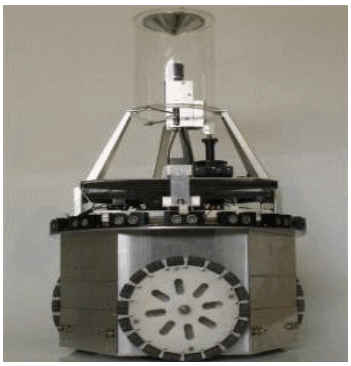
UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO



GUARDA-REDES ROBÓTICO OMNIDIRECCIONAL

Catarina Esteves, nº 47946, AE de Sistemas, Decisão e Controlo
Sérgio Lopes , nº 48122, AE de Sistemas, Decisão e Controlo

LICENCIATURA EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES
Relatório de Trabalho Final de Curso
26/2004/L

Prof. Orientador: Pedro Lima

Prof. Acompanhante: Luís Custódio

Novembro de 2005

Agradecimentos

Queremos expressar o nosso agradecimento a toda a equipa do SocRob com especial destaque para o Professor Pedro Lima por nos ter acompanhado ao longo de todo o ano e ter tido paciência para nos aturar, ao Professor Luís Custódio por todos os esclarecimentos relativos a comportamentos e afins, ao Hugo Costelha pelas inúmeras vezes em que o chateamos com o SuSe (ninguém o manda ser defensor acérrimo do Linux).

Resumo

Este Trabalho Final de Curso consistiu em criar comportamentos para um robot holonómico de forma a que este se comporte como um guarda-redes. A criação dos comportamentos passou pelo seu desenho e pelo encadeamento dos mesmos, onde foi implementado um sistema de decisão de comportamentos baseado em regras, onde cada um dos comportamentos foi desenhado com base em *Finite State Machines*. Seguiu-se posteriormente uma implementação de acordo com a nova arquitectura do projecto do SocRob, tendo sido depois realizados testes de performance dos comportamentos numa simulação criada em Matlab.

Palavras-chave

Robot Omnidireccional, Autómatos Finitos, Sistema Baseado em Comportamento.

Abstract

This Final Course Work consisted in the creation of behaviors for a holonomic robot with the finality that he behaves like a goalkeeper. The creation of the behaviors have passed by it design and it linkage, where was implemented a decision system of behaviors based in rules, where which one of the behaviors was drawn with base in Finite State Machines. It was followed by an implementation respecting the new SocRob project architecture, and then performance tests of those behaviors in a simulation made in Matlab.

Keywords:

Goalkeeper, Omnidirectional, Predicate, Behavior.

Índice

Acrónimos

MSL	Middle Size League
SVN	Subversion
FCT	Fundação para a Ciência e Tecnologia
ISR	Instituto de Sistemas e Robótica
IST	Instituto Superior Técnico
SocRob	Sociedade de Robots ou *Soccer Robots*
FSM	Finite State Machine
RAPs	Reactive Action Packages
SR	Stimulus Response
FSA	Finite State Acceptor

1. Introdução

1.1. Futebol Robótico

Nos dias de hoje, existe um interesse crescente relativamente ao estudo da robótica, e a cada dia que passa aparecem robots com um nível de desenvolvimento que superam os seus antecessores. Uma área específica desse estudo é a cooperação entre robots uma vez que trabalhando em equipa, existe uma maior probabilidade de se conseguir atingir objectivos específicos com maior facilidade que um único individuo. É neste âmbito que aparece o futebol robótico para tornar a ideia mais atractiva, porque, tendo em conta que o futebol é um desporto que tem milhões de adeptos em todo o mundo, existirão mais pessoas a interessar-se pela ideia e o seu estudo torna-se na generalidade mais apelativo.

Por esse mundo fora, existem cada vez mais competições, onde são postos à prova o desenvolvimento conseguido por cada equipa participante, não só a nível de futebol como em outras variadas categorias, sendo que a principal competição existente de momento é a RoboCup, que todos os anos se realiza num pais diferente, tendo em 2004 passado por Portugal.

O grande objectivo desta competição e deste "desporto", é criar uma equipa que em 2050 seja capaz de vencer os campeões mundiais no que diz respeito a Humanos.

1.2. O Projecto SocRob

Na área da investigação em Robótica Cooperativa e Sistemas Multi-Agente no âmbito da Inteligência Artificial e Robótica, surgiu em 1997 o projecto denominado de SocRob (Sociedade de Robots ou "Soccer Robots") do ISR/IST (Instituto de Sistemas e Robótica, pólo do Instituto Superior Técnico), financiado presentemente pela FCT

(Fundação para a Ciência e Tecnologia).

Um dos seus objectivos é a participação no RobCup, que corresponde a um encontro anual científico internacional, onde se realizam conferências no âmbito do futebol robótico dadas por investigadores nesta área e competições de futebol robótico (e não só) entre as diversas equipas participantes. Estas competições dividem-se em várias ligas. O projecto SocRob desenvolve equipas participantes nas ligas "Simulation-League", "4-Legged League", "Middle Size League".

A "Simulation-League" consiste em jogos apenas realizados por software.

A "4-Legged League" é constituída por equipas de quatro robots de quatro pernas (SONY AIBO).

A "Middle Size League" (MSL) tem como base robots de não mais de 50 cm de diâmetro em equipas de quatro indivíduos.

A equipa de MSL do SocRob (IsocRob MSL) foi composta até 2005 por quatro robots não holonómicos "Nomadic Super Scouts II", a cada um deles era atribuído um papel em campo, sendo atribuído a um dos robots o papel de guarda-redes. Derivado ao facto de serem não holonómicos os robots apresentavam grandes restrições de movimento, o que afectava em muito o seu desempenho na execução do seu papel.

De maneira a aumentar o sucesso da equipa no desenvolvimento dos comportamentos adequados a cada circunstância a adoptar por cada robot, a IsocRob decidiu substituir a equipa existente, por uma equipa de robots omnidireccionais, esperando com isto, implementar soluções mais criativas e portanto de maior interesse para o presente estudo.

Apoiado num projecto financiado pela FCT, o ISR/IST desenvolveu novos robots omnidireccionais juntamente com as companhias portuguesas, IdMind (responsável pelo hardware electrónico) e a ServiLog (responsável pelo hardware mecânico).

1.3. Especificação do Problema

O objectivo deste TFC é implementar toda a lógica necessária para que um dos novos robots omnidireccionais da equipa IsocRob MSL consiga comportar-se como um guarda-redes, pelo que se espera que não permita a entrada de bolas na baliza e coloque a bola em

campo quando tal for necessário e evidentemente é pretendido um bom desempenho na representação deste papel.

A concepção do papel de guarda-redes particulariza-se na escolha de uma arquitectura baseada em comportamentos que se adeqúe aos objectivos pretendidos pelo mesmo. Para este problema específico foi utilizada uma codificação discreta de comportamentos utilizando uma estratégia baseada em regras, ou seja SE `antecedente` ENTÃO `consequente`, sendo que neste caso o consequente representa cada um dos comportamentos desenvolvidos, ou seja, um micro-agente (MACHINE) é que fica encarregue de seleccionar o comportamento a ser executado tendo em conta a situação do mundo em cada instante. Relativamente aos comportamentos, estes foram criados baseados em Máquinas de Estado Finitas, ou, *Finite State Acceptors* [1]*, ou, Finite State Machines* (sendo daqui em diante designadas por FSA ou FSM)

1.4. Arquitectura Baseada em Comportamentos

Como criar uma arquitectura de comportamentos de forma a que um robot faça aquilo para que é criado?

Esta questão leva-nos a uma série de outras perguntas que devem ser respondidas para dotar um robot com um mecanismo de coordenação de comportamentos.

- Quais são os blocos de construção correctos para sistemas robóticos.
- O que é realmente um comportamento primitivo?
- Como são esses comportamentos efectivamente coordenados
- Como são esses comportamentos ligados a sensores e actuadores

Infelizmente não existe actualmente uma concordância universal quanto à resposta a essas perguntas.

Seguidamente são descritos alguns métodos utilizados para especificar e criar comportamentos robóticos.

Etologicamente guiados/desenho restrito.

O estudo dos comportamentos dos animais pode fornecer uma poderosa visão de formas de como os comportamentos podem ser construídos.

Existem casos onde alguns investigadores procura criar modelos altamente fieis do substrato neurológico de um animal, numa tentativa de emular uma resposta apropriada por parte do robot. Esses investigadores restringem deliberadamente os seus modelos dos comportamentos de modo a serem equivalentes aos dos animais em estudo.

A metodologia para desenhar este tipo de comportamentos é feito da seguinte forma: o modelo é fornecido por um estudo científico. Este é então alterado de modo a ser computacionalmente realizado, sendo depois relacionado com um robot a nível de sensores e actuadores. Os resultados da experiência com o robot são então comparados com os resultados do estudo biológico, sendo depois ambos os modelos alterados para produzirem resultados que estejam mais em conformidade com o comportamento original do animal em estudo [1].

Desenho baseado em actividade situada:

Actividade situada significa que existem para as acções do robot, predicados para as situações nas quais se encontra. Assim sendo, o problema de percepção fica reduzido ao reconhecimento de, em que situação o robot se encontra e aí escolher uma acção a tomar. Assim que se encontra noutra situação, selecciona uma acção mais apropriada a tomar.

Este tipo de situações pode ser visto como micro-comportamentos, ou seja, comportamentos específicos que são úteis apenas para circunstâncias limitadas. O desenho de um robot baseado nesta metodologia requer uma sólida compreensão da relação entre o agente robótico e o ambiente em que se encontra.

Assumindo que não existe limite para o número de condições situacionais que podem ser enumeradas, temos uma versão mais expansiva desta teoria de actividade situada: planos universais. Estes planos universais requerem que o agente robótico tenha a habilidade para reconhecer cada situação e aí escolher a acção apropriada para cada estado do mundo em que se encontra. Esses planos universais cobrem todo o plano de interacção, utilizando os sensores para levar à classificação, e não se pressupõe qualquer ordem nas situações ou no tipo de situações que dai podem advir. Os sensores estão

constantemente activos, logo tanto a avaliação sensorial, como a resposta apropriada estão continuamente activos.

Mas existe o argumento que enumerar todas as situações possíveis é impraticável e matematicamente intratável.

Temos depois também os designados *Reactive Action Packages* (RAPs), que consiste num conjunto de métodos específicos para certas situações, e, para esses métodos é dada uma sequência de passos para alcançar o pretendido.

A principal diferença entre RAPs e a maioria dos sistemas reactivos é que não são verdadeiramente baseados em comportamentos, mas sim baseados em tarefas e, esse sistema assenta num forte modelo do mundo.

Desenho de robots baseado na experimentação:

Os desenhos de robots baseados na experimentação são invariavelmente criados de baixo para cima. A premissa básica de operação é municiar um robot com um conjunto básico de competências, fazer então experiências no mundo real, ver o que funciona e o que não funciona, fazer debug dos comportamentos imperfeitos e aí ir sucessivamente introduzindo novos comportamentos até o sistema exibir uma performance que seja satisfatória.

Expressão de comportamentos:

Existem vários métodos para exprimir os comportamentos dos robots.

Vamos destacar três: Diagramas *Stimulus-Response* (SR), notação funcional e diagramas *Finite State Acceptor* (FSA).

Para uma representação gráfica de configurações de um comportamento específico são usados os diagramas SR, notação funcional utiliza-se para haver uma maior clarificação no desenho dos sistemas, sendo os diagramas FSA usuais quando é necessária uma sequência temporal dos comportamentos.

Os diagramas *Stimulus-Response* são o mais intuitivo e menos formal método de expressão. Qualquer comportamento pode ser representado por uma resposta gerada por um dado estimulo.

Quanto à notação funcional podem-se utilizar métodos numéricos para descrever uma relação utilizando esta notação. Por exemplo: b(s)=r significa que o comportamento

b, sendo-lhe dado um estimulo **s**, gera uma resposta **r**.

Os esquemas *Finite State Acceptors* têm propriedades muito úteis para descrever agregações e sequências de comportamentos, tornando explícitos os comportamentos activos em qualquer altura e as transacções entre eles.

1.5. Abordagem ao Problema

Temos, primeiro que tudo definir algumas premissas que são essenciais para uma correcta análise baseada em comportamentos, premissas essas que são descritas de seguida.

- Papel: Caracteriza-se por um conjunto de comportamentos e um conjunto de regras para selecção dos mesmos.

- Comportamentos: São máquinas de estados, cujos estados correspondem a acções primitivas e as transições entre os mesmos estão associadas à validação dos predicados que lhes estão associados. Existem três tipos de comportamentos: organizacionais, relacionais e individuais.

- Comportamentos Organizacionais: Correspondem aos comportamentos apresentados pela equipa distribuição de papéis por indivíduo.

- Comportamentos Relacionais: Apresentados quando um conjunto de robots se envolve numa execução conjunta de acções.

- Comportamentos Individuais: Baseiam-se na execução de acções primitivas.

- Acção Primitiva: Elemento atómico de um comportamento, uma acção primitiva não pode ser decomposta em mais acções. Uma acção primitiva normalmente necessita de analisar dados sensoriais e proceder à activação de um actuador.

- Evento: Denota uma mudança no modelo do mundo. Na análise comportamental é feita a descriminação das mudanças sobre as quais pretendemos ter percepção. Esses são

os eventos considerados para o estudo.

- Predicado: Atribui o valor de verdadeiro ou falso a um aspecto considerado relevante sobre o mundo em análise. A mudança de valor atribuído (falso - verdadeiro ou verdadeiro - falso) indica a ocorrência de um evento.

- Estado: Designação atribuída a um agente e normalmente é associada à descrição do presente desenrolar de uma determinada acção primitiva por parte deste. Não faz sentido falar em estado, pelo menos no caso em estudo, se este não for relativo a um agente.

- Agente: Os agentes serão todos aqueles que se encontrem dentro do campo e tenham a possibilidade de acção sobre o meio através dos seus actuadores, neste caso os agentes terão também a capacidade de análise do meio através dos seus sensores e por comunicação com outros agentes. Neste caso os agentes serão todos os robots presentes no campo, pertencentes às duas equipas que se confrontam.

- Mundo: Neste caso a realidade em análise está limitada ao campo onde se irá realizar o jogo de futebol, campo este delimitado por linhas e as balizas que se encontram nos extremos do campo, nada que se encontre fora destas fronteiras será considerado relevante e portanto fará parte da matéria em estudo.

- Modelo do Mundo: Todos os dados provenientes dos sensores da equipa considerados relevantes para representação do estado do mundo.

- Máquina de Estados: Indica quais os estados que o agente deverá se encontrar através da análise de eventos. A máquina deverá ter sempre um estado de início e outro de fim. O objectivo da máquina é o cumprimento de um determinado objectivo. Uma boa prática de desenho é indicar sempre quais os estados que o agente deverá se colocar no caso da ocorrência e não ocorrência de um evento.

Os passos seguidos na resolução do problema em causa coincidem em ordem e em conteúdo, com a estrutura deste relatório.

O primeiro passo foi estabelecer quais os objectivos pretendidos para o guarda-

redes da o seu papel no jogo e na equipa. Como objectivo mínimo estabelecemos que o guarda-redes deveria não permitir a entrada da bola na baliza. Para além deste objectivo propusemos um objectivo complementar de ao efectuar uma defesa minimizar as hipóteses de golo na própria baliza e maximizar as hipóteses de golo na baliza adversária.

Seguiu-se um estudo de matérias relevantes para responder aos objectivos por nos propostos, inclusive do trabalho efectuado para o guarda-redes anterior.

Para cumprir o primeiro objectivo foram criados dois comportamentos: FollowBall e InterceptBall. O primeiro destina-se a seguir a bola de modo a estarmos sempre colocados na melhor posição possível para uma defesa em caso de remate, o que melhorará o tempo de resposta da defesa, o segundo serve para interceptar a bola na eventualidade de um remate à baliza. Estes comportamentos já eram efectuados pelo guarda-redes anterior, no entanto tiveram que pensados de raiz uma vez os robots terem cinemáticas diferentes.

O objectivo seguinte foi atendido na construção de mais três comportamentos, que o robot poderia exibir: Face Opponent, KickOut, PassBall. O primeiro pretende que o guarda-redes se aproxime da bola, e se apodere dela assim que um adversário entra na área com a bola controlada, reduzindo assim o ângulo com que o adversário pode acertar na baliza e ao mesmo tempo, caso se consiga apoderar da bola, lançar no imediato um contra-ataque, tendo sido esta um comportamento pensado de raiz. O comportamento KickOut e PassBall foram também criados de raiz e resumem a colocação da bola por parte do guarda-redes após se ter aproximado desta. O primeiro chuta a bola para fora, ou seja, não se limita a apanhar a bola e rematar para onde esta virado, procurando um ângulo onde não existam obstáculos e mandando a bola nessa direcção. No caso do segundo chuta a bola para um ponto próximo de um colega de equipa que ele considera estar bem colocado. Todos estes comportamentos são comportamentos individuais, ou seja, a sua escolha só depende do robot das decisões do robot e não das da restante equipa.

Assim sendo foi pensada uma arquitectura baseada em regras, onde existe um agente, que coordena, entre os comportamentos disponíveis, qual esta activo em cada momento, tendo em conta as condições do mundo em que se enquadra, sendo que cada comportamento é executado, mas, caso existam alterações desse mesmo mundo, pode ser interrompido, sendo substituído por outro comportamento. Resumidamente, o trabalho

concretizou-se por uma análise de toda a documentação respectiva ao caso em estudo, pela descriminação dos objectivos pretendidos para o robot, desenvolvimento de uma arquitectura baseada em comportamentos para o papel em causa, implementação dessa arquitectura com base na arquitectura do projecto já existente, simulação dessa arquitectura, realização de testes de desempenho da mesma e finalmente crítica aos resultados de modo a dar sugestões para o futuro que pensamos que uma vez desenvolvido irá melhorar o presente trabalho realizado.

2. Descrição da Equipa

2.1. Software

O software da equipa é implementado e executado em ambiente Linux. O sistema Operativo a ser usado é o SUSE LINUX. O código fonte está a ser desenvolvido com base no conceito de Programação por Objectos em linguagem C++.

O software da MSL (Middle Size League) está sempre sujeito a alterações, no sentido de alcançar uma melhor resposta aos problemas associados ao futebol robótico. Como tal o código fonte é controlado por um sistema de controlo de versões SVN, que faz a gestão dos ficheiros ao longo do tempo. O SVN coloca os ficheiros num repositório central, acessível a todos os membros do projecto. Cada membro trabalha numa cópia local privada (denominada de *working copy*), que a poderá depois publicar no repositório passando esta a fazer parte de uma nova versão, sem no entanto se ter perdido a antiga.

Para que os algoritmos desenvolvidos possam ser testados e avaliados antes de serem executados nos robots reais, é usado um simulador que proporciona um ambiente semelhante ao real, e embora não se espere que os resultados sejam os mesmos dá-nos algumas garantias em termos das suas semelhanças com o comportamento real dos robots.

A importância da simulação é a maior facilidade de teste e uma protecção aos robots reais uma vez que quando os algoritmos são executados neles já dão algumas garantias de sucesso.

O SocRob neste momento está a usar um simulador baseado na ferramenta Webots [12].

Os robots não têm interfaces de I/O logo a comunicação com estes é feita através de um PC externo que corre um interface gráfico X e usa `ssh` para desenvolvimento. A comunicação entre o robot e o PC externo é feita por intermédio de uma rede wireless.

2.1.1. Descrição da Arquitectura

A nova arquitectura do projecto esta dividida em quatro grandes blocos que são de seguida descriminados (ver Anexo D).

- Aquele em que existe uma interacção com o mundo, quer seja através de sensores, quer de actuadores.
- A zona onde é guardada toda a informação acerca desse mesmo mundo.
- O bloco onde cada robot "decide" a acção a tomar.
- Finalmente, existe também o local onde são feitas todas as comunicações.

A ligação entre todos estes blocos é feita da seguinte forma:

Cada robot, através de transdutores, transforma tudo o que é fisicamente adquirido em sinais eléctricos que são posteriormente fornecidos aos vários sensores. Dá-se depois um a fusão de informação, que é feita com todos os dados fornecidos pelos sensores e também por tudo o que é já conhecido e que se encontra na *WorldInfo* (zona de memória partilhada por todos os robots e onde é guardada toda a informação processada pela fusão sensorial, assim como dados referentes à *referee-box*, às transformações referentes tanto à cinemática como à imagem, e ainda outras variadas informações relativas ao estado da equipa).

O resultado dessa fusão de informação é depois guardado na *WorldInfo* e é também utilizado para se saber que acções primitivas devem ser executadas (para cada robot individualmente).

O *WorldInfo* fornece informação para o *Event Manager* (onde em cada instante se sabe os eventos que foram detectados, através da validação dos predicados correspondentes).

O resultante (*WorldInfo* com *Event Manager*), é então utilizada pelo *Team Organizer* (que selecciona o papel a ser executado por cada um dos robots), pelo *Behavior Coordinator* (que junta essa informação à proveniente do *Team Organizer*, para seleccionar o comportamento a executar no momento entre os que estão disponíveis dado o papel que foi seleccionado) e pelo *Behavior Executor* (que utilizando a informação recebida do Behavior *Coordinator*, executa a máquina de estados do comportamento

escolhido). De notar que o *Event Manager* é que tem a iniciativa de fornecer informação ao *Team Organizer*, ao *Behavior Coordinator* e ao *Behavior Executor*, sendo que estes é que pedem informação ao *WorldInfo*.

A informação resultante do *Behavior Executor* (acção primitiva seleccionada), assim como a da fusão de informação (que vai fornecer informações sobre a navegação) juntamente com as referências a serem utilizadas pelos actuadores dadas pela zona das primitivas de navegação (parameterizadas depois de executado o algoritmo de navegação desejado), é utilizada na zona das acções primitivas, como resultado esta envia para aos actuadores o sinal que estes vão ter que aplicar (velocidade das rodas, força do *kicker*).

Finalmente as comunicações, onde existem três tipos diferentes de comunicação. Uma que é utilizada para difundir a *WorldInfo* respectiva a cada robot para o resto da equipa (tipo *Multicast*), outra para comunicação entre os robots e interfaces remotas e finalmente as terceiras referentes ao envio de mensagens de *commit* entre os membros da equipa que estejam envolvidos em comportamentos relacionais.

3. Descrição do Guarda-Redes

3.1. Hardware

A equipa possui cinco robots omnidireccionais. Os robots são todos iguais, sendo a constituição de cada um deles a nível de hardware composta por:

- NEC FS900 laptop responsável por todo o processamento necessário, com um processador INTEL Centrino 1.6 GHz, com 512MB RAM e um disco de 30GB. O laptop inclui um CD-ROM, wireless 802.11b, 3 portos USB 2.0, e 1 porto mini-firewire, e uma bateria Li-Ion.

- 3 rodas omnidireccionais: cada roda têm rolotes ao longo de todo o perimetro para permitir o movimento omnidireccional do robot.
- 3 motores MAXON DC, modelo RE35/118776, tensão nominal 15 V (a operar a 12 V) e 91:6 (ou 15.1(6):1) gear ratio (ref. MAXON 203116), um por roda.
- Sistema de chuto electromecânico de força controlável: para fazer remates à baliza ou passar a bola a um outro jogador.
- rolo frontal: usado para segurar uma bola aquando da sua recepção. -Sensores: -sensor de visão catadióptrico omnidireccional: permite uma visão aérea do campo, preservando as distâncias relativas dos elementos na imagem.
- 16 sonares (SRF04 RangeFinder): colocados em volta da base do robot com o objectivo de detectar obstáculos.
- 1 codificador 1 500 CPR por motor: para controlo do motor e odometria.
- 1 AnalogDevices rate-gyro XRS300EB: para melhorar a determinação da orientação.
- 1 Creative Notebook Optical Mouse: para melhorar o cálculo da posição por odometria.

- 2 sensores de infra-vermelhos Sharp: usados para detectar a bola entre os dedos (saliências usadas ajudar a agarrar a bola) do robot e para medir o deslocamento do dispositivo usado para chutar a bola.
- 2 packs de baterias 9Ah NiMH por robot.

3.2. Navegação

A navegação é o subsistema que permite ao robot determinar a sua postura actual e movimentar-se autonomamente de uma postura inicial para uma postura objectivo, sem colidir com obstáculos.

Um dos problemas da navegação consiste na movimentação de um robot num ambiente povoado com obstáculos.

No futebol robótico, a capacidade de um jogador se movimentar a velocidades relativamente grandes através de um ambiente constituído por obstáculos dinâmicos e estáticos, como por exemplo as balizas, as linhas de fora de campo, os robots em campo e a bola, influência bastante o sucesso deste no cumprimento de uma dada tarefa.

Relativamente aos aspectos da localização do guarda-redes foram usadas as funções disponíveis no projecto, cuja localização é calculada através da informação recolhida por odometria.

3.2.1. Descrição da Cinemática

Os veículos omnidireccionais proporcionam um capacidade de manobra superior aos veículos não holonómicos. A capacidade de se movimentarem em qualquer direcção independentemente da orientação do veículo, é uma opção que se adapta bem a ambientes dinâmicos.

3.2.1.1. Holonómica

Os veículos são constituídos por um grupo de três rodas omnidireccionais igualmente espaçadas de 120º umas das outras.

A informação quanto à cinemática do robot foi retirada de [9]

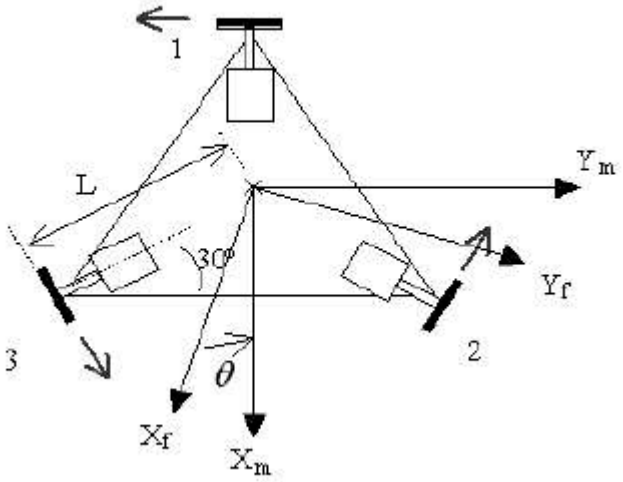


Fig 3.1. Geometria de um veiculo omni direccional

A actual cinemática directa do robot é dada por:

$$\begin{bmatrix} v_x \\ v_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{\sqrt{3}}r & \frac{1}{\sqrt{3}}r \\ -\frac{2}{3}r & \frac{1}{3}r & \frac{1}{3}r \\ \frac{1}{3L}r & \frac{1}{3L}r & \frac{1}{3L}r \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \quad (1)$$

onde V_x e V_y são as velocidades lineares do robot, $\dot{\theta}$ a sua velocidade angular e w_i, i=1,2,3 representa a velocidade angular das rodas em rad/s.

Onde o raio das rodas é dado por r=100mm e L=200mm é o raio do robot.

3.3. Software

O guarda-redes terá sempre o mesmo papel do início ao fim do jogo, ao passo que os outros jogadores poderão alternar entre os restantes papeis consoante o estado da equipa. O guarda-redes não irá apresentar nenhum comportamento relacional na proposta actual. O papel de guarda-redes tem à sua disposição cinco comportamentos que são seleccionados com base no estado do mundo segundo regras criadas, sendo executada a sequência de acções primitivas associadas a cada comportamento após a selecção do mesmo.

3.3.1. FSA

Os esquemas *Finite State Acceptors* [1] têm propriedades muito úteis para descrever agregações e sequências de comportamentos, tornando explícitos os comportamentos activos em qualquer altura e as transições entre eles.

Um *Finite State Acceptor* [1] **M** pode ser especificado por um quádruplo (Q, δ, q_0, F), onde:

- **Q** representa o conjunto de estados do comportamento permitidos.
- **δ** é a função de transição, mapeando a entrada e o estado actual para outro, ou mesmo de um estado para ele próprio.
- $\mathbf{q_0}$ denota a configuração inicial dos estados.
- **F** representa um conjunto de estados aceites ou marcados, sendo que este é um subconjunto Q.

δ pode ser representado de forma a que os arcos representam a transição entre os no diagrama *Final State Acceptor* e são invocados por estímulos.

A melhor utilização para os FSA's é para especificar sistemas complexos de controlo de comportamentos, onde todo o subconjunto de primitivas dos comportamentos é alterado durante a execução dos comportamentos, enquanto são cumpridos alguns

objectivos de alto nível.

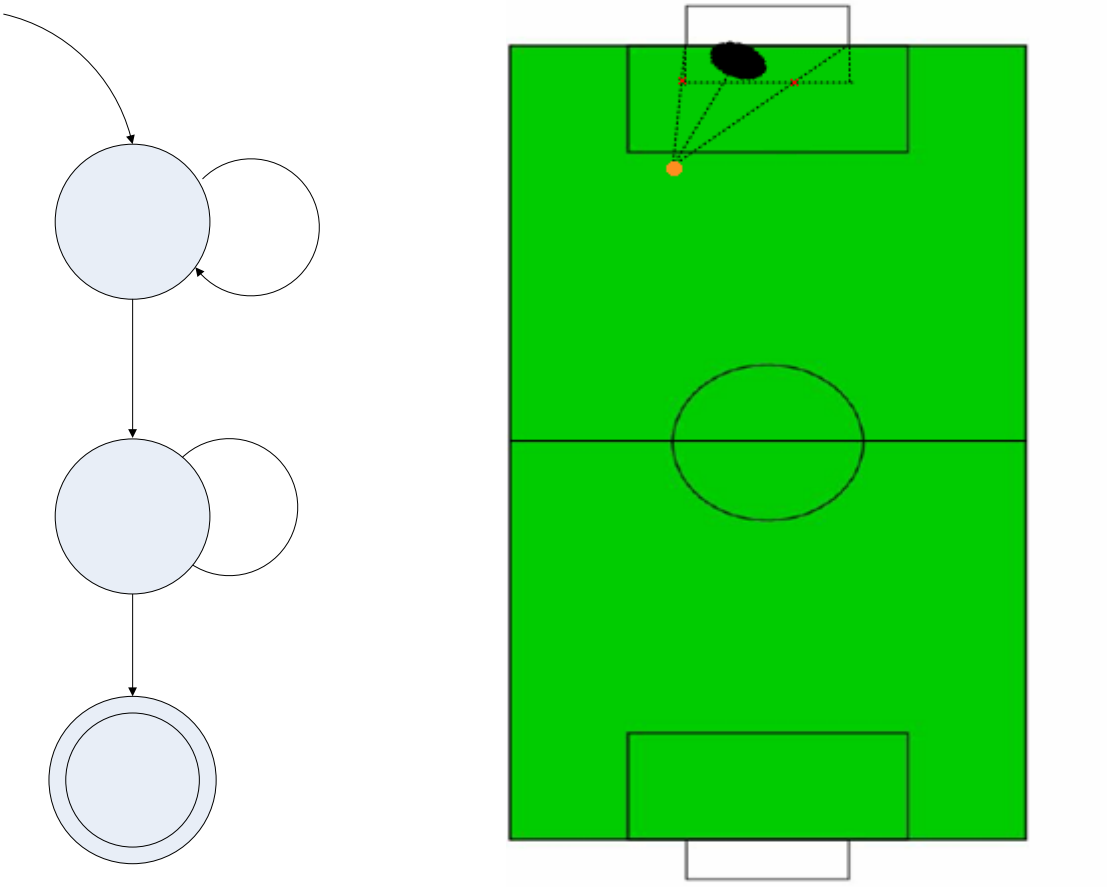
FSA's permitem também exprimir comportamentos sequenciais entre vários conjuntos de comportamentos e foram bastante utilizados em robótica para exprimir sistemas de controlo, sendo que no nosso caso foram utilizados para seleccionar acções primitivas dentro dos comportamentos. Os FSA's fornecem um grande nível de abstracção com o qual conseguimos exprimir a relação entre conjuntos de comportamentos.

3.3.2. Comportamentos

Os comportamentos são compostos unicamente por acções primitivas e por transições entre estas acções originadas com base em predicados. Para mostrar o ciclo de execução destes comportamentos, com base nas acções primitivas e predicados, usamos máquinas de estados, onde cada estado corresponde a uma execução de uma acção primitiva e as transições entre estados são dadas por um conjunto de condições lógicas com base em predicados. Cada comportamento tem um objectivo a cumprir e portanto espera-se que no fim da execução da máquina de estados tenha cumprido com sucesso o seu objectivo. Neste caso o desenho das máquinas de estados foi com base em FSA's, visto que as primitivas que levam a cada comportamento estão constantemente a ser alteradas. De seguida descrevem-se sumariamente os comportamentos utilizados (uma vez que serão descritos com mais detalhe na secção 4.1), tendo sido por nós remodelados os comportamentos Follow Ball e Intercept Ball, ao passo que os outros (Kick Out, Pass Ball e Face Opponent) foram por nós criados.

3.3.2.1 FollowBall

Segue a bola num segmento de recta paralela à linha de golo e tendo em conta uma relação entre ângulos e distâncias entre a bola e os postes da baliza, calcula a melhor posição onde se pode colocar, indo posteriormente para essa posição.



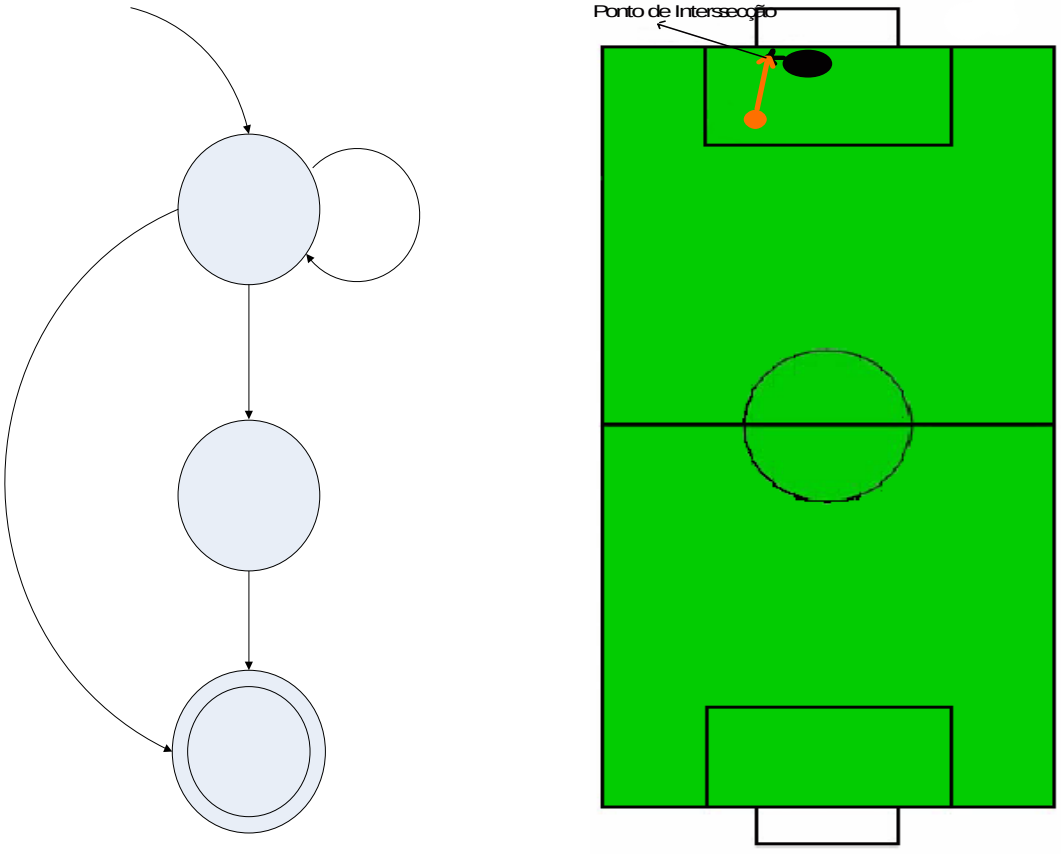
(a) (b)

Fig 3.2.

(a) Máquina de Estados do comportamento Follow Ball onde Inicial Position indica a posição no centro do segmento de recta paralelo à linha de baliza, que ´onde o guarda-redes se posiciona inicialmente e Follow Position indica a posição calculada para que o guarda-redes proteja melhor a baliza. (b) Figura ilustrativa dessa mesma situação.

3.3.2.2 InterceptBall

Procura o ponto de intersecção entre a trajectória da bola e a linha onde o guarda-redes se movimenta e tenta interceptar a bola quando esta atingir a linha. À semelhança do comportamento anterior as figuras seguintes complementam a descrição do comportamento.



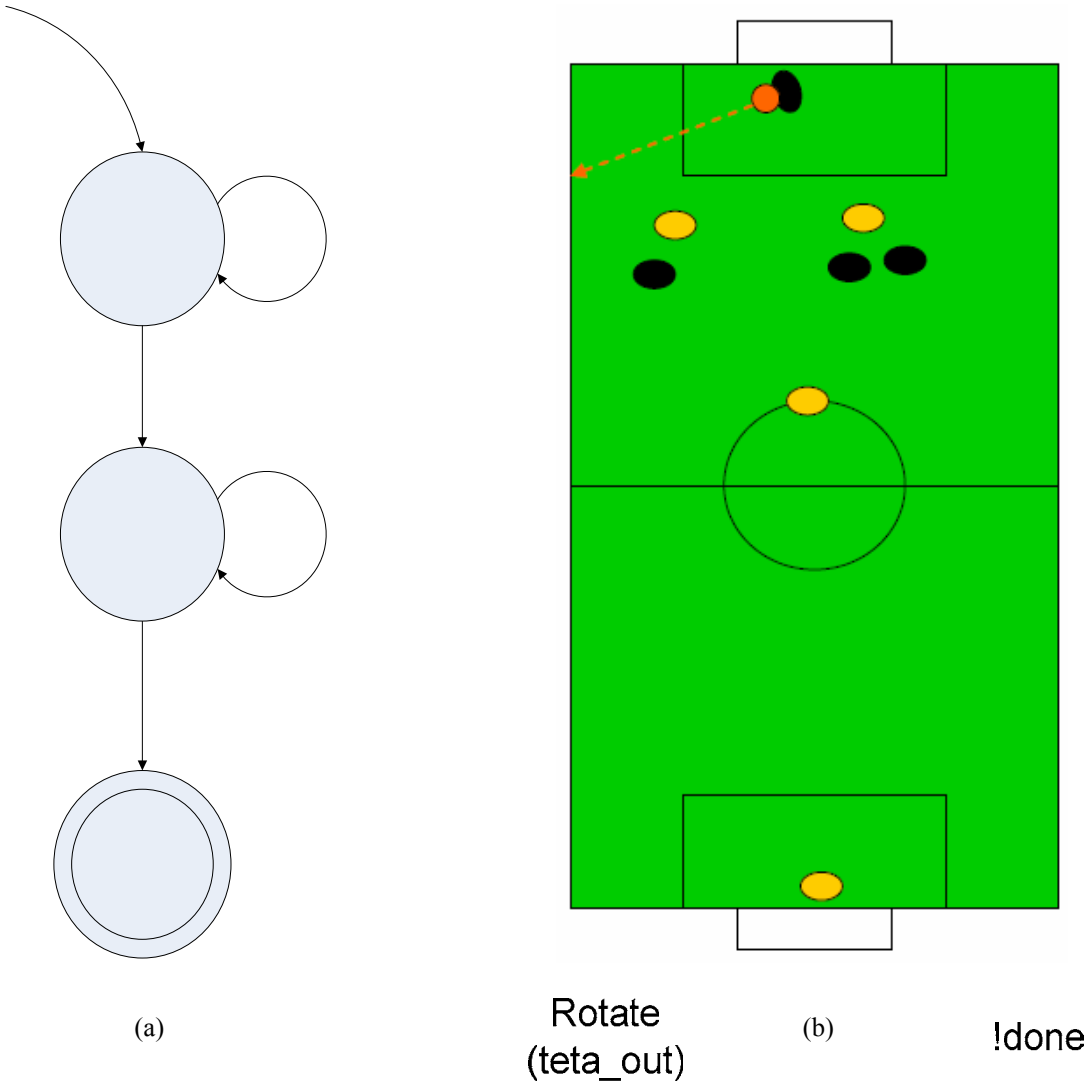
(a) (b)

Fig 3.3.

(a) Máquina de Estados do comportamento Intercept Ball, onde Point of Intersection representa o ponto calculado, como sendo o ponto em que a boa se cruza com a linha onde o guarda-redes se movimenta. (b) Figura que ilustra o pretendido.

3.3.2.3 KickOut

Quando o guarda-redes tem a bola controlada, e não existe um colega bem colocado de modo a iniciar um ataque, remata a bola para fora do campo.



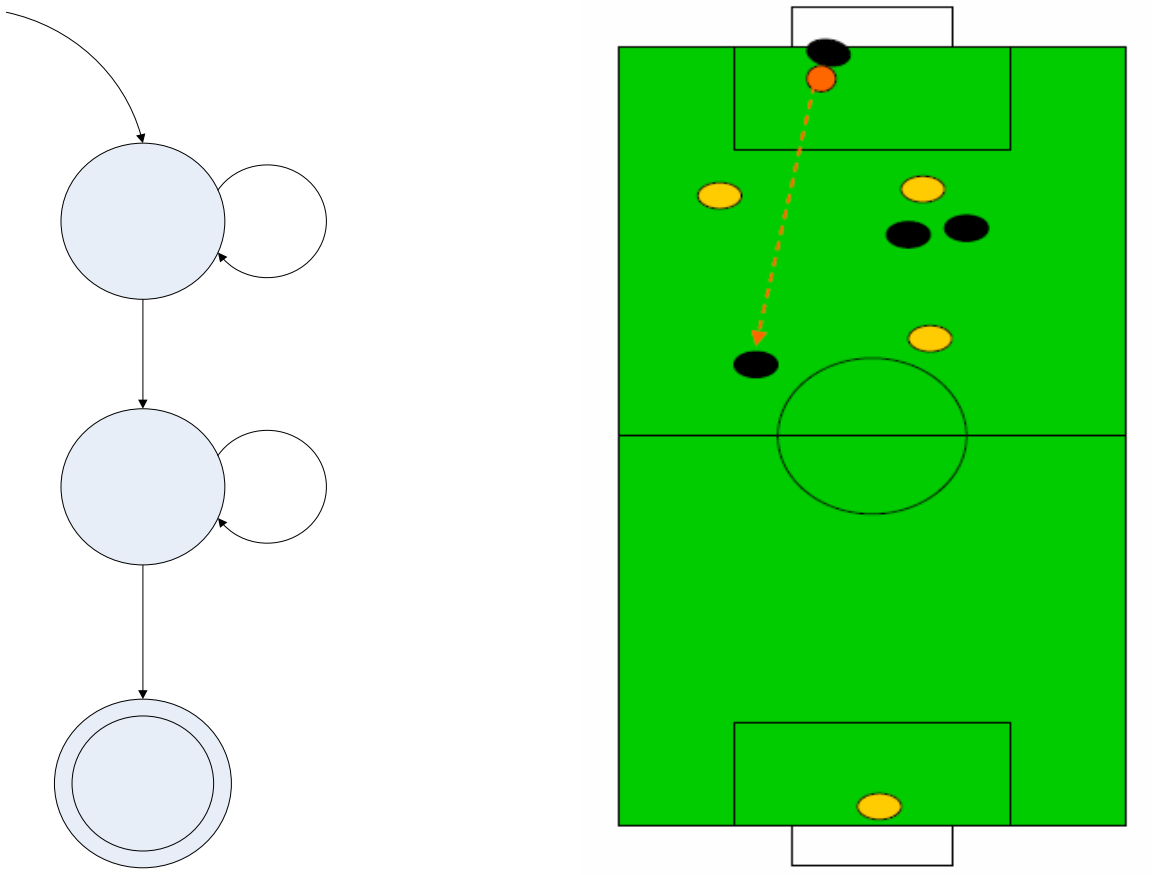
(a) (b)

Fig 3.4.

(a) Máquina de Estados do comportamento KickOut, onde teta_out representa a primeira direcção encontrada entre o guarda-redes e a linha lateral que não tem obstáculos. (b) Possível situação de jogo, onde o guarda-redes, caso não tenho nenhum companheiro em condições para receber a bola. opta por colocá-la fora de campo

3.3.2.4. PassBall

Quando tem a bola controlada, e existe um colega bem colocado de modo a iniciar um ataque, passa a bola para o colega. Este comportamento não está pensado de momento como um comportamento relacional, o guarda-redes apenas chuta a bola para a posição do campo onde o jogador melhor colocado se encontra.



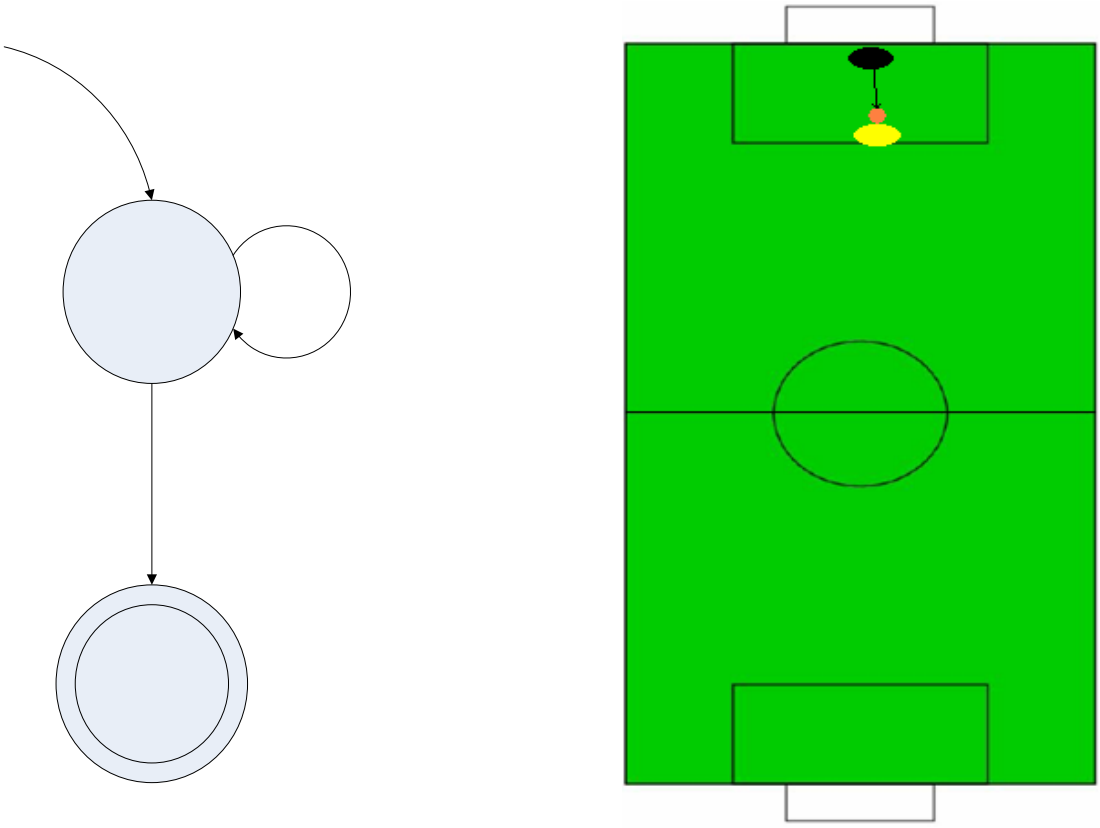
(a) (b)

Fig 3.5.

(a) Máquina de Estados do comportamento PassBall onde teta_teammate representa a direcção do companheiro de equipa em boa posição para receber a bola. (b) Figura ilustrativa de uma possível situação de jogo, onde o guarda-redes ao ver um companheiro em colocado passa-lhe a bola.

3.3.2.5. FaceOpponent

Embora o seu funcionamento seja semelhante ao funcionamento do InterceptBall, visto que ambos tentam ficar com a bola, diferem no sentido em que no InterceptBall o guarda-redes vai ter com a bola sem sair da linha onde se encontra e no GetClose2Ball sai da linha de encontro à bola, sendo que este comportamento é útil, em situações em que o adversário tem a bola controlada dentro da área e se prepara para rematar, onde este comportamento vai retirar ao adversário hipóteses de sucesso visto que vai reduzir o ângulo e eventualmente apoderar-se da bola.

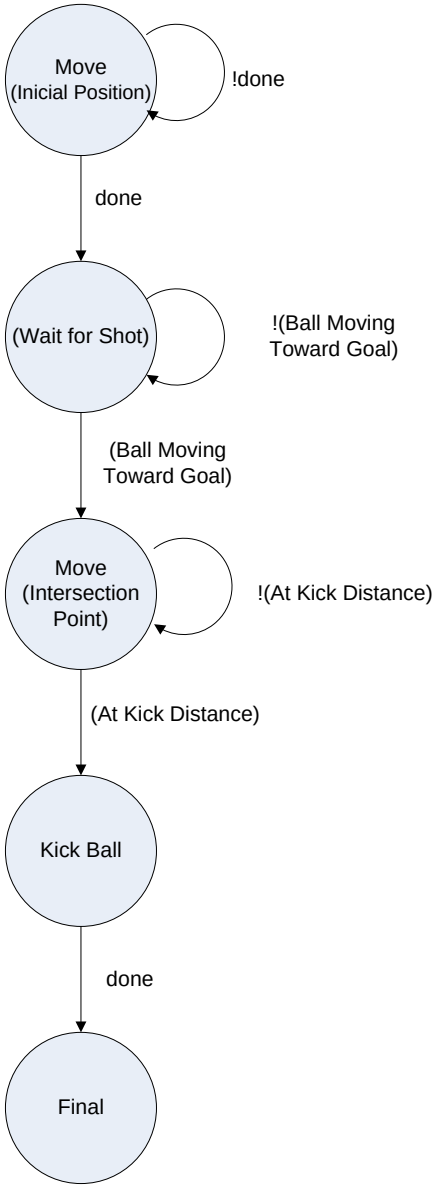


(a) (b)

Fig 3.6

(a) Máquina de Estados do comportamento FaceOpponent. (b) Possível situação de jogo.

3.3.2.6. Defend Penalty



(a)

Fig 3.7.

(a) Maquina de Estados do comportamento Defend Penalty onde o guarda-redes espera que exista um remate por parte do adversário para depois tentar defender a bola num comportamento semelhante ao Intercept Ball

3.3.2.7. Goal Kick

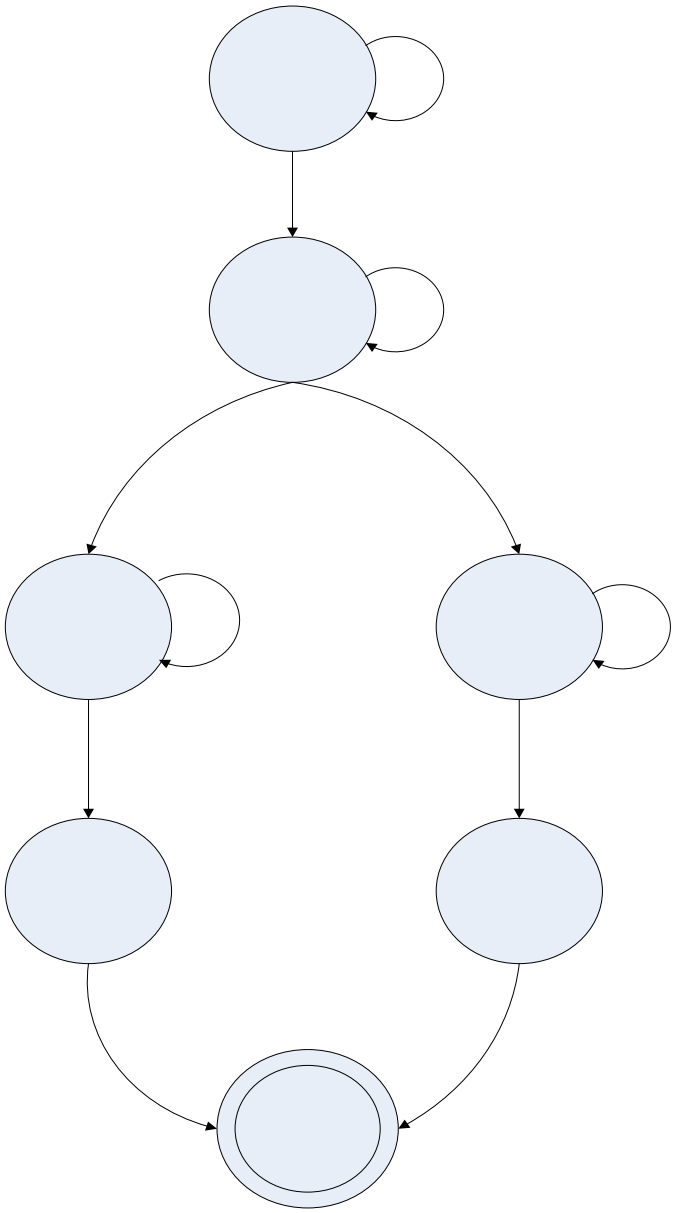


Fig 3.8.

(a) Máquina de Estados do comportamento Goal Kick, onde o guarda-redes, aquando de um pontapé de baliza verifica se tem algum colega bem colocado e caso tenha passa-lhe a bola. Caso contrário envia-a para fora do campo

3.3.3. Acções Primitivas

São acções que se limitam a uma só acção como ir para um ponto, ou rodar. As acções primitivas são acções atómicas que são executadas com a ajuda das funções primitivas de orientação. As funções primitivas de orientação são funções criadas com o objectivo de comunicar com o hardware de mais baixo nível do robot. Todas as acções primitivas foram implementadas com base em acções já existentes (move, rotate e kick), no entanto, foram todas construídas de novo, onde foi adicionada uma analise sensorial às acções já existentes, exceptuando o `Wait_for_Shot`.

- `KickBall`: Remata a bola com força máxima.dando ordem ao dispositivo de remate.

- `PassBall`: Passa a bola para uma dada posição com metade da força máxima.

- `MoveToInitialPosition`: Para cada comportamento, move-se para a posição definida como inicial para esse mesmo comportamento.

- `MoveToFollowPosition e MoveToInterceptPosition:` Move-se para uma dada posição de seguimento da bola, ou de intercepção da mesma (a explicação para a determinação destes pontos, encontra-se descrita na secção 4.1)

- `Rotate_Team_Mate e Rotate_Out`: Roda sobre si mesmo na direcção de um companheiro de equipa ou de forma a que a bola saia de campo, direcção essa encontrada através de uma análise do mundo (explicada em pormenor na secção 4.1).

- `MoveToBall`: Identifica a posição da bola e tenta ir ao seu encontro.

- `Wait_for_Shot`: No caso de penalty, o guarda-redes espera pelo remate do adversário.

3.3.4. Predicados

Os predicados dão-nos informação sobre o meio e o seu valor poderá ser verdadeiro ou falso. A transição entre os seus valores resulta da observação do meio e dão-se na ocorrência de um evento no meio. Destes predicados os que tivemos de acrescentar à lista dos já existentes foram: Teammate_Well_Positioned, Ball_in_Area, Ball_Alone, Ball_Other_Team, Miedfield_Alone, Ball_in_Attacking_Field e Closer_to_Ball, Ball_Moving_Towards_Goal, Penalty, Gk_Ready_for_Penalty?.

- `Found_ball`: Verdadeiro se consegue localizar a bola. A posição da bola é determinada por visão, a chamada deste predicado limita-se a devolver um valor lógico guardado numa variável local do blackboard. Este predicado já existia.

- `At_kick_distance`: A bola posiciona-se próxima do jogador e a essa distância é possível chutar a bola. O valor da distância considerado como sendo próximo do jogador, de maneira a que a essa distância ele a possa chutar, encontra-se guardada no Blackboard, a função limita-se a determinar a posição do jogador e da bola, calcular a distância entre esses dois pontos, caso essa distância seja menor ou igual à distância predefinida como sendo de chuto, retorna verdadeiro, caso contrário retorna falso. Este predicado já tinha sido implementado anteriormente.

- `Teammate_Well_Positioned`: Verdadeiro quando existe um elemento da equipa bem posicionado para passar a bola, o critério de bom

posicionamento é não ter adversários próximos. Para implementação do mesmo necessitámos da posição dos colegas recolhida do BlackBoard, da posição da bola, da própria posição do próprio jogador e da posição de todos os obstáculos. Com esses dados determinamos quais dos obstáculos eram adversários ficando com o registo das suas posições e verificámos se havia algum colega de equipa cuja sua posição não estivesse próxima de nenhumas das posições dos adversários. Caso ele encontre algum colega de equipa nessa situação retorna verdadeiro, se não existir nenhum retorna falso. Este predicado foi por nós pensado e implementado de origem usando funções já existentes para determinar a posição da bola, a própria posição e para aceder ao BlackBoard.

- `Ball_Alone`: Quando a bola não está na posse de nenhuma das equipas. Recolheu-se a posição da bola e a posição de todos os obstáculos, caso ele não estivesse a uma distância considerada por nós como próxima de nenhum dos obstáculos então retorna verdadeiro.

- `Ball_in_Area`: A bola situa-se na grande área. Através da posição da bola e dos limites da área consegue-se verificar se a bola se encontra dentro da área. Os limites da área são guardados em variáveis do BlackBoard e já estão predefinidos, a posição da bola é determinada através de funções já existentes.

- `Ball_Other_Team`: A bola encontra-se na posse dos adversários. Determina-se a posição da bola e a posição dos adversários pelo mesmo método descrito no predicado Teammate_Well_Positioned verica-se depois se a distância da bola a algum dos adversários se encontra dentro do limite da distância considerada por nós como próxima.

- `Ball_Moving_Towards_Goal`: A bola move-se em direcção à baliza da equipa. Através de amostras sucessivas da posição da bola em instantes diferentes, verifica-se se a bola na ultima amostra se encontra mais próxima do guarda-redes que as anteriores.

- `Miedfield_Alone`: A metade do campo da equipa encontra-se apenas com o guarda-redes. Através do BlackBoard ficamos com a informação dos limites do nosso meio campo, essa informação está é predefinida. Recolhemos os dados relativos às posições de todos os obstáculos. Verificamos se alguma posição relativa aos obstáculos se encontra dentro dos limites do nosso meio campo.

- `Ball_in_Attacking_Field`: A bola encontra-se na metade do campo atacante. Recolhemos a posição da bola, recolhemos a posição dos limites do campo atacante através do BlackBoard e vericamos se a bola se encontra algures nesses dentro desses limites.

- `Closer_To_Ball`: É o jogador que consegue chegar primeiro à bola. Essa conclusão é retirada por análise de distância à bola, do jogador e de todos os restantes jogadores se ele estiver mais próximo então o predicado é válido.

- Penalty: Quando é assinalada pela *referee-box* uma situação de *penalty* contra a equipa.

- Gk_Ready_for_Penalty?: Quando o guarda-redes já está colocado e pronto para defender um *penalty*. Devolve o valor lógico guardado numa variável global do *BlackBoard* que o guarda-redes actualiza quando está na posição de defesa de *penalty*.

- Goal_Kick?: Quando é assinalada pela *referee-box* uma situação de *goal kick* contra a equipa.

3.3.5. O Papel Guarda-Redes

3.3.5.1. Sistema de Decisão do Guarda-Redes.

O sistema de decisão usado para o guarda-redes é o sistema de decisão reactivo, pois a mudança de comportamento a adoptar por este depende única e exclusivamente da ocorrência de eventos pré determinados, ou seja do estado do mundo em cada momento.

A razão pela qual não usamos um sistema híbrido prende-se com o facto de não fazer qualquer tipo de sentido dada a simplicidade da escolha de um comportamento a tomar num determinado instante e devida à rapidez com que terá que decidir e agir.

No entanto pensamos que no futuro seja uma boa hipótese o sistema híbrido, aliado ao facto de o guarda-redes poder tomar comportamentos relacionais e organizacionais e poder-se assim inserir numa estratégia de equipa, como por exemplo participar numa tentativa de golo através de uma boa colocação de bola.

A implementação do nosso sistema de decisão reactivo na selecção do comportamento a adoptar pelo guarda-redes foi feita com base num sistema de decisão baseada em lógica (BLDU), que se implementa com regras. Cada regra é constituída por um conjunto de pré condições que terão que ser todas verdadeiras de modo a que a regra seja aplicada.

Estas regras foram implementadas em linguagem C++ e cada regra é constituída por uma frase lógica que relaciona um série de predicados cujo seu valor poderá ser verdadeiro ou falso dado a análise do mundo que assinala a ocorrência ou não ocorrência de um evento.

A BLDU é constituída por um ciclo em que o seu corpo é constituído por sequências de IF-ELSE, que testam a ocorrência de uma determinada regra.

A ocorrência de uma determinada regra, acarreta a escolha de um comportamento a adoptar.

A razão pela qual é utilizada a BLDU e não máquinas de estados ou árvores de decisão deve-se ao facto de o guarda-redes não ter estados específicos, ou seja, não entrar dentro de um comportamento especifico e só sair de lá apenas quando tiver cumprido a

sua missão, mas sim um comportamento que se enquadra com o mundo que o rodeia, podendo saltar entre comportamentos, bastando que para isso se alterem as condições à sua volta.

3.3.5.2. Coordenador de Comportamentos.

O coordenador de comportamentos (definido na arquitectura como *Behavior Coordinator*) (neste caso estará a executar o papel do guarda-redes), é responsável por seleccionar um comportamento, de uma lista de comportamentos disponíveis para aquele papel, como base em regras. Ao seleccionar o comportamento será responsável por criar uma *thread* que irá conter no seu ciclo de execução o *plugin* referente ao comportamento a executar, este bloco está referido na arquitectura como o Executor de Comportamentos (Behavior Executor).

No anexo [C] está apresentado a título de exemplo, meramente ilustrativo no sentido de um melhor entendimento do papel do guarda-redes, o ciclo de execução dos comportamentos que não deverá ser confundido com uma máquina de estados (apenas serve para ilustrar as possíveis transições entre diversos comportamentos com base em características de jogo), pois como já foi referido a escolha de um comportamento a executar numa dada altura é feita pela *thread*, que executa o papel do guarda-redes, com base em regras, por nós previamente estipuladas.

3.3.5.3. Regras de escolha de comportamentos.

Foram escolhidas várias regras nas quais o *Behavior_Coordinator* se baseia para escolher num dado momento um dado comportamento. As regras consistem em condições lógicas baseadas em predicados. Portanto cada comportamento terá seleccionado para si uma condição lógica que garanta que quando esta ocorrer a sua máquina de estados será executado. São indicadas a seguir as regras atribuídas a cada comportamento:

Comportamento	Condições necessarias para a sua execução
Follow Ball A	! Miedfield Alone && ! Ball in Area
Follow Ball B	Miedfield Alone && ! Ball in Area
Get Close to Ball A	Ball in Area && Ball Other Team
Get Close to Ball B	Miedfield Alone && ! Ball Attacking Field && !Ball in Area
Intercept Ball	Ball in Area && Ball Alone
Kick Out	At Kick Distance && !Teammate Well Positioned
Pass Teammate	At Kick Distance && Teammate Well Positioned
Todos	&& (!Penalty)&&(!Goal_Kick)

Tabela 3.1. Regras necessárias para a ocorrência de cada comportamento

Para a criação destas regras, foi necessário que não existisse sobreposição dos diferentes comportamentos, ou seja, teve que se formular as regras de modo a que as mesmas condições não levassem à execução de dois comportamentos diferentes, pois se tal acontecesse o comportamento do guarda-redes iria ser incoerente e provavelmente a escolha certa não seria a mais apropriada para cada momento.

Dois casos específicos são a diferenciação entre os comportamentos Follow Ball A e B, e entre os comportamentos Get Close to Ball A e B.

Relativamente ao Follow Ball, a única diferença entre os dois comportamentos é a posição no campo onde o guarda-redes segue a bola.

No caso dos comportamentos Get Close to Ball, a maior dificuldade prendia-se

com o facto de diferenciar em que caso o guarda-redes devia ir ter com a bola, estando esta fora da área, de modo a que não colocasse em risco a guarda da baliza, numa situação em que pode aproveitar um adiantamento no campo por forma a apoderar-se mais facilmente da bola.

4. Implementação do Software

Como vimos em cima toda a arquitectura foi desenvolvida em linguagem C++, com base nos conceitos de programação orientada por objectos.

O objectivo proposto era inserir na arquitectura existente do projecto a máquina de estados dos comportamentos que pretendíamos que o guarda-redes realizasse, o processo de selecção das possíveis máquinas de estados e a execução da escolhida.

Na concepção do código, a criação das classes necessárias para responder correctamente ao problema, foi feita de forma a garantir uma correcta generalização do problema, de maneira a permitir uma reutilização do código criado.

Foram portanto, criadas classes abstractas que possuem os métodos e atributos que achámos que seriam comuns a todos os comportamentos e papeis a implementar.

As classes abstractas criadas foram a classe *Behavior*, onde todos os comportamentos por nós criados são suas subclasses e a classe *Role*, cuja classe que representa o papel do guarda-redes é sua subclasse. Ambas as classes são subclasses da classe *Thread* , classe esta que agrupa todos os métodos e atributos inerentes à criação de uma *thread*, execução da mesma e sua destruição, isto porque todos os comportamentos e papeis serão no fundo *threads* cujo seu ciclo de execução será uma máquina de estados ou uma sequência de escolha de comportamentos e sua criação respectivamente.

A classe Behavior pode ser descrita com base nos seus atributos e métodos.

Os atributos comuns a todos os comportamentos são uma *String* identificadora que guardará o nome do comportamento, um inteiro que guardará uma constante que representará o estado em que o robot se encontra actualmente, e um contador de estados que servirá para indicar quantos ciclos de execução do comportamento existiram em que o robot se encontrava num determinado estado, todos estes atributos são acessíveis pelas subclasses desta classe.

Os métodos comuns a todos os comportamentos são, o método que retorna o identificador do comportamento, um método chamado *setup* herdado da super classe

Thread que servirá para fazer todas as inicializações necessárias antes de ser lançada a *thread*, ou seja todas as inicializações necessárias antes da execução de um comportamento, e um método intitulado de *execute* também herdado da super classe *Thread* que conterá o ciclo de execução da *thread* criada, ou seja a máquina de estados a executar. As subclasses criadas fazem um overriding dos métodos *setup* e *execute* herdados e ainda terão alguns métodos e atributos que apenas a elas dirão respeito, não estando portanto estes declarados na super classe *Behavior*. As subclasses criadas no âmbito deste trabalho correspondem cada uma delas a um comportamento e como tal foram baptizadas com o mesmo nome: *FollowBall*, *InterceptBall*, *Face_Opponent*, *KickOut*, *PassBall*, *Defend_Penalty*, *Goal_Kick*. A classe *Followball* tem ainda para além dos atributos herdados a posição inicial onde se irá colocar no inicio da sua execução e um método chamado *follow_posture* que retorna a posição onde o guarda-redes se deverá colocar na linha onde se desloca, dada a posição da bola, para uma melhor defesa da baliza. A classe *InterceptBall* terá ainda um método (*Intercept_Posture*) próprio da classe que irá retornar a posição de intersecção da bola na linha onde este se desloca. A classe *Defend_Penalty* irá ter os dois métodos citados acima (*follow_posture* e *Intercept_Posture*) e ainda um atributo com a posição onde se deverá colocar inicialmente.

Finalmente a descrição da classe Role, Os atributos comuns a todos os papeis são uma *String* identificadora que guardará o nome do papel, um inteiro que guardará uma constante que representará o comportamento em que o robot se encontra actualmente, e um contador de comportamentos que servirá para indicar em quantos ciclos de execução do papel o robot se encontrava num determinado comportamento, todos estes atributos são acessíveis pelas subclasses desta classe.

Os métodos comuns a todos os papeis são, o método que retorna o identificador do papel, um método chamado *setup* herdado da super classe *Thread* que servirá para fazer todas as inicializações necessárias antes de ser lançada a *thread* e um método intitulado de *execute* também herdado da super classe *Thread* que conterá o ciclo de execução da *thread* criada. As subclasses criadas fazem um overriding dos métodos *setup* e *execute* herdados. Só foi criada uma subclasse da classe Role a que chamamos Omni_Goalkeeper e representa o papel do guarda-redes. O método *setup* terá todas as inicializações

necessárias antes da execução do papel e o método *execute* que conterá a sequência de regras (IF-ELSE´s encadeados) em que uma vez uma regra validada, cria uma *thread* responsável pela execução do comportamento a ela referente.

Criamos também uma classe que agrupa todos os predicados (*Predicate*), só contendo portanto esta classe métodos estáticos (*static*).

Achámos necessário a identificação das classes por nós criadas, bem como os métodos, atributos e relações entre as mesmas. Usamos para tal ilustração um diagrama UML que se encontra no anexo [E].

4.1. Descrição da Implementação dos Comportamentos

As subclasses que foram criadas dizem respeito aos cinco comportamentos que achámos que o guarda-redes deveria exibir na execução do seu papel e são eles:

- *Follow Ball*
- *Intercept Ball*
- *Get Close to Ball*
- *Kick Out*
- *Pass Ball*
- *Defend_Penalty*
- *Goal_Kick*

Relativamente a cada um deles achámos relevante focar algumas escolhas de implementação que irão ser discriminadas a seguir:

- **`Followball A:`** A forma utilizada para calcular a posição em que o robot se deve colocar, teve a ver com a linha que passa à frente do robot, ou seja, supondo que o robot não vai estar exactamente em cima da linha de golo, foi projectado que estaria cerca de

15cm (e não mais para evitar que a bola entre ele e o poste) à frente dessa mesma linha. Tendo em conta que o robot tem de diâmetro aproximadamente 50cm, a linha utilizada para calcular a posição onde o guarda-redes se deve situar está cerca de 65cm à frente da linha final. Depois utilizando a posição da bola e a posição dos postes, é calculada a intersecção entre as rectas que vão da bola a cada um dos postes e a linha situada à frente do robot. É então calculado o ângulo que fazem essas duas rectas (que unem os postes à bola). Daí calcula-se então o ponto de intersecção entre a linha do guarda-redes e um ponto que passe no meio desse ângulo, deslocando-se posteriormente o robot para essa posição.

- **`Omni_Followball B`:** O raciocínio seguido foi equivalente ao do Follow Ball A com, onde a única diferença foi a linha escolhida para o guarda-redes se colocar e consequentemente, a linha utilizada para efectuar os cálculos é diferente

- **`Omni_InterceptBall`:** Neste caso é avaliada a direcção da bola, e, com essa direcção e com a posição da bola é calculada a intersecção entre a bola e a linha do guarda-redes, dirigindo-se este, posteriormente para esse ponto.

- **`Omni_GetClose2Ball`:** Quando este comportamento é chamado, o guarda-redes apenas verifica onde esta a bola e tenta aproximar-se dela de modo a poder manda-la para fora ou endereçá-la a um colega

- **`Omni_KickOut`:** Aqui, quando o guarda-redes tem a bola à distância de remate, é procurada uma direcção onde não existam obstáculos. Este cálculo faz-se a partir de uma linha perpendicular à linha lateral da qual o guarda-redes está mais próximo, sendo que no caso de não existirem obstáculos, remata a bola nessa direcção, caso existam, volta a verificar mas numa direcção de cinco graus mais para o interior, e assim sucessivamente ate encontrar uma trajectória que esteja livre.

- **`Omni_PassBall`:** Através da posição que os companheiros de equipa enviam para a *blackboard*, analisa um a um, e o primeiro para o qual não existam obstáculos, entre ele e

o guarda-redes, é-lhe endereçada a bola.

- **`Defend_Penalty:`** Muito parecido com o comportamento intercept ball, mas com a diferença que só acontece no caso do sinal de `penalty` ter sido dado pela `refeeree box`. Aí situa-se no centro da baliza e espera que a bola seja rematada, tentando depois intercepta-la.

- **`Goal_Kick:`** Também só é executado quando `referre box` da sinal de `goal kick`, e aí caso tenha um companheiro de equipa bem colocado passa-lhe a bola e, caso contrário envia-a para fora.

5. Descrição dos Testes

Foram efectuados testes à parte do código efectuado em C++, para inserir no projecto o papel do guarda-redes. Não nos foi possível apresentar testes, sobre o código efectuado como era de esperar com o simulador Webots [11] e com o robot real omnidireccional. Os testes foram portanto feitos escrevendo os algoritmos usados no projecto por nós criado em MatLab e criando uma interface com o utilizador com um campo com as medidas correspondentes com as medidas do campo do LRM (onde cada centímetro no simulador corresponde a um metro no campo real.

A interface com o utilizador apresenta-se no anexo D, Fig D.1., espera-se que o utilizador insira uma situação de jogo. O utilizador tem a possibilidade de colocação de todos os elementos do jogo, que são eles os quatro adversários, os três colegas de equipa, a bola e ainda o guarda-redes. Não é obrigatória a colocação de todos os elementos, os elementos que não forem colocados assume-se que se encontram na origem do referencial do campo, que como foi mencionado anteriormente se encontra no meio do campo. Recolhida toda a informação colocada pelo utilizador, então todos esses dados são usados para primeiro de tudo calcular o valor lógico de todos os predicados. Com esses valores lógicos dos predicados, testamos a validação das regras existentes que nos indicam qual o comportamento que o guarda-redes irá efectuar. Uma vez escolhido o comportamento é calculada a situação na qual o guarda-redes se irá colocar. Toda a informação sobre a nova situação do guarda-redes e o comportamento escolhido é impressa na interface. Como foi dito os algoritmos para cálculo dos predicados, escolha do comportamento e cálculo da posição de colocação do guarda-redes é feito tendo por como base no código efectuado para o projecto.

Na simulação criada em Matlab, foram testada várias situações de jogo, e em todas elas, era sempre escolhido o comportamento pretendido assim como todos comportamentos escolhidos faziam o que lhes era pedido.

Foram testadas situações em que a bola estava na área sem ninguém por perto mas sem estar ao alcance do guarda-redes, ao alcance do guarda-redes e também situações em

que a bola estava na área juntamente com adversários, situações em que o guarda-redes se encontrava sozinho no seu meio campo, com a posse de bola e com companheiros em condições de a receber, assim como sem companheiros que não podiam receber a bola, tendo nestas situações a simulação correspondido ao que era pretendido.

6. Resultados

Dada a não possibilidade de teste do código dos comportamentos criados a inserir no projecto, no simulador e no robot real, os resultados que iremos apresentar dizem respeito aos algoritmos implementados e como foi visto os testes são efectuados em MatLab.

No anexo D encontram-se várias situações de jogo.

6.1. Escolha do comportamento adequado.

Os resultados efectuados indicam que é efectuada uma correcta escolha do comportamento adequado dada uma determinada situação de jogo, o que indica uma correcta implementação dos predicados e regras, nesta secção apenas foram indicadas algumas situações a titulo de exemplo, mas foram efectuadas muitas experiências e todas elas revelaram os resultados pretendidos.

6.2. Cumprimento do objectivo pedido.

Uma vez escolhido o comportamento, o estado objectivo do guarda-redes é coerente com o estado objectivo pretendido para aquele comportamento, revela uma boa implementação dos cálculos efectuados, referentes à posição de seguimento da bola, direcção de intercepção da bola e direcção de chuto para colocação da mesma.

7. Conclusão

O objectivo final deste trabalho não foi cumprido, pois era esperado conseguir colocar o novo robot omnidