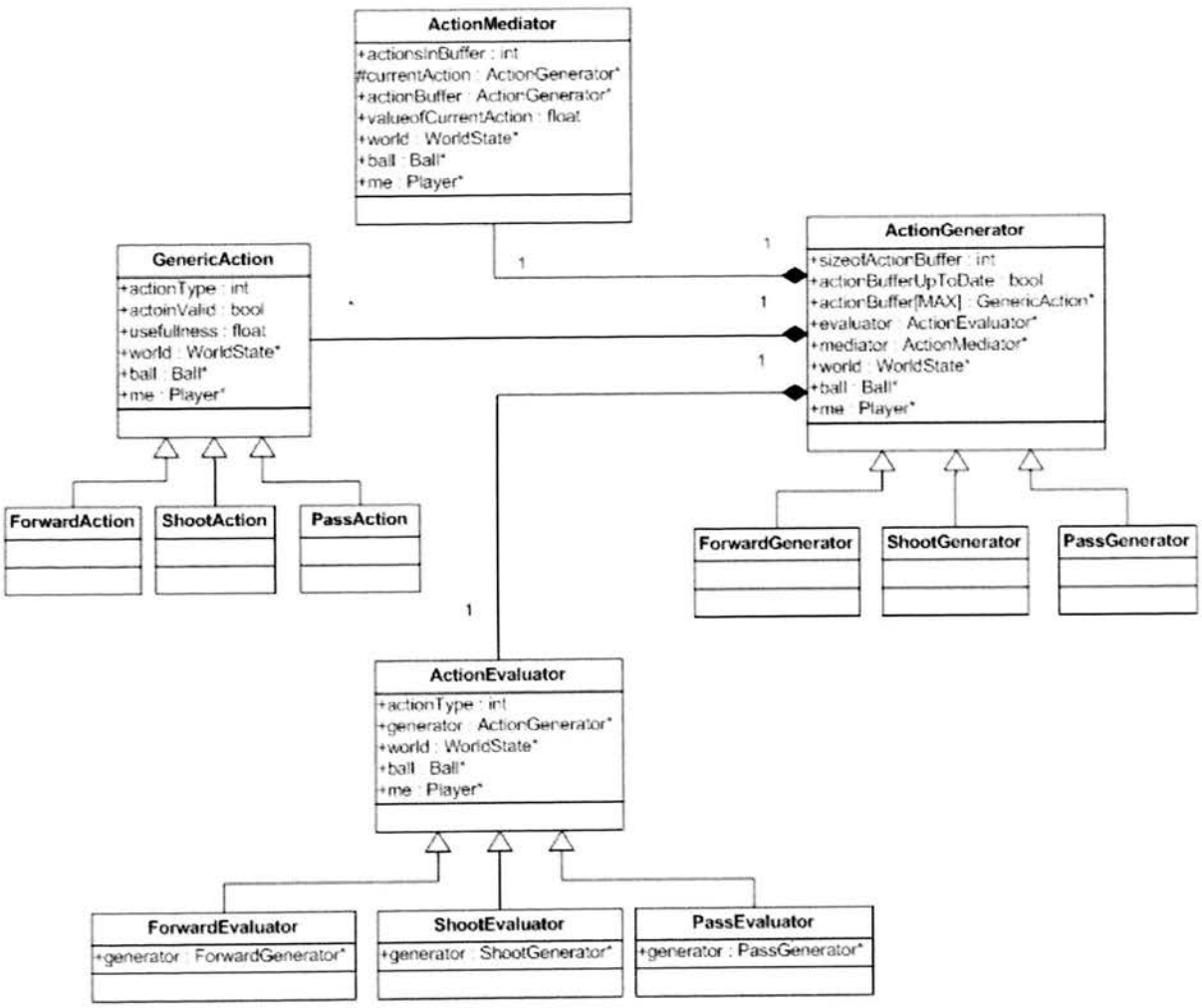

Figura 4: Arquitectura do Actions

Para produzir uma acção 4 classes são utilizadas: o *mediator,* o *evaluator,* o *generator* e a classe *actions.* O *generator* (*ActionGenerator*) é o responsável pela criação das acções que vão ser executadas. A ele estão associadas mais 3 classes: *pass*, *shoot* e *foward*. Cada uma destas podem produzir outras acções que podem ser consideradas em ciclos seguintes. As acções retornadas por cada *generator* apresentam propriedades próprias de acordo com o tipo que lhes estão associados. A evolução de uma acção é gerada pelo *evoluator* (*ActionEvoluator*). Esta é a classe que permite ao agente estimar a utilidade e importância de cada acção gerada. Para juntar tudo, o FCPagent usa o *mediator (ActionMediator)* que tem a capacidade

de chamar as funções necessárias para gerar e evoluir todos os tipos de acções e decidir qual acção que vai ser executada.

3.2.6 Utils

Este pacote foi implementado para conter classes que têm relevância directa sobre o comportamento, mas facilita na execução de algumas tarefas. Por exemplo temos a criação dos *logs files*, comunicação com o servidor, recepção das mensagens e composição das mensagens para enviar uma acção para o servidor.

3.3 Comportamento dos Agentes

O comportamento dos agentes em campo está intimamente ligado com o SBSP - *Situation Based Strategic Positioning* – [Reis e Lau, 2001]. O SBSP começou a ser usado pelo FCPortugal na versão 2D e consiste numa atribuição estratégica de posições para cada um dos agentes para uma dada posição da bola e situação de jogo. O jogador que se desloca para a bola é aquele que a sua posição estratégica estiver mais próximo da bola.

Antes do *kick-off* ou quando a bola estiver na posse do adversário, por alguma razão (canto, lançamento, etc), os jogadores do FCPortugal posicionam segundo o algoritmo SBSP.

Na situação contrária, quando a bola estiver na posse do FCPortugal, o jogador mais próximo da bola (*bestFCPortugalInterceptor*) tenta segurar ou chuta-la enquanto os outros se deslocam de acordo com o SBSP. Se a bola estiver próxima da baliza do FCPortugal os jogadores tentam chuta-la para os lados mas de maneira a não a entregar ao adversário, caso contrario chutam na direcção da baliza adversária. O algoritmo é apresentado na tabela 1.

```
Case (Playmode) equals
{
   BeforeKickOff:
      MoveAccordingSBSP();

   FCPortugalKickOff:
   {
      If (nyNumber == 9)
         RunToTheBallAndKickIt();
      Else
         MoveAccordingSBSP();
   }

   OpponentKickOff or OpponentKickIn or
   OpponentCornerKick or OpponentGoalKick:
   {
      MoveAccordingSBSP();
```

```
    }

    FCPortugalKickIn or FCPortugalCornerKick or
    FCPortugalGoalKick or PlayOn:
    {
       If (MyMoveToTheBallAccordingSBSP())
          RunToTheBallAndKickIt();
       Else
          MoveAccordingSBSP();
    }
}
```

Tabela 1: Comportamento do Agente

3.4 Conclusão

Foi escolhida uma arquitectura apropriada a uma melhor implementação da estratégia da equipa. Nesta arquitectura o agente apresenta um tradicional ciclo de controlo usando a percepção, interpretação e predição para actualizar o estado do mundo para depois com sucesso decidir qual a melhor acção a tomar e em seguida executar essa acção [Reis, Lau, Oliveira, 2000]. O domínio do conhecimento está estruturado em tácticas, formação, protocolos de comunicação e das *sensations* vindas do mundo envolvente.

No próximo capítulo compara-se o problema *standard* inicial em Inteligência Artificial, o xadrez, com o RoboCup focando o futebol robótico e analisando mais em detalhe cada uma das *sensations, perceptions* e a comunicação. Mais adiante descrevemos o *soccerserver* destacando o simulador 3D (instalação e execução), o DebugMonitor e a comunicação agente/servidor.

Capítulo 4

4. Futebol Robótico Simulado

4.1 As Ligas de Simulação

Como foi dito anteriormente na liga de simulação existem três competições que não envolvem robôs físicos. De seguida passamos a descrever cada uma delas:

4.1.1 Simulação 2D

Todo o ambiente é processado em duas dimensões – Figura 5. Esta foi a primeira liga de simulação a existir.

Figura 5: Monitor 2D

A liga de simulação 2D é baseada no sistema de simulação livre *soccerserver*. Aqui simula-se um campo contendo 2 equipas de 11 jogadores e uma bola que se conectam ao simulador, numa arquitectura cliente-servidor, através de sockets UDP. Quer na execução dos comandos enviados pelos jogadores quer na informação enviada pelo servidor é adicionado ruído de forma a dar realismo ao jogo.

4.1.2 Simulação 3D

Competição de simulação 3D, tema do nosso trabalho, processa-se de forma semelhante à competição 2D, com a diferença do ambiente ser todo a três dimensões – Figura 6. O servidor usado é diferente, mas a lógica de funcionamento é semelhante. A competição iniciou-se no ano de 2004.

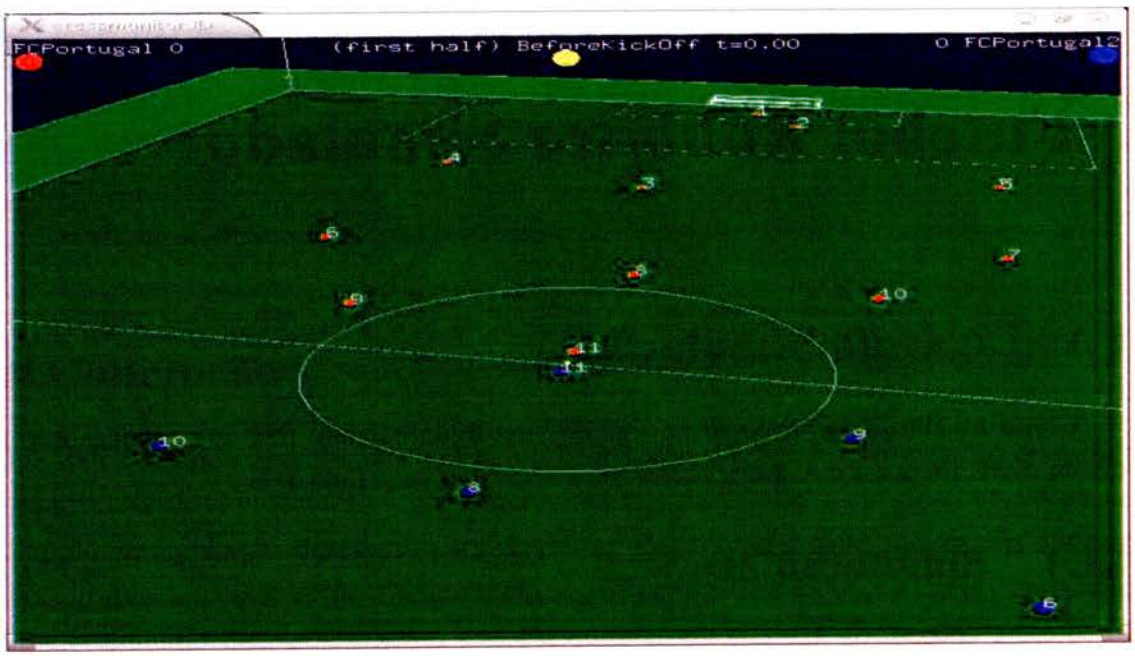

Figura 6: Monitor 3D

4.1.3 Simulação coach

Nesta liga o que se pretende avaliar não é o desempenho da equipa através da programação dos jogadores, mas sim através do treinador. O desafio é utilizar o treinador para coordenar e definir tácticas para esta equipa, de forma a ganhar à equipa adversária.

FCPortugal é a equipa portuguesa de futebol robótico simulado que compete no evento mundial de inteligência artificial aplicada ao futebol, o RoboCup, e compete nestas três últimas ligas referidas.

4.2 RoboCup Vs Xadrez

Até há alguns anos, a visão humana estava muito direccionada ao jogo de xadrez efectuado por computadores. Após o confronto com Kasparov em 1997, ficou provado que embora a rapidez seja importante o método usado para a resolução do problema podia ser o mais simples e menos complicado possível. Sendo assim, Kitano et al.[1995] sugeriu que o RoboCup passasse a ser o substituto do xadrez como problema *standard* em Inteligência Artificial. Nos pontos seguintes vamos comparar o problema *standard* inicial com o novo, fazendo uma pequena análise aos *Perceptors*, *Actions*, *Effectors* e a Comunicação, focando o Futebol Robótico.

4.2.1 Sensores - Perceptors

Um *Perceptor* é um mecanismo que permite a um agente receber informações do mundo exterior. As percepções num jogo de xadrez podem ser, num modelo mais simples, apenas um tabuleiro e a posição das peças no tabuleiro. Num modelo mais complexo podemos incluir as estratégias em cada instante do adversário e ainda prever o número das várias jogadas possíveis, tendo em conta que cada peça pode executar um número limitado de acções. Por exemplo a rainha, a peça com mais graus de liberdade, pode teoricamente mover em oito direcções e até mais sete a seguir a cada uma destas direcções.

No *Soccer challenges* encontramos características semelhantes ao xadrez: temos um campo e as posições dos jogadores. A diferença na percepção dos jogadores e das peças são claras, pois no xadrez temos informações perfeitas sobre tudo o que se encontra no tabuleiro. Porém no Soccer os jogadores têm apenas a percepção dos agentes que conseguem ver, isto é, a posição dos jogadores no seu ângulo de visão com um alcance limitado.

Devido à característica muito realística do *SoccerServer* existe muita incerteza nos dados recebidos por parte dos agentes ao contrário do que acontece no xadrez em que as posições são discretas e sem erros. A informação torna-se ainda mais complexa quando adicionarmos a larga gama de atributos – direcção do corpo e pescoço, velocidade, etc - que compõe um agente sabendo que cada um destes atributos apresenta erros.

Na presente versão do servidor temos 4 tipos de *Perceptors*: *Vision, GameState, AgentState, e o Hear.*

4.2.1.1 Vision

Permite ao agente ter informações acerca da posição da bola bem como da posição de todos os jogadores e bandeiras no terreno de jogo. A posição dos objectos é dada por coordenadas polares - Figura 3 -, com origens no centro do agente e com o eixo horizontal na direcção da baliza adversária. Na anterior versão do servidor os agentes tinham uma visão de 360º. Com o *RestrictedVisionPerceptor*, que veio com a nova versão, a visão foi restrita a apenas 180º.

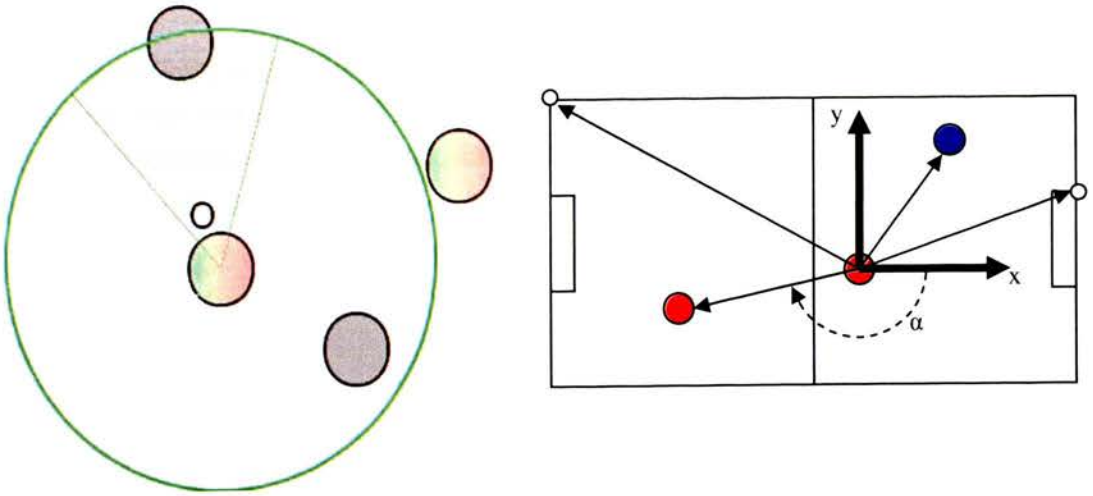

Figura 7: Vison Perceptor e modo como o agente indica a localização de um objecto

A este *Perceptor* estão associados dois tipos de erros: o erro de calibração da posição da câmara e ruído nos parâmetros. O erro de calibração está uniformemente distribuído, para cada eixo, no intervalo [-0.005 0.005]. O ruído nos parâmetros tem uma distribuição normal, a volta de 0.0 e a variância varia consoante o tipo de parâmetro. Para a distancia temos um σ = 0.0965, para o ângulo horizontal (plano x-y) temos σ = 0.1225 e para o ângulo longitudinal um σ = 0.1480.

```
Sint int (Vision
            (Flag (id string) (pol float float float)) ...
            (Goal (id string) (pol float float float)) ...
            (Ball (pol float float float))
            (Player (team string) (id int) (pol float float float)) ...
        )

S12 10 (Vision (Flag (id 1_1) (pol 77.13 27.82 -0.32)) ...
               (Goal (id 1_1) (pol 68.31 3.07 -0.35)) ...
               (Ball (pol 14.88 0.03 -1.22))
               (Player (team FCPortugal) (id 1) (pol 29.53 57.84 -0.40))

   )
```

Tabela 2: Exemplo do vison perceptor

4.2.1.2 GameState

O *GameState perceptor* fornece ao agente informações relativas ao jogo, como o número da equipa, o tamanho dos objectos, o tempo de jogo e o estado actual do jogo. O exemplo do *Exemplo* na tabela 3 mostra-nos informações acerca do agente 1 do FCPortugal antes do início do jogo.

```
S12 10 (GameState (unum 1) (team FCPortugal) (FieldLength
104)(FieldWidth 72)
                  (FieldHeight 40) (GoalWidth 7.32) (GoalDepth 2)
                  (GoalHeight 0.5) (BorderSize 10) (AgentMass 75)
                  (AgentRadius 0.22) (AgentMaxSpeed 10) (BallRadius
0.111)
                  (BallMass 0.43) (time 0) (playmode BeforeKickOff))
```

Tabela 3: Exemplo do GameState perceptor

4.2.1.3 AgentState

Fornece informações acerca do estado do agente. O ângulo actual da câmara (em graus), o nível da bateria e a temperatura.

```
Sint int (AgentState (battery float) (temp float))

Exemplo:
  S12 10 (AgentState (battery 100) (temp 23))
```

Tabela 4: Exemplo do AgentState perceptor

4.2.1.4 Hear

Para o campeonato europeu na Holanda, a parte da comunicação ainda não tinha sido implementada. Portanto o nosso agente só podia usar este *perceptor* se usasse o *SayEffector* para enviar mensagens.

Quando um jogador recebe uma mensagem a sua capacidade de audição é decrementada de um factor – *hearDecay* – e é incrementada a cada ciclo por um factor – *hearInc*. Para evitar que uma equipa torne a comunicação do seu oponente inútil, sobrecarregando o canal, a capacidade de comunicar de cada jogador é diferente para cada equipa. A mensagem dita por um jogador apenas é transmitida para os jogadores que estiverem a uma distância *audioCutDist*. Na tabela 5 apresentamos os parâmetros do servidor que afectam o *Hear Perceptor* e o exemplo de uma mensagem recebida.

Parameter	Values	Example Hear output
audioCutDist	50.0	(hear 0.8 -179.99 Test_1) (hear 0.4 self Test_2)
hearMax	2	
hearInc	1	
hearDecay	2	
sayMsgSize	512	

Tabela 5: Parâmetros e um exemplo do Hear perceptor

4.2.2 Acções - Effectors

Effector é um mecanismo que permite a um agente produzir uma acção sobre o mundo. No mundo do xadrez um *Action* é muito simples, pois temos um número limitado de acções em cada jogada, desde um simples avanço de um peão ao mais complexo movimento de um cavalo.

No domínio do RoboCup, a deslocação de um único jogador é mais complexo. Para se efectuar um simples movimento são precisos em média nove *actions* em cada instante de tempo. Nos pontos seguintes descrevemos alguns dos 8 *effectors*, mais importantes, disponíveis na última versão do servidor.

4.2.2.1 Create

Inicialmente quando conectamos ao simulador, o nosso agente não tem uma representação física. Com o *createEffector* nós podemos pedir ao simulador diferentes tipos de *effectors*, *perceptors* ou tipos de robôs. De momento só podemos pedir um único tipo de robô, ao contrário do que acontece no rescue simulation. Portanto o servidor ignora todos os outros parâmetros que possamos passar.

```
Action (create)

Exemplo:
A(create)
```

Tabela 6: Create Effector

4.2.2.2 Init

O *initEffector* deve proceder o *createEffector*. Permite associar o agente a uma dada equipa.

```
Action (init (unum int) (teamname str))

Exemplo:
A(init (unum 0) (teamname FCPortugal))
```

Tabela 7: Init Effector

O exemplo na tabela anterior permite inicializar o agente como sendo pertencente à equipa do FCPortugal. O unum informa sobre o *ID* do agente e caso este valor seja nulo, o servidor atribui automaticamente um valor e informa-o usando o *GameState Perceptor*.

4.2.2.3 Drive

Permite aplicar uma força sobre o agente e faze-lo mover para uma determinada posição.

```
Action (drive (pol float float float))

Exemplo:
A(drive 20.0 10.0 0.0)
```

Tabela 8: Init Effector

O exemplo mostra como aplicar uma força (20.0, 10.0, 0.0) no plano horizontal. O agente ainda não pode saltar, por isso se aplicarmos uma força segundo a vertical será inútil. A máxima força que pode ser aplicada é de 100.

4.2.2.4 Kick

O kickEffector permite ao agente chutar a bola se esta estiver a uma distância de chuto - *kickable distance*.[1]

```
Action (kick float float))
Exemplo:
A(kick 20.0 100.0))
```

Tabela 9: Kick Effector

O exemplo acima descreve um chuto à potência máxima (100.0) com um ângulo vertical de 20º. O ângulo de chuto varia entre [0º, 50º]. Para converter a potência de chuto para a força, em newton, temos que multiplicar a potência por *ForceFactor* = 0.7. A força é aplicada na bola durante 10 ciclos (0.1ms) .

O kick é radial no plano horizontal – Figura 8 - o que implica a que o agente tenha que se posicionar de acordo com a direcção a que vai chutar, como ilustrado na figura seguinte.

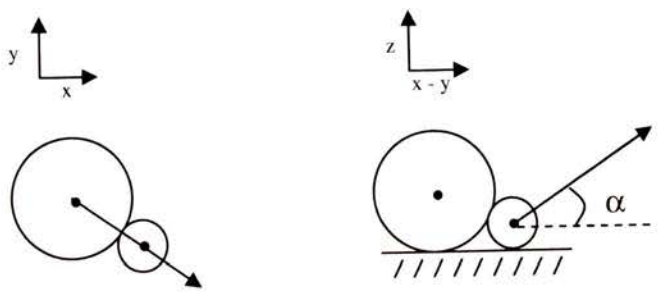

Figura 8: A direcção do kick

4.2.2.5 Catch

O Guarda-redes – *Goalie* – é o único jogador com a habilidade de agarrar a bola. Ele pode segurar a bola se esta estiver dentro da área de penalty ou se esta estiver junto a este numa margem denominada de *catchmargin*. O valor actual é de 1 metro. O *catchEffector* coloca a bola à frente do guarda-redes a faz deslocar os outros jogadores a uma distância de 5 metros.

4.2.2.6 Say

Como foi dito anteriormente, para se enviar uma mensagem aos outros agentes o agente tem que usar o *SayEffector*. As mensagens têm que ter um tamanho máximo de 512 caracteres e destes não se pode usar espaços e parênteses. As mensagens podem ser ouvidas por aque-

---

[1] *Kickable distance* = *AgentRadius* + *BallRadius* + *KickMargin*. – Raios do jogador e da bola somados a *KickMargin* que estão definidos no ficheiro de configuração *rcssserver.rb*. Os valores actualmente definidos são 0.22, 0.111 e 0.07 respectivamente.

les que estiverem a uma distância de 50 metros por membros de ambas as equipas. O uso do *SayEffector* é apenas restrita pela limitação da capacidade dos agentes ouvirem ou não a mensagem.

O *Kick* e o *Say*, temas centrais do nosso projecto, são descritos mais em pormenores no próximo capítulo.

4.3 Simulador - *SoccerServer*

Funcionando numa arquitectura cliente-servidor, o *SoccerServer* (simulador) permite às duas equipas compostas por 11 agentes (clientes) autónomos comunicarem entre si. O tipo de linguagem ou sistema operativo para a construção dos agentes é opcional mas a comunicação só pode ser feita usando sockets UDP. Uma das vantagens do *SoccerServer* é permitir um certo nível de abstracção de problemas que os robôs têm, por exemplo, como fazer um robô mover-se. Esta abstracção permite aos investigadores concentrarem-se num nível de conceitos mais elevados, tais como a cooperação e a aprendizagem.

O *SoccerServer* é composto por dois programas, *soccerserver* e *soccermonitor*. O *Soccerserver* é um programa que simula os movimentos da bola e dos jogadores, comunica com os clientes e controla um jogo de acordo com as regras. O *soccermonitor* é um programa que mostra o campo virtual do *soccerverver* numa janela do sistema. Vários programas *soccermonitor* podem se conectar a um só *soccerserver*. Assim podemos ver um "campo-janela" em exposições múltiplas. Um cliente conecta-se ao *soccerserver* por um socket UDP, e pode enviar comandos para controlar um seu jogador e receber informações dos sensores do jogador. Ou seja, um programa cliente é o cérebro do jogador: O cliente recebe a informação visual e auditora do servidor, e envia comandos de controlo para o servidor.

4.3.1 Comunicação Agente/Servidor

O servidor estabelece a comunicação com os agentes enviando uma mensagem do tipo "D" – *done* – e aguarda a execução da inicialização por parte dos agentes que devem enviar uma mensagem *"I"* – *init* – A partir daqui o servidor começa a enviar *Sensations* e os agentes respondem com *Actions*. As *Actions* devem ser finalizadas com uma mensagem do tipo "D".

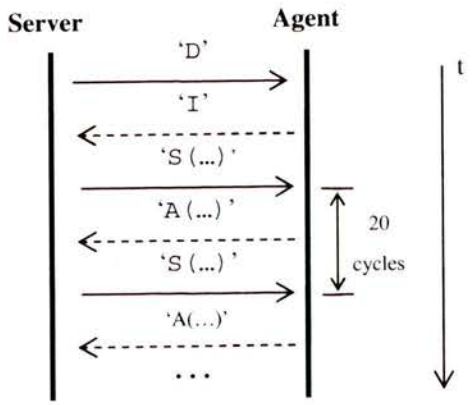

Figura 9: Comunicação entre agente e servidor

4.3.2 Simulador 3D

Para se proceder a instalação da última versão do simulador 3D (0.51), que vai ser utilizada no campeonato do mundo de RoboCup 2006 em Bremen, é necessário um computador com Linux com as livrarias de OpenGL e Glut instaladas. Adicionalmente, são necessários ainda os pacotes Spades, Boost, Ruby, ODE e o pacote do simulador Rcssserver3D.

Aqui fica uma breve descrição de cada pacote necessário e de onde podem ser obtidos:

- **SPADES** (System for Parallel Agent Discrete Event Simulation)

  Sistema que permite a criação de simulações multi-agente. Permite uma distribuição robusta dos agentes por várias máquinas e controla o tempo de decisão dos agentes.

  http://prdownloads.sourceforge.net/spades-sim/spades-1.10.tar.gz?download

- **BOOST**

  Proporciona um conjunto de bibliotecas que são necessárias pelos outros pacotes.

  http://prdownloads.sourceforge.net/boost/boost_1_33_1.tar.gz?download

- **RUBY**

  Ruby é uma linguagem de interpretação de scripts utilizada na programação orientada a objectos. Permite o processamento de ficheiros de texto e a realização de tarefas de gestão.

  ftp://ftp.ruby-lang.org/pub/ruby/ruby-1.8.3.tar.gz

- **ODE** (Open Dynamics Engine)

  Simula a dinâmica de corpos rígidos articulados e permite a detecção de colisões entre esses objectos.

  http://prdownloads.sourceforge.net/opende/ode-0.5.tgz?download

- **RCSSSERVER3D**

  O RoboCup Soccer Simulator é um simulador 3D de futebol que permite que duas equipas de 11 jogadores autónomos joguem entre si.

  http://prdownloads.sourceforge.net/sserver/rcssserver3d-0.5.1.tar.gz?download

4.3.3 Instalação do Simulador 3D

De seguida encontram-se os comandos (Linux) necessários à instalação do simulador. Deve-se substituir "Caminho_Completo_Dir_Destino" pelo caminho completo do directório onde se quer instalar o simulador e substituir "Caminho_Completo_dos_Pacotes" pelo caminho do directório para onde os pacotes necessários foram descarregados.

```
mkdir /Caminho_Completo_Dir_Destino
cd /Caminho_Completo_Dir_Destino/
export LDFLAGS="-L/usr/X11R6/lib -L/Caminho_Completo_Dir_Destino/lib"
```

```
#Spades
cp /Caminho_Completo_dos_Pacotes/spades-1.10.tar.gz .
tar xvf spades-1.10.tar.gz
rm spades-1.10.tar.gz
mv spades-1.10 spades-root
cd spades-root
configure --prefix=/Caminho_Completo_Dir_Destino/
make
make install
export SPADES=/Caminho_Completo_Dir_Destino/
cd ..
rm -r -d spades-root
```

```
#Boost
cp /Caminho_Completo_dos_Pacotes/boost_1_33_1.tar.gz .
tar xvf boost_1_33_1.tar.gz
rm boost_1_33_1.tar.gz
mv boost_1_33_1 boost-root
export CPPFLAGS=-I/Caminho_Completo_Dir_Destino/boost-root/
```

```
#Ruby
cp /Caminho_Completo_dos_Pacotes/ruby-1.8.3.tar.gz .
tar xvf ruby-1.8.3.tar.gz
rm ruby-1.8.3.tar.gz
mv ruby-1.8.3 ruby-root
cd ruby-root
configure --prefix=/Caminho_Completo_Dir_Destino/ --enable-shared
make
make install
export CXXFLAGS=-I/Caminho_Completo_Dir_Destino/lib/ruby/1.8/i686-linux/
export RUBY=/Caminho_Completo_Dir_Destino/bin/ruby
cd ..
```

```
#ODE
cp /Caminho_Completo_dos_Pacotes/ode-0.5.tgz .
tar xvf ode-0.5.tgz
rm ode-0.5.tgz
mv ode-0.5 ode-root
cd ode-root
make
export ODE=/Caminho_Completo_Dir_Destino/ode-root/
cd ..
```

```
#Rcssserver3D
cp /Caminho_Completo_dos_Pacotes/rcssserver3d-0.5.1.tar.gz .
tar xvf rcssserver3d-0.5.1.tar.gz
rm rcssserver3d-0.5.1.tar.gz
cd rcssserver3d-0.5.1
configure --prefix=/Caminho_Completo_Dir_Destino/
make
make install
```

Tabela 10: Instalação do Simulador 3D

4.3.4 Execução do Servidor

Depois da instalação deve-se criar um ficheiro "agentdb.list" com os caminhos dos binários dos agentes que vão jogar ou editar o já existente e adicionar as equipas que queremos que joguem. De seguida, mostra-se o conteúdo típico deste ficheiro:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<agentdb
  xmlns:adb="http://spades-sim.sourceforge.net/agentdbxml.html"
  xmlns="http://spades-sim.sourceforge.net/agentdbxml.html"
  adb:version="0.91">
<!--  This file specifies the different agent types and how to execute
the agents -->
<agent_type_external name="fcpagent3D">
      <inputfd>3</inputfd>
      <outputfd>4</outputfd>
      <timer>jiffies 2000</timer>
      <working_dir>/usr/users2/deec/fcp3d/equipas/Eindhoven-
fcp3D/</working_dir>
  <exec_line>./fcpagent --teamname Eindhoven-fcp3D</exec_line>
</agent_type_external>
<agent_type_external name="ARZ">
      <inputfd>3</inputfd>
      <outputfd>4</outputfd>
      <timer>jiffies 2000</timer>
      <working_dir>/usr/users2/deec/fcp3d/equipas/ARZ/</working_dir>
  <exec_line>./fcpagent --teamname ARZ</exec_line>
</agent_type_external>
</agentdb>
```

Tabela 11: Ficheiro Xml com a lista dos agentes

Neste exemplo podem-se ver duas equipas Eindhoven-fcp3D e ARZ e os respectivos caminhos para os binários.

De seguida deve-se verificar a existência do ficheiro de configuração do servidor "rcss-server.rb" no directório "~/.rcssserver3d/". Deve-se então editá-lo de modo a adicionar e pôr a jogar as equipas desejadas. Caso não exista copiar para lá o ficheiro com o mesmo nome que se encontra no directório de instalação "rcssserver3d-0.5.1". Por pré-definição defrontam-se as equipas de teste "foo" e "bar" (últimas duas linhas), que como foram feitas para a versão .3 do servidor (em que os agentes tinham conhecimento total do mundo) não devem conseguir encontrar a bola (na versão actual os agentes têm um campo de visão mais reduzido e por isso têm que "olhar" para os objectos para os verem). Por exemplo, se o ficheiro agentdb.list anterior for utilizado pode-se por a jogar a equipa Eindhoven-fcp3D contra a equipa ARZ alterando as linhas seguintes:

spadesServer.queueAgents('foo',11)

spadesServer.queueAgents('bar',11)

para

spadesServer.queueAgents('fcpagent3D',11)

spadesServer.queueAgents('ARZ',11)

De seguida para executar o servidor corre-se na shell o comando:

```
/Caminho_Completo_Dir_Destino/bin/rcssserver3D --ipc_force_remove --
agent_db_fn /path_completo_do_ficheiro_agentdb/agentdb.list
```

Noutra shell pode-se então correr o visualizador enquanto o servidor está a correr para ver o jogo:

```
/Caminho_Completo_Dir_Destino/bin/rcssmonitor3D-lite
```

Para iniciar o jogo carrega-se na tecla "k" (de kick-off) para dar início à primeira parte e outra vez depois do intervalo para dar início à segunda. Para controlar a câmara de jogo utilizam-se as teclas de cursor ou W, A, S e D.

Depois de um jogo ter acabado pode também ser visualizado acedendo aos logs criados e correndo o visualizador com o seguinte comando:

```
/Caminho_Completo_Dir_Destino/bin/rcssmonitor3D-lite --logfile
path_do_log/nome_do_log.log (tipicamente
/Caminho_Completo_Dir_Destino/bin/Logfiles/monitor.log)
```

4.4 Obtenção dos dados

4.4.1 Logfiles

Para obtermos os dados que descrevem o comportamento dos jogadores em campo, o servidor e os agentes geram *logfiles*. A partir destes *logs* conseguimos retirar informações respeitantes aos objectos em campo (posição, velocidade, direcção, etc.), mensagens enviadas e recebidas pelos agentes bem como informação estatística necessária para a análise do comportamento da equipa. Na tabela 12 apresentamos um exemplo.

```
Msg: C0
 Msg: S39 49 (Vision (Flag (id 1_l) (pol 75.1264 -65.2808 -19.19)) (Flag
(id 2_l) (pol 38.9976 8.55031 -38.9127)) (Flag (id 1_r) (pol 151.559 -
25.5884 -9.34223)) (Flag (id 2_r) (pol 137.358 1.99813 -10.2077)) (Goal
(id 1_l) (pol 51.3238 -48.2647 -28.4973)) (Goal (id 2_l) (pol 46.8462 -
41.2644 -31.5221)) (Goal (id 1_r) (pol 141.254 -14.0214 -10.0048)) (Goal
(id 2_r) (pol 139.709 -11.0419 -10.1016)) (Ball (pol 91.1625 -19.8944 -
15.4539)) ( (id 0) (pol 85.3132 -14.1721 -15.159)) ( (id 0) (pol 81.274 -
13.7479 -14.8137)) ( (id 0) (pol 76.8974 -13.941 -14.9673)) ( (id 0) (pol
72.5703 -14.0442 -15.0347)) ( (id 0) (pol 68.2266 -14.2163 -15.1828)) (
(id 0) (pol 63.9316 -14.2855 -15.2339)) ( (id 0) (pol 59.7216 -14.1471 -
15.1132)) ( (id 0) (pol 55.5392 -13.9121 -14.9124)) ( (id 0) (pol 51.2446
-13.9744 -14.9578)) ( (id 0) (pol 46.9373 -14.0902 -15.0476)) ( (id 0)
(pol 42.6647 -14.1055 -15.0541)) ( (id 0) (pol 38.4398 -13.935 -14.9068))
( (id 0) (pol 34.1409 -14.064 -15.0202)) ( (id 0) (pol 29.8648 -14.1026 -
15.0461)) ( (id 0) (pol 26.4588 -13.6032 -14.5171)) ( (id 0) (pol 24.6332
-12.1557 -13.0265)) ( (id 0) (pol 22.2442 -10.6257 -11.466)) ( (id 0) (pol
18.9916 -9.16138 -9.96237)) ( (id 0) (pol 14.5334 -7.73642 -8.50179)) (
(id 0) (pol 8.39146 -6.64115 -7.33259)))(GameState (time 0) (playmode Be-
foreKickOff))(AgentState (pan_tilt 0 0) (battery 100) (temp 23))
X Collision pos=-30.012 posCalc=0
Y Collision pos=10.913 posCalc=0
updateCalc bVel=(393.7134, -142.4787, -115.7
updateCalc corrected bVelZ=-120.477
```

Tabela 12: Exemplo criado por um agente

4.4.2 DebugMonitor

Com os *logs* gerados durante uma simulação podemos usar o *DebugMonitor*, que nos permite analisar passo a passo o movimento dos nossos jogadores, da equipa adversária e da bola bem como ver as linhas de passe em cada instante de jogo.

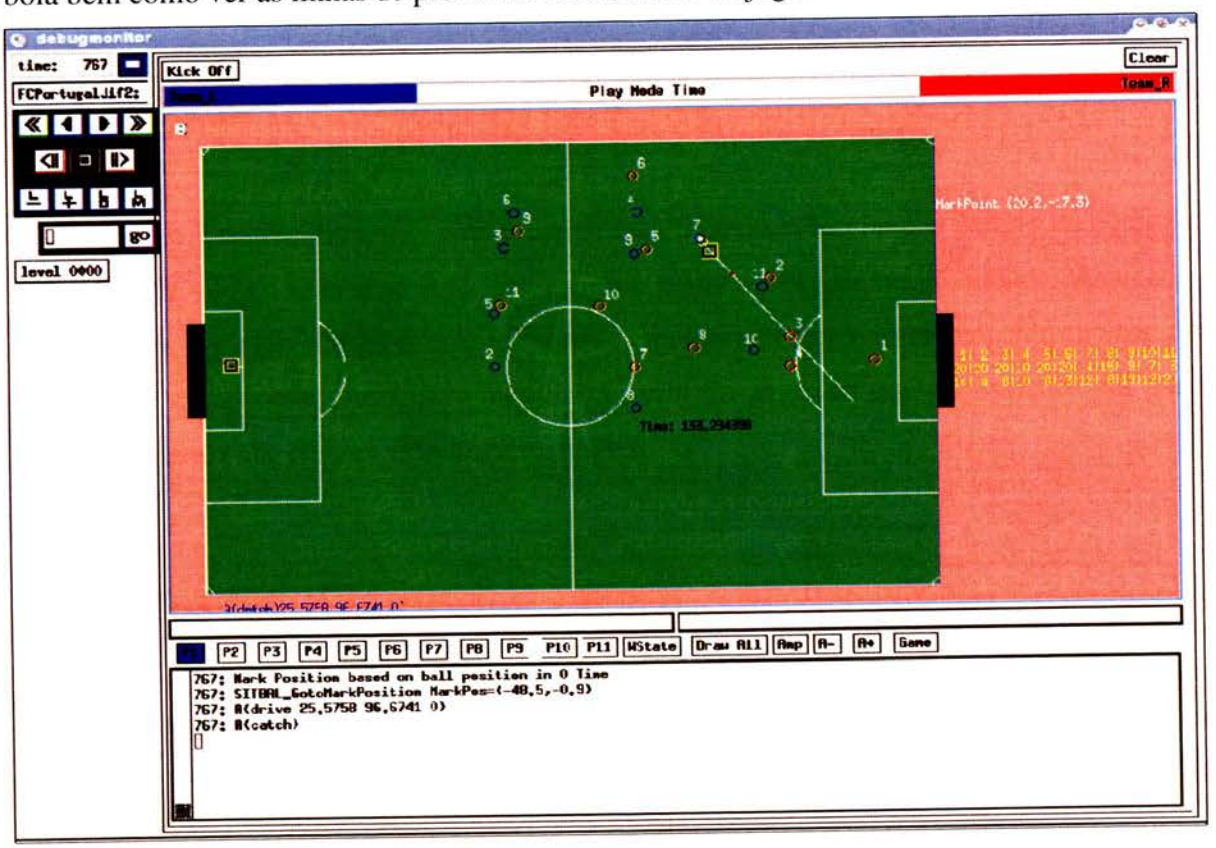

Figura 10: DebugMonitor

4.5 Nota adicional

O simulador está em constante evolução. Do campeonato europeu para o mundial foram adicionos novos pacotes e ao que existia antes foi actualizado corrigindo assim alguns bugs. A arquitectura actual foi construída tendo estes pressupostos em mente, permitindo criar novos desafios tornado cada vez mais complexo e realístico o simulador.

Com as novas regras e funcionalidades as equipas têm que adaptar o seu código, o que muitas vezes implica uma mudança ou melhoramento na estatégia das equipas.

Nos próximos capítulos abordamos o Kick e a comunicação temas centrais do nosso trabalho.

Capítulo 5

5. Kick

Para o desenvolvimento deste projecto foi utilizado como base o código da equipa do FCPortugal3D 2005. O código fonte foi mantido praticamente na sua totalidade, mantendo a sua estrutura, e a esse código foram adicionados melhoramentos e implementadas novas classes. Com o campeonato europeu e mundial à porta foi sugerido que pegássemos em partes chaves que ainda não tinham sido implementadas.

5.1 Implementação

Cada agente para comunicar a sua intenção de chutar a bola tem que enviar ao servidor um *Action-Effector*. Como dito anteriormente, o ângulo do *kick* – *kickAngle* – é o ângulo em relação ao plano horizontal (aceitando valores entre *[0º, 50º]* e a potência – *kickPower* – é a percentagem de potência a ser aplicada à bola (aceitando, por isso, valores entre *[0, 100]*).

O jogador chuta sempre na direcção para que está virado, o que implica que se tenha que mover o agente para que a bola possa ir na direcção desejada. Depois de efectuado o comando, o servidor valida o pontapé verificando se o jogador estava à distância de chuto da bola (*Kickable distance)* e se no momento de chuto o jogador estava no chão. Se estas condições se verificarem, é aplicada uma força à bola durante 10 ciclos (aproximadamente 1/10s em tempo de simulação). Este comando tem vários erros associados. Há ruído nos dois ângulos de chuto (em relação ao chão e à direcção em que o jogador está virado). No plano horizontal o erro é muito baixo (distribuído entre 0 e 0.02) enquanto que o erro em relação ao chão varia: está compreendido entre *[0, 0.9]* nos ângulos extremos [0º, 50º], aumentando até 4.5 nos ângulos a meio da escala (20º~35º).

Também se tem que se considerar os erros associados à movimentação do agente para chutar, que aumentam os erros no plano horizontal.

5.1.1 Trajectória da bola

A trajectória da bola à partida, usando as leis da física, podia ser calculada comparando o seu comportamento ao do lançamento de um projéctil. Mas estando perante um ambiente muito dinâmico e um pouco realista, existem vários erros associados. Sendo assim as trajectórias da bola foram estimadas a partir de experiências.

Para obter uma aproximação o mais exacta possível da trajectória, foram realizadas algumas centenas de pontapés, fazendo variar a potência e o ângulo de maneira controlada. A partir dos dados obtidos tentou-se obter funções interpoladoras para a distância e altura percorridas pela bola em função do tempo, ângulo e potência do chuto.

5.1.1.1 Função Distância

Primeiro, obtiveram-se funções da variação do ângulo entre *[0º, 50º]* (com incrementos de 5º) e do tempo para potência constante e igual à potência máxima, ou seja, 100%. Estas funções foram obtidas a partir dos dados experimentais por interpolação de Newton de grau 6.

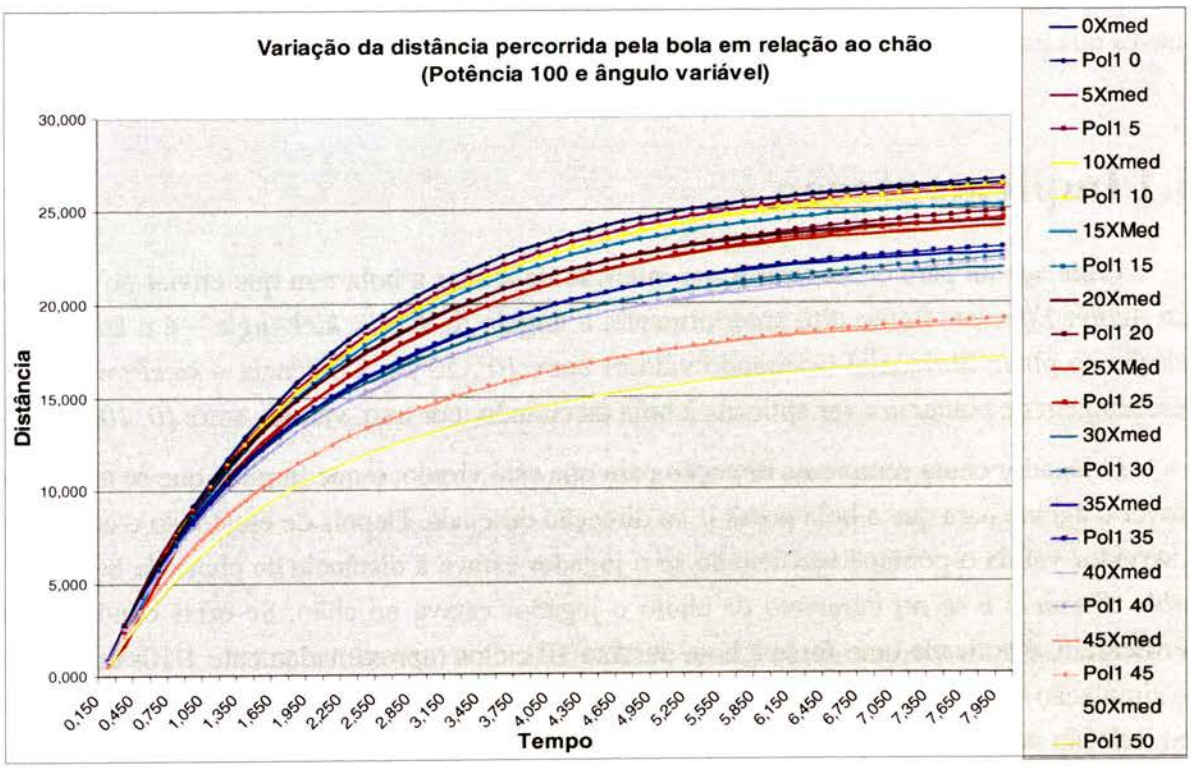

Figura 11: Variação da distancia para a potência máxima e ângulo variável.

Como seria de esperar há uma grande aceleração nos primeiros momentos com velocidades que vão dos 8 aos 14 m/s nos primeiros instantes (até ao 1,5 s). A partir dos 10.0s as velocidades são inferiores a 0.1 m/s e a posição da bola é aproximadamente constante. Verifica-se que à medida que o ângulo aumenta, a distância percorrida diminui de forma cada vez mais acentuada. A partir dos 40º perde-se aproximadamente 2 metros de alcance por cada 5º de elevação adicionados. Fizeram-se também testes para ângulos superiores a 50º, constatando-se que o servidor arredonda estes valores para os 50º.

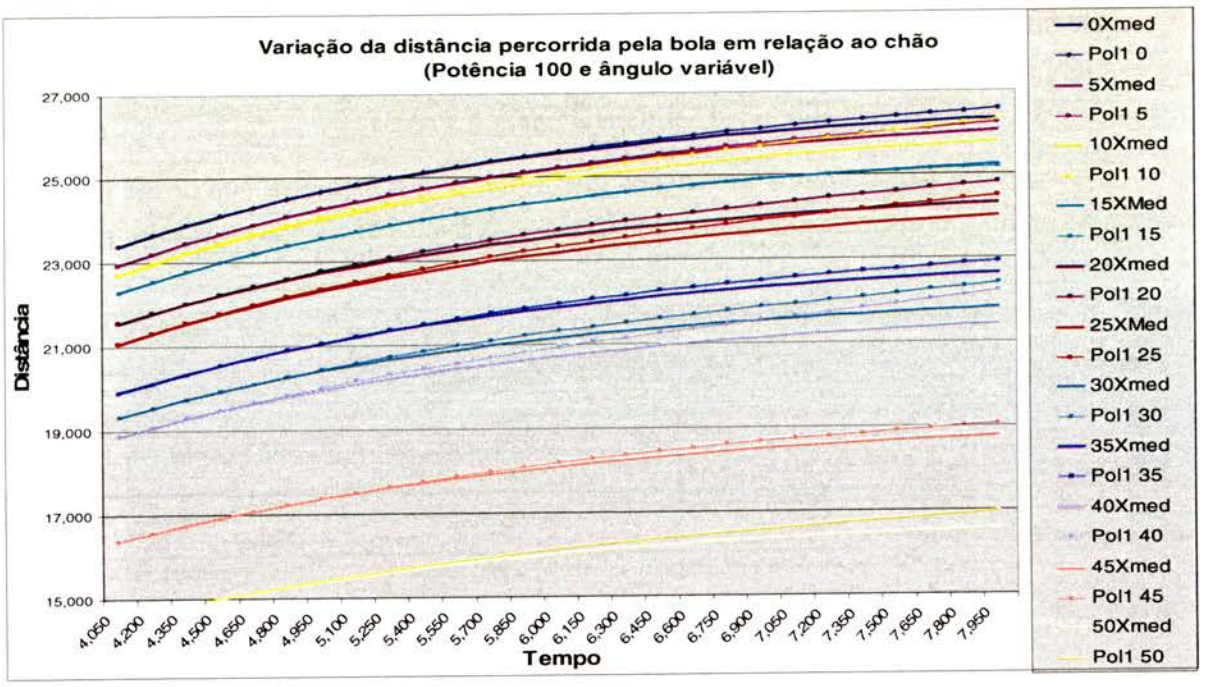

Figura 12: Variação da distancia. Uso de polinómios interpoladores

Verifica-se na figura acima – Figura 12 - que o erro introduzido pelo uso dos polinómios interpoladores é bastante reduzido, sempre abaixo dos 20 cm até aos 7 segundos.

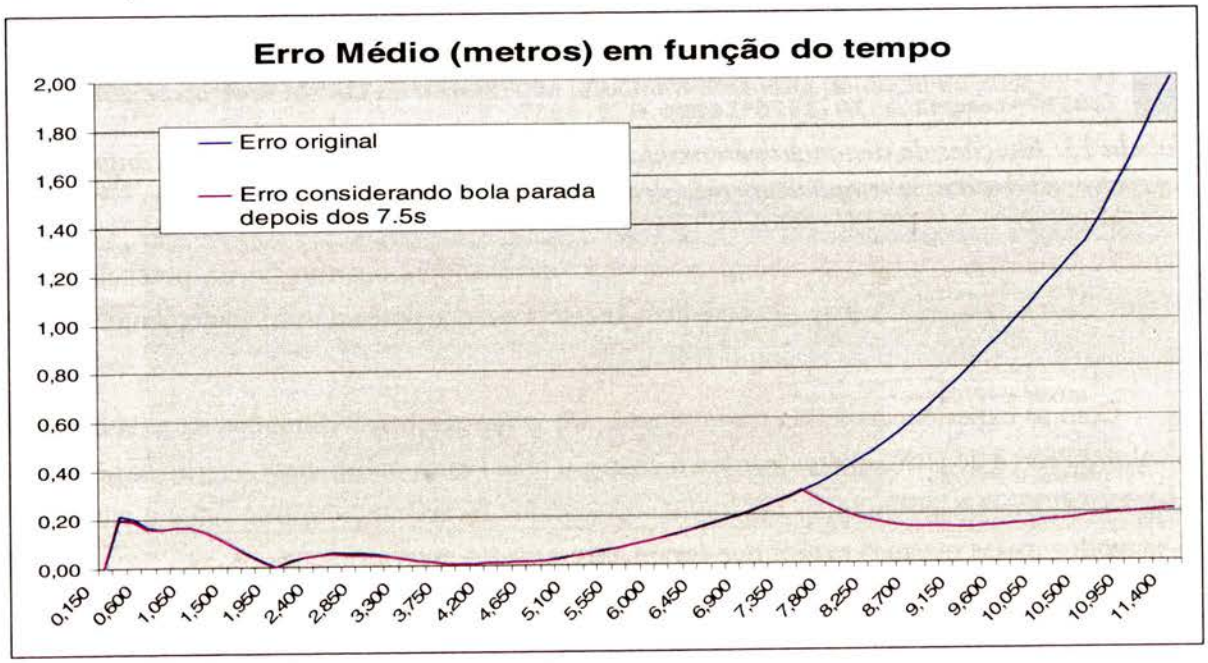

Figura 13: Erro médio em função do tempo

Podem-se definir três zonas de erro a partir da curva obtida no gráfico da figura 13. Na primeira zona que vai até aos 2 segundos, tem-se um erro que vai diminuindo aproximadamente linearmente dos 20 cm até aos 0 cm. Este erro deve-se às velocidades elevadas neste período, que se traduzem numa certa incerteza acerca da posição da bola. Segue-se um período de baixo erro entre os 2.4 e os 5 segundos em que o erro fica abaixo dos 6 cm.

Na última zona o erro cresce exponencialmente, devido à divergência entre as funções interpoladoras (devido ao arredondamento dos factores do polinómio – foram consideradas 4 casas decimais) e dos dados obtidos experimentalmente.

De seguida apresentam-se as funções que descrevem a distância percorrida pela bola para cada ângulo, obtidas através da interpolação dos dados experimentais e válidas para tempos inferiores a 7.5 segundos.

0°	-0.0001*tempo6 + 0.0037*tempo5 - 0.0606*tempo4 + 0.5889*tempo3 - 3.7066*tempo2 + 14.6069*tempo - 1.3675
5°	-0.0001*tempo6 + 0.0037*tempo5 - 0.0600*tempo4 + 0.5733*tempo3 - 3.5637*tempo2 + 14.0911*tempo - 1.1879
10°	-0.0001*tempo6 + 0.0038*tempo5 - 0.0626*tempo4 + 0.5989*tempo3 - 3.6626*tempo2 + 14.1880*tempo - 1.2999
15°	0.0008*tempo5 - 0.0275*tempo4 + 0.3942*tempo3 - 3.0559*tempo2 + 13.3155*tempo -1.1636
20°	-0.0001*tempo6 + 0.0042*tempo5 - 0.0729*tempo4 + 0.6933*tempo3 - 4.0004*tempo2 + 14.2992*tempo - 1.2976
25°	-0.0001*tempo6 + 0.0038*tempo5 - 0.0618*tempo4 + 0.5766*tempo3 - 3.4357*tempo2 + 13.1537*tempo - 1.2058
30°	-0.0002*tempo6 + 0.0076*tempo5 - 0.1168*tempo4 + 0.9566*tempo3 - 4.6499*tempo2 + 14.1766*tempo - 1.3455
35°	-0.0001*tempo6 + 0.0044*tempo5 - 0.0768*tempo4 + 0.7102*tempo3 - 3.909*tempo2 + 13.4117*tempo - 1.1683
40°	-0.0002*tempo6 + 0.0074*tempo5 - 0.1110*tempo4 + 0.8938*tempo3 - 4.3236*tempo2 + 13.3890*tempo - 1.1351
45°	-0.0001*tempo6 + 0.0044*tempo5 - 0.0758*tempo4 + 0.6792*tempo3 - 3.5472*tempo2 + 11.4949*tempo - 1.0725
50°	-0.0001*tempo6 + 0.0043*tempo5-0.0729*tempo4+0.6454*tempo3- 3.3197*tempo2 + 10.4926*tempo - 0.9907

Tabela 13: Funções da distância percorrida pela bola no plano XY em função do ângulo de chuto e do tempo decorrido desde o mesmo (até aos 7.5 segundos)

Foi necessário obter, também, a variação da distância em função da potência. Para obter os dados relativos a esta dependência fez-se variar a potência em incrementos de 10 entre os 10% e os 100%, para os ângulos 0º e 25º.

Com as experiências feitas com o ângulo 0º, pode-se obter a dinâmica da bola no chão e o valor da força de atrito exercida entre o chão e a bola (valor obtido para o atrito experimentalmente é 0.72). Os polinómios interpoladores obtidos foram também truncados a partir dos 7.5 segundos, pelas mesmas razões que foram apresentadas anteriormente.

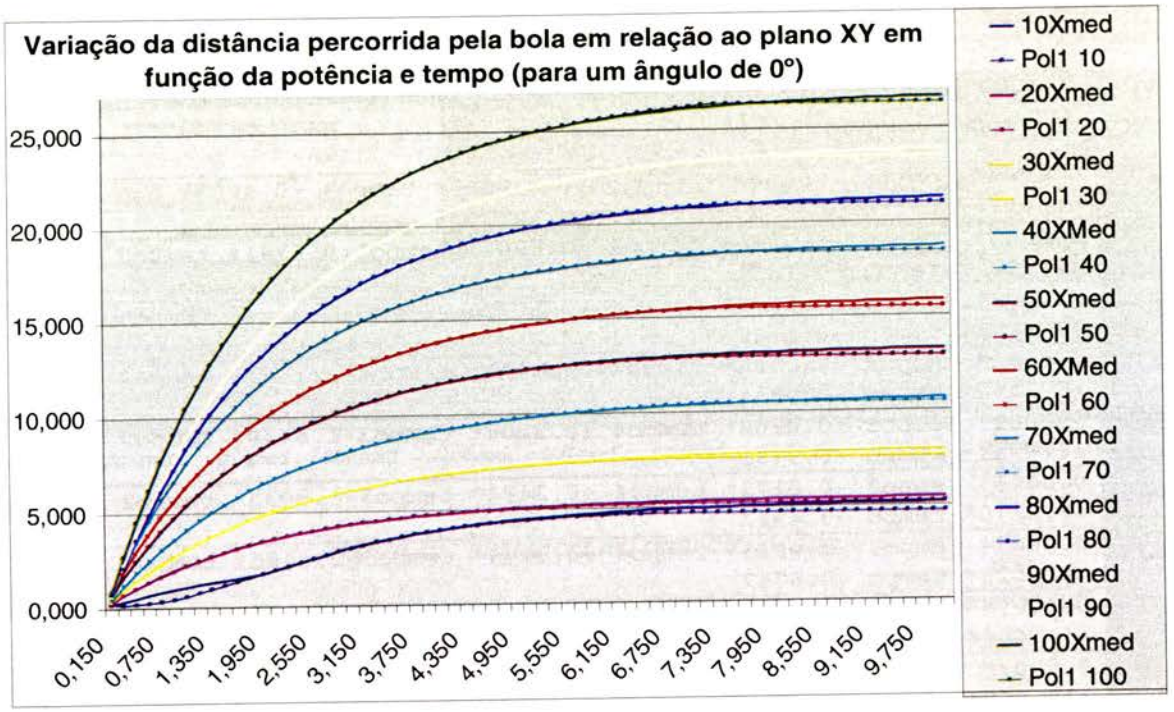

Figura 14: Variação da distância percorrida em função da potência

Pode-se verificar – Figura 15 - que a variação com a potência é aproximadamente linear, tirando para a potência 10. Neste caso, o jogador batia num primeiro momento com o corpo na bola impulsionando-a lentamente e só depois é que a chutava. Na prática chutar com uma potência de 10% resulta no mesmo que um chuto com uma potência de 20%.

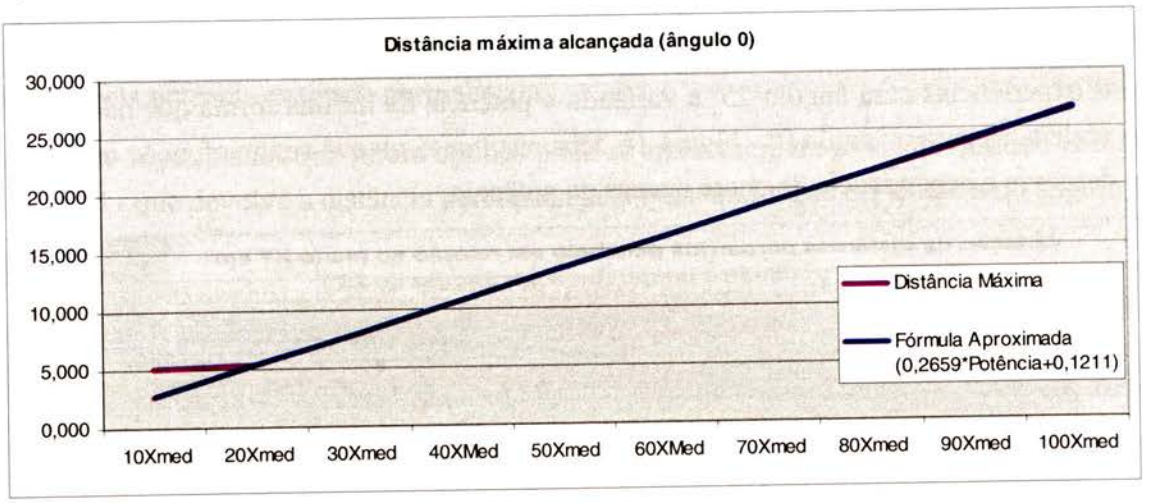

Figura 15: Distância máxima alcançada com ângulo de 0º

As funções que descrevem a posição da bola no chão (bola chutada com ângulo 0º) em função da potência e tempo são:

10%	0.0001* tempo6 - 0.0039* tempo5 + 0.0588* tempo4 -0.4283* tempo3+1.4138* tempo2 -0.7626* tempo +0.2698
20%	0.0002* tempo5 -0.0066* tempo4 +0.0904* tempo3-0.6781* tempo2 +2.8682* tempo -0.1531
30%	0.0002* tempo5 -0.0073* tempo4 +0.1079* tempo3-0.8644* tempo2 +3.9156* tempo -0.2162
40%	0.0003* tempo5 -0.0103* tempo4 +0.1492* tempo3-1.1761* tempo2 +5.2740* tempo -0.3254
50%	0.0005* tempo5 -0.0162* tempo4 +0.2204* tempo3-1.6418* tempo2 +6.9561* tempo -0.3733
60%	0.0005* tempo5 -0.0173* tempo4 +0.2474* tempo3-1.9035* tempo2 +8.2170* tempo -0.4543
70%	0.0007* tempo5 -0.0229* tempo4 +0.3149* tempo3-2.3486* tempo2 +9.8925* tempo -0.5767
80%	0.0007* tempo5 -0.0243* tempo4 +0.3486* tempo3-2.6707* tempo2 +11.4088* tempo -1.0857
90%	0.0007* tempo5 -0.0239* tempo4 +0.3441* tempo3-2.7027* tempo2 +12.0604* tempo -1.1474
100%	-0.0001* tempo6 +0.0037* tempo5 -0.0606* tempo4 +0.5889* tempo3-3.7066* tempo2 +14.6069* tempo -1.3675

Tabela 14: Funções da distância percorrida pela bola no plano XY em função da potência de chuto e do tempo decorrido desde o mesmo (para tempo < 7.5 s) para 0º

Para os valores intermédios de potência é feita uma interpolação linear entre os dois valores mais próximos da potência desejada.

Para descrever a variação da distância em função da potência no ar foram realizadas várias experiências com ângulo 25º e variando a potência da mesma forma que nas experiências realizadas para o ângulo 0º - Figura 16. Dos resultados obteve-se uma função normalizada que descreve a dependência do chuto apenas da potência.

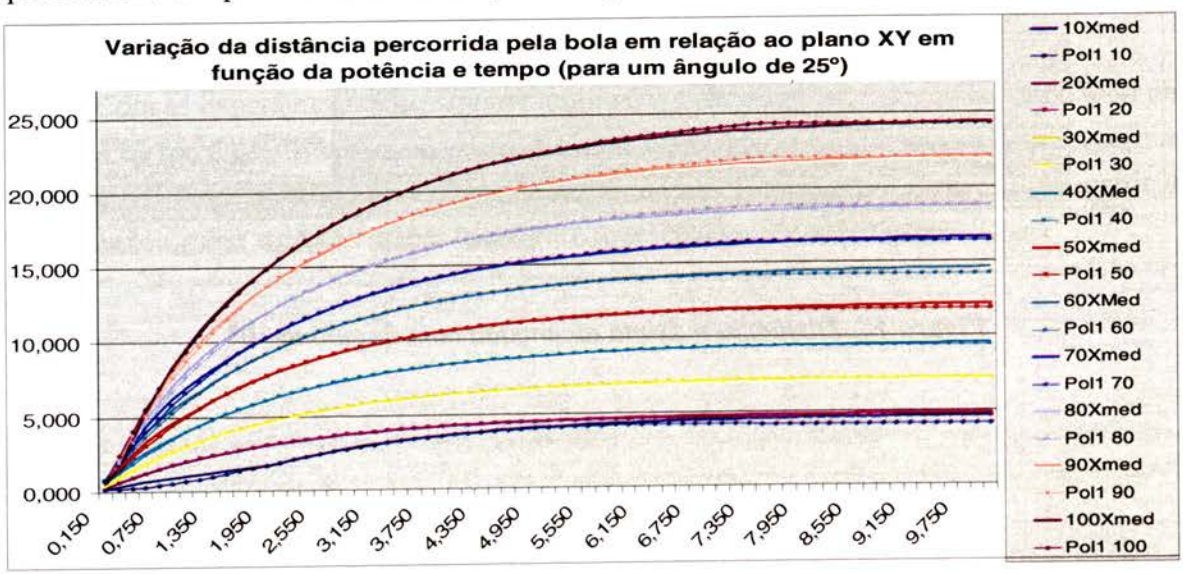

Figura 16: Variação da distancia percorrida em função da potencia e ângulo de 25º

Como esperado a variação da distância em função da potência é aproximadamente linear – Figura 17, sendo os desvios causados provavelmente pelos vários erros introduzidos pelo servidor.

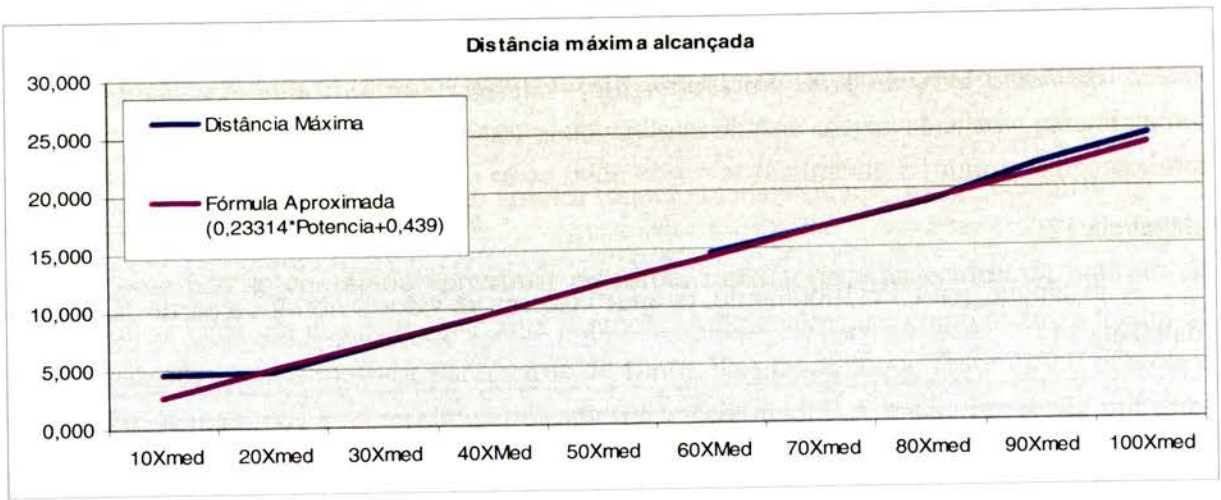

Figura 17: Distância máxima alcançada

Obteve-se assim que a variação da potência normalizada para um ângulo maior que 0º era dado por:

$$dist1 \times \frac{0.23314 \times \text{potência} + 0.439}{0.23314 \times 100 + 0.439}$$

Que é por sua vez aproximadamente:

$$dist1 \times \frac{\text{potência}}{100},$$

em que *dist1* é a distância calculada em função do ângulo para o tempo desejado e para a potência 100%. Percebe-se assim que a variação com a potência é dada directamente pela multiplicação pela potência desejada normalizada.

Com os resultados que foram obtidos pode-se então escrever a função (pseudo-código) – Tabela 15 - que devolve a distância percorrida pela bola em função do tempo e do ângulo e potência de chuto:

```
float Kick::distancia(float potência,float ângulo,float tempo){
      ângulo = múltiplo de 5 mais próximo de ângulo;
      SE (tempo >= 7.5) ENTÃO {
dist1 = distância_máxima_para_Pot100(ângulo)
} SENÃO {
   SE (ang == 0) ENTÃO {
RETORNA distância_para_Ang0(potência, tempo)
   } SENÃO {
dist1 = distância_ para_Pot100(ângulo, tempo)
         }
      }
      RETORNA dist1*potencia/100
}
```

Tabela 15: Função Kick::distância

em que o valor de:

distância_máxima_para_Pot100(ângulo) retorna os valores máximos obtidos experimentalmente para cada ângulo testado com potência 100;

distância_ para_Ang0(potência, tempo) retorna os valores obtidos a partir das funções da tabela 12;

distância_ para_Pot100(ângulo, tempo)) retorna os valores obtidos a partir das funções da tabela 11.

5.1.1.2 Função CalcPot – Cálculo da Potência

Para calcular a potência de chuto sabendo a distância máxima a ser atingida pela bola e o ângulo em que a bola vai ser chutada faz-se uma pesquisa utilizando a função *Kick::Distancia*. Se a distância retornada por essa função for igual ou superior à distância desejada então retorna-se essa potência. O tempo passado à função *Kick::Distância* vai ser 7.5 segundos pois é o tempo a partir do qual se considera a bola parada. Se não for encontrada solução chuta-se à potência máxima.

```
float Kick::CalcPot(float distância, float ângulo){
      SE (distância > 26.75) ENTÃO RETORNA Potência_Máxima;
      ENQUANTO (potencia <= 90){
            SE distancia(potencia, ângulo, 7.5) >= distância) ENTÃO
RETORNA potencia
            INCREMENTA potencia em 5%
      }
      RETORNA Potência_Máxima
}
```

Tabela 16: Função Kick::CalcPot

5.1.1.3 Função Distância Inversa

É também possível fazer a operação inversa, ou seja, passando os parâmetros de chuto e a distância a que a bola vai estar, obter o tempo que a bola demora a chegar a esse ponto. Para isto é feita uma pesquisa utilizando a função *Kick::distancia*. As distâncias retornadas por esta função vão sendo comparadas com a distância desejada: se a distância devolvida for menor que a distância desejada então tem-se o tempo que a bola leva a atingir essa distância, senão decrementa-se o tempo e chama-se de novo a função *Kick::distancia* com os novos parâmetros até se obter o tempo.

```
float Kick::distancia_inversa(float potência, float ângulo, float dis-
tância){
 tmax = 7.5
 ENQUANTO (distancia(potência, ângulo, tmax) > distância) tmax =- 0.2;
 RETORNA tmax
}
```

Tabela 17: Função Kick::distancia_inversa

As funções seguintes descrevem o movimento da bola no ar - Figura 18 - de forma aproximada. Como não se conseguiu aproximar de forma satisfatória a trajectória da bola no

ar, aproximou-se cada um dos saltos por uma parábola. Adicionalmente, como se fez a média de várias pontapés com os mesmos parâmetros de chuto, isso levou a que fosse difícil descobrir onde eram os máximos e os mínimos de cada trajectória média. A média provocou um arredondamento das curvas, pelo que só se pode observar facilmente o mínimo e os máximos dos saltos da bola de maior amplitude.

Como não se conseguiu aproximar de forma satisfatória a trajectória da bola no ar, aproximou-se cada um dos saltos por uma parábola. Adicionalmente, como se fez a média de várias pontapés com os mesmos parâmetros de chuto, isso levou a que fosse difícil descobrir onde eram os máximos e os mínimos de cada trajectória média. A média provocou um arredondamento das curvas, pelo que só se pode observar facilmente o mínimo e os máximos dos saltos da bola de maior amplitude.

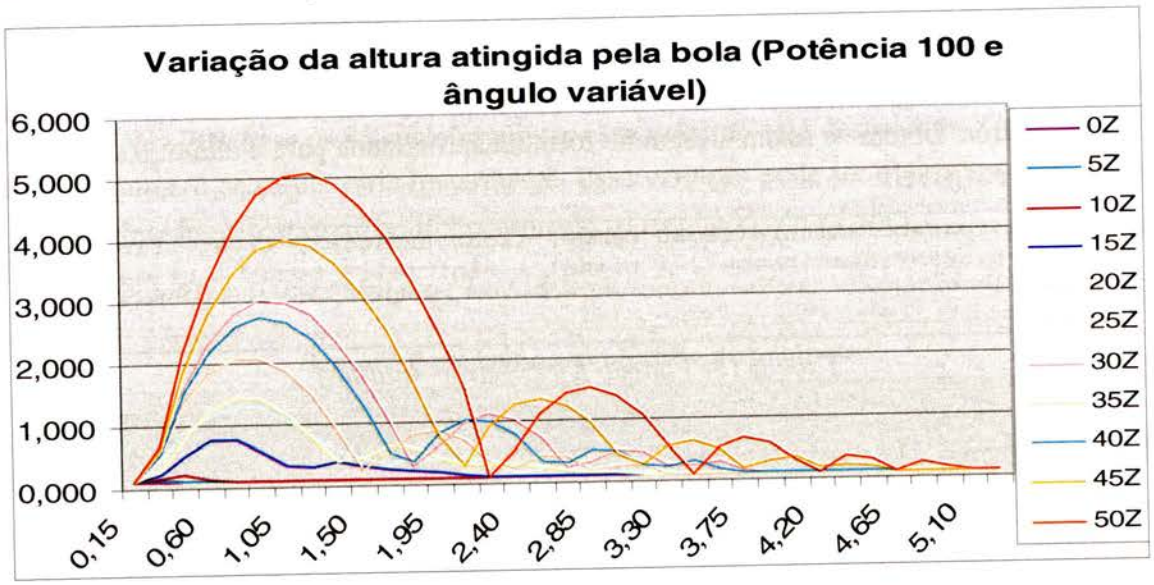

Figura 18: Variação da altura atingida pela bola

5.1.1.4 Função Parábola

Para calcular a altura a que se encontra a bola em qualquer momento utiliza-se esta função que recebe como parâmetros, para além do ângulo e potência de chuto, também o tempo. A trajectória da bola é aproximada pelas fórmulas que descrevem a trajectória parabólica descrita por um projéctil. Uma parábola que tem como vértice (V_1; V_2) e passa pelo ponto (P_1; P_2) tem a seguinte função:

```
h = (P2- V2)/( P1- V1)^2 * (t- V1)^2 + V2
```

`(V1; V2)` é o vértice da parábola

`(P1; P2)` é um ponto pelo qual passa a parábola

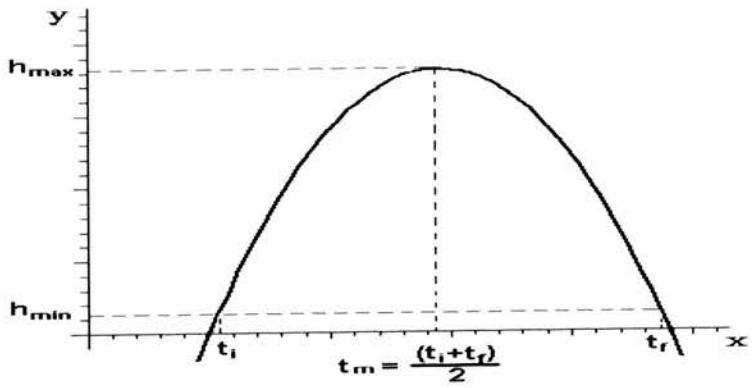

Figura 19: A trajectória parabólica descrita por um projéctil

Devido a não se estar a considerar nem o atrito nem a perda de energia cinética da bola em cada impacto com o chão, esta função apresenta valores que excedem em média os valores obtidos experimentalmente em 25% destes últimos. Atenuando os resultados obtidos multiplicando-os por 0,8 (que é o mesmo que 1/125%), o erro médio diminui de 37,2 centímetros para 21,7 centímetros. Obtém-se assim a seguinte fórmula aproximada para a altura: *0.089-7.848*(t2- (ti + tf) * t + tf * ti)*

```
float Kick::parabolaaltura(float tempo, float tinicial, float tfinal){
  RETORNA 0.089-7.848*(tempo^2-( tfinal + tinicial)* tempo + tfinal *
tinicial);}
}
```

Tabela 18: Função Kick::parabolaaltura

5.1.1.5 Função NumeroSaltos

Esta função retorna o número de saltos efectuados pela bola sabendo a potência e ângulo de chuto. Considerou-se que a bola efectuava um salto quando ultrapassava aproximadamente 75% da altura dos jogadores, ou seja, 0.30 m. A partir da variação da potência com ângulo fixo a 25º obteve-se a variação aproximada do número de saltos em função da potência e variando o ângulo mantendo a potência a 100% obteve-se a variação com o ângulo.

	1º	2º	3º
10			
20			
30			
40	0,348		
50	0,477		
60	0,555	0,330	
70	0,929	0,476	
80	1,501	0,667	0,364
90	1,135	0,537	0,329
100	1,456	0,632	0,307

Tabela 19: Alturas dos máximos atingidos em função da potência

	1º	2º	3º	4º
0				
5				
10				
15	0,780	0,392		
20	1,299	0,442		
25	1,456	0,632	0,307	
30	3,030	1,132	0,529	0,318
35	2,099	0,886	0,463	
40	2,739	1,034	0,522	0,319
45	3,998	1,383	0,650	0,357
50	5,084	1,542	0,687	0,379

Tabela 20: Alturas dos máximos atingidos em função do ângulo

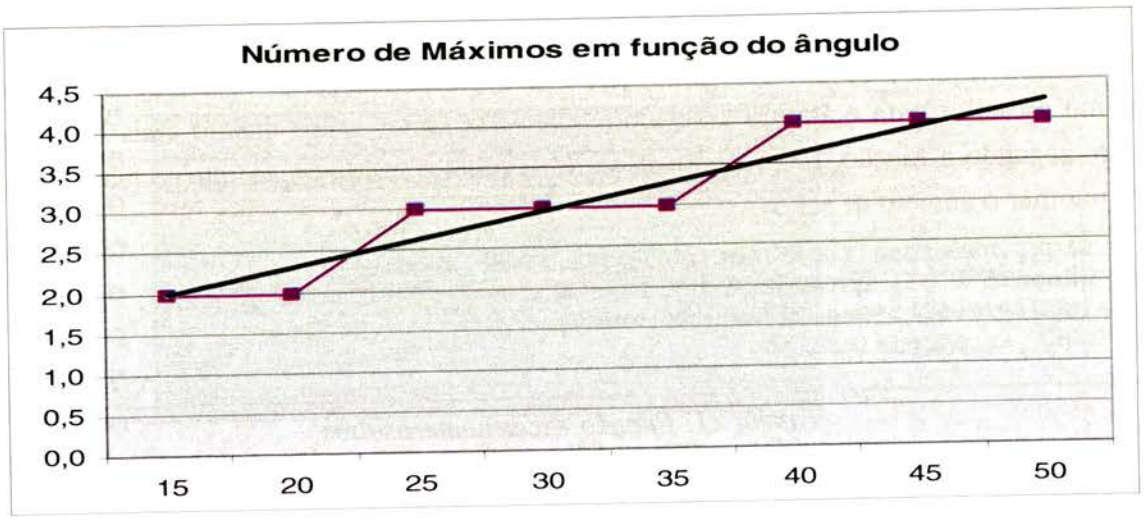

Figura 20: Número de Máximos em função do ângulo

Na tabela com os máximos em função do ângulo – Tabela 20 -, pode ver-se que há um 4º máximo para 30º. Para os cálculos seguintes esse máximo será desprezado. Com esta aproximação o número de saltos varia linearmente. Essa variação pode ser obtida aproximadamente pela seguinte função aproximando linearmente os resultados da tabela:

*F1(ângulo) = 0,0643*ângulo+1,0358*

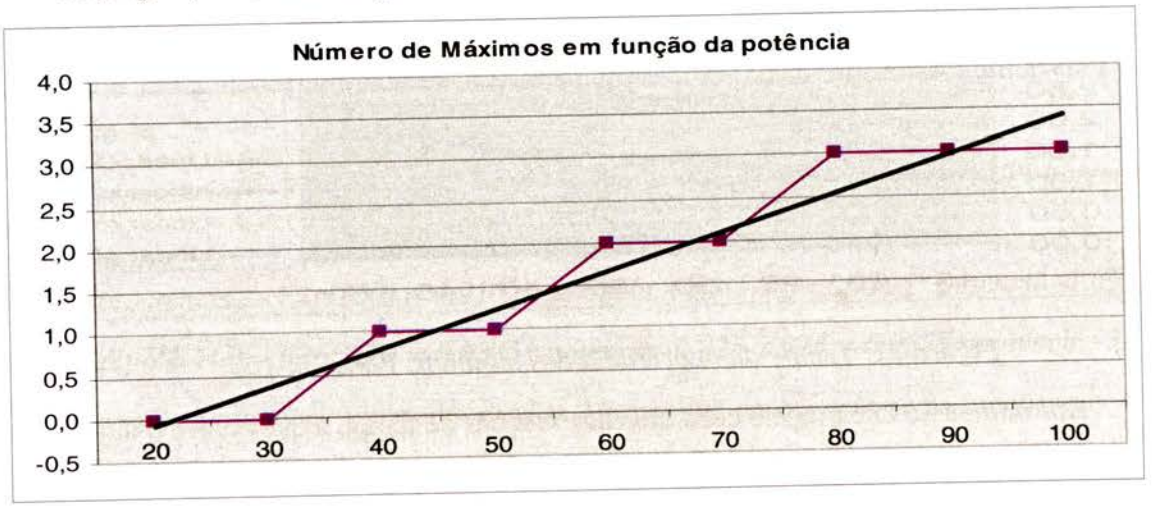

Figura 21: Número de Máximos em função da potência

A variação do número de saltos com a potência é também aproximadamente linear e tem como função:

*F2(potência) = 0,0433*potencia-1,33*

Subtraindo-lhe 3 (número máximo de saltos) obtém-se a função:

*F2(potência) = 0,0433*potencia-4,33*

Somando as duas funções, F1 e F2, tem-se:

*F(ângulo, potência) = 0.0643*ângulo + 0.0433*potência-3.2942*

Deste modo obtém-se a função descritiva da dependência do número de saltos da potência e ângulo de chuto. Ou seja, a partir da primeira função, *F1*, obtém-se o número máximo de saltos para o ângulo desejado e subtrai-se tantos saltos quanto mais baixo for o ângulo segundo a função, *F2*. Arredonda-se de seguida o resultado ao inteiro mais próximo para retornar o número de saltos.

```
int Kick::numerosaltos(float potência,float ângulo){
 SE (ângulo < 15) ENTÃO RETORNA 0;
 nsaltos = 0.0643*ang+0.0433*pot-3.2942;
 RETORNA arredonda(nsaltos);
}
```

Tabela 21: Função Kick::numerosaltos

5.1.1.6 Função tminimos – Tempo dos mínimos

Os tempos em que ocorrem os mínimos podem ser obtidos utilizando esta função. Esta função recebe como parâmetros a potência e ângulo de chuto e também a ordem do mínimo. Nesta função aproximou-se linearmente o tempo a que ocorrem os mínimos da mesma ordem.

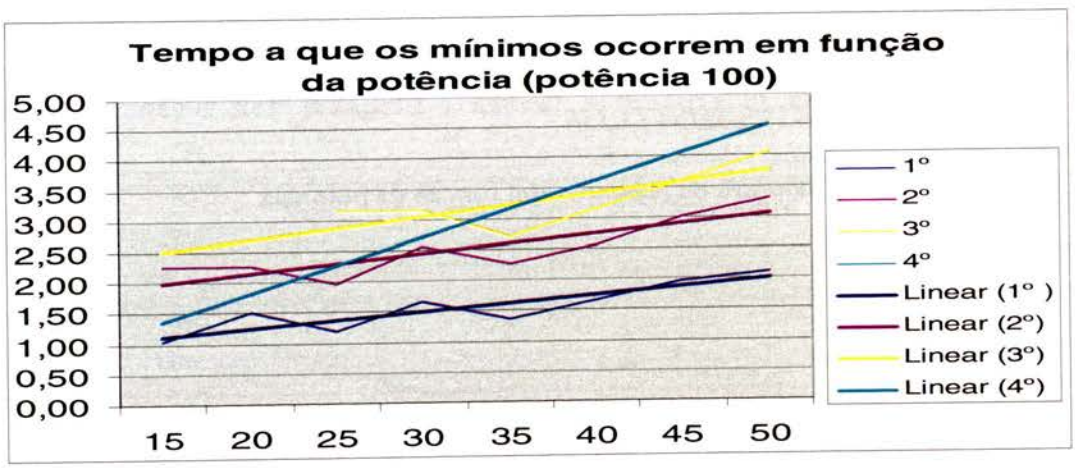

Figura 22: Tempo da ocorrência dos mínimos. Potencia 100

Aproximou-se linearmente cada uma das funções do tempo a que ocorre o mínimo.

Para os 1°s mínimos obteve-se a função:

$F1a(\text{ângulo}) = 0{,}0254 * \text{angulo} + 0{,}7321$

$F2a(\text{ângulo}) = 0{,}03 * \text{angulo} + 1{,}5375$, *para os 2°s*

$F3a(\text{ângulo}) = 0{,}036 * \text{angulo} + 1{,}95$, *para os 3°s*

e finalmente para os 4°s a função interpoladora é:

$F4a(\text{ângulo}) = 0{,}09 * \text{angulo}$

Estas funções apresentam os tempos máximos a que os mínimos podem ocorrer e dependem apenas do ângulo.

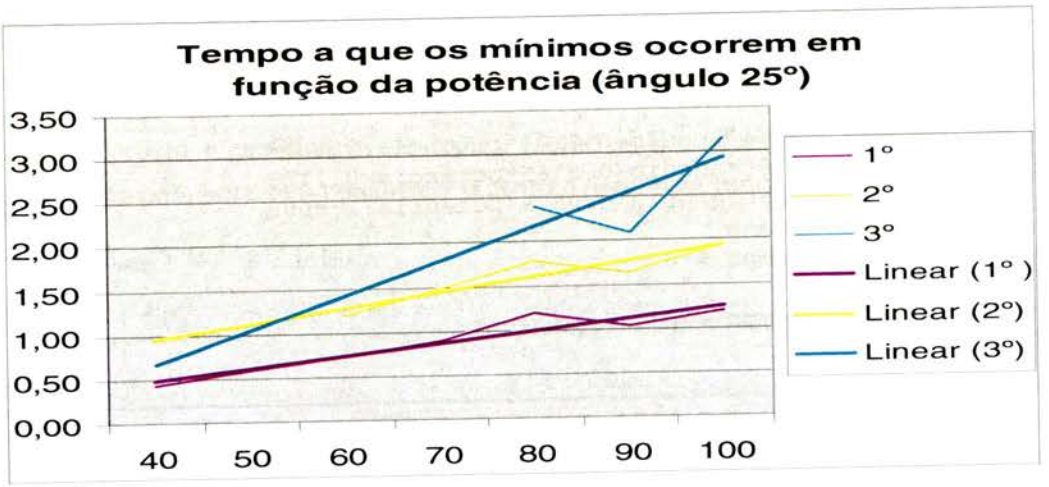

Figura 23: Tempo da ocorrência dos mínimos em função da potência. Ângulo 25

Para também haver variação com a potência, conseguiu-se através da interpolação linear dos valores dos tempos dos mínimos obtidos variando só a potência e para um ângulo constante e igual a 25°, as seguintes funções:

F1p(potência) = 0,0129 potencia -0,0215 para o 1º mínimo em função da potência*

F2p(potência) = 0,0165 potencia +0,3 idem para o 2º mínimo*

F3p(potência) = 0,0375 potencia -0,825 e para o terceiro*

Substituindo nas funções anteriores o seu valor para a potência 100 e subtraindo-lhes esse valor obtêm-se:

F1p(potência) - F1p(100) = 0,0129 potencia -0,0215-1.2685 = 0,0129* potencia - 1.29*

F2p(potência) – F2p(100) = 0,0165 potencia +0,3-1.95 = 0,0165* potencia -1.65*

F3p(potência) – F3p(100) = 0,0375 potencia -0,825-2.925 = 0,0375* potencia -3.75*

Somando as funções anteriores com as 3 primeiras conseguem-se as funções que descrevem a variação do tempo a que ocorrem os mínimos em função da potência e ângulo:

*F1(ângulo, potência) = 0.0254*angulo+0.0129*potencia-0.5579 para os tempos dos 1ºs mínimos*

*F2(ângulo, potência) = 0.0300*angulo+0.0165*potencia-0.1125 para os 2ºs mínimos*

*F3(ângulo, potência) = 0.0360*angulo+0.0375*potencia-1.8000 para os 3ºs mínimos*

F4(ângulo, potência) = 0,09 angulo para os 4ºs.*

Devido à dificuldade de separar um salto de outro, provocado por se ter feito a média de vários chuto, esta função tem um erro médio associado de 0.3 segundos ou seja dois ciclos. Este erro verifica-se sobretudo nos saltos de menor amplitude, ou seja, para os mínimos de maior ordem.

```
float Kick::tminimos(float potência, float ângulo,int ordem){
 CASO ordem SEJA
      1: tempo = 0.0254*angulo+0.0129*potencia-0.5579
      2: tempo = 0.0300*angulo+0.0165*potencia-0.1125
      3: tempo = 0.0360*angulo+0.0375*potencia-1.8000
      4: tempo =0,09*ang
      outro valor: tempo = 0
 }
 SE tempo <0 ENTÃO tempo = 0
 RETORNA tempo
}
```

Tabela 22: Função Kick::tminimos

5.1.1.7 Função tmaximos – Tempo dos máximos

Esta função devolve o tempo a que ocorre os máximos de ordem n da trajectória da bola, sabendo o ângulo e potência de chuto. Esse tempo é obtido através da média dos tempos a que ocorrem os mínimos anterior e seguinte. O erro cometido nesta aproximação é muito reduzido na maior parte dos casos, sendo que só para os saltos de menor amplitude é que se começam a verificar erros, pois é aí que ocorrem também os maiores erros de tminimos, função utilizada para o cálculo do tempo a que ocorrem os mínimos.

```
float Kick::tmaximos(float potencia, float angulo,int ordem){
 RETORNA (tminimos(potencia, angulo, ordem)+ (potencia, angulo,
ordem))/2
}
```

Tabela 23: Função Kick::tmaximos

5.1.1.8 Funções dmaximos e dminimos – Cálculo da distância

Pode calcular-se o tempo a que ocorre um máximo ou mínimo utilizando as funções tmaximos e tminimos respectivamente, substituindo esse tempo na função distância. Deste modo, é obtida a distância no plano XY (chão) a que ocorre esse máximo ou mínimo em relação ao ponto de chuto. Estas funções precisam por isso de receber como parâmetros o ângulo e potência de chuto, assim como a ordem do máximo ou do mínimo a calcular.

```
float Kick::dmaximos(float potencia,float angulo,int ordem) {
 tempo = tmaximos(potencia, angulo, ordem)
 RETORNA distancia(potencia, angulo, tempo)
}
```

Tabela 24: Função Kick::dmaximos

```
float Kick::dminimos(float potencia,float angulo,int ordem) {
 tempo = tminimos (potencia, angulo, ordem)
 RETORNA distancia(potencia, angulo, tempo)
}
```

Tabela 25: Função Kick::dminimos

5.1.1.9 Função maximos – Máximos de ordem *n*

O valor do máximo de ordem n pode ser obtido utilizando esta função que recebe como argumentos a ordem e parâmetros de chuto. Como já foi calculado anteriormente a altura máxima atingida pela bola sem considerar o atrito é dada pela função:

$$h_{max} = h_{min} + 0.5*g*(t_f-t_m)^2$$

em que $t_m = (t_i + t_f)/2$

Daqui se tem que:

$$h_{max} = h_{min} + 0.125*g*(t_f-t_i)^2$$

Para diminuir o erro cometido por não se considerar o atrito tem que se atenuar o valor obtido multiplicando-o por 0.8. [2] A fórmula obtida é então:

$$h_{max} = 0.0888 + 0.981*(t_f-t_i)^2$$

```
float Kick::maximos(float pot, float ang, int n){
 tmin1 = tminimos(pot, ang, n-1);
 tmin2 = tminimos(pot, ang, n);
 SE (tmin1 == tmin2) RETORNA 0.111; //A bola está pousada
 diferenca = tmin2-tmin1;
 RETORNA 0.0888+0.981*(tmin2-tmin1)2;
}
```

Tabela 26: Função Kick::minimos

5.1.1.10 Função alturat – Cálculo da altura

É necessária uma função que dê a altura da bola em função do tempo e potência e ângulo de chuto. Esta função retorna a altura calculando e passando à função anterior `Kick::parabolaaltura` os parâmetros referentes aos tempos em que a bola se encontra à altura mínima. Para isso calcula o número de saltos que a bola dá para saber quantos mínimos há e calcula os tempos a que esses mínimos ocorrem. De seguida escolhe o mínimo anterior e o seguinte ao tempo passado à função `Kick::alturat`.

```
float Kick::alturat(float potencia, float angulo, float tempo) {
 nsaltos = numerosaltos(potencia, angulo);
 tmin1=0.0
 tmin2=0.0
 i = 1
 ENQUANTO i FOR <= nsaltos FAZ {
   tmin2 = tminimos(potencia, angulo, i); //Calculo do tempo a que ocor-
re o mínimo de ordem i
   SE (tmin2 > tempo) {
      Sai do ciclo
   } SENÃO tmin1 = tmin2;
 }
 SE (tmin1 == tmin2) RETORNA 0.111; //A bola está pousada
 RETORNA parabolaaltura(tempo, tmin1, tmin2);
}
```

[2] Esta valor para a atenuação é justificado na explicação da função `Kick::parabolaaltura`

Tabela 27: Função Kick::alturat

5.1.1.11 Função calculate

Finalmente, é necessário uma função que calcule a potência (*KickPower*) e ângulo (*KickAngle*) para fazer um passe para um determinado ponto ou a melhor maneira de rematar.

Nesta função é definido um cone com uma abertura de 30º – Figura 24 – que tem como início a posição da bola e direcção do objectivo.

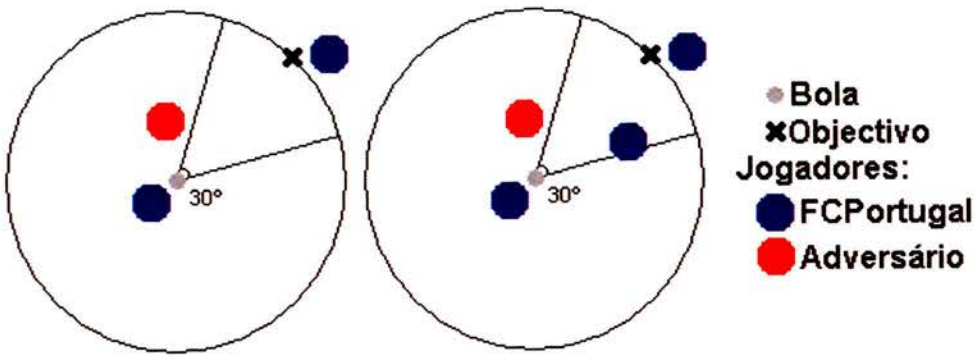

Figura 24: Evitar jogadores dentro do cone

Caso a acção a efectuar pelo agente seja um passe e se dentro deste cone não estiverem jogadores, a bola é passada rasteira (ângulo 0º) e potência necessária para atingir esse ponto (calculada pela função `Kick::CalcPot`), caso sejam encontrados jogadores no cone (da equipa FCPortugal ou adversários) a bola é chutada com o ângulo mais baixo necessário para ultrapassar todos os jogadores dentro dessa área também com a potência necessária para parar no objectivo.

Se o agente tiver decidido rematar poderá fazê-lo de três maneiras: remate rasteiro e à potência máxima, fazer um chapéu aos jogadores entre ele e a baliza ou ainda um remate colocado para que a bola entre na baliza o mais próxima da trave. O agente remata pelo chão sempre que não ajam jogadores no cone. Quando o guarda-redes estiver afastado da baliza o agente tentará fazer-lhe um chapéu se estiver dentro da grande área (se estiver mais longe a potência actualmente utilizada não chega para a bola chegar à baliza se chutada com ângulo). Poderá ainda tentar rematar com a potência máxima, tentando colocar a bola abaixo da trave se o guarda-redes estiver muito próximo da linha.

Para calcular a direcção de remate o agente calcula o interesse e segurança para várias linhas de remate. Se a linha passar pela baliza o interesse é elevado e se passar pela linha final fora da baliza o interesse é nulo. A segurança é tanto maior quanto maior a distância ao guarda-redes. O agente tenta então maximizar a distância da trajectória da bola ao guarda-redes assegurando que tem uma boa probabilidade de acertar na baliza.

```
void Kick::Calculate()
{
   aberturacone=30.0;
   numero_jogadores_cone = JogadoresCone(&Jogadores_Cone, ball-
>positionCalc, goal, aberturacone, true)
   SE passe ENTÃO SE numero_jogadores_cone  > 0 ENTÃO {
      kickAngle=0.0;
      kickPower=CalcPot(distToGoal,kickAngle);
   } SENÃO {
      kickAngle=30.0;
      kickPower=CalcPot(distToGoal,kickAngle);
   }
   SE remate ENTÃO {
      SE numero_jogadores_cone  == 0 ENTÃO {
        kickAngle=0.0;
        kickPower=100.0;
      } SENÃO SE guarda-redes afastado da baliza ENTÃO chapéu
      SENÃO remate ao ângulo
   }
}
```

Tabela 28: Função Kick::Calculate

Capítulo 6

6. Comunicação

6.1 Considerações iniciais

A partir da versão 0.5 do servidor os jogadores passaram a ter visão limitada a 180º. Assim, deixaram de ter conhecimento total do estado do mundo. Para compensar esta perda foram introduzidos um novo effector (Say) e um novo perceptor (Hear). Os agentes são agora capazes de transmitir e receber mensagens uns dos outros (indirectamente através do servidor). Neste momento, não há qualquer ruído associado à comunicação. Os únicos limites impostos são o tamanho da mensagem e o número de mensagens que um agente pode receber.

6.2 Comunicação Agente/Servidor

6.2.1 Say Effector

Como dito anteriormente, para transmitir mensagens aos outros agentes é necessário utilizar o Say Effector. As mensagens podem ter até 512 caracteres válidos[3]. As mensagens podem ser ouvidas pelas duas equipas até uma distância máxima definida em audioCutDist. Este effector pode ser enviado todos os ciclos e o seu formato é simplesmente:

`A(say <messagem>)`

Tabela 29: Say Effector

[3] No manual do servidor lê-se que os caracteres permitidos têm que estar na gama 0x20 a 0x7E de ASCII e não podem incluir espaço e parentesis. Veio a descobrir-se que há outros caracteres que não podem ser utilizados durante os testes à comunicação

6.2.2 Hear Perceptor

Este *perceptor* é recebido sempre que algum dos agentes envia uma mensagem utilizando o *Say Effector* e está a menos de *audioCutDist* metros do agente que enviou a mensagem e que tem a sua capacidade de audição à pelo menos *hearDecay*. As mensagens têm também um tempo de propagação que varia tipicamente entre os 25 e 35 ciclos de servidor.

O *perceptor* recebido tem o seguinte formato:

```
(hear <tempo> <direcção em graus> <messagem>)

<tempo> indica o tempo em que a mensagem foi recebida

<direcção em graus> indica a direcção relativa do emissor em graus ou self
se a mensagem recebida tiver sido enviada pelo próprio agente

<messagem> a mensagem transmitida
```

Tabela 30: Hear Perceptor

Os parâmetros do servidor que afectam este perceptor podem ser vistos na tabela 4.

Um jogador só pode ouvir a mensagem se a sua capacidade de audição é pelo menos *hearDecay* visto que cada vez que é ouvida uma mensagem é subtraído este valor à sua capacidade de audição. A cada ciclo[4] esta capacidade é incrementada por *hearInc*. A capacidade máxima de audição está definida por *hearMax*. Com os valores actuais, cada agente só pode ouvir uma mensagem de dois em dois ciclos[3].

Como referimos anteriormente, de modo a evitar que uma equipa sature as comunicações da outra enviando mensagens todos os ciclos, cada equipa tem canais diferentes com capacidades de audição associadas a cada um destes canais. Deste modo, a cada dois ciclos, cada agente pode ouvir uma mensagem de um adversário e de um colega de equipa (pode ainda ouvir uma transmissão efectuada por ele próprio).

Se mais do que uma mensagem for recebida simultaneamente, na implementação actual do servidor, a mensagem ouvida é a mais antiga dessas mensagens ou seja a primeira recebida pelo servidor.

O alcance da mensagem como já foi dito é de 50 metros (definido em *audioCutDist*) e, por isso, um defesa próximo da sua baliza receberá uma mensagem do seu guarda-redes enquanto um avançado no meio campo adversário poderá estar afastado demais para a ouvir.

---

[4] Este ciclo é um ciclo de percepção equivalente a 20 ciclos do servidor, ou seja, 150 ms de tempo real de jogo. O servidor envia uma mensagem de percepção visual e auditiva (caso tenha sido transmitida uma mensagem) todos os 20 ciclos.

6.3 Implementação

Cada agente possui um modelo do estado do mundo que descreve o estado corrente do seu ambiente. Cada um dos objectos deste mundo tem uma confiança associada. Se o objecto (agente ou bola) tiver sido visto no último ciclo de percepção a confiança está a um (100%). Caso o objecto não tenha sido visto, a confiança é decrementada multiplicando a confiança anterior por 0.9.

Do mesmo modo, existe outro estado do mundo que descreve o que foi transmitido pelos agentes da equipa. As confianças deste mundo são actualizadas por ciclo do servidor em vez de ciclo de percepção, para se ter um valor de confiança mais exacto, pois o tempo de propagação da mensagem não é constante apesar de ser sempre recebido no ciclo de percepção. A confiança nos dados da mensagem é então 0.994 elevado ao número de ciclos que a mensagem demorou a ser ouvida. Ao fim de 20 ciclos o valor para a confiança é ligeiramente inferior a 0.9 (0.89) à semelhança da confiança do mundo visto.

Cada agente possui também uma tabela com as acções que os outros agentes da equipa tencionam realizar (passe, drible, remate, etc) e respectiva direcção, também com uma confiança associada. Estas acções são actualizadas quando um colega de equipa comunica. À semelhança do estado do mundo a confiança em cada acção é decrementada 10% a cada ciclo de percepção.

De seguida, descrevem-se as funções utilizadas pelo módulo de comunicação do agente.

6.3.1 Função Init_ say_message_allowed

Esta função cria uma lista com os caracteres do dicionário que irá ser utilizado na codificação das mensagens. Segundo o manual do simulador 3D, são válidos todos os caracteres entre 0x20 e 0x7E, exceptuando o espaço e os parêntesis curvos. Destes 91 caracteres são utilizados neste momento apenas 86[5] devido a incompatibilidades com a versão do simulador utilizada. Os caracteres que não podem ser utilizados são, para além do espaço e parêntesis curvos, a vírgula, o ponto e vírgula, aspas, barra invertida (*backslash*) e pelica.

Para além disso é também criada uma lista para a descodificação, utilizando o valor dos caracteres (os caracteres equivalem a um número inteiro) como índice.

5 Este valor é guardado na variável NALLOWED

```
Communic::Init_say_message_allowed_array()
{// OS caracteres permitidos são de 0x20 a 0x7E excepto " " e "()"
      //Inicialização da lista de codificação allowed[]
        ch = 0x20;
        i = 0;
        ENQUANTO (ch <= 0x7e) {
            if(ch!=' ' && ch!='\\'
               && ch!='"'
               && ch!=','
               && ch!=';'
               && ch!=0x27     //Single quote
               && ch!='('
               && ch!=')')
            { allowed[i]=ch;
              i++;
            }
            ch++;
        }
      //Inicialização da lista de descodificação Chardecode[]
       for(i=0;i<256;i++) Chardecode[i]=255;
       for(i=0;i<=NALLOWED;i++) Chardecode[(int)allowed[i]]=i;
}
```

Tabela 31: Função Communic::Init_say_message_allowed_array

6.3.2 Função decideCommunication

Apesar de um agente poder transmitir uma mensagem em todos os ciclos, apenas consegue receber uma transmissão da sua equipa a cada 40 ciclos (pode ainda ouvir-se também a ele próprio e a um dos adversários uma vez em cada 40 ciclos). Caso dois agentes da mesma equipa transmitam no mesmo intervalo de 40 ciclos, é apenas recebida a mensagem que chegou em primeiro lugar ao servidor. Assim, como o meio de transmissão está limitado, é importante que o agente seja capaz de decidir se deve ou não transmitir o seu conhecimento do mundo sem se sobrepor aos outros agentes da sua equipa. Como cada agente pode ligar-se ao servidor em ciclos ligeiramente diferentes, os ciclos em que as mensagens são recebidas é também diferente. Torna-se por isso necessário sincronizá-los, dividindo por 20 o ciclo em que o agente recebe as mensagens, obtendo assim o ciclo de transmissão. É na função `decideCommunication` que é decidido se o agente transmite a mensagem ou não.

O agente que está mais próximo da bola transmite sempre a cada dois ciclos de transmissão. Deste modo, é garantido que todos os outros agentes o ouvem. Os outros ciclos de transmissão são divididos entre os jogadores que estão a menos de 15 metros de distância da bola e que a viram.

Está também implementado uma *flag* que permite a comunicação independente de ser a vez do agente comunicar ou não, para o caso de um agente ter algo importante para comunicar, mas actualmente não é utilizada em nenhuma função.

```
Communic::decideCommunication(){
      ciclo = Ciclo_do_agente / 20  //Sincroniza os ciclos dos agentes
      Decrementa confiança nos objectivos dos jogadores
      ciclo = (ciclo + (ciclo % 2))/2;          //Considera os ciclos pares
e impares o mesmo ciclo
      SE (agente é o mais proximo da bola) e (ciclo % 2 == 1) ENTÃO Comu-
nica
      SE (
viu a bola &&
distância à bola < 15 &&
ciclo % 2 == 0 &&
ciclo%(2*numero de jogadores a menos de 15 metros da bola))/2 == numero do
jogador)
ENTÃO Comunica
SE Prioritario ENTÃO Comunica
Não comunica nos outros casos
}
```

Tabela 32: Função Communic::decideCommunication

6.3.3 Funções add_infoChar e add_info

São funções auxiliares que adicionam ao fim da mensagem, respectivamente, um caracter ou um conjunto de caracteres. Recebem como argumentos o apontador para a mensagem e o conjunto de caracteres a adicionar.

```
Communic::add_info(Mensagem, Grupo de Caracteres a adicionar)
{
      Mensagem = Mensagem + Grupo de Caracteres a adicionar
}

void Communic::add_infoChar(Mensagem, Caracter a adicionar)
{
      Mensagem = Mensagem + Caracter a adicionar
}
```

Tabela 33: Funções add_infoChar e Communic::add_info

6.3.4 Função encodeChar

Codifica um inteiro com um valor entre 0 e o tamanho do dicionário, utilizando para isso a lista *allowed*. O número é codificado directamente utilizando o seu valor como índice desta lista. Caso o número a codificar seja um número negativo ou maior que o tamanho do dicionário é devolvido o último caracter do dicionário e escrita uma mensagem de erro no *log* de erros.

```
Communic::encodeChar(Inteiro){
      SE (Inteiro < 0) ou (Inteiro > NALLOWED) ENTÃO {
            Escreve erro nos logs
DEVOLVE allowed[NALLOWED]
      }
      DEVOLVE allowed[Inteiro]
}
```

Tabela 34: Função Communic:encodeChar

6.3.5 Função decodeChar

Faz a operação inversa de `encodeChar`, ou seja, a partir de um caracter codificado devolve o inteiro correspondente. Neste caso é o valor do caracter que é utilizado como índice na lista de descodificação `Chardecode`. Caso o número obtido na descodificação seja menor que 0 ou menor que NALLOWED, é escrita uma mensagem de erro no *log* de erros é devolvido –1.

```
Communic::decodeChar(Caracter){
  Valor = Chardecode[Caracter];
  SE (Valor < 0) ou (Valor > NALLOWED) ENTÃO {
escreve mensagem de erro
DEVOLVE -1
  } SENÃO DEVOLVE Valor
}
```

Tabela 35: Função Communic:decodeChar

6.3.6 Funções encodeint e encodeintaux

Encodeint codifica um inteiro num determinado número de caracteres utilizando a função auxiliar `encodeintaux`. Recebe como argumentos o valor a ser codificado e o número de caracteres em que este vai ser codificado. Codifica qualquer inteiro de 0 até NALLOWED elevado ao número de caracteres utilizado. Se um número for superior a este último número é escrito um erro no *log*, codifica-se o valor máximo. Esta função vai codificando recursivamente cada um dos caracteres do menor para o maior ficando o resultado no *array* `str_conv2`.

```
Communic::encodeint(Inteiro, NumeroCaracteres) {
      SE Inteiro < 0 ENTÃO {
Inteiro = 0
Escreve erro no log
      }
      SE Inteiro > NALLOWED^NumeroCaracteres ENTÃO {
Inteiro = NALLOWED^NumeroCaracteres
Escreve erro no log
}
      encodeintaux(number, nchars)
      DEVOLVE str_conv2
}

Communic::encodeintaux(Inteiro, NumeroCaracteres) {
SE NumeroCaracteres == 2 ENTÃO add_infoChar(str_conv2, encodeChar(number %
coef))
SE NumeroCaracteres > 2 ENTÃO {
encodeint(Inteiro % NALLOWED, NumeroCaracteres-1)
      }
add_infoChar(str_conv2, encodeChar(number / coef))
}
```

Tabela 36: Funções Communic:encodeint e Communic::encodeintaux

6.3.7 Função decodeint

Esta é a função inversa de `encodeint`, ou seja, descodifica um inteiro recebendo como parâmetros os caracteres e o número de caracteres passados. Cada caracter é descodificado utilizando a função e depois multiplicando o valor obtido pelo peso desse caracter. (a primeira posição tem peso 1, a segunda tem peso NALLOWED, a terceira tem peso NALLOWED^2, etc.). Somando tudo é obtido o valor codificado que é devolvido pela função.

```
Communic::decodeint(Grupo_de_Caracteres, Numero_de_Caracteres) {
      Numero_Descodificado = 0
      Coeficiente = 1
      i = 0
      ENQUANTO (Numero_de_Caracteres > 1) {
            Numero_ Descodificado = Numero_Codificado + decode-
Char(Grupo_de_Caracteres[i])* Coeficiente
            Numero_de_Caracteres = Numero_de_Caracteres +1
            i = i+1
            Coeficiente = Coeficiente * NALLOWED;
      }
      Numero_ Descodificado = Numero_ Descodificado + decode-
Char(Grupo_de_Caracteres[i])* Coeficiente
      DEVOLVE Numero_ Descodificado
}
```

Tabela 37: Função Communic::decodeint

6.3.8 Função encodeConfidence

A confiança, que é um valor decimal que pode variar de 0 a 1 é codificada com um caracter. Na codificação o número a codificar é multiplicado por 100 e truncado de maneira a obter-se só a parte inteira. Subtrai-se de seguida MINCONF[6] de modo a desprezar todas as confianças abaixo deste valor. O valor assim obtido é então codificado com auxílio da função `encodeChar`.

```
Communic::encodeConfidence(Confiança){
      SE (Confiança < 0) ou (Confiança > 1) {
Confiança = 0
Escreve erro no log
      }
Confiança = round(Confiança *100.0) - MINCONF
      SE (Confiança < 0) ENTÃO Confiança = 0
      DEVOLVE encodeChar(Confiança);
}
```

Tabela 38: Função Communic::encodeConfidence

[6] MINCONF é a confiança mínima e está definida como (100 – ALLOWED). Qualquer valor da confiança que esteja abaixo desta diferença a dividir por 100 é codificado com o valor 0.

6.3.9 Função decodeConfidence

Esta função descodifica o caracter com que se codificou a confiança com auxílio da função `decodeChar` , soma-se MINCONF se o valor obtido for maior que 0 e divide-se o valor por 100 para voltar a ter confianças entre 0 e 1.

```
Communic::decodeConfidence(Caracter){
      Confiança  = decodeChar(Caracter);
      SE (Confiança < 0) Escreve erro no log
      SE (Confiança > 0) {
            DEVOLVE Confiança = (Confiança + MINCONF) / 100
      }
      DEVOLVE 0
}
```

Tabela 39: Função Communic::decodeConfidence

6.3.10 Funções encodecycle e decodecycle

O ciclo é um número inteiro que pode variar entre 0 e 80000[7]. Este número é codificado com 3 caracteres pelo que a gama permitida vai de 0 a NALLOWED[3] muito superior ao necessário, permitindo que a comunicação não seja alterada mesmo que no futuro o tempo de jogo seja aumentado ou que o tempo por ciclo diminua.

```
Communic::encodecycle(ciclo) {
  DEVOLVE encodeint(ciclo, 3);
}

Communic::decodecycle(string) {
  DEVOLVE decodeint(string, 3);
}
```

Tabela 40: Funções Communic::encodecycle e Communic::decodecycle

6.3.11 Função encodePositionandSpeed

A posição e velocidade de cada objecto (bola e agente) utilizam 16 caracteres. No primeiro caracter são codificados os sinais (se é positivo ou negativo) de cada componente cartesiana (x, y e z) da posição e velocidade. Nos 9 caracteres seguintes é codificado o módulo de cada componente da posição (3 por cada componente) e nos últimos 6 é codificada a velocidade (2 caracteres/componente).

---

[7] Calculado dividindo o tempo de simulação (actualmente 600 segundos) pelo tempo de cada ciclo de servidor (.0075)

Os sinais são codificados bit a bit (1 se a componente for positiva, 0 se for negativa) sendo que os bits menos significativos correspondem respectivamente às componentes da posição x, y e z e os 3 bits seguintes ao x, y e z da velocidade (do menos significativo para o mais significativo). Obtém-se assim um valor para os sinais entre 0 e 63 que é depois convertido no caracter correspondente.

O módulo de cada componente da posição é multiplicado por 1000 e arredondado ao inteiro mais próximo e codificado com `encodeint` utilizando os 3 caracteres na codificação. Deste modo é obtido um erro de menos de 1 mm devido ao arredondamento.

Para a velocidade o módulo de cada componente é multiplicado por 100 e também arredondado ao inteiro mais próximo. É então codificado com `encodeint` de 2 caracteres. É obtido assim um erro de menos de 1 cm/s, desprezável face às velocidades atingidas pela bola.

```
Communic::encodePositionandSpeed(posição, velocidade) {
      sinais = 0;
      //Posicao
      SE (posição x >= 0) sinais ++;
      add_info(str, encodeint((int)round(abs(posição x)*1000), 3));
      SE (posição y >= 0) sinais += 2;
      add_info(str, encodeint((int)round(abs(posição y)*1000), 3));
      SE (posição z >= 0) sinais += 4;
      add_info(str, encodeint((int)round(abs(posição z)*1000), 3));

      //Velocidade
      SE (velocidade x >= 0) sinais += 8;
      add_info(str, encodeint((int)round(abs(velocidade x)*100), 3));
      SE (velocidade y >= 0) sinais += 16;
      add_info(str, encodeint((int)round(abs(velocidade y)*100), 3));
      SE (velocidade z >= 0) sinais += 32;
      add_info(str, encodeint((int)round(abs(velocidade z)*100), 3));

      add_infoChar(str2, encodeChar(sinais));
      add_info(str2,str);
      DEVOLVE str2;
}
```

Tabela 41: Função Communic::encodePositionandSpeed

6.3.12 Função decodePositionandSpeed

Esta função faz a descodificação dos valores de posição e velocidade de um objecto, devolvendo por referência os valores descodificados. A descodificação é feita multiplicando o inteiro correspondente a cada componente pelo respectivo sinal e dividindo por 1000 (se for uma componente da posição) ou 100 (caso seja da velocidade).

```
Communic::decodePositionandSpeed(str, & posição, & velocidade) {
      Descodificação dos sinais de cada componente da posição e velocidade
a partir do primeiro caracter;
      // Posicao
posição x = (signposx*decodeint(do 2°, 3° e 4° caracteres), 3))/1000;
      posição y = (signposy*decodeint(dos 3 caracteres seguintes),
3))/1000; str+=3;
      posição z = (signposz*decodeint(dos 3 caracteres seguintes),
3))/1000; str+=3;
//Speed
      s.x = (signspdx*decodeint(strncpy(dos 2 caracteres seguintes),
2))/100; str+=2;
      s.y = (signspdy*decodeint(strncpy(dos 2 caracteres seguintes),
2))/100; str+=2;
      s.z = (signspdz*decodeint(strncpy(dos 2 últimos caracteres),
2))/100;
}
```

Tabela 42: Função Communic::decodePositionandSpeed

6.3.13 Funções encodegoal e decodegoal

Estas funções codificam e descodificam os objectivos dos jogadores. A cada objectivo está associado um tipo (se é passe, remate, drible, etc) que já é um caracter e por isso não precisa ser codificado e uma posição (para onde rematar ou passar). À semelhança da codificação da posição de um objecto a direcção é codificada com 10 caracteres, sendo que o primeiro codifica os sinais de cada componente e cada três seguintes o x, o y e o z dessa posição. O último caracter corresponde ao tipo de acção.

Na descodificação a posição do objectivo e o tipo de objectivo são devolvidos por referência.

```
Communic::encodegoal(Objectivo, Tipo) {
      str = Codificação dos sinais;
      str += Codificação do módulo de cada componente da posição;
      str += Tipo;
      DEVOLVE str;
}

Communic::decodegoal(str, &Objectivo, &Tipo) {
      Descodificação do sinal do objectivo;
      Descodificação de cada componente do objectivo;
      Extracção do tipo de objecto;
}
```

Tabela 43: Funções Communic::encodegoal e Communic::decodegoal

6.3.14 Função communicate

Quando o agente decide comunicar é chamada esta função. `Communicate` recebe quatro parâmetros: o tipo de mensagem (sempre a 1 para já), o objectivo e o tipo de objectivo que o agente que vai comunicar decidiu e actions que é um parâmetro utilizado pela função de envio da mensagem para o servidor.

A mensagem é constituída por um cabeçalho e pelo corpo da mensagem. O cabeçalho começa por "FCP" seguido pelo tipo de mensagem, lado do campo em que o jogador joga, o seu número e o ciclo actual codificados. Actualmente só há um tipo de mensagem (1) em que se comunicam a posição e velocidade de todos os jogadores e da bola assim como o objectivo do jogador que está a falar. Transmitindo estes dados só são gastos 411 dos 512 caracteres permitidos pelo que para já não há necessidade de compactar mais a mensagem.

O corpo da mensagem depende do tipo de mensagem e tem o seguinte formato (para já só há um tipo): Posição e tipo do objectivo do agente, posição da bola e respectiva velocidade e confiança, posição, velocidade e confiança do jogador 1 da equipa do agente, seguido pela posição, velocidade e confiança do jogador 1 da equipa adversária, do jogador 2 de cada equipa, e assim por diante até ao jogador 11 da equipa adversária.

```
Communic::Communicate(Tipo Mensagem, Objectivo, Tipo Objectivo, actions) {
      add_info(mensagem,"FCP");
      add_infoChar(mensagem, Tipo Mensagem);          //Tipo de Mensagem
      add_infoChar(mensagem, encodeChar(worldoriginal->game->side));
      add_infoChar(mensagem, encodeChar(worldoriginal->agentUnum));
      add_info(mensagem, encodecycle(worldoriginal->game->cycle));

      switch (MessageType)
      {
      CASO '1':
      add_info(mensagem, encodegoal(Objectivo, Tipo Objectivo));
      add_info(mensagem,encodePositionandSpeed(worldoriginal->ball-
>position), worldoriginal->ball->velocity));
      add_infoChar(mensagem,encodeConfidence(worldoriginal->ball->conf));

      DE i=1 ATÉ 11 {
      add_info(mensagem,encodePositionandSpeed(worldoriginal-
>GetFCPortugalPlayer(i)->position, worldoriginal->GetFCPortugalPlayer(i)-
>velocity));
      add_infoChar(mensagem,encodeConfidence(worldoriginal-> GetFCPortu-
galPlayer(i)->conf));

      add_info(mensagem,encodePositionandSpeed(worldoriginal-
>GetOpponentPlayer(i)->position, worldoriginal->GetOpponentPlayer(i)-
>velocity));
add_infoChar(mensagem,encodeConfidence(worldoriginal->GetOpponentPlayer(i)-
>conf));
      }
      Envia mensagem
   }
}
```

Tabela 44: Função Communic::Communicate

6.3.15 Função ReceiveCommunication

Quando a mensagem é recebida é chamada esta função, que recebe como argumentos o tempo em que a mensagem foi recebida e a direcção (em graus) de onde foi recebida e a mensagem. Se o valor para a direcção for -1000, então a mensagem é do próprio agente e a mensagem é descartada. A mensagem é também descartada caso as primeiras 3 letras do cabeçalho da mensagem forem diferentes de "FCP", se o valor para o lado do campo em o emissor está a jogar for diferente do campo do receptor (permitindo assim que se defrontem duas equipas FCPortugal sem utilizarem as comunicações uma da outra), se a mensagem for demasiado antiga ou ainda se tiver um tamanho menor que o esperado (caso o tipo da mensagem seja 1) ou a mensagem tenha um tipo inválido.

A idade da mensagem é calculada subtraindo o ciclo actual pelo ciclo codificado na mensagem. Os valores típicos para a idade da mensagem situam-se entre os 25 e os 35 ciclos, havendo no entanto algumas que chegam aos 55. Se esta diferença for superior a MAXAGE[8] a mensagem é descartada, pois as confianças associadas ao que foi transmitido vão ser muito reduzidas. No entanto, esta condição nunca foi verificada com os valores actuais. Depois de calculada a idade da mensagem, é calculado um factor de confiança pelo qual todas as outras confianças serão multiplicadas, para compensar o atraso da mensagem. Este factor, `conf-const`, é igual a 0.994^{idade}, sendo o seu valor mínimo 0.54que acontece quando a mensagem demora 100 ciclos a propagar-se (neste caso só há 50% de confiança global).

De seguida, faz a descodificação do corpo da mensagem consoante o tipo. Se for do tipo 1 descodifica a posição e tipo do objectivo do emissor, a posição, velocidade da bola e a confiança que o emissor tinha nestes dados e a posição, velocidade e confiança associados a cada jogador. Se os valores foram recebidos sem erros as confianças associadas a cada objecto são então multiplicados por `conffactor`, actualizando assim o valor para as confianças recebidas.

Segue-se uma mensagem típica da equipa FCPortugal e uma descodificação parcial do seu conteúdo.

```
(hear 3.38 88.2388 FCP1$/%-
!*^U!&F%@#!pVw!!J!!8#!f!=!!!}Kf8+p!!4$!I#D!!!3cwu+:!!0!!h!j!!!}O/Y#wZ#w!!
t#+!!!&edT*`F###!H#?#!!}Su&$IW!R#!D#>#!!3asS*d!!L!!O!&!!!}U_+$HT!@!!4!s#!
!3g?T*VG#$!!N!L!!!}Y5ua#$#!D!1#!!3TSh$Jz!u!!H!6!!!}bT/#v4!?!!V#!!3H1E#.1#
%!!i!U!!!}SS%!iQ$j!!t#/!!!&VOG#?1#Q#!^!_!!!}U%#Q$3$!K!8!!!}g!i$kz!G#!H!.!
!!}U_#!|$!Y$!N!&!*!^TgU!2G##$!U!7!!!}deu+/z%H$!m#U!!!&_E7$p#!=$!.#E
!!!}U+&!1F#h#!Q!=!!!}VUU!HH#5$!B!3!!!})
```

Tabela 45: Exemplo de uma mensagem do FCPortugal

8 Esta variável está actualmente inicializada com o valor 100, ou seja, 5 ciclos de percepção.

Este *hear perceptor* foi recebido aos 3.38 segundos de jogo, de um jogador posicionado a `88.2388`° em relação ao receptor. É uma mensagem de tipo 1 (4° caracter da mensagem). O emissor está a jogar do lado direito do campo ($) e é o número 9 (/). De seguida, vê-se o ciclo em que a mensagem foi enviada codificado (`%-!`) o ciclo 598. A idade da mensagem é de 41 ciclos pelo que a confiança máxima nos dados de qualquer objecto comunicado é de 78.13%. O objectivo do jogador é um passe (20° caracter). A bola está na posição (`0.0790, 0.0350, 0.1030`) aproximadamente no centro do campo (acabou de ser dado o kick-off) e a confiança nessa posição é de 78.13%, ou seja a confiança máxima, pelo que se sabe que esse jogador viu a bola nesse ciclo.

```
Communic::ReceiveCommunication(tempo da mensagem, direcção do emissor,
messagem){
      Descodifica cabeçalho;
      SE (
        direcção do emissor == -1000 || número do emissor == número do
receptor    //Mensagem enviada por este agente
   || 3 primeiros caracteres da mensagem != "FCP"
   || lado do campo do emissor != lado do campo do receptor
   || tipo inválido
   || idade > MAXAGE
) ENTÃO Descarta mensagem e sai da função
      conffactor = 0.994idade
CASO tipo de mensagem SEJA
1:
      SE comprimento da mensagem < 411 Descarta mensagem e sai
      Descodifica posição e tipo de objectivo
      Descodifica posição, velocidade e confiança da bola
      SE erro na descodificação ENTÃO confiança = 0
      SENÃO confiança = confiança*conffactor

      DE i=1 ATÉ número de jogadores {
            Descodifica posição, velocidade e confiança do jogador i da
equipa FCPortugal
            SE erro na descodificação ENTÃO confiança = 0
            SENÃO confiança = confiança*conffactor
            Descodifica posição, velocidade e confiança do jogador i da
equipa adversária
            SE erro na descodificação ENTÃO confiança = 0
            SENÃO confiança = confiança*conffactor
            i++:
      }
Outros tipos: Descarta mensagem
}
```

Tabela 46: Função Communic::ReceiveCommunication

6.3.16 Actualização do estado do mundo

Depois de uma mensagem ter sido recebida e descodificada com sucesso (se a mensagem não foi descartada) é necessário actualizar o estado do mundo e a tabela de objectivos. Com este fim, comparam-se as confianças do estado do mundo do agente com as confianças do estado do mundo comunicado, para cada objecto, e caso a confiança comunicada seja superior para um determinado objecto, é actualizado o estado do mundo para esse objecto.

Capítulo 7

7. Análise de Resultados

7.1 Europeu

No campeonato da Europa realizado em Eindhoven a equipa de simulação 3D teve o seu primeiro teste oficial sagrando-se campeã europeia nesta categoria. A equipa não sofreu golos marcando 12 golos em 11 jogos. Neste campeonato os guarda-redes tinham um *catchmargin* de 2 metros o que tornou muito difícil a marcação de mais golos, especialmente na última fase em que só as melhores equipas se encontravam apuradas. Isto provocou muitos empates a 0, incluindo na decisão pelos primeiros 4 lugares. Na decisão pelo 3º e 4º lugar e na final, ao fim do primeiro jogo todas as equipas empataram a 0. Foi necessário então diminuir o alcance do guarda-redes para 1 metro. Com esta alteração, os Rezvan ganharam por 1-0 aos UtUtd conquistando o terceiro lugar. No entanto, para a decisão do primeiro lugar o jogo acabou em empate a 0. A equipa FCPortugal ganhou o título de campeã por ter ficado em primeiro lugar no grupo de apuramento e ter derrotado os vice-campeões, os Seu3D, nesta fase.

Na tabela seguinte encontram-se os resultados obtidos neste europeu e a percentagem de tempo em que a bola esteve na área da equipa FCPortugal, no quarto de campo seguinte, na entrada do meio campo adversário e na área adversária respectivamente.

1ª Ronda	
Grupo 1	
Arman - FC Portugal	0-4
Posicionamento da Bola:	0.00% 8.66% 37.15% 54.19%
FC Portugal - HumanLike	0-0
Posicionamento da Bola:	0.00% 11.19% 30.60% 58.21%
2ª Ronda	
Grupo 1	
FC Portugal - Rapotic	4-0
Posicionamento da Bola:	0.00% 9.63% 26.20% 64.16%
FC Portugal - VWerder	1-0
Este log não está disponível	
Ronda Final	
Rezvan - FCPortugal	0-0
Posicionamento da Bola:	19.90% 50.41% 24.37% 5.32%
HumanLike - FCPortugal	0-2
Posicionamento da Bola:	0.00% 12.09% 45.48% 42.43%
SEU - FCPortugal	0-1
Posicionamento da Bola:	9.05% 31.54% 29.65% 29.75%
FCPortugal - VWerder	0-0
Posicionamento da Bola:	1.74% 8.52% 32.71% 57.04%
FCPortugal - UtUtd	0-0
Posicionamento da Bola:	0.55% 7.09% 42.95% 49.41%
Jogos da Final	
FC Portugal - SEU	0-0
Posicionamento da Bola:	5.20% 23.26% 40.81% 30.73%
FC Portugal - SEU	0-0 (catch margin 1 metro)
Posicionamento da Bola:	15.36% 37.95% 33.53% 13.15%
3/4 Lugares	
Rezvan - UtUtd	0-0
Rezvan - UtUtd	1-0 (catch margin 1 meter)

Tabela 47: Resultados obtidos em Eindhoven pela equipa FCportugal

Na maioria dos jogos a equipa por nós implementada dominou, como se pode ver pela posição da bola ao longo do jogo. Deste modo, pode-se concluir que a maioria das equipas não conseguiu chegar à área de FCPortugal ou chegava pontualmente enquanto a equipa FCPortugal jogava quase exclusivamente no meio campo adversário e com grande frequência na área adversária. Em média, a bola estava 75% do tempo no meio campo adversário, estando mais de 40% do tempo na metade mais próxima da baliza respectiva.

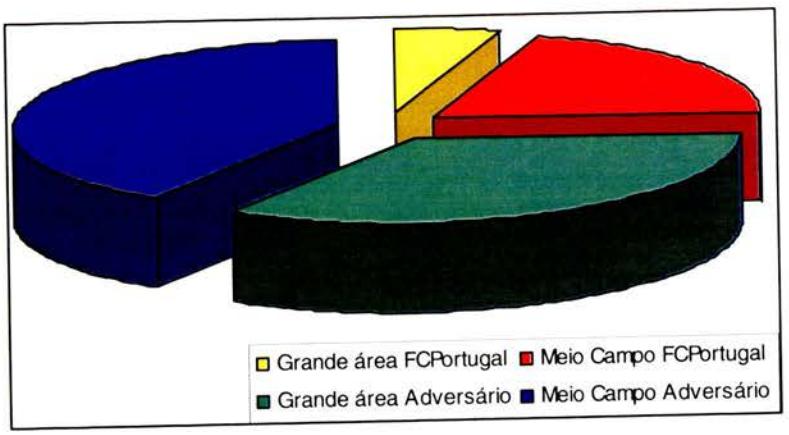

Figura 25: Média do posicionamento da bola nos jogos no europeu

No entanto há algumas excepções: o jogo contra os Rezvan e as finais contra os SEU. Os Rezvan tinham um bom algoritmo de intercepção de bola pelo que a conseguiam recuperar rapidamente, o que dificultou o jogo à nossa equipa. No primeiro jogo da final, a equipa FCPortugal dominou (a bola esteve aproximadamente 70% do tempo no meio campo adversário) mas devido a uma defesa coesa e a uma boa implementação do guarda-redes pelos SEU e algumas oportunidades de golo perdidas pela nossa equipa, o jogo acabou em empate. No último jogo, a táctica foi mais defensiva pois sabia-se que em caso de empate o campeão era decidido pela posição na última fase de grupos. Mesmo jogando mais defensivamente, o jogo foi equilibrado e os SEU nunca conseguiram rematar.

Figura 26: Os novos campeões da Europa e do mundo

7.2 Mundial

O campeonato mundial de futebol robótico – RoboCup 2006 – foi realizado em Bremen na Alemanha. Participaram nesta liga equipas de diversos países e foram defrontadas equipas de algumas das melhores universidades mundiais (Alemanha, Holanda, Japão, Reino Unido, China, Irão, Brasil e México).

Foram realizados 25 jogos nos quais foram marcados 78 golos sem sofrer nenhum. O FCPortugal dominou praticamente todos os jogos, sofrendo menos de 10 remates à baliza durante todo o campeonato.

A diminuição da *catchmargin*, o aumenta das balizas em altura e o melhoramento da nossa equipa nos dois meses que antecederam este acontecimento levaram a um aumento significativo de golos marcados. No entanto, o servidor continuou com alguns *bugs*, especialmente na marcação de foras-de-jogo, utilizados pela primeira vez neste mundial.

De seguida podem-se ver os resultados obtidos neste mundial e o cálculo do posicionamento da bola ao longo do jogo (Grande área e entrada do meio-campo da equipa FCPortugal, entrada do meio campo adversário e respectiva grande área). Alguns dos logs ainda não foram disponibilizados pela organização, pelo que nestes casos só é mostrado o resultado do jogo:

1ª Fase de grupos:

```
FC Portugal - TsinghuAeolus (China)                5:0
Posse Bola                                         0.00%  6.76% 36.65% 56.58%
FC Portugal - AllemaniACs3D (Germany)              5:0
Posse Bola                                         0.00%  3.96% 30.08% 65.96%
FC Portugal - MRL (Iran)                           2:0
Posse Bola                                         4.34% 17.49% 51.34% 26.82%
FC Portugal - RoboLog3D (Germany)                  5:0
Posse Bola                                         0.00%  0.99% 48.87% 50.14%
FC Portugal - Aria (Iran)                          4:0
Posse Bola                                         9.25% 36.22% 35.27% 19.26%
FC Portugal - MainzRollingBrains (Germany)         9:0
Posse Bola                                         0.00% 14.82% 36.51% 48.67%
```

2ª Fase de grupos:

```
FC Portugal - JU-TsubameGaeshi (Japan)             9:0
Posse Bola                                         0.00%  3.53% 58.77% 37.69%
FC Portugal - WrightEagle (China)                  3:0
Posse Bola                                         0.00% 11.89% 35.86% 52.25%
FC Portugal - MainzRollingBrains (Germany)         8:0
Posse Bola                                         0.00% 20.60% 48.49% 30.91%
FC Portugal - AmoiensisNQ (China)                  5:0
Posse Bola                                         0.00% 19.21% 31.30% 49.49%
FC Portugal - Arman (Iran)                         1:0
Log não disponível
```

3ª Fase de grupos:

```
FC Portugal - SEU (China)                          0:0
Posse Bola                                         0.40% 15.89% 40.10% 43.61%
```

FC Portugal - MRL (Iran) Posse Bola FC Portugal - Rezvan (Iran) Posse Bola FC Portugal - Brainstormers (Germany) Log não disponível FC Portugal - Caspian (Iran) Posse Bola FC Portugal - CZU2006 (China) Posse Bola 0.00% 19.35% 44.24% 36.41% FC Portugal - Arman (Iran) Log não disponível	3:0 0.00% 19.54% 35.48% 44.98% 1:0 15.70% 29.87% 33.47% 20.96% 5:0 1:0 0.00% 5.24% 50.89% 43.87% 5:0 2:0
Quartos-de-final:	
FC Portugal - Virtual Werder (Germany) Este log ainda não está disponível	3:0
Meia-final:	
FC Portugal - Aria (Iran) Este log ainda não está disponível	1:0 (depois de 2 empates a 0)
Final:	
FC Portugal - WrightEagle (China) Este log ainda não está disponível	1:0 (depois de 2 empates a 0)

Tabela 48: Resultados obtidos em Bremen pela equipa FCportugal

Pelos resultados dos jogos é de notar uma melhoria na capacidade dos agentes concretizarem. É de notar também a boa prestação na defesa que manteve a bola sempre no meio campo adversário e na posse da nossa equipa. Neste mundial o domínio do jogo foi ainda mais acentuado que no europeu, mesmo com os melhoramentos efectuados pelas outras equipas. Em média mais de 80% do tempo de jogo a bola encontrava-se no meio campo adversário, sendo que em metade deste período a bola se encontrava no último quarto do campo. O período de tempo em que a bola chegou à grande área do FCPortugal diminuiu também significativamente para uns 1,6%. Este domínio pode ser verificado no gráfico da figura 27.

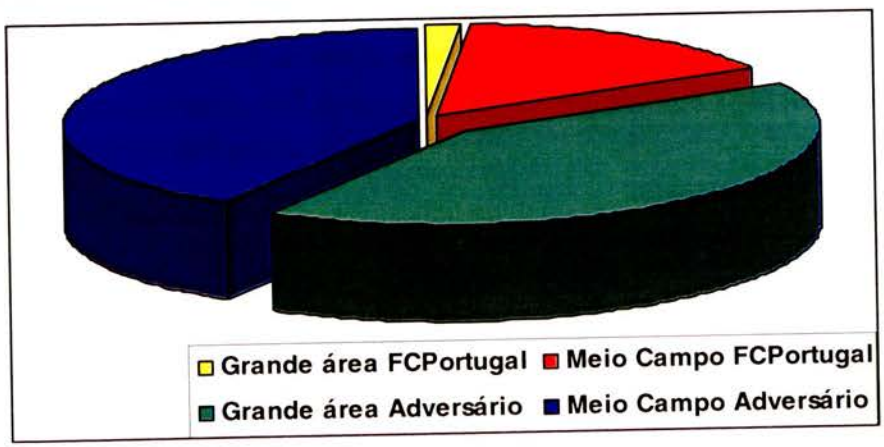

Figura 27: Média do posicionamento da bola nos jogos do mundial

Nas raras ocasiões em que a equipa adversária chegou à área da equipa FCPortugal poucas vezes conseguiu rematar com perigo e, nesses casos, a nova implementação do guarda-redes mostrou ser eficaz permitindo ao guarda-redes defender a bola. O jogo contra os Rezvan, apesar de acabar numa vitória de FCPortugal, foi o mais equilibrado, pois devido a um problema na implementação da decisão de passe, já corrigido, caso os defesas dos Rezvan estivessem encostados à linha de meio campo, os nossos agentes não passavam desta linha para evitar a situação de fora de jogo.

Comparando a média de golos marcados por jogo no europeu e no mundial vê-se um aumento de 65% marcados, obtendo-se uma média de 3.1 golos por jogo na fase de grupos do mundial. Já em fases finais, a jogar com as melhores equipas, este valor cai para 0.7 por jogo (devido aos empates a 0 na meia final e final) como se pode ver no gráfico seguinte:

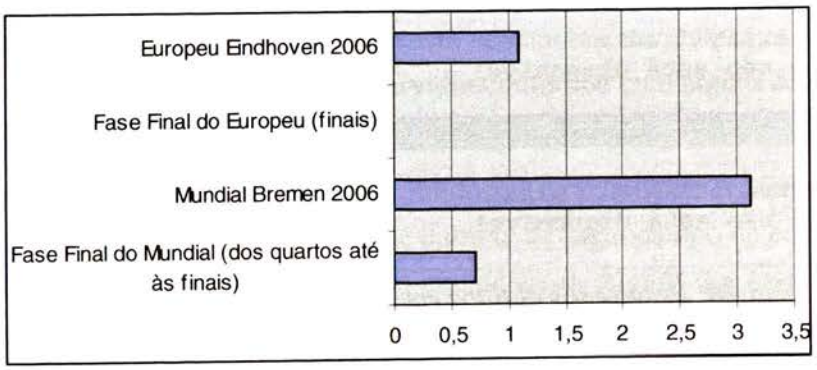

Figura 28: Média de golos marcados em jogos oficiais

Na meia-final e final as equipas defrontadas eram muito defensivas pelo que os nossos agentes, apesar de manterem quase sempre a bola no ataque, frequentemente não conseguiam chegar com perigo à baliza adversária. No entanto, existiram várias oportunidades perdidas que por pouco não foram concretizadas, e acabou-se sempre por conseguir o golo.

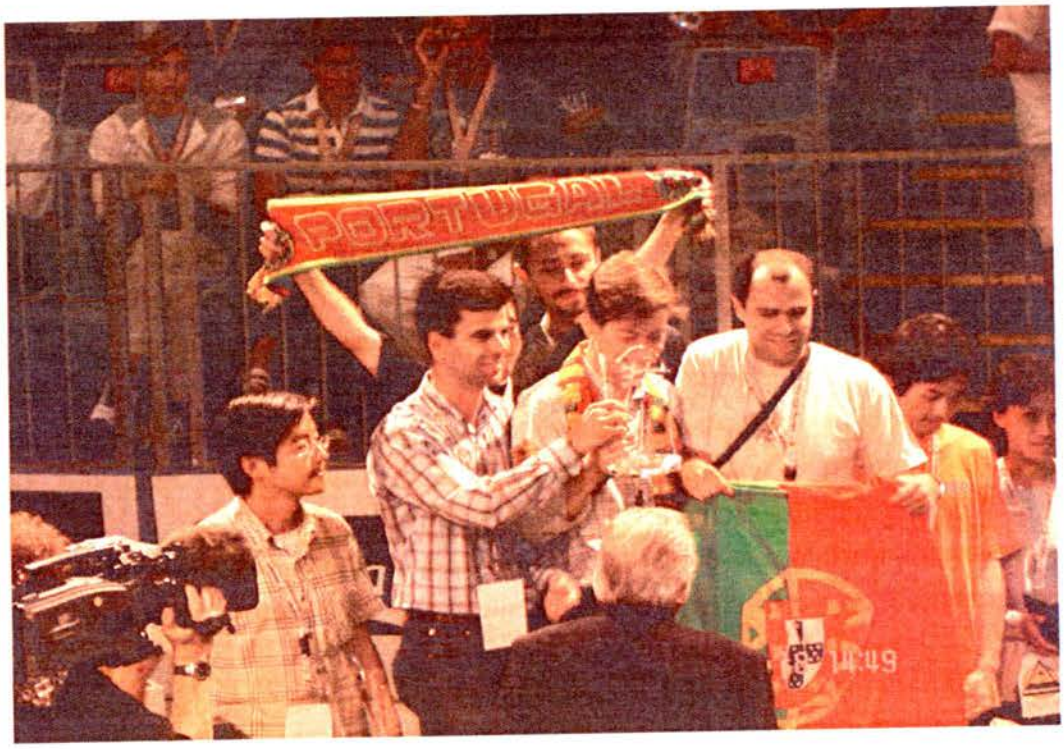

Figura 29: Subida ao pódio – RoboCup2006

7.3 Kick

Como já foi dito, a execução do pontapé tem vários erros associados: posição do jogador, erro na direcção de chuto, erro na potência e ângulo aplicados à bola e os erros provocados pelas aproximações feitas nas fórmulas descritivas da trajectória. Apesar de no total esses erros poderem atingir valores consideráveis, na ordem das poucas dezenas de centímetros (10-30 cm como descrito na implementação das funções de kick), na prática os jogadores conseguem passar e rematar com bastante precisão sendo possível colocar a bola muito perto do objectivo. Estes erros poderão ser ainda diminuídos aumentando o número de amostras de chuto de forma a obter fórmulas que descrevam melhor as trajectórias e considerando também a massa da bola, o que de momento ainda não está a ser feito (já há fórmulas que descrevem esta dependência mas que ainda não foram testadas suficientemente pelo que ainda não foram implementadas no código da equipa).

7.4 Comunicação

Na comunicação como não há erros de canal, aquilo que é enviado é sempre recebido com sucesso desde que não haja mais do que um agente a comunicar simultaneamente. Os erros associados à comunicação provêm de erros na codificação da confiança (valores abaixo de 100-NALLOWED são desprezados), das posições e velocidade. Estes podem ser, no entanto, desprezados, pois os erros em cada componente de posição são no máximo 0.5 mm e na velocidade de 5 mm/s, que face aos erros provocados pela visão são muito reduzidos. Todas as outras variáveis transmitidas são codificadas sem erro. Há também um erro devido à propagação da mensagem, quando esta é ouvida o estado do mundo comunicado já difere do estado actual do mundo. Como a diferença é tipicamente de 20 a 40 ciclos, ou seja de 0.15 a 0.3 segundos, os objectos do mundo não tiveram tempo de se mover significativamente. No entanto, o erro devido a este atraso poderá ser reduzido se se utilizar a velocidade transmitida para actualizar a posição do objecto.

Para diminuir o erro do estado do mundo importa também maximizar as confianças de cada objecto, especialmente daqueles que estão próximos da bola. Com esse fim, os jogadores mais próximos da bola são os que comunicam. No entanto, isto resulta num afastamento entre as posições do estado do mundo de cada agente e as posições reais de cada agente que esteja muito afastado da bola e que não seja visto por nenhum agente que esteja próximo desta. Estes erros não afectam o modo de jogo dos jogadores, pois é na área mais próxima da bola que interessa ter o melhor conhecimento do mundo. À medida, que a bola se vai movendo no campo, o estado do mundo vai sendo actualizado com maior precisão nessas zonas. No futuro, o algoritmo vai ser alterado de modo a permitir que um agente, que tenha mais e melhor informação, a comunique em vez daqueles que estão mais próximos evitando parcialmente estes erros para os objectos mais afastados. Quanto maior a diferença entre o seu mundo e o mundo comunicado, assim como quanto maior for a confiança em cada um dos objectos em relação às confianças comunicadas pelos outros agentes, especialmente aqueles próximos da bola, maior será a importância da informação a comunicar.

Capítulo 8

8. Conclusões e Perspectivas de Desenvolvimento

O objectivo fundamental do trabalho apresentado neste documento relaciona-se com o estudo de metodologias de coordenação entre múltiplos agentes autónomos computacionais em ambientes que implicam a execução cooperativa de tarefas, nomeadamente a liga de simulação 3D no RoboCup. Esta liga tem particular interesse devido a simular complexidades existentes nos sistemas robóticos, como erros nos sensores e actuadores dos agentes, simulando um ambiente simultaneamente adverso e cooperativo, contínuo e não determinístico.

Com o fim de cumprir este objectivo, foi melhorado e adaptado às novas condições do ambiente de simulação o código de um Sistema Multi-Agente totalmente funcional – FC Portugal 3D. Foram desenvolvidos algoritmos de decisão de passe e chuto assim como a implementação de uma nova funcionalidade no código: a comunicação. Na comunicação, a utilização de dois mundos separados para descrever o estado do mundo do agente e o estado do mundo comunicado ao agente permite ao agente actualizar só o seu mundo se as confianças comunicadas forem superiores. Deste modo, fica com a melhor informação de cada estado aproximando-se mais do estado real.

Estes novos algoritmos foram testados com sucesso quer em experiências controladas quer em competições oficiais, levando a equipa FCPortugal a sagrar-se campeão da Europa em Eindhoven e posteriormente campeã do mundo em Bremen. Nestas competições, esta equipa defrontou algumas das melhores universidades mundiais (Alemanha, Holanda, Japão, Reino Unido, China, Irão, Brasil e México) nunca sofrendo um golo em 32 jogos e marcando 90 golos no total. Estes resultados comprovam a eficácia das metodologias utilizadas pela equipa.

A principal limitação do trabalho efectuado prende-se com o facto de o único domínio de validação e aplicação directa das funcionalidades implementadas ser a liga de simulação 3D do RoboCup. No entanto, algumas das metodologias utilizadas pela equipa poderão ser adaptadas para outras liga do RoboCup e para outros domínios que impliquem a coordenação de robôs.

No futuro, poderão ser melhorados os algoritmos de decisão de passe e remate, de forma a torná-los mais precisos e o algoritmo de comunicação, alterando a forma como um agente decide comunicar de modo a que só o faça quando tem algo pertinente a comunicar. Este segundo algoritmo já foi parcialmente implementado, mas ainda não é utilizado. É necessário também realizar mais testes de modo a optimizar alguns dos parâmetros utilizados. Quanto à comunicação poderão ser comunicados mais parâmetros como por exemplo pedir a outro jogador para se desmarcar ou ir para um certo ponto. Deverá também acabar de ser implementado o novo algoritmo de decisão de comunicação que permitirá a um agente calcular a importância da informação que possui (quanto maior a diferença entre o seu mundo e o mundo comunicado, assim como quanto maior for a confiança em cada um dos objectos em relação às confianças comunicadas pelos outros agentes, especialmente aqueles próximos da bola, maior será a importância da informação a comunicar).

Referências Bibliográficas

[Asada e Kitano, 1999] Asada, M. e Kitano, H. editores. RoboCup-98: Robot Soccer World Cup II, Springer, Lecture Notes in Artificial Intelligence, Vol. 1604, 1999

[Balch e Arkin, 1994] Balch, T. e Arkin, R. C. Communication in Reactive Multi-Agent Robotic Systems, Autonomous Robots, Vol. 1 (1), pp. 27-53, 1994

[Barr e Feigenbaum, 1981] Barr, A. e Feigenbaum, A. The Handbook of Artificial Intelligence, Vol. 1,Los altos, Morgan Kaufmann, 1981

[Birk et al., 2002] Birk, Andreas; Coradeschi, S. e Takadoro, S., RoboCup 2001: Robot Soccer World Cup V. Springer LNAI 2377, Berlin, 2002

[Booch, 1994,] Booch, G., Object-Oriented Analysis and Design, 2nd, Edition, Addison-Wesley, Reading, MA, 1994

[Bronson, 1997] Bronson, Gary J., Program Development and Design Using C++, PWS Publishing Company; 1997.

[Coelho, 1994] Coelho, Hélder, Inteligência Artificial em 25 Lições, Fundação Calouste Gulbenkian,1994

[Dorer, 2000] Dorer, Klaus, The Magma Freiburg Soccer Team, em [Veloso et al., 2000], pp.600-603, 2000

[FCPortugal, 2006] FCPortugal: [Em Linha] Disponível em *ttp://www.ieeta.pt/robocup/*

[Consultado em Junho de 2006]

[Kitano, 1997] Kitano, Hiroaki, RoboCup: The Robot World Cup Initiative, Proceedings of the 1st International Conference on Autonomous Agent (Agents' 97), Marina del Ray, The ACM Press, 1997.

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

[Lau e Reis, 2000] Lau, Nuno e Reis, Luis Paulo, FC Portugal Most Interesting Research: Overview, [Em Linha] Disponível em http://www.ieeta.pt/robocup/documents/ FCPortugalInteresting.ps.zip, Novembro de 2000

[Lau e Reis, 2001] Lau, Nuno e Reis, Luis Paulo, FC Portugal 2001: Low Level Skills and World State Update, LIACC (NIAD&R) – Research Report, Outubro de 2001

[Lau e Reis, 2002] Lau, Nuno e Reis, Luis Paulo, FC Portugal 2001 Team Description: Configurable Strategy and Flexible Teamwork, em [Birk et al., 2002], pp. 515-518, 2002

[Reis e Lau, 2001] Reis, Luís Paulo e Lau, Nuno, FC Portugal Approach Overview, [Em Linha] Disponível em Http://www.ieeta.pt/robocup/overview.htm

[Consultado em Junho de 2006]

[Reis e Lau, 2001a] Reis, Luís Paulo e Lau, Nuno, FC Portugal Team Description: RoboCup 2000 Simulation League Champion, em Peter Stone, Tucker Balch and Gerhard Kraetzschmar, editors, RoboCup-2000: Robot Soccer World Cup IV, Springer Verlag Lecture Notes in Artificial Intelligence, Vol. 2019, pp.29-40, Berlin, 2001

[Reis e Lau, 2001b] Reis, Luís Paulo e Lau, Nuno, How to Give Intelligence to Soccer Playing Agents, [Em Linha] Disponível em Http://www.fe.up.pt/~lpreis/Individual.ps.zip

[Consultado em Junho de 2006]

[Reis e Lau, 2001c] Reis, Luís Paulo e Lau, Nuno, Coach Unilang, [Em Linha] Disponível em Http://www.fe.up.pt/~lpreis/ Coach_Unilang.ps.zip,

[Consultado em Junho de 2006]

[Reis e Lau, 2001d] Reis, Luís Paulo e Lau, Nuno, FCPortugal Home Page [Em Linha] Disponível em Http://www.ieeta.pt/robocup

[Consultado em Junho de 2006]

[Reis e Lau, 2002] Reis, Luís Paulo e Lau, Nuno, COACH UNILANG – A Standard Language for Coaching a (Robo) Soccer Team, em Andreas Birk, Silvia Coradeschi and Satoshi Tadokoro, editors, RoboCup-2001: Robot Soccer World Cup V, Springer Verlag Lecture Notes in Artificial Intelligence, Vol. 2377, pp.183-192, Berlin, 2002

[Reis e Lau, 2003a] Reis, Luís Paulo e Lau, Nuno, FC Portugal 2002 Team Description: Flexible Coordination Techniques em Gal Kaminka, Pedro Lima e Raul Rojas editors, RoboCup-2002: Robot Soccer World Cup V, Springer Verlag Lecture Notes in Artificial Intelligence, Berlin, 2003

[Reis e Lau, 2003b] Reis, Luís Paulo e Lau, Nuno, FC Portugal 2002 Coach: High-Level Coaching of RoboSoccer Games em Gal Kaminka, Pedro Lima e Raul Rojas editors, RoboCup-2002: Robot Soccer World Cup V, Springer Verlag Lecture Notes in Artificial Intelligence, Berlin, 2003

[Reis, 2000] Reis, Luís Paulo, How to Give Intelligence to Football Playing Agents, LIACC(NIAD&R) – Research Report, Abril de 2000

[Reis, 2001b] Reis, Luís Paulo, How to Win in Robotic Football? Strategy, Cooperation and Communication to build an Artificial Football Team, WIAA2001, Instituto Superior Técnico - Lisboa,Março de 2001

[Reis, 2002e] Reis, Luís Paulo, Coordenação de Agentes Cooperativos e Trabalho de Equipa, LIACC(NIAD&R) – Research Report, Maio de 2002

[Reis, 2002f] Reis, Luís Paulo, Coordenação Estratégica – Aplicação no Futebol Robótico Simulado, LIACC (NIAD&R) – Research Report, Junho de 2002

[Riedmiller et al., 2000] Riedmiller. Martin, et al., Karlsruhe Brainstormers 2000 - A Reinforcement Learning Approach to Robotic Soccer. Proceedings of the Fourth International Workshop on RoboCup. Melbourne, Agosto de 2000

[RoboCup, 2001] The Goals of RoboCup. by RoboCup Federation, [Em Linha] Disponível em Http://www.robocup.org/overview/22.html, 2000

[Consultado em Junho de 2006]

[RoboCup, 2006] RoboCup: [Em Linha] Disponível em *http://www.robocup.org*

[Consultado em Junho de 2006]

[Soccermonitor] RoboCup Soccer Simulador: [Em Linha] Disponível em *http://sserver.sourceforge.net*

[Consultado em Junho de 2006]

[Stone et al., 2000a] Stone, Peter; Riley, Patrick; Veloso, Manuela, The CMUnited-99 Champion Simulator Team, em [Veloso et al., 2000]

[Stone et al., 2001a] Stone, Peter; Balch, Tucker; Kraetzschmar, Gerhard, editores, RoboCup-2000: Robot Soccer World Cup IV, Springer Verlag, Berlin, 2001

[Stone et al., 2001b] Stone, Peter et al. editores, RoboCup-2000 The Forth Robotic Soccer World Championships, AI magazine, Vol. 22, No. 1, pp. 11-38, 2001.

[Stone et al., 2001c] Stone, Peter et al. editors, Overview of RoboCup-2000 in [Stone et al., 2001a], pp.1-28, 2001

Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL
www.fe.up.pt