Instituto Superior Técnico


# Desenvolvimento de Modelos de Cooperação para uma Sociedade de Agentes

113/1999/L

Luis Filipe Toscano

Lisboa, Portugal
30 de Novembro de 2001

Trabalho orientado por:

**Luis Manuel Marques Custódio**
Professor responsável
Secção de Sistemas e Controlo
Dept. Engenharia Electroténica e de Computadores
Instituto Superior Técnico

**Rodrigo Martins de Matos Ventura**
Docente acompanhante
Secção de Sistemas Digitais e Computadores
Dept. Engenharia Electroténica e de Computadores
Instituto Superior Técnico

# Agradecimentos

Depois de tantos anos de casa, escrever uma dedicatória que enumere muitos dos que contribuiram para o meu desenvolvimento enquanto adulto reguila é complicado. Quase tão complicado como acabar o curso. :)

Aos membros do **Projecto ISocRob** – Prof. Luis Custódio, Prof. Pedro Lima, Rodrigo Ventura, Carlos Marques, Manuel Lopes, Bruno Damas – em particular aos orientadores do presente trabalho a quem devo o incentivo para a realização deste estudo, bem como o apoio amigo e orientação científica, e para eles vai o meu primeiro e profundo reconhecimento.

A toda a rapaziada da **SCDEEC**! Aos do passado (NAF, OM, VR), presente (JG, PF, PO, VF) e futuro (FC, HC, JO, JG, RMS).

A todos os meus **Camaradas Controlo Engenheiros** sem os quais teria prestado o mínimo de atenção nas aulas! Ao Deus João Menano, amén. Ao Bruno Belbute e Vítor Oliveira por sobrevivermos juntos a OA. Aos *Peters*, versões nacional e importada. Ao João Santos pelas sempre elucidativas conversas sobre fauna e flora europeia. Ao Hugo Constantino pela excelente imitação do Diabo. Ao Eduardo Catarino por, no meio de tanto *cego*, ser o único com um olho. Ao Herr Bernardo Motta, e o seu fiel *side-kick* Tiago Mendes, na contra-divulgação da nova revolução. Ao Ricardo Oliveira por me fazer companhia até ao fim. Ao super-trio Egídio *Humpty*, Elvino *Dumpty* e Alberto Vale-*Tudo*. E ainda o *fala fininho* Brás.

À minha **trupe**! Ao futuro vizinho Silva, que vá aquecendo as brazas do churrasco. Ao fusso mor Baptista (o outro), que aprenda a marcar golos de jeito. Ao teórico Sá, palavras para quê? Ao Ricardo Fradinho, *Angélico* Azevedo e *Flip* Saraiva por saberem sempre mais que eu. Ao *urso* Edgar e ao *outdoor man* Jóta Paus pela companhia na BaíaNet.

Sem esquecer o meu ***clone***! Fpmmmmppmppffmm mmmmmfmmm-mmpmmmpff mmm ppmmpppffmpmmmm mpmppf mmffmfpfffmmppf! Ao vizinho, ao Duke Nuker, ao Zéner Power, ao *one and only* jB!

À minha ***Pwetty* Catarina**. Vemo-nos em São Francisco e Las Vegas.

# Resumo

Concepção de uma arquitectura de *software* multi-agente baseada em trabalho anterior realizado no âmbito do Projecto ISocRob, tendo como principais objectivos a abstração da camada de decisão e modularidade nas várias componentes. Implementação da arquitectura desenvolvida aplicada ao futebol robótico.

# Palavras chave

Arquitectura de *software*, Cooperação, Agentes, Sociedade de Agentes, Futebol robótico, RoboCup, Engenharia de *software*

# Índice

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

Tomando como objectivo final deste trabalho realizar cooperação entre agentes, optou-se por fazê-lo única e exclusivamente usando como plataforma de desenvolvimento e teste o futebol robótico, inserido no Projecto ISocRob [1] do ISR/IST. Pretende-se assim realizar cooperação entre robôs, adaptados às necessidades daquele desporto, num meio ambiente real.

Por iniciativa da RoboCup Federation [2] surge em 1997 o futebol robótico [3] como fórum de investigação nas áreas de Inteligência Artificial e Robótica, sendo então formadas três ligas de características diferentes que, por força das diferenças do meio com que lidam, acabaram por privilegiar àreas de investigação diferentes. As três ligas criadas são:

- Liga de Simulação ou *Simulation League*;
- Liga de Pequenos Robôs ou *Small Size League*;
- Liga de Robôs Autónomos ou *Middle Size League*.

Na Liga de Simulação, que consiste em colocar frente-a-frente duas equipas de onze agentes, num meio simulado, em que as percepções do mundo não estão sujeitas a ruído e em que a intervenção do agente no meio tem normalmente o efeito pretendido, a cooperaçao entre os agentes surge como o principal elemento diferenciador entre as duas equipas envolvidas.

Relativamente à *Small Size League* e dada a existência de uma câmara disposta sobre o campo com visão para a totalidade do mesmo, a ênfase principal reside no processamento de visão em tempo-real. O problema sentido nesta competição é de que enquanto for possível construir robôs mais rápidos, terá de continuar a haver um esforço em tornar rápida e cada vez melhor a percepção do mundo, apesar de já haver várias equipas que efectuam cooperação nesta liga.

O desafio da *Middle Size League* (MSL) [4] consiste em "jogar futebol" no palco mais realista: ao contrário da *Small Size League* não existe centralização de decisão nem percepção total do mundo. A robótica surge como o primeiro desafio às equipas, pois é necessário dispor de robôs com uma dinâmica que permita manobrar com eficácia num meio povoado de outros robôs, bem como empurrar a bola de uma forma controlada.

Resolvida a questão da robótica, surge um segundo desafio: dotar os robôs de uma percepção local em tempo-real do mundo. O processamento de visão tornou-se assim na principal ênfase da MSL, uma vez que a visão local proporciona a maior parte, senão a totalidade, da percepção que o robô dispõe do mundo.

O presente trabalho pretendia, tal como o título do mesmo sugere, conceber e implementar um modelo formal de cooperação entre os agentes robóticos, adaptados às necessidades do ambiente em causa, tendo como ponto de partida a arquitectura de *software* existente [5, 6], arquitectura essa desenvolvida no âmbito do Projecto ISocRob em anos anteriores. Fruto de inúmeras deficiências apresentadas por essa arquitectura, o foco deixou de consistir em implementar cooperação na arquitectura actual, mas sim conceber e implementar uma que o permitisse.

Os três objectivos do presente trabalho visavam colmatar as duas deficiências observadas mas mantendo a característica dominante da arquitectura existente. Os objectivos estabelecidos são então:

- **Abstracção**: Não deverá ser necessário um conhecimento profundo da arquitectura de *software* para estabelecer o comportamento exibido pelo agente robótico;
- **Modularidade**: Terá de ser possível introduzir novas vertentes ao código existente e, na medida do possível, facilitar a inclusão desse acréscimo de funcionalidade;
- **Desempenho**: Uma estrutura que obedecendo aos dois melhoramentos anteriores não comprometa um tempo de resposta suficientemente pequeno para fazer face às vicissitudes do futebol robótico.

No capítulo 2 serão apresentadas as duas arquitecturas de *software* desenvolvidas antes do presente trabalho como forma de descrever o percurso escolhido pelo Projecto ISocRob na resolução do problema. Serão também descritas as duas arquitecturas de *software* desenvolvidas no âmbito do trabalho.

O capítulo 3 descreverá em maior detalhe a estrutura e funcionamento das várias componentes da segunda arquitectura de *software* concebida e implementada.

Não obstante este trabalho incidir fundamentalmente sobre a concepção de uma arquitectura de *software* que possa servir para a implementação de cooperação, é feito um contributo no capítulo 4 quanto a uma possível forma de o efectuar.

O presente relatório termina com o capítulo 5 onde serão apresentadas as conclusões finais e os possíveis desenvolvimentos futuros.

# Capítulo 2

# Arquitectura de *Software*

A tarefa de conceber uma arquitectura de *software* é uma matéria ingrata, visto ser necessário um ciclo completo de desenvolvimento de forma a retirar conclusões da sua validade enquanto solução viável para um problema real, ciclo este que consiste em três etapas distintas: concepção, implementação e experimentação.

O presente capítulo começa por descrever o problema em questão e os desafios que são colocados na resolução do mesmo. Primeiro serão apresentadas as duas arquitecturas existentes antes da realização do presente trabalho, fazendo-se um ponto de situação quanto à medida de desempenho corrente dos robôs enquanto jogadores de futebol robótico. Tendo como base a segunda arquitectura de *software*, serão então apresentadas as duas novas arquitecturas que foram objecto do ciclo completo de desenvolvimento, detalhando as alterações e melhoramentos em cada novo ciclo bem como as razões que motivaram tais medidas.

## 2.1 Futebol Robótico

O futebol robótico pretende imitar o futebol real no seu princípio mais simples: marcar golos por forma a vencer o jogo. Objectivo esse que é naturalmente adoptado por ambas as equipas, pelo que o jogo consiste em tentar marcar golos, introduzindo a bola na baliza adversária e evitando que o adversário o consiga fazer. Logo, tal como no desporto a sério, o jogo resume-se a duas equipas que lutam pela posse da bola num campo verde. Contudo, as semelhanças terminam aqui.

O meio ambiente limitado e altamente estruturado que a equipa de agentes, doravante referidos como agentes robóticos visto coincidirem com a equipa de robôs, terá de enfrentar está ilustrada na figura 2.1. Os objec-

tos com significado para o agente robótico são identificáveis por cores:

- Balizas, uma azul e outra amarela;
- Bola laranja, de tamanho semelhante ao usado no futebol de onze;
- Campo verde, de nove metros de comprimento por cinco metros de largura, com marcações a branco;
- Paredes[1] que delimitam o espaço de jogo a branco;
- Robôs integralmente em preto, sendo que as duas equipas são distinguidas por marcas[2] em azul celeste e magenta.

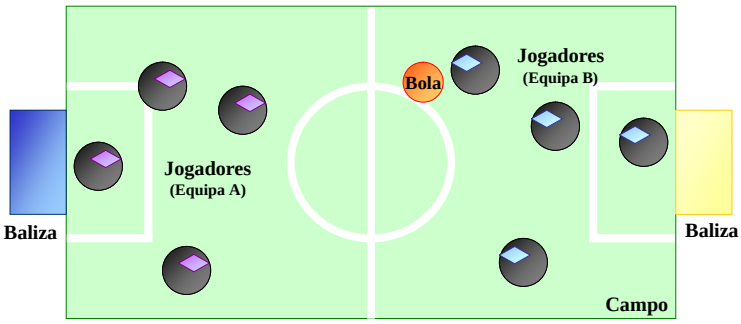



Figura 2.1: O mundo do Futebol Robótico

Uma partida de futebol robótico consiste em duas partes de dez minutos sendo declarada vencedora a equipa que marcar mais golos. De referir que, tal como no futebol real, existe um conjunto de regras[3] que define todos os aspectos da modalidade: dimensões do campo, comportamentos não permitidos, sanções disciplinares, etc. A título de exemplo, apesar de ser permitido o contacto entre robôs, se um persistir ou forçar a passagem pelo robô adversário será sancionado de acordo com as mesmas[4].

[1] Há intenção de vir a retirar brevemente as paredes de forma a aumentar o realismo na MSL.

[2] Tiras de cartolina colorida dispostas de forma a que seja sempre visível pelo menos uma única marca. De referir que nas regras não é imposta uma forma específica para estas marcações.

[3] Totalidade das regras disponíveis em [4].

[4] Na primeira violação da regra será punido com um cartão amarelo. Uma segunda advertência implica que seja mostrado um cartão azul, sendo retirado do campo até ao próximo recomeço de jogo. A acumulação de dois cartões azuis resulta num cartão vermelho e consequente expulsão do jogo.

Estabelecidos os objectivos da equipa e o meio ambiente há que definir a estratégia que o agente robótico irá adoptar para ser bem sucedido nas suas funções como jogador de futebol: procurar a bola e ir até ao encontro dela, assim como enquadrar-se com a baliza adversária de forma a progredir no campo com a bola até junto da baliza na tentativa de marcar um golo.

Fica assim por satisfazer a necessidade de impedir o adversário de marcar, adicionando ao ataque o conceito de defesa que é obtido de duas formas: um único robô, denominado guarda-redes, que pode permanecer dentro da grande-área de forma a cobrir a baliza ou através de um comportamento defensivo em que o robô pretende apenas afastar a bola da própria baliza.

## 2.2 Primeira Geração: `robocup98`

A primeira iteração do *software*, designada por `robocup98`, resultou de uma leitura simples do problema e pretendia demonstrar o robô SocRob [7, 8] como uma plataforma válida para o futebol robótico: a imagem capturada pelo robô é processada, a seguir é decidida a acção a efectuar e por fim essa acção é posta em prática. Este ciclo perpétuo, ilustrado na figura 2.2, foi denominado *STA Loop*.

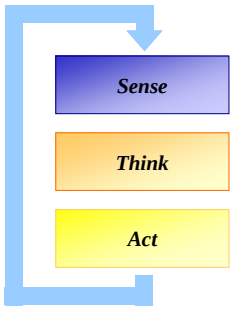



Figura 2.2: *STA Loop*

No código `robocup98` os três blocos do *STA Loop*, implementados em série, eram constituídos pelo código estritamente necessário que no seu conjunto fazia com que o robô exibisse o conjunto mínimo de aptidões necessárias de forma a participar nas competições organizadas pela RoboCup. Os blocos STA eram então:

- ***Sense***: Estima as posições, relativas ao robô, da bola e das duas balizas com base nos centros de massa dos *pixeis* discriminados como laranja e azul/amarelo, respectivamente[5];

[5]Em 1998, a colocação de marcadores coloridos nos robôs das equipas era opcional.

- ***Think***: Uma máquina de estados[6], ilustrada na figura 2.3, em que as transições de estado dependem dos predicados enumerados na tabela 2.1, ou de combinações lógicas destes;

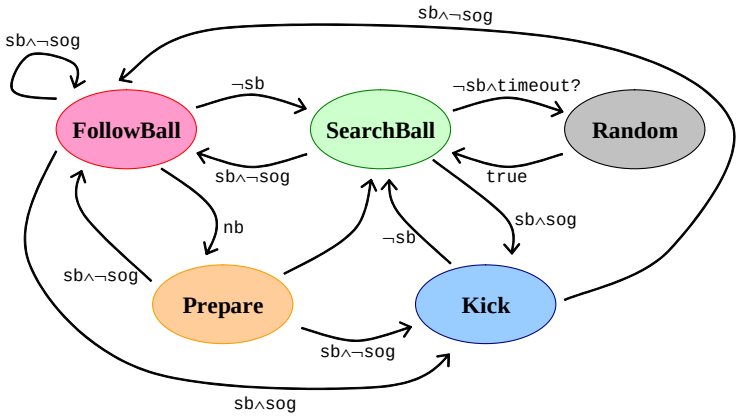



Figura 2.3: Máquina de estados `robocup98`

| Sigla | Predicado | Descrição |
|---|---|---|
| `sb` | `see_ball?` | Verdadeiro caso o robô esteja a ver a bola |
| `sog` | `see_other_goal?` | Verdadeiro caso o robô esteja a ver a baliza adversária |
| `nb` | `near_ball?` | Verdadeiro caso a bola se encontre a menos de uma determinada distância do robô |
| | `timeout?` | Verdadeiro caso o robô não veja a bola há algum tempo |

Tabela 2.1: Predicados

- ***Act***: Envio das actuações, calculadas no passo anterior, para os motores.

Com uma estrutura de código muito simples, conseguia-se o principal para um projecto que dava então os seus primeiros passos: colocar o robô à procura da bola e a empurrar a dita para a baliza.

[6]De referir que a máquina de estados era ainda complementada com um conjunto de estados específicos para o guarda-redes.

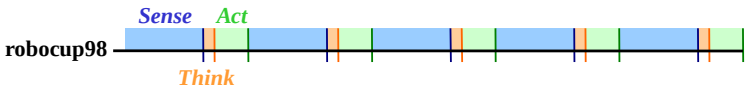



Figura 2.4: Diagrama temporal `robocup98`

## 2.3 Segunda Geração: `robocup99`

O *STA Loop* em série implica uma perda de tempo de aquisição e processamento de imagem, visto fazer parte do mesmo ciclo de execução que o processo de decisão e envio dos comandos para o robô. Pela figura 2.4 este facto é evidente, pelo que ao executar o código dos três blocos do `robocup98` paralelamente passa a ser possível obter a cadência máxima de processamento sensorial. Assim, o agente robótico foi dividido em pequenos agentes de *software*, que passam a lidar exclusivamente com uma pequena parte do problema total numa estratégia clara de *divide and conquer*.

Esta é a primeira característica fundamental, partilhada por todas as arquitecturas de *software* realizadas desde a concepção dos pequenos agentes de *software*, denominados micro-agentes, ou abreviadamente $\mu$A. Esta denominação surge da intenção inicial de povoar o agente com inúmeros $\mu$A relativamente simples, de reduzida complexidade de código, mas que por força das interacções com os restantes $\mu$A da arquitectura resultariam num funcionamento complexo [9].

Sendo desconexos, os $\mu$A dispõem de duas formas distintas de comunicarem entre si: por envio e recepção de sinais ou através de um *blackboard*. A primeira forma é a mais elementar, servindo apenas para sincronizar dois $\mu$A: um primeiro $\mu$A coloca-se num estado dormente até que um outro qualquer $\mu$A lhe envie um sinal, ficando o primeiro $\mu$A prontamente reactivado.

O *blackboard* é uma entidade de *software* onde é efectuado uma associação entre uma chave, um símbolo codificado numa *string*, e um valor. A estrutura em que os pares de chaves e valores são guardados foi implementada de tal forma que seja rápido aceder ao valor actual associado a uma chave, mas permitindo que novas chaves sejam criadas ou introduzidas em qualquer instante.

A arquitectura `robocup99`, ilustrada na figura 2.5, composta por seis $\mu$A tem como centro de operações o trio composto pelos $\mu$A `vision`, `machine` e `motors`. O funcionamento é extremamente simples visto tratar-se de uma transposição inspirada no código anterior ao enquadrar a *STA Loop* nos três $\mu$A referidos. O $\mu$A `vision` captura uma imagem e extrai todas as características relevantes colocando esses dados no *blackboard*. Após a recepção

do sinal vindo do $\mu$A de visão, o $\mu$A `machine` efectua uma única mudança de estado e antes de bloquear coloca no *blackboard* as actuações dos motores e envia um sinal para o $\mu$A `motors`. Por esta descrição, e pela análise da figura 2.6, é notório que todos $\mu$A trabalham à mesma cadência.

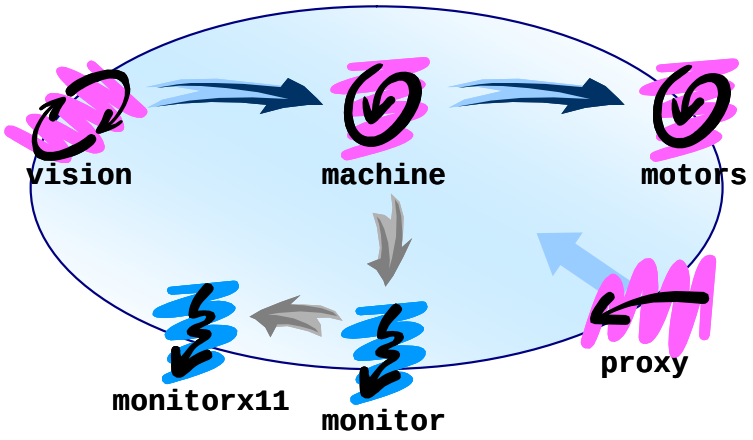



Figura 2.5: Arquitectura de *software* `robocup99`

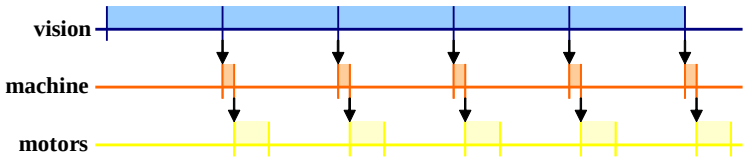



Figura 2.6: Diagrama temporal `robocup99`

A concluir a descrição da arquitectura resta referir qual a funcionalidade dos restantes elementos ilustrados na figura 2.5. Foram adicionados dois $\mu$A de monitorização que são opcionalmente lançados no arranque do programa com o único objectivo de facilitar a depuração do código. O $\mu$A `monitor` envia para o ecrã em modo texto, uma vez por ciclo do $\mu$A `machine`, um conjunto de parâmetros definidos pelo programador. O $\mu$A `monitorx11` envia para o ecrã a última imagem capturada onde as características detectadas são identificadas (figura 2.7).

O conceito do *blackboard* foi ainda estendido para que algumas chaves fossem transparentemente partilhadas por todos os agentes robóticos. Quando essas chaves, ditas distribuídas, fossem modificadas eram enviadas pela rede *wireless*, sendo interceptadas pelo $\mu$A `proxy` que as colocava no *blackboard* local de cada agente. O $\mu$A `proxy` e o mecanismo de difusão de chaves será descrito em maior pormenor na secção 3.4.1.

Figura 2.7: Janela criada pelo $\mu$A `monitorx11`

## 2.4 Terceira Geração: `eurobocup2000`

O principal facto a reter da terceira geração de *software* ISocRob foi um novo conjunto de pré-requisitos que tinha de satisfazer face à nova composição da equipa, composta agora exclusivamente por robôs Nomadic SuperScout II [10] (figura 2.8). Havia portanto que suportar todo o conjunto de sensores instalados de raiz, como um sensor de odometria, dezasseis sonars e dois conjuntos de três *bumpers* instalados na frente e na retaguarda do robô.

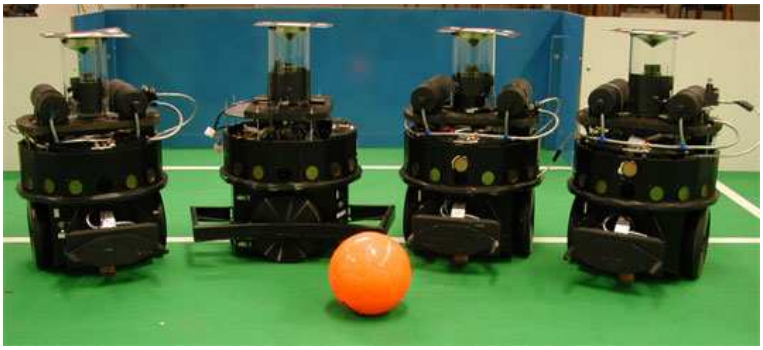
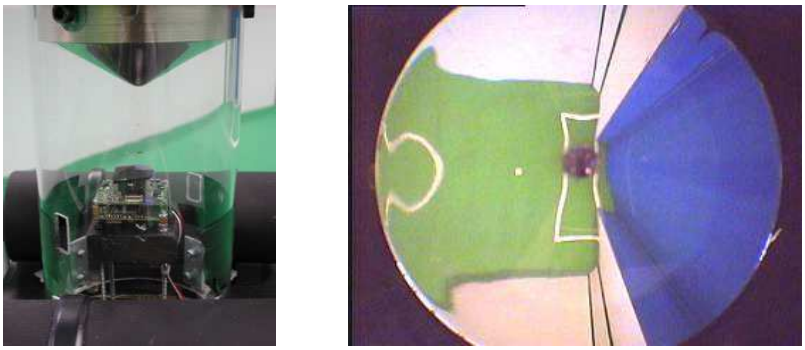


Figura 2.8: Equipa futebol robótico

Foram ainda feitas duas alterações adicionais ao nível de *hardware* que visam melhorar a percepção e desempenho do robô no ambiente de futebol robótico. Para facilitar a percepção do campo foi concebido um sistema catadióptrico constituído por espelho cónico e uma câmara disposta verticalmente (figura 2.9) que resulta numa imagem ortográfica do campo de futebol [11, 12]. Para efectuar um remate foi construído um sistema de chuto pneumático (figura 2.10) que foi colocado na frente do robô.

Um dos pressupostos dos $\mu$A é que toda a comunicação, sendo o sincronismo uma das forma possíveis de comunicação, era feita através dos dois mecanismos já referidos. Desta forma, na delegação das responsabilidades de *hardware* pelos $\mu$A é necessário ter o cuidado de atribuir a um único $\mu$A toda a responsabilidade associada a um determinado dispositivo ou recurso



(a) Detalhe (b) Imagem obtida

Figura 2.9: Conjunto câmara/espelho catadióptrico

(a) Topo (b) Lado

Figura 2.10: Dispositivo de chuto. Dois pares de "bigodes" permitem ao robô efectuar trajectórias curvilíneas mantendo a bola controlada. Bigodes desenvolvidos por Bruno Damas no âmbito do Projecto ISocRob

de *hardware*, evitando assim eventuais recursos a semáforos. Seguindo esta ideia, a atribuição foi a seguinte:

- **Câmara de cima**: Visto ambas as câmaras de visão estarem ligadas à mesma placa de captura de vídeo, continua apenas a existir um $\mu$A de visão, que no entanto tem de ser reprogramado para efectuar uma multiplexagem de qual o canal de visão que se encontra activo;

- **Dispositivo de Chuto**: O sinal de comando binário que regula o circuito eléctrico que controla o fluxo de ar, permitindo encher e vazar o pistão, é accionado pela porta paralela, dispositivo que não era usado anteriormente. Logo, foi criado um $\mu$A `kicker`;

- **Sensores e Actuadores da Scout**: O acesso ao microcontrolador da Scout é efectuado através da porta série, pelo que todo o *software* relacionado quer com os sensores (odometria, sonars e *bumpers*) quer com os actuadores (motores) deve ficar no mesmo $\mu$A.

Perante estas alterações de *hardware* importa esclarecer porque é que se procurou um novo modelo e se reescreveu a totalidade do código em detrimento de eventuais melhoramentos à arquitectura de *software* já existente. Esta opção deveu-se à percepção de que o contexto em que o projecto se encontrava era radicalmente diferente daquele que existia aquando a escrita do `robocup98`.

O facto dos robôs terem passado a dispor de uma câmara omnidireccional e um algoritmo de autolocalização, significava que era possível passar todos os dados para um referencial absoluto, comum a todos os robôs. Havia assim a noção de que estar a efectuar alterações significativas na estrutura de um código que já tinha sido objecto de modificações[7] iria conduzir a conflitos e contradições desnecessárias.

A principal motivação da segunda geração de *software* era dar os primeiros passos rumo a uma equipa de robôs cooperantes sendo que, para tal, era fulcral retirar complexidade do $\mu$A `machine`. A solução passava por delegar a responsabilidade de efectuar os cálculos necessários para deslocar o robô entre dois pontos para o $\mu$A `guidance`. Pretendia-se assim que este $\mu$A fosse uma colecção de algoritmos de condução, numa analogia a usar controladores P, PI ou PD para controlar um sistema conforme as situações.

[7] O código `robocup99` consiste em forçar todo o código `robocup98` na arquitectura de *software* pretendida, paralelizando os três blocos que na primeira geração eram efectuados em série.

Com base nesta motivação e face aos novos requisitos de *hardware*, a arquitectura `eurobocup2000` [13], ilustrada na figura 2.11, apresenta um conjunto de diferenças face à anterior embora mantendo o característico *STA Loop*. O $\mu$A `machine` passa a ser o centro nevrálgico das operações: em cada estado da máquina de estados é determinado qual a câmara a utilizar e qual o algoritmo de visão que deverá processar a imagem. Com base nas observações, o $\mu$A tem de escolher qual o algoritmo de condução desejado e colocar os dados relevantes no *blackboard*. É ainda o $\mu$A `machine` que, ao colocar essa instrução no *blackboard*, informa o $\mu$A `kicker` que deverá actuar sobre o dispositivo de chuto.

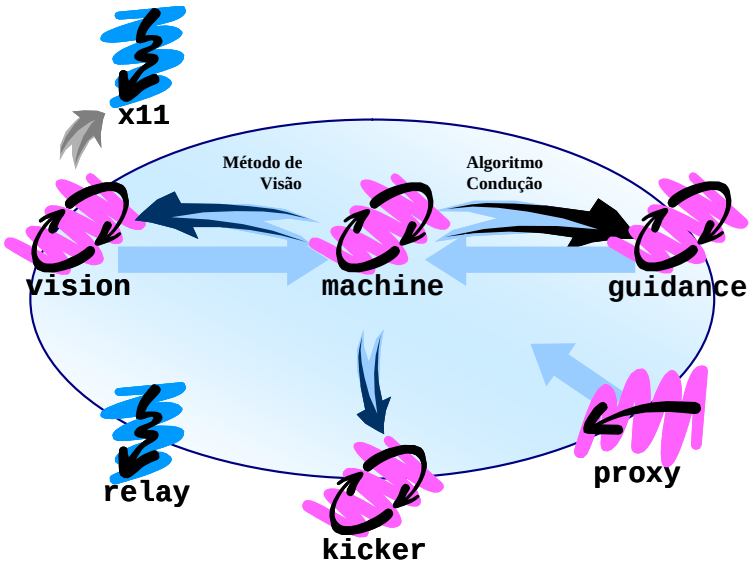



Figura 2.11: Arquitectura de *software* `eurobocup2000`

Uma grande diferença entre as duas arquitecturas de *software* está no diagrama temporal (figura 2.12): os três $\mu$A correspondentes às três componentes do STA Loop deixaram de estar sincronizados entre si. Desta forma, o $\mu$A `machine` efectua um maior número de iterações na máquina de estados por cada ciclo de aquisição e processamento do $\mu$A `vision`, possibilitando assim uma transição célere para um estado estável[8]. Uma vez que o sensor de odometria está à responsabilidade do $\mu$A `guidance` e atendendo à importância de o actualizar à máxima frequência possível[9], foi necessário não o

[8]Estável no sentido de que nas próximas iterações o valor dos predicados manterá o estado inalterado.

[9]De forma a efectuar a transformação de coordenadas do referencial do robô para o referencial partilhado por todos é essencial que os dados relativos à localização e orientação do robô sejam o mais actuais possíveis.

sincronizar com o $\mu$A `machine`.

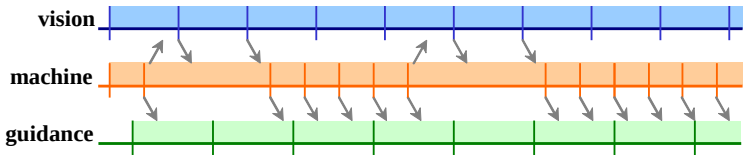



Figura 2.12: Diagrama Temporal `eurobocup2000`

Uma outra diferença visível no diagrama temporal é a existência para o $\mu$A `machine` de períodos com durações diferentes devido à interacção deste com o $\mu$A `vision`. Quando o $\mu$A `machine` informa, através do *blackboard*, o $\mu$A `vision` que deseja mudar de câmara e/ou algoritmo é necessário esperar que este termine o processamento da imagem actual e que efectue o processamento da primeira imagem capturada na nova câmara e/ou com um algoritmo diferente.

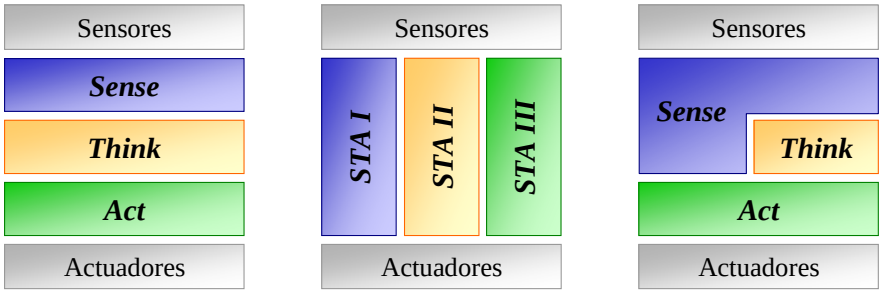
O segundo novo $\mu$A da arquitectura, o $\mu$A `relay`, foi inspirado num trabalho desenvolvido anteriormente no âmbito do Projecto ISocRob [14] ao possibilitar um acesso externo ao conteúdo do blackboard, permitindo a visualização e alteração do valor das chaves nele contido. Este $\mu$A será descrito com maior detalhe na secção 3.4.3.

## 2.5 Quarta Geração: `seattle2001`

Apesar da terceira geração de *software* ISocRob aproveitar muitas das ideias implementadas no baixo-nível que foram ensaiadas pela primeira vez na arquitectura `eurobocup2000`, esta falhou rotundamente o seu principal objectivo: conceber uma abstração adequada para o $\mu$A `machine`, que evitasse as interacções complexas com os outros $\mu$A, especialmente o $\mu$A `vision`.

A diferença da nova arquitectura `seattle2001` relativamente às anteriores, sendo que esta foi a segunda desenvolvida no âmbito do presente trabalho, prende-se com o significado dado aos dois últimos blocos do *STA Loop*, numa leitura diferente dos verbos *Think* e *Act*. O salto qualitativo foi dado ao distinguir entre dois tipos de decisões: uma, de mais alto nível, em que o agente decide **o que** fazer e outra em que é decidido **como** o fazer. A primeira determina qual o objectivo a realizar e a segunda determina a melhor forma de o atingir.

À luz desta nova interpretação, o bloco *Think* do agente decide qual o comportamento a exibir perante o estado actual do mundo: tendo a bola procura marcar golo, avistando a bola aproxima-se da mesma. Esta intenção




(a) Horizontal (b) Vertical (c) Misto

Figura 2.13: Dos sensores aos actuadores

é indicada ao bloco *Act* que, de acordo com a percepção do meio ambiente determina a melhor forma de levar a cabo o comportamento pretendido: se é mais fácil marcar golo do lado esquerdo do guarda-redes, qual a trajectória que deverá efectuar de aproximação à bola.

A consequência desta nova leitura é que parte do cálculo anteriormente efectuado no $\mu$A `machine` foi delegado para o $\mu$A `control`, que passa assim a depender simultaneamente do processamento sensorial efectuado pelo $\mu$A `vision` e pela primeira camada de decisão do $\mu$A `machine`.

O caminho seguido é um misto da solução clássica utilizada na robótica, estrutura composta por sucessivas camadas dos sensores aos actuadores como ilustrado na figura 2.13(a), com uma estrutura vertical [15], composta por camadas que efectuam ligações directas entre os sensores e actuadores (figura 2.13(b)). Baseando-se assim na filtragem de informação, compreensão do meio e análise de características associadas à solução clássica, introduz-se o conceito de ritmos de decisão distintos ao providenciar dois caminhos alternativos dos sensores aos actuadores: um directo pelas camadas *Sense* e *Act*, para alterações rápidas e subtis no meio ambiente, e um comparativamente mais lento, para maiores mudanças de contexto, passando pelas três camadas do *STA Loop* (figura 2.13(c)).

Assim, foi possível delegar para o uA `control` o controlo sobre a câmara de visão e algoritmo de processamento, bem como a decisão sobre quando activar o dispositivo de chuto. O $\mu$A `machine` passa assim a funcionar como uma caixa-negra: retira do *blackboard* a informação sensorial desejada e coloca no *blackboard* única e exclusivamente o comportamento desejado. Esta abstração tem as seguintes consequências:

- Uma vez que não é necessária a determinação da câmara activa nem

efectuar o cálculo da actuação ao nível do $\mu$A `machine`, fica mais fácil a tarefa de estabelecer o comportamento a exibir pelo robô;

- Passa a ser possível pensar num modelo formal de cooperação entre os agentes robóticos.

A segunda diferença assinalável da arquitectura `seattle2001` foi a introdução do terceiro conceito que, juntamente com os $\mu$A e o *blackboard*, completa o esqueleto da arquitectura que depois será devidamente instanciado para o problema desejado. Este novo conceito surge pela observação de que determinados $\mu$A, nomeadamente os $\mu$A `vision` e `control`, são meros multiplexadores de módulos, ou *plugins*.

Como tal, esta função do $\mu$A foi explorada de forma a que este não tenha qualquer conhecimento do funcionamento interno dos seus módulos. A responsabilidade do $\mu$A passa assim por identificar simbolicamente qual o *plugin* desejado, e de efectuar chamadas às funções por este implementadas.

A maior vantagem dos módulos, reside, não no facto de ser trivial acrescentar novas componentes a um $\mu$A, mas ao permitir que *plugins* antigos sejam transparentemente substituidos por novos, sem que tais alterações obriguem a modificações noutros $\mu$A que dependiam dos primeiros.

Nas secções 3.1 e 3.2 serão apresentados os *plugins* de algoritmos de visão do $\mu$A `vision` e os *plugins* de comportamentos individuais do $\mu$A `control`, respectivamente.

À arquitectura `seattle2001` [16] foram ainda adicionados dois $\mu$A: o $\mu$A `heartbeat` que actualiza, uma vez por segundo, o valor de uma chave no *blackboard* global e que serve para os outros agentes saberem quais os agentes que estão "vivos"; o $\mu$A `halt`, explicado mais detalhadamente na secção 3.4.5, que monitoriza os pequenos movimentos relativos de um rato USB colocado na frente do robô de forma a detectar quando este está em movimento, visando assim resolver o problema de quando o robô está parado mas tem os motores em esforço.

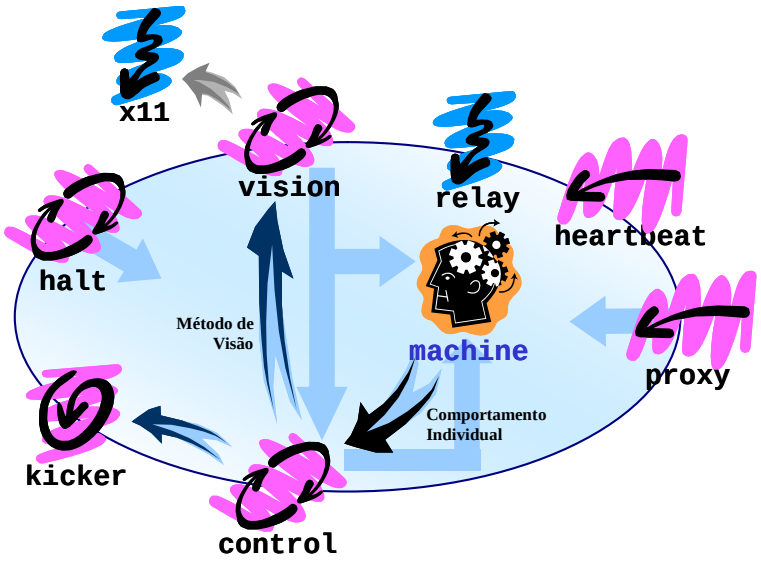



Figura 2.14: Arquitectura de *software* `seattle2001`

# Capítulo 3

# Implementação da Arquitectura de *Software*

Pretende-se no presente capítulo desenvolver ao pormenor os $\mu$A da arquitectura `seattle2001`, apresentados na secção 2.5, quanto à sua estrutura e às suas responsabilidades, assim como detalhar as interacções entre estes de forma a estabelecer-se o funcionamento geral da arquitectura, dando particular ênfase aos três $\mu$A principais: `vision`, `control` e `machine`.

Em engenharia de *software* a tarefa de implementar uma arquitectura de *software* pode ser executada de duas formas: a correcta, ao conceber um modelo em que se identificam e se relacionam as várias componentes e posteriormente se implementa a solução, ou a errada ao dar esses dois passos pela ordem inversa. A implementação da arquitectura utilizada para o `eurobocup2000` foi feita desta última forma, atendendo ao curto espaço de tempo de que se dispunha para participar na competição. Verificou-se que, apesar da arquitectura, apresentada na secção 2.4, estar bem definida aquando da sua escrita, o mesmo não acontecia com o arranjo interno dos $\mu$A, resultando assim em indefinições que se resolviam pontualmente.

A arquitectura do código `seattle2001` usufruiu da experiência recolhida durante o ciclo de desenvolvimento do código da primeira, concluindo-se que quaisquer indefinições possibilitavam o desenvolvimento de situações que futuramente iriam colidir com o objectivo pretendido. Assim, teve-se o cuidado de precisar minuciosamente as funções dos $\mu$A e dos módulos neles inseridos.

## 3.1 $\mu$A `vision`

O $\mu$A `vision`, responsável pela aquisição e processamento de imagem, é conceptualmente dos $\mu$A mais fáceis de descrever pois não necessita de qual-

quer interacção com outro $\mu$A. A sua responsabilidade na sociedade de $\mu$A que compõem o agente robótico é a de actualizar, com a máxima frequência possível, os dados extraídos da imagem contidos no *blackboard*. Contudo, a sua estrutura interna, originalmente muito simples, sofreu alterações de fundo por forma a poder multiplexar entre as duas câmaras ligadas à mesma placa de aquisição. Consequentemente, era necessário dispor de um mecanismo que permitisse a execução de diferentes algoritmos de processamento de visão em ambas as câmaras.

Na arquitectura `seattle2001` esta multiplexagem de câmaras e algoritmos de visão foi abstraída de tal forma que o $\mu$A apenas necessita de multiplexar entre câmaras virtuais, que depois são mapeadas para o *hardware*. Desta forma, aos quatro algoritmos de processamento de visão desenvolvidos, no âmbito do projecto ISocRob, correspondem as seguintes câmaras virtuais (figura 3.1):

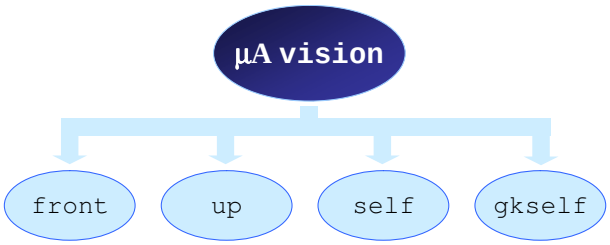



Figura 3.1: $\mu$A `vision`

- `front`: Numa imagem da câmara da frente são discriminados os pixeis laranja, amarelo e azul de forma a actualizar os dados referentes à bola e às balizas, respectivamente. A posição relativa da bola é estimada com base no centro de massa e do número de pixeis laranja. Para as balizas é determinado o ponto médio do maior bloco de pixeis, por força da oclusão resultante da presença do guarda-redes;

- `up`: Numa imagem da câmara de cima são discriminados os pixeis laranja. Com base em dois centros de massa em janelas progressivamente mais pequenas, é determinado qual o pixel com a menor distância ao centro da imagem. Devido à projecção ortográfica efectuada pelo espelho, se esse pixel pertencer à bola, então deverá indicar a posição do centro da bola;

- `self`: Numa imagem capturada na câmara de cima, são discriminadas as transições de pixeis branco/verde e verde/branco. Após a aplicação da transformada de Hough, obtêm-se as rectas do campo que, com

base num conhecimento prévio das verdadeiras dimensões do campo, permitem estimar a posição e orientação do robô no campo;

- `gkself`: Consiste num segundo método de autolocalização usando a câmara de cima, mas concebido especificamente para os jogadores que se encontrem dentro da grande área, face ao reduzido número de marcações do campo visíveis dessa posição. Com base nas transições amarelo/branco e branco/amarelo ou azul/branco e branco/azul, conforme a baliza em que o robô se encontre, são obtidas as duas rectas correspondentes aos postes da baliza que através de equações trigonométricas permitem estimar a posição e orientação do robô no campo.

O primeiro aspecto a reter é que face ao excelente sensor de odometria e havendo a possibilidade de reavaliar periodicamente a posição real do robô através destes últimos dois algoritmos de autolocalização, é possível transformar as estimativas da posição da bola do referencial do robô para um referencial fixo no campo (figura 3.2). Para um robô na posição $(x_R, y_R, \theta_R)$ no referencial do campo, uma observação local $(x', y')$ corresponde a um objecto na posição $(x, y)$, equação 3.1.

$$\begin{cases} x &= x_R + x' \cos\theta_R + y' \sin\theta_R \\ y &= y_R - x' \sin\theta_R + y' \cos\theta_R \end{cases} \tag{3.1}$$

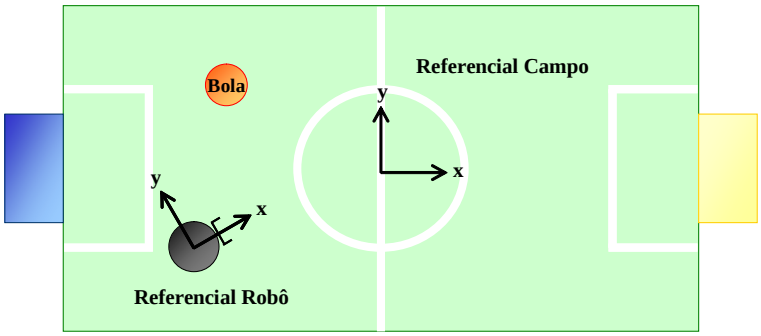



Figura 3.2: Referenciais do campo e do robô

Visto ser possível esta transformação, as estimativas da localização da bola obtidas por vários algoritmos são comparáveis, pelo que passou a ser efectuada uma fusão sensorial da posição da bola, calculada pelos vários algoritmos de visão. O $\mu$A fica assim responsável pelo armazenamento dos dados relativos às últimas posições da bola e por colocar no *blackboard* a última posição da bola e uma estimativa da sua velocidade com base no conjunto de valores disponível.

Os módulos das câmaras virtuais foram implementados através de *shared objects*, o equivalente UNIX das *Dynamic Loaded Libraries* (DLL) do *Windows*. Estes são localizados no arranque do programa e incorporados no programa principal caso possuam todas as funções de um *plugin* de visão, funções essas que são:

- `getId`: Função que devolve o símbolo que identifica o *plugin*. Ao ser colocado no *blackboard*, pelo $\mu$A `control`, o símbolo da câmara virtual que deverá ser activada, o $\mu$A `vision` procede à sua comparação com o valor devolvido pela função `getId` de cada *plugin*. Como estão dispostos num vector, o primeiro *plugin* cujo símbolo seja identificado será o escolhido. Daqui se constata que as identificações dos *plugins* não necessitam de ser únicas, uma vez que o $\mu$A escolherá sistematicamente o primeiro candidato;

- `getChannel`: Função que devolve qual a câmara que a câmara virtual utiliza e a dimensão da imagem a adquirir. A dimensão da imagem é um aspecto determinante, uma vez que envolve o compromisso entre obter maior precisão dos dados recorrendo a uma imagem maior e o custo de tempo necessário para a processar;

- `init`: Função que é chamada uma única vez no arranque do $\mu$A, sendo o local indicado para efectuar a alocação da memória dinâmica necessária;

- `destroy`: Função chamada quando termina o processamento do $\mu$A e que visa libertar a memória alocada na função `init`.

- `change`: Sempre que o $\mu$A muda, por indicação do $\mu$A `control`, para uma nova câmara virtual, é efectuada uma única chamada à função `change` desse *plugin*. Não obstante não ser usada em nenhum dos *plugins*, é útil para algoritmos que vão construindo um historial de um determinado valor ao permitir limpar esse historial antes da primeira chamada ao algoritmo. A título de exemplo, poder-se-ia começar por tentar detectar a bola numa janela pequena em torno da última observação;

- `method`: Função contendo o algoritmo de processamento, recebendo como argumento uma imagem obtida da câmara correcta. A única condição imposta sobre esta função é a de informar o $\mu$A no caso de ter observado a bola, e em caso afirmativo qual a posição da bola no referencial do campo;

- `show`: Uma função de depuração visual do algoritmo que não é chamada pelo $\mu$A `vision` mas pelo $\mu$A `x11`. Esta função será descrita na secção 3.4.2 onde o $\mu$A `x11` é apresentado em pormenor.

Este mecanismo de *plugins* é versátil visto não haver por parte do $\mu$A qualquer conhecimento sobre o *plugin* que está a executar. É assim possível que um *plugin* seja substituído por outro equivalente, retirando o *plugin* indesejado e alterando o novo de forma a que este se identifique da mesma forma que o primeiro.

A interacção do $\mu$A `vision` com o $\mu$A `control` é relativamente simples, estando ilustrada na figura 3.3. Quando o $\mu$A `control` deseja mudar de câmara virtual activa informa, pelo *blackboard*, o $\mu$A `vision` qual o *plugin* a activar e coloca-se prontamente num estado dormente à espera que lhe seja enviado um sinal. O $\mu$A de visão apenas toma conhecimento do desejo do $\mu$A `control` no fim do processamento da última imagem adquirida, mas não envia o sinal imediatamente para garantir que o *blackboard* ainda disponha de dados actualizados. Como tal, só após o processamento de uma primeira imagem é que o $\mu$A envia o sinal, reactivando o $\mu$A `control`.

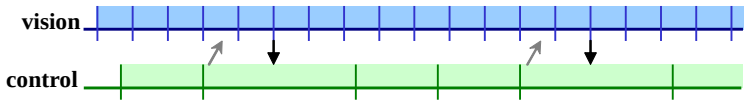



Figura 3.3: Interacção do $\mu$A `vision` com o $\mu$A `control`

O último pormenor a realçar relativamente ao $\mu$A de visão reside num módulo de *software* desenvolvido de forma a acelerar a discriminação de cores dos *pixeis* baseada numa segmentação no espaço de côr HSV. Para tal, é necessário efectuar o seguinte conjunto de passos:

- A imagem é adquirida da placa de aquisição Bt848 no formato RGB555 (figura 3.4), implicando que para cada pixel é necessário extrair dos dois *bytes* as três componentes do espaço RGB, $R = r_4 \cdots r_0$, $G = g_4 \cdots g_0$ e $B = b_4 \cdots b_0$;

- Efectuar uma transformação do espaço de cor de RGB para HSV através da equação 3.2 [17];

$$\begin{cases} H & = & \begin{cases} \frac{G-B}{\max\{R,G,B\}-\min\{R,G,B\}} & \text{se } R = \max\{R,G,B\} \\ 2 + \frac{B-R}{\max\{R,G,B\}-\min\{R,G,B\}} & \text{se } G = \max\{R,G,B\} \\ 4 + \frac{R-G}{\max\{R,G,B\}-\min\{R,G,B\}} & \text{c.c.} \end{cases} \\ S & = & \frac{\max\{R,G,B\}-\min\{R,G,B\}}{\max\{R,G,B\}} \\ V & = & \max\{R,G,B\} \end{cases} \tag{3.2}$$

- Cada cor $C^i$ é definida pelo tuplo $\langle H^i_{min}, H^i_{max}, S^i_{min}, S^i_{max}, V^i_{min}, V^i_{max} \rangle$, contendo os valores máximos e mínimos de cada componente do espaço de cor HSV. Um pixel é da cor $C^i$ se e só se as seguintes desigualdades forem satisfeitas:

$$\begin{cases} H^i_{min} \leq H \leq H^i_{max} \\ S^i_{min} \leq S \leq S^i_{max} \\ V^i_{min} \leq V \leq V^i_{max} \end{cases} \tag{3.3}$$

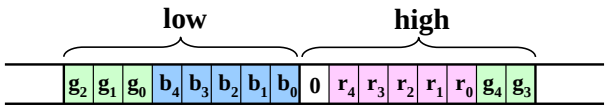



Figura 3.4: Pixel RGB555

Uma das principais inovações do código `seattle2001` consistiu em efectuar numa única operação todos os passos acima descritos, recorrendo a um vector que estabelece uma relação directa entre os dois *bytes* do *pixel*, no formato RGB555, e a cor correspondente. As três consequências directas desta inovação foram:

- A segmentação da imagem passou a ser feita muito rapidamente, permitindo assim que os algoritmos sejam no futuro mais complexos, mas mantendo uma frequência de captura elevada;
- A abstração efectuada permite que os *plugins* deixem de ter qualquer conhecimento do modo como a cor do *pixel* é determinada;
- A possibilidade de desenvolver uma interface gráfica em que o utilizador determina os seis parâmetros de cada cor gerando assim os dois vectores, um para cada câmara.

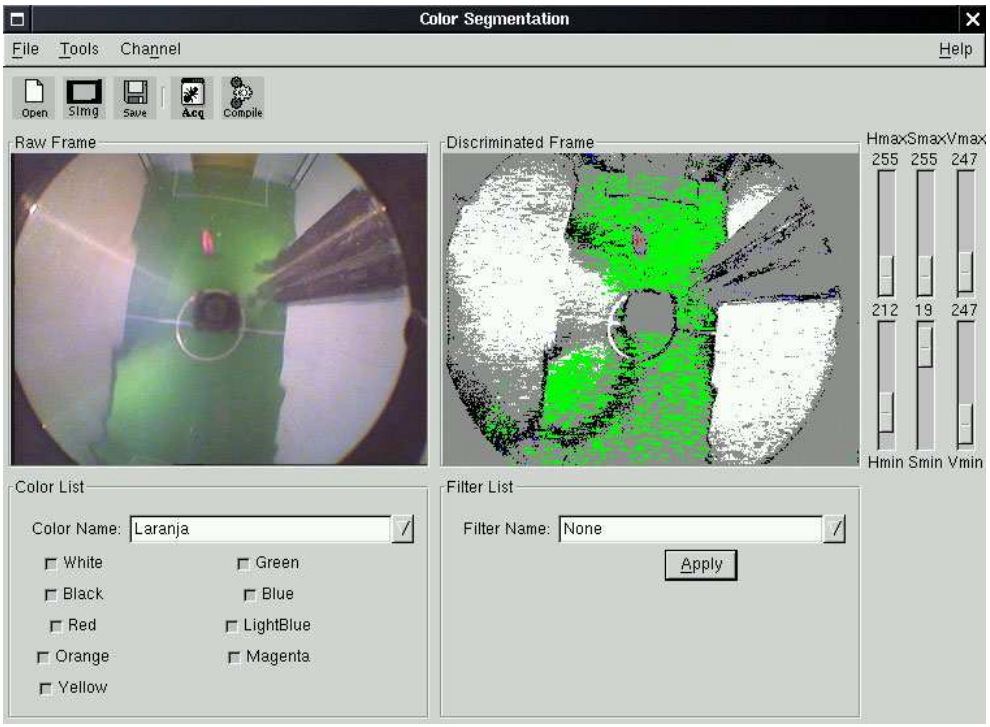



Figura 3.5: Interface gráfica de segmentação de cor. Aplicação desenvolvida por Manuel Lopes no âmbito do Projecto ISocRob

## 3.2 $\mu$A `control`

À parte das interacções com outros $\mu$A, a estrutura do $\mu$A `control` é, em tudo, idêntica à do $\mu$A `vision`, mas em que os *plugins* são comportamentos individuais em vez de algorimos de processamento de visão. Como tal, a principal responsabilidade do $\mu$A consiste em multiplexar entre os vários comportamentos pretendidos pelo $\mu$A `machine` (figura 3.6). Os onze comportamentos individuais implementados são:

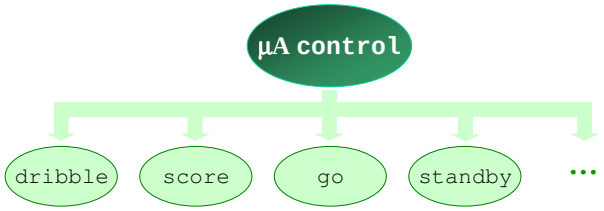



Figura 3.6: $\mu$A `control`

- `go`: Desloca o robô para uma região do campo. Ao contrário da interacção entre os $\mu$A `machine` e `guidance` do código `eurobocup2000`, em que a posição desejada era explicitamente indicada pelo $\mu$A `machine`, no código `seattle2001` foi dado o primeiro passo rumo a uma descrição simbólica das zonas do campo. De momento, apenas foram definidos dois símbolos: `HOME` e `EMPTYSPOT` que levam o robô para a sua posição base ou para uma zona adjacente que é considerada mais vazia, respectivamente. Contudo, é ilustrado na figura 3.7 uma possível codificação futura de todo o campo;

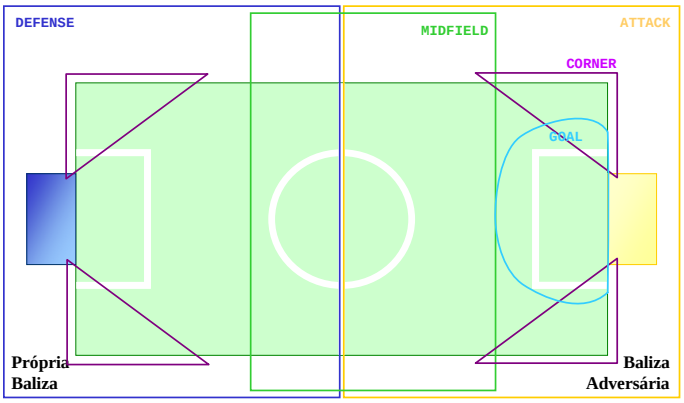



Figura 3.7: Símbolos descrevendo zonas do campo

- `dribble`: Um comportamento em tudo idêntico ao `go`, mas em que são introduzidas restrições na velocidades linear e angular do robô de forma a manter a bola sempre encostada ao dispositivo de chuto;

- `approachball`: Efectua uma aproximação à bola. Em situações típicas, este comportamento será seguido pelo `dribble`, pelo que o comportamento `approachball` tem de se aproximar de tal forma que permita colocar a bola dentro dos bigodes laterais do dispositivo de chuto. O problema principal deste comportamento é na postura que este deve assumir numa aproximação a uma bola que se encontre encostada a uma parede;

- `clearball`: O objectivo deste comportamento é de afastar a bola da própria baliza, numa atitude defensiva. De momento, é utilizado quando um robô, por força das limitações da sua dinâmica, já não pode prosseguir com uma acção de `dribble`, mas poderá ser futuramente utilizada para fins exclusivamente defensivos;

- `score`: O comportamento de efectuar um remate à baliza é um caso extremo de condução da bola, em que, ao contrário do `dribble`, o objectivo consiste em imprimir na bola uma tal velocidade que a mesma saia do dispositivo de chuto, preferencialmente para dentro da baliza do adversário. A ideia subjacente ao comportamento actual é a de que o robô deverá empurrar a bola, ao mesmo tempo que acelera, em direcção ao centro da maior região da baliza adversária não coberta pelo guarda-redes. Uma vez atingida uma determinada distância da baliza, é efectuado um chuto e o robô é parado;

- `nothing`: Como o nome sugere, o agente robótico a executar este comportamento não faz nada. Este comportamento é utilizado enquanto o robô aguarda o início da partida;

- `selflocalize`: Tal como o `nothing`, também este comportamento se reflecte numa ausência de movimento por parte do robô. O comportamento `selflocalize` é uma consequência directa de ser o $\mu$A `control` a decidir qual a câmara virtual que deve estar activa no $\mu$A `vision`. Conceptualmente, no entanto, este é um comportamento individual perfeitamente aceitável, sendo idêntico à reacção dos humanos quando estão perdidos ou perante uma situação inesperada;

- `standby`: Comportamento em que o robô permanece parado, rodando sobre si próprio de forma a manter a bola centrada na imagem adquirida na câmara frontal. É um comportamento necessário face à regra do

futebol robótico que obriga a que sejam retirados do campo robôs que não se mexam há mais de trinta segundos;

- `halt`: Este comportamento visa solucionar um problema que até ao código de `seattle2001` se revelou de difícil resolução. Este problema e a solução encontrada serão apresentados na secção 3.4.5;

- `goalkeeper`: Defender a baliza, impedindo que a bola seja introduzida na própria baliza, é um comportamento que tem de fazer face a duas restrições: o campo de visão do robô e o tempo de resposta perante um remate forte. Para colmatar o primeiro problema, a câmara frontal do guarda-redes encontra-se montada no lado direito, de forma a que quando o robô se movimenta paralelamente à linha de fundo, a câmara esteja virada para o centro do campo. Esta alteração veio melhorar o comportamento do robô quando a bola se encontra na área de oclusão da câmara de cima, estando, no entanto, visível na câmara frontal embora com uma grande oclusão dos lados (figura 3.8). O segundo problema foi resolvido no código `seattle2001` ao obter-se a frequência máxima no $\mu$A `vision`, permitindo assim detectar mais cedo a trajectória da bola;

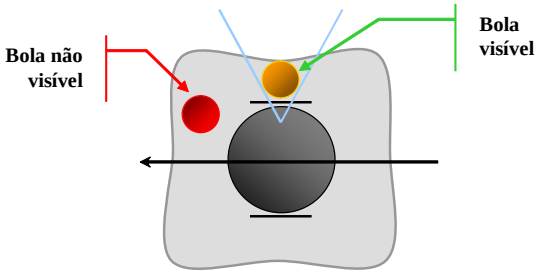



Figura 3.8: Oclusão

- `teamkeeper`: Este comportamento é idêntico ao `goalkeeper`, mas sem recorrer à câmara da frente. Apesar de não ser utilizado de momento, a intenção é de que, caso o guarda-redes seja retirado do campo, qualquer outro robô o possa substituir na função de guarda-redes.

Tal como no $\mu$A `vision`, também há um elemento comum entre todos os *plugins* do $\mu$A `control`: a actuação dos mesmos. Esta fica a cargo do $\mu$A que com base na indicação dos *plugins* envia o comando pela porta-série. Como os acessos pela porta-série são muito lentos – cada comando enviado para o microcontrolador do robô demora $40ms$ – foi possível reduzir

o número de actuações que o robô efectua ao comparar a actuação actual desejada pelo *plugin* e a última actuação enviada. Se a diferença for inferior a $0{,}1ms^{-1}$, a nova actuação é ignorada ganhando assim preciosos milisegundos de execução.

Uma outra diferença relativamente aos *plugins* de visão é a lista de funções dos *plugins* dos comportamentos. As seis funções exportadas pelo *plugin* são:

- `getID`, `init` e `destroy` que servem o mesmo propósito que as funções do mesmo nome dos *plugins* de visão;
- `method` é uma função onde é implementado o comportamento. Apesar de não haver qualquer restrição quanto à forma de implementação, todos os comportamentos foram codificados através de máquinas de estados, aproveitando, desta forma, muito do código de $\mu$A `machine` do `eurobocup2000`;
- `changeTo` e `changeFrom` são duas funções que são chamadas imediatamente antes da primeira chamada à função `method` do *plugin* e após a última chamada a essa mesma função. A primeira é usada para estabelecer o estado inicial da máquina de estados, bem como para impor quaisquer condições iniciais à dinâmica do robô. Por exemplo, no comportamento `selflocalize` o robô é imobilizado na chamada à função `changeTo`. A função `changeFrom` não é utilizada por nenhum *plugin* actualmente, mas poderá servir para forçar uma última actuação sobre os motores, na transição para o próximo comportamento.

Sendo o centro nevrálgico das operações, o $\mu$A `control` interage com três $\mu$A da seguinte forma:

- Determina a camâra virtual activa, comandando o $\mu$A `vision` da forma já referida na secção 3.1;
- Activa o dispositivo de chuto enviando sinais ao $\mu$A `kicker`, como será descrito na secção 3.4.4;
- É comando pelo $\mu$A `machine` que indica ao $\mu$A qual o comportamento pretendido.

Resta assim explicar esta última interacção, semelhante à relação entre os $\mu$A `vision` e `control` (figura 3.9). Quando o $\mu$A `machine` deseja efectuar uma mudança do comportamento exibido, coloca no *blackboard* o símbolo do novo *plugin* passando de seguida a um estado dormente, aguardando a recepção do sinal vindo do $\mu$A `control`. Apenas ao terminar a chamada à

função `method` do comportamento actual é que o $\mu$A `control` tem conhecimento da indicação do $\mu$Amachine efectuando os seguintes passos:

- uma chamada à função `changeFrom` do *plugin* actual, terminando assim a execução do comportamento actual;
- uma chamada à função `changeTo` do novo *plugin*, iniciando a execução do novo comportamento;
- desbloqueia o $\mu$A `machine` ao enviar-lhe um sinal.

A diferença para a relação entre os $\mu$A `control` e `machine` reside no conceito de medida de desempenho associado aos *plugins* de comportamento: estes executam até terminarem, com ou sem sucesso, os seus objectivos. Como tal, um comportamento pode ser descrito como estando em um de três estados possíveis: a decorrer, terminado com sucesso ou terminado sem sucesso. Em cada chamada à função `change`, o *plugin* terá de colocar no *blackboard* em qual dos três estados se encontra.

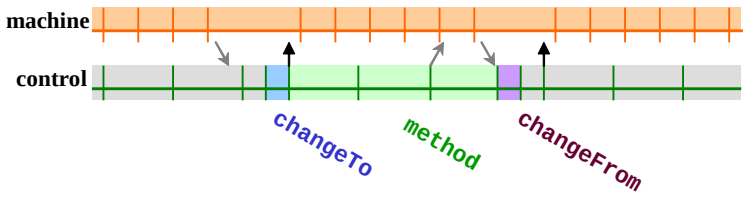



Figura 3.9: Interacção do $\mu$A `control` com o $\mu$A `machine`

## 3.3 $\mu$A `machine`

O $\mu$A `machine` foi concebido como a caixa-negra do robô, um bloco que extrai do *blackboard* a informação sobre o estado actual do jogo decidindo com base nessa percepção do mundo qual o comportamento que o robô deverá exibir. Face a esta descrição, é evidente que este $\mu$A apresenta o maior grau de liberdade uma vez que apenas interage, de uma forma muito simples, com o $\mu$A `control`, tal como foi descrito na secção anterior.

Apesar de não ser relevante para a melhor compreeensão da arquitectura, uma vez que o seu principal objectivo era precisamente abstrair a decisão de alto-nível, fica aqui a descrição de como o $\mu$A `machine` foi implementado no código `seattle2001` (figura 3.10). O robô poderá estar em um de três estados:

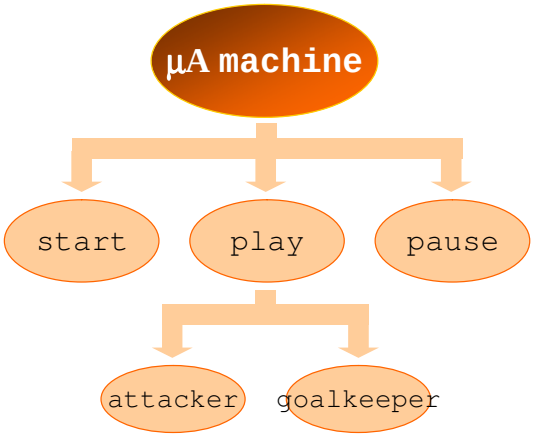



Figura 3.10: $\mu$A `machine`

- `Pause`: O robô encontra-se parado à espera de uma indicação de qual será o próximo estado;
- `Start`: O robô está a dirigir-se para a sua posição inicial;
- `Play`: O robô está a jogar em uma de duas posições: robô de campo ou guarda-redes.

Estes três estados são comuns a todos os robôs, havendo um programa externo que permite transitar entre eles da forma assinalada na figura 3.11.

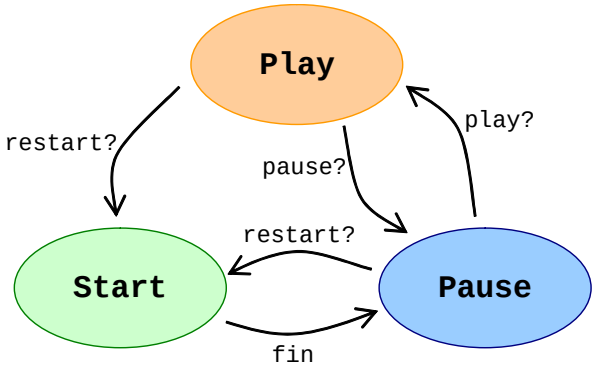



Figura 3.11: Estado do jogo

A figura 3.12 ilustra a máquina de estados actual de um atacante. É importante frisar que apesar do maior número de estados e transições, a sua codificação foi extremamente simples.

O principal ponto a reter da implementação do $\mu$A `machine` é de que devido à simplicidade de codificação da máquina de estados, foi possível conceber uma mais complexa (figura 3.12). A cada estado corresponde a execução de um comportamento no $\mu$A `control`, sendo que a maior parte das transições é efectuada com base na medida de desempenho do comportamento a ser executado.

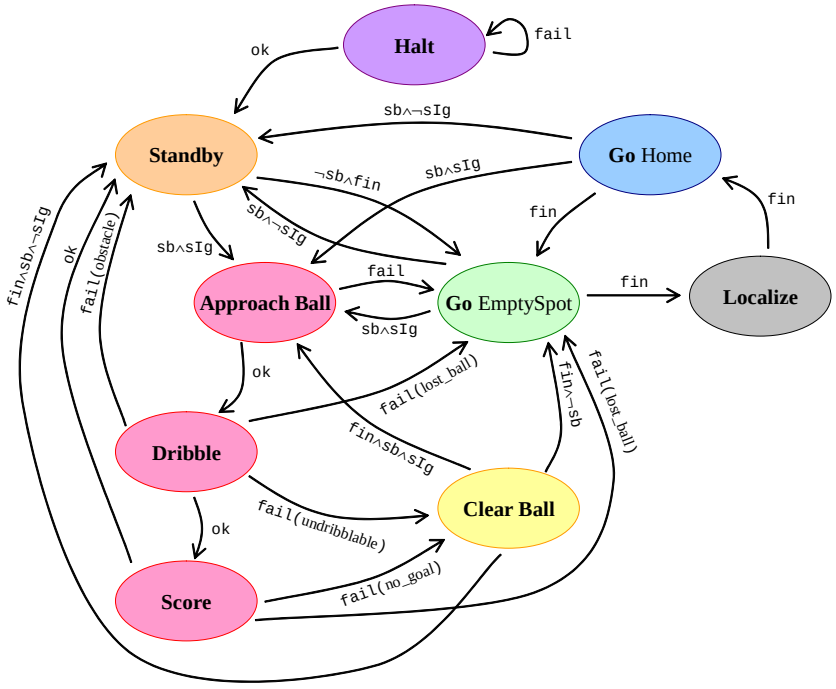



Figura 3.12: Máquina de estados atacante. Consultar tabela 3.1

## 3.4 Restantes $\mu$A

Apresentados os três principais $\mu$A resta descrever os restantes que compõem a arquitectura de *software* `seattle2001`.

### 3.4.1 $\mu$A `proxy`

O $\mu$A `proxy` trata de toda a comunicação, por rede *wireless*, entre os agentes robóticos, estabelecendo um servidor que intercepta toda a comunicação *multicast* efectuada para o porto 2000. De momento, a única forma de comunicação implementada é a partilha de chaves de *blackboard* distribuído, mas

| Sigla | Predicado | Descrição |
|---|---|---|
| `sb` | `see_ball?` | Verdadeiro caso o robô esteja a ver a bola |
| `fin` | `behaviour_finished?` | Verdadeiro caso o robô tenha terminado o comportamento |
| `ok` | `behaviour_success?` | Verdadeiro caso o robô tenha terminado com sucesso o comportamento |
| `fail` | `behaviour_failure?` | Verdadeiro caso o robô tenha terminado sem sucesso o comportamento |
| `sIg` | `should_i_go?` | Verdadeiro caso o robô seja o que se encontre mais perto da bola (ver secção 4.1) |

Tabela 3.1: Predicados

| Mensagem | Descrição |
|---|---|
| ≪*type*`\|`*body*≫ | Mensagem *multicast* do tipo *type* com a informação *body* |
| ≪`BB\|`*key*§*type*§*value*≫ | Actualização da chave *key* com o novo valor *value* do tipo *type* |
| ≪`MB\|`*to*§$from$§*msg*≫ | Uma mensagem de correio destinada à *mailbox* do agente *to* vinda do agente $from$ |

Tabela 3.2: Mensagens *multicast*

havendo a possibilidade de recorrer a formas alternativas de comunicação, como por exemplo através de caixas de correio ou *mailboxes*. O formato das mensagens *multicast* enviadas pelos agentes foi entendido de forma a encapsular vários tipos de comunicação. Este formato e dois exemplos de mensagens *multicast* são apresentados na tabela 3.2.

É importante desenvolver o termo de *blackboard* distribuído visto não se tratar de uma única entidade centralizadora, mas de uma cópia local, em cada agente, de um conjunto de chaves partilhadas por todos. Quando uma chave partilhada é actualizada, a cópia local não é actualizada directamente uma vez que a mensagem *multicast* enviada será interceptada pelo $\mu$A `proxy` do próprio agente.

Numa rede *ethernet*[1] o tempo de difusão de mensagens é negligenciável desde que a rede não esteja saturada. Caso a ocupação da largura de banda de canal exceda os 30% de utilização haverá atrasos na recepção das mensagens,

[1]A rede *wireless* 802.11b tem um desempenho semelhante a uma rede *ethernet* desde que a distância entre pontos seja inferior a 25m.

| Comando | Descrição |
|---|---|
| `ADDW` *key* | Adiciona a chave *key* ao conjunto de chaves cujo valor é periodicamente enviado ao cliente |
| `DELW` *key* | Remove a chave *key* do conjunto de monitorização |
| `LSTW` | Lista de chaves monitorizadas |
| `CLRW` | Apaga a lista de chaves monitorizadas |
| `SETV` *key*§*type*§*value* | Actualiza a chave *key* com o valor do tipo *type* |
| `GETV` *key* | Devolve o valor actual da chave *key* |
| `QUIT` | Termina a ligação entre cliente e servidor |

Tabela 3.3: Comandos servidor $\mu$A `relay`

havendo ainda a possibilidade de se perderem algumas.

### 3.4.2 $\mu$A `x11`

O $\mu$A `x11` é a principal forma de depuração dos *plugins* de visão, revelando a última imagem capturada e representando graficamente nesta os dados extraídos da mesma.

No arranque do $\mu$A ele cria uma janela em XWindows por cada câmara virtual especificada na linha de comando do programa principal, já com as dimensões correctas das imagens capturadas nessa câmara virtual. De seguida, e atendendo à câmara virtual que se encontre activa, o $\mu$A `x11` efectua uma chamada à função `show` do *plugin* correspondente.

### 3.4.3 $\mu$A `relay`

O $\mu$A `relay` é um mecanismo simples de depuração de código ao interagir com os clientes que se ligam ao servidor estabelecido no porto 2001, satisfazendo os pedidos de acesso para leitura ou escrita no *blackboard* local do agente. Através de um protocolo muito simples de comandos é possível visualizar o valor actual de uma chave, monitorizar um conjunto de chaves e alterar o valor das chaves. É importante realçar que alterações a chaves partilhadas, pertencentes ao *blackboard* distribuído, não são difundidas, pelo que a alteração não é propagada para os restantes *blackboards*. A lista de comandos implementados é apresentada na tabela 3.3.

Com base neste protocolo foi desenvolvida uma interface gráfica para monitorizar todos os robôs simultaneamente (figura 3.13). Esta aplicação

estabelece uma ligação com o servidor criado pelo $\mu$A `relay` de todos os agentes e coloca na lista de monitorização um conjunto de chaves que descrevem o estado interno do robô. Dados como posição e orientação do robô, o comportamento activo e o estado da máquina de estados do *plugin*, bem como os dados do robô sobre a bola são visualizados de uma forma gráfica permitindo uma observação simultânea de todos os robôs.


Figura 3.13: Interface gráfica para monitorização dos robôs. Aplicação desenvolvida por Rodrigo Ventura no âmbito do Projecto ISocRob

### 3.4.4 $\mu$A `kicker`

O $\mu$A `kicker` está quase sempre dormente à espera de um sinal vindo do $\mu$A `control`. Até este momento apenas se fez referência a um dos dois sinais[2] que podem ser enviados para o $\mu$A: este faz com que o pistão se extenda e retraia rapidamente, numa acção denominada por chuto.

No entanto, desenvolveu-se outro comportamento que é efectuado ao receber o outro sinal: se o pistão estiver recolhido o $\mu$A coloca o valor '`1`' em todos os pinos da porta paralela, activando o circuito externo que controla o fluxo de ar extendendo o pistão. Uma vez actuada a porta paralela o $\mu$A retorna ao seu estado dormente. Ao receber novamente esse sinal o pistão retrai-se.

[2]Os dois sinais reconhecidos são `SIGUSR1` e `SIGUSR2`.

### 3.4.5 $\mu$A `halt`

O $\mu$A `halt` foi criado devido a uma situação muito perigosa que pode resultar na destruição parcial do robô. Esta situação ocorre quando o robô colida com um obstáculo ficando preso, permanecendo contudo a actuação sobre os motores que assim ficam em esforço. Se os motores ficarem algum tempo[3] neste estado, o sobreaquecimento provocado resulta na destruição de alguns andares de saída do controlador dos motores.

Um segundo cenário menos perigoso mas igualmente problemático surge quando o robô se liberta depois de estar em esforço durante breves momentos. O controlador efectua uma integração das actuações, pelo que quando este se consegue libertar efectua um *unwind* aos motores de uma forma muito pouco graciosa. O movimento efectuado pode pôr em perigo outros robôs e o próprio, mas acima de tudo poderá atingir com alguma força uma pessoa que o estivesse a tentar libertar.

A solução encontrada passou por colocar um rato óptico na frente do robô (figura 3.14) permitindo assim verificar se o robô se está a mexer ou não. Juntamente com a informação colocado no *blackboard* pelo $\mu$A `control`, o $\mu$A `halt` consegue determinar se o robô se está a tentar mexer mas não o está a fazer. Detectada a situação de *halt*, é prontamente actualizada uma chave do *blackboard* a indicar o estado perigoso em que o robô se encontra.

Figura 3.14: Rato óptico

### 3.4.6 $\mu$A `heartbeat`

O $\mu$A `heartbeat` é o mais simples de todos os $\mu$A implementados, acordando do estado dormente segundo a segundo de forma a avisar os outros agentes de que ele ainda está "vivo". Este aviso é conseguido através de uma chave partilhada específica a cada agente robótico.

[3]Por razões óbvias não foi feito nenhum estudo sobre o tempo necessário para derreter a primeira componente. Contudo depois de apenas cinco segundos em esforço, ambos os lados do robô ficam muito quentes.

Este facto não é usado pelos outros agentes, mas foi desenvolvido uma pequena aplicação que monitoriza os pacotes *multicast* transmitidos na rede *wireless*. Este programa indica quando já não recebe nenhuma actualização nos últimos cinco segundos vindo de um agente. Esta situação pode ocorrer quando:

- o robô se desliga por falta de baterias;
- o agente em questão está sem rede *wireless*.
- o programa principal efectua uma operação ilegal e termina.

# Capítulo 4

# Cooperação

Numa equipa de quatro agentes robóticos em que cada um tem os mesmos objectivos e forma de os realizar, é evidente que existe um grave problema: para os oitos robôs em campo que desejam ir ter com a bola e empurrá-la para a baliza adversária existe apenas uma única bola. Esta escassez do recurso fundamental do futebol obriga a que haja um elemento de cooperação entre os agentes.

Por um robô cooperante entende-se como sendo um agente robótico que demonstra um comportamento que não seria possível sozinho, caso não houvesse uma partilha de informação ou troca de intenções com outros agentes da própria equipa.

Como tal, e pela anterior definição, a fusão sensorial entre vários robôs deve ser entendida como uma das formas possíveis de cooperação, visto melhorar a percepção do mundo que o robô tem face à percepção que teria caso dependesse unicamente dos seus próprios sensores.

Contudo, não é este tipo de cooperação que será o foco do presente capítulo. Evidentemente que a fusão sensorial é necessária ao proporcionar aos agentes robóticos um melhor modelo do mundo sobre o qual tomam decisões mais correctas, ao invés de actuarem correctamente mas para uma percepção errada, com ruído. A ênfase do presente trabalho sempre foi no tipo de cooperação que envolva coordenação de comportamentos de um grupo de agentes.

Nesta óptica, o presente capítulo começa por abordar a pouca cooperação que foi implementada na segunda arquitectura de software e termina com um esboço de um possível modelo de cooperação susceptível de se enquadradar na arquitectura de software actual.

## 4.1 Máquina de Estados

Se é inevitável que ambas as equipas disputem a posse da bola, é de extrema importância que os robôs da mesma equipa não o façam. A câmara omnidirecional criou um problema ao colocar a bola visível para todos, pelo que não havendo um mecanismo de exclusão, todos os robôs se dirigem à bola sendo que nenhum ficará em condições de empurrá-la para a frente, face à obstrução dos colegas de equipa.

Na arquitectura `seattle2001` é usada uma técnica de exclusão, já ensaiada nas arquitecturas `robocup99` e `eurobocup2000`, que permite que um único robô se desloque à bola enquanto os restantes são imobilizados. A decisão dos agentes é efectuada com base na avaliação de cada robô, colocada no *blackboard* distribuído, do custo de aproximação à bola. Este valor é periodicamente calculado pelo $\mu$A `vision` através da heurística 4.1 que pondera os seguintes três factores:

- Penalizar os robôs que estejam à maior distância $d_{ball}$ da bola (figura 4.1(a));
- Penalizar os robôs que estejam "para lá da bola", ou seja quando estão mais perto da baliza que a bola (figura 4.1(b));
- Penalizar os robôs que tenham de efectuar uma correcção angular $\theta^R_{ball}$ de forma a colocar a bola centrada com o dispositivo de chuto (figura 4.1(c)).

$$\begin{aligned} d_{ball} &= \sqrt{(x_b - x_R)^2 + (y_b - y_R)^2} \\ h_{ball} &= \begin{cases} k_1 d_{ball} - k_2(x_b - x_R) + k_3\theta^R_{ball} & \text{caso bola visível,} \\ 100 & \text{c.c.} \end{cases} \end{aligned} \tag{4.1}$$

Com base nos valores das heurísticas $h_{ball}$ dos robôs de campo, foi definido o seu predicado `should_i_go?` que apenas devolve o valor lógico verdadeiro se e só se o robô tiver o menor valor de $h_{ball}$. Desde que não haja latência na difusão das chaves do *blackboard* distribuído, é garantido que a cada instante apenas um dos agentes terá este predicado com valor lógico verdadeiro. Este predicado é utilizado na máquina de estados do atacante (figura 3.12) para excluir todos excepto um de executar os estados que lidam com a bola, identificados a rosa.

Embora esta técnica muito simples resulte na maioria das situações é possível colocar dois robôs a deslocarem-se para a bola num determinado

encadeamento de eventos[1]. Este problema é contudo resolvido através do conceito de um semáforo distribuído, dos sistemas de informação distribuídos.

O pormenor técnico relevante reside na ginástica necessária ao conceber toda a máquina de estados do atacante em torno de um único acto pontual de cooperação entre os agentes robóticos. Caso se pretendesse estabelecer o comportamento em que a cada instante teria de haver apenas um robô no campo adversário seria necessário recorrer a mais um semáforo distribuído e, o mais difícil, reescrever toda a máquina de estados em torno desse novo predicado `should_i_attack?`.

## 4.2 Regras e Planos

O contributo, que se apresenta de seguida, sobre um possível percurso para a realização de cooperação sobre a actual arquitectura de *software* não é uma solução completa. Nela constam apontamentos do que se pretende e sugestões para consideração daqueles que venham a realizar esta tarefa.

Desde o início do Projecto ISocRob que a arquitectura pretendeu seguir os passos da arquitectura multi-agente proposta por Alex Drogoul [18, 19] que distingue três níveis de interacções na sociedade de agentes:

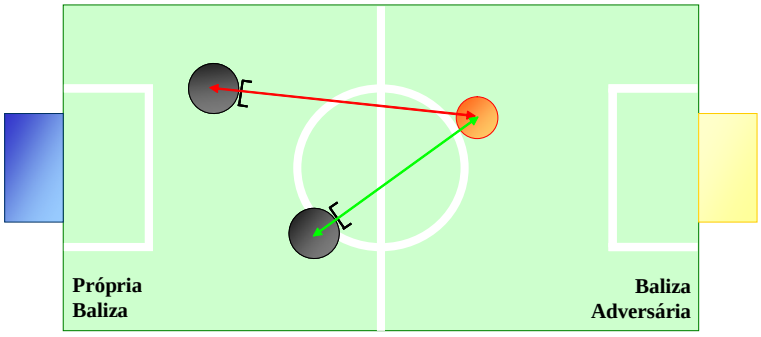
- **Organizacional**: Camada responsável pelos aspectos relativos a todos os elementos da sociedade;
- **Relacional**: Camada que comporta as interacções entre dois ou mais agentes mas que não afectam a sociedade como um todo;
- **Individual**: Camada que lida com as questões relativas a um único indivíduo.

Uma leitura possível destes três níveis aplicada ao futebol robótico seria a seguinte:

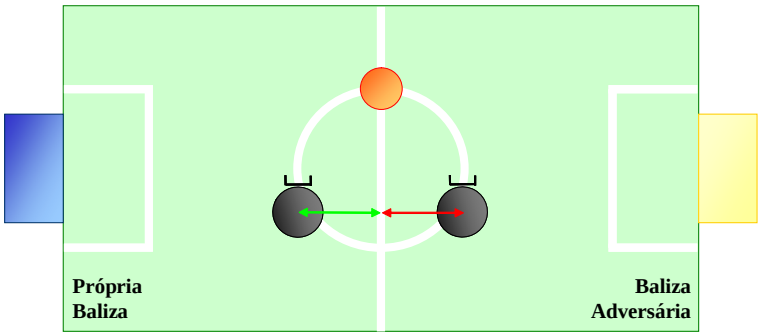
- No nível organizacional seria estabelecida a formação da equipa[2] haveria uma representação do estado actual do jogo, seria designado um

[1] A situação não é descrita por ser relativamente extensa. Fica o apontamento de que o problema surge quando o robô que está mais perto da bola não a consegue ver, permitindo que outro robô mais distante tome a iniciativa de se dirigir à bola.

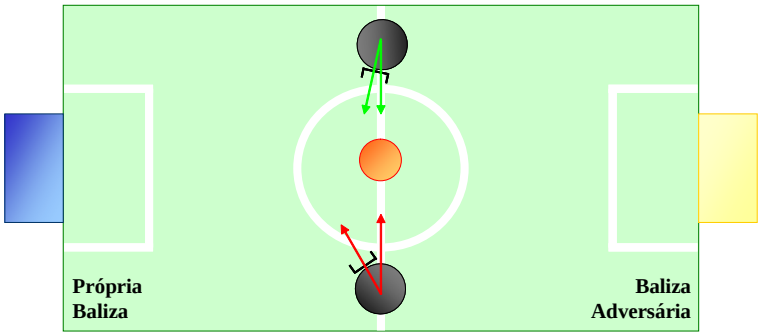
[2] Entende-se por formação da equipa o número de robô de cada classe de jogadores implementada. De momento existem os perfis atacante e guarda-redes, mas no futuro poderão haver defesas e médios. A determinação da formação deverá ser efectuada com base num conjunto de parâmetros tais como: resultado e tempo de jogo, características do adversário, importância do jogo, estratégia do treinador.




(a) Menor Distância




(b) Alinhamento com Baliza Adversária




(c) Menor desvio angular

Figura 4.1: Heurística: Custo de aproximação à bola

robô como sendo capitão de equipa[3];

- No nível relacional estariam todas as acções conjuntas, como passar a bola entre jogadores, formação de barreiras para cobrir a baliza, troca de funções entre dois robôs;

- No nível individual estarão os comportamentos individuais exibíveis pelo robô: ir ao encontro da bola, empurrar a bola, movimentar-se no campo, chutar.

A ideia central do contributo é de que os níveis individual e relacional são mapeados para um único comportamento individual, sendo que o nível organizacional condiciona o conjunto de acções exequíveis consoante o decorrer da partida. Esta ideia está subjacente na decomposição de objectivos de um colectivo em objectivos de sub-grupos que são subsequentemente decompostos até obtermos o objectivo de cada um dos agentes [20]. O esboço do modelo (figura 4.2) contempla três níveis:


Figura 4.2: Novo $\mu$A `machine`

- **Nível O** (organizacional) presente apenas no capitão da equipa e dormente nos restantes agentes. Caso o capitão termine, deverá ser eleito novo capitão activando o nível organizacional do agente em questão;

- **Nível RI** (relacional e individual) onde o agente determina, de um conjunto de regras, qual será o plano que irá tentar seguir nos próximos instantes. Cada regra é definida por um conjunto de condições e um plano. As condições determinam se o plano associado é passível de ser executado pelo robô em questão;

[3]As decisões ao nível organizacional são tomadas pelo capitão de equipa que depois informa os restantes jogadores.

- **Nível E** (execução) onde é executado o plano escolhido no nível RI. Este nível traduz o plano numa máquina de estados idêntica à que corre actualmente no $\mu$A `machine`.

A ênfase do contributo vai para a sintaxe dos planos: uma representação simbólica de eventos que facilite a tarefa de adicionar novos comportamentos, quer individuais quer relacionais. Como é ilustrado nas tabelas 4.3 e 4.4 os planos são facilmente codificados segundo as definições expostas na tabela 4.1.

```
Plano     :=  Instante(;Instante;...)
Instante  :=  Termo(,Termo,...)
Termo     :=  Agente Operador Argumento
```

Tabela 4.1: Decomposição de um plano em operadores

A dificuldade prende-se com a avaliação das regras de forma a escolher o plano mais adequado. Apresenta-se de seguida um conjunto de características que essa mesma avaliação deverá apresentar:

- Terá de haver uma função de avaliação $f_t$ por operador (tabela 4.2) que pondere dois factores: o factor risco ou dificuldade em executar o termo e o interesse ou benefício associado;
- De forma semelhante, terá de haver uma função de avaliação $f_i(f_{t_1}, f_{t_2}, ..., f_{t_n})$ para um instante e uma função de avalição $f_p(f_{i_1}, f_{i_2}, ..., f_{i_m})$ para um plano;
- Uma ponderação na avaliação (impossível) de instantes no futuro de forma a que os planos mais compridos não sejam penalizados;
- A cada regra terá de ser associado um valor estabelecendo assim prioridades entre planos de forma a que certas regras não sejam ignoradas devido à existência de outra(s) melhor(es). Por exemplo, um robô que esteja há nove segundos na grande-área adversária deverá sair prontamente mesmo que esteja prestes a marcar um golo[4];
- Uma vez que o agente estará periodicamente a re-avaliar o conjunto de regras, terá de haver um factor de penalização que impeça que o agente mude demasiadas vezes o plano que está a executar.

[4]No início da partida é designado um robô e apenas esse poderá permanecer dentro da própria grande-área por um tempo superior a dez segundos.

| Operador | Comportamento | Descrição |
|---|---|---|
| & | approachball | Aproximar-se da bola |
| # | clearball | Afastar a bola da própria baliza |
| ~ | dribble | Deslocar-se para outra posição na posse da bola |
| @ | go | Deslocar-se para outra posição no campo |
| ++ | go | Avançar no campo |
| -- | go | Recuar no campo |
| [ | goalkeeper | Defender a baliza |
| { | teamkeeper | Defender a baliza |
| % | halt | Tentar sair da situação de *halt* |
| = | nothing | Ficar parado |
| > | pass | Passar a bola para outro robô |
| < | receive | Receber a bola vindo de outro robô |
| ? | selflocalize | Determinar a posição no campo |
| " | standby | Ficar parado, seguindo a bola |
| ! | score | Marcar golo |

Tabela 4.2: Operadores Individuais

| Plano | Descrição |
|---|---|
| x&;x++Goal;x! | Atacante |
| x&;x# | Defesa |
| x? | Guarda-Redes |

Tabela 4.3: Exemplos Planos Individuais

| Plano | Descrição |
|---|---|
| x++Corner,A++Goal;x>A;A! | Cruzamento do canto para golo |
| x=,A++Goal;x>A | Desmarcação |
| x>A;A<x,x++;A>x,x++;x<A | Triangulação |

Tabela 4.4: Exemplos Planos Relacionais

# Capítulo 5

# Conclusões

Não há resultados passíveis de mostrar a mais valia da nova arquitectura de *software* `seattle2001` face à que existia aquando do início do trabalho. Os únicos números derivados ocorrem nas partidas de futebol realizadas, mas dada a natureza de competição acaba por ser mais valiosa a potência de um dispositivo de chuto do que uma arquitectura de *software* modular, virada para os próximos passos a serem dados no futuro.

Resta assim a única comparação possível: uma análise subjectiva da arquitectura desenvolvida face a todas as outras. Esta análise passa por indicar o contributo para o Projecto ISocRob, uma vez que foi adiada a primeira iteração rumo à cooperação entre os robôs. Esse contributo efectuou-se nos três objectivos estabelecidos: Abstração, Modularidade e Desempenho.

A **Abstração** da decisão do comportamento individual deverá ser o contributo mais relevante, dada a intenção inicial de desenvolver um modelo formal de cooperação.

Não obstante, a **Modularidade** da arquitectura é sem dúvida o ponto forte da solução desenvolvida. O ciclo concepção, implementação e depuração de código para a arquitectura `seattle2001` decorreu de uma forma muito pacífica. Um dos grandes problemas surgidos até então passava por conseguir repetir toda uma situação de forma a averiguar qual é que seria o problema, que tanto poderia estar na lógica da máquina de estados no $\mu$A `machine`, como poderia resultar de uma interacção incorrecta com o $\mu$A `guidance`.

A possibilidade de na arquitectura $\mu$A `seattle2001` isolar um comportamento e depurar o *plugin* isoladamente, acelerou o processo de escrita de código, uma vez que se tornou possível o desenvolvimento em paralelo dos vários comportamentos.

Contudo, e apesar dos inúmeros melhoramentos, o **Desempenho** manteve-se igual. Apesar do peso computacional ter aumentado consideravelmente,

o $\mu$A `vision` continua a processar o máximo possível de 25 imagens por segundo. Devido ao percurso alternativo dos sensores aos actuadores, sem passar pela (possivelmente lenta) primeira camada de decisão, o robô tornou-se visivelmente mais reactivo que no código `eurobocup2000`.

A arquitectura desenvolvida demonstrou ser capaz de resolver um problema exigente de aplicação ao mundo real e satisfez plenamente as expectativas, sendo que deverá concerteza ser uma solução passível de ser aplicada na resolução de outros problemas [21].

Não faz sentido falar em melhoramentos a esta arquitectura ao nível da cooperação, uma vez que um dos seus objectivos principais é precisamente abstrair toda essa camada de decisão.

O único melhoramento a apontar à arquitectura surge em função da aplicação que esta arquitectura teve. Como os robôs dispõe essencialmente de um único sensor, o $\mu$A `vision` acabou por funcionar como toda a camada de sensores, ou *Sense*.

Numa aplicação mais rica, dotada de maior número de sensores, faz sentido esclarecer a concepção da arquitectura actual. A camada *Sense*, na forma de um $\mu$A, fará uma fusão sensorial dos dados colocados no *blackboard* por um vector de $\mu$A responsáveis pelos vários sensores disponíveis (figura 5.1).

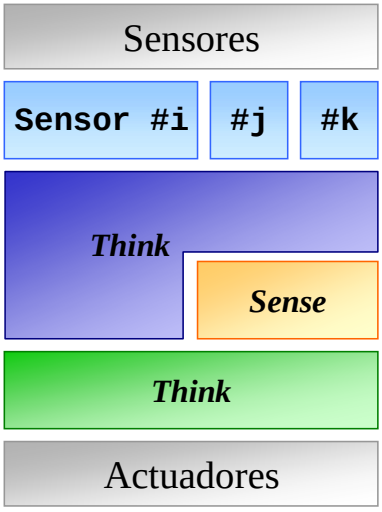



Figura 5.1: Fusão sensorial

# Apêndice A

# Participação em Competições

Apresentam-se de seguida os resultados obtidos aquando da participação da equipa em duas competições internacionais da modalidade.

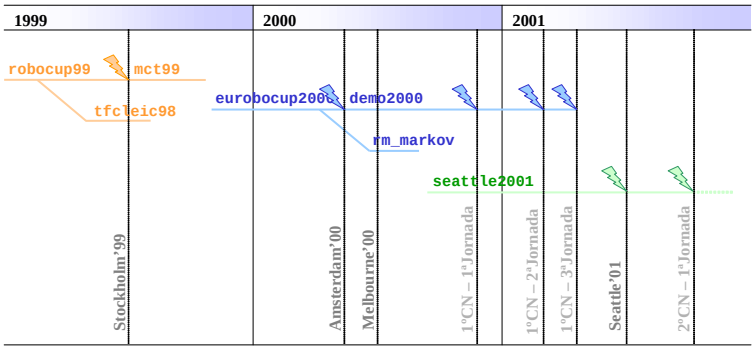



Figura A.1: Roadmap

| | ART 2000 | Dutch Team | ISocRob | Ulm Sparrows | Uppsala |
|---|---|---|---|---|---|
| ART 2000 | | 1-0 | 3-0 | 2-2 | 3-0 |
| Dutch Team | 0-1 | | 0-2 | 0-3 | 3-0 |
| ISocRob | 0-3 | 2-0 | | 1-1 | 1-0 |
| Ulm Sparrows | 2-2 | 3-0 | 1-1 | | 1-0 |
| Uppsala | 0-3 | 0-3 | 0-1 | 0-1 | |

(a) Resultados

| | W | D | L | F | A | GD | Pts |
|---|---|---|---|---|---|---|---|
| ART 2000 | 3 | 1 | 0 | 9 | 2 | 7 | **10** |
| Ulm Sparrows | 2 | 2 | 0 | 7 | 3 | 4 | **8** |
| ISocRob | 2 | 1 | 1 | 4 | 4 | 0 | **7** |
| Dutch Team | 1 | 0 | 3 | 3 | 5 | -2 | **3** |
| Uppsala | 0 | 0 | 4 | 0 | 8 | -8 | **0** |

(b) Classificação

Figura A.2: Resultados Europeu Amesterdão'2000

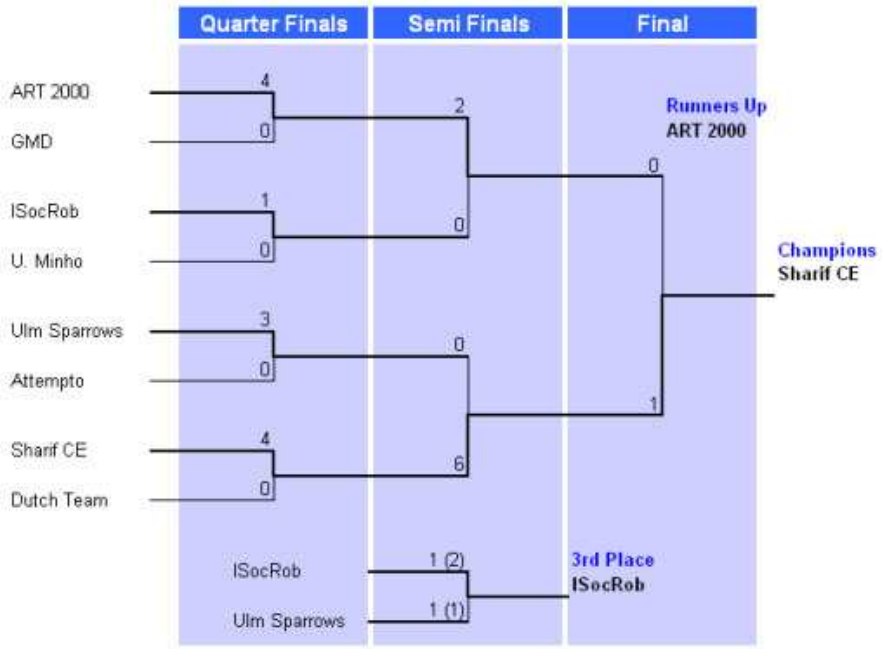



Figura A.3: Fase Final Europeu Amesterdão'2000

| | Trackies | Cops Stuttgard | Fun2maS | ISocRob | Clockwork Orange | JayBots |
|---|---|---|---|---|---|---|
| Trackies | | 3-0 | 8-0 | 9-0 | 8-0 | 2-0 |
| Cops Stuttgard | 0-3 | | 7-1 | 4-0 | 2-2 | 2-0 |
| Fun2maS | 0-8 | 1-7 | | 0-1 | 0-5 | 2-0 |
| ISocRob | 0-9 | 0-4 | 1-0 | | 1-3 | 1-0 |
| Clockwork Orange | 0-8 | 2-2 | 5-0 | 3-1 | | 2-0 |
| JayBots | 0-2 | 0-2 | 0-2 | 0-1 | 0-2 | |

(a) Resultados

| | W | D | L | F | A | GD | Pts |
|---|---|---|---|---|---|---|---|
| Trackies | 5 | 0 | 0 | 30 | 0 | 30 | **15** |
| Cops Stuttgard | 3 | 1 | 1 | 15 | 6 | 9 | **10** |
| Clockwork Orange | 3 | 1 | 1 | 12 | 13 | -1 | **10** |
| ISocRob | 2 | 0 | 3 | 3 | 16 | -13 | **6** |
| Fun2maS | 1 | 0 | 4 | 3 | 21 | -18 | **3** |
| JayBoys | 0 | 0 | 5 | 0 | 9 | -9 | **0** |

(b) Classificação

Figura A.4: Resultados Mundial Seattle'2001

# Bibliografia


[1] Intelligent Society of Robots. URL: http://socrob.isr.ist.utl.pt/.

[2] RoboCup Federation. URL: http://www.robocup.org/.

[3] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of IJCAI-95*, 1995.

[4] RoboCup Middle Size League. URL: http://smart.informatik.uni-ulm.de/ROBOCUP/f2000/.

[5] Rodrigo Ventura, Pedro Aparício, Carlos Marques, Pedro Lima, and Luis Custódio. Isocrob - intelligent society of robots. *RoboCup-99: Robot Soccer World Cup III*, 1999.

[6] Pedro Lima, Rodrigo Ventura, Pedro Aparício, and Luis Custódio. A functional architecture for a team of fully autonomous cooperative robots. *RoboCup-99: Robot Soccer World Cup III*, 1999.

[7] Rodrigo Ventura, Pedro Aparício, Pedro Lima, and Carlos Pinto-Ferreira. Socrob - a society of cooperative mobile robots. In *Proc. of 1998 IEEE International Conference on Systems, Man, and Cybernetics*, 2000.

[8] Socrob. URL: http://socrob.isr.ist.utl.pt/socrob.php.

[9] Marvin Minsky. *Society of Mind*. Simon & Schuster, 1988.

[10] Nomadic Technologies Inc. Nomadic SuperScout II. URL: http://socrob.isr.ist.utl.pt/scouts-inside.php.

[11] Carlos Marques and Pedro Lima. A Localization Method for a Soccer Robot Using a Vision-Based Omni-Directional Sensor. In *Proceedings of EuRoboCup Workshop 2000, Amsterdam, The Netherlands*, 2000.

[12] Carlos Marques and Pedro Lima. Vision-based self-localization for soccer robots. In *Proceedings of IROS 2000, Takamatsu, Japan*, 2000.

[13] Rodrigo Ventura, Filipe Toscano, Carlos Marques, Luis Custódio, and Pedro Lima. Eurobocup 2000 technical report. Technical Report RT-701-00, RT-402-00, Instituto Sistemas e Robótica, October 2000.

[14] Bruno Correia, Sérgio Saraiva, Rodrigo Ventura, and Pedro Lima. Interface gráfica para operação de robots cooperantes. Technical Report RT-407-99, Instituto Sistemas e Robótica, September 1999.

[15] Rodney Brooks. New approaches to robotics. *Science*, 253:1227–1322, September 1991.

[16] Pedro Lima, Luis Custodio, Bruno Damas, Manuel Lopes, Carlos Marques, and Luis Toscano. Isocrob 2001 team description paper. *Robot Soccer World Cup V*, 2001.

[17] James Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 1996.

[18] Alex Drogoul and A. Collinot. Applying an agent-oriented methodology to the design of artificial organizations: A case study in robotic soccer. *Autonomous Agentes and Multi-Agent Systems*, 1:113–129, 1998.

[19] Alex Drogoul and J. Ferber. Multi-agent simulation as a tool for modeling societies: Application to social differentiation in ant colonies. *Actes du Workshop MAAMAW'92*, 1992.

[20] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[21] RoboCup Rescue. URL: http://www.r.cs.kobe-u.ac.jp/robocup-rescue/.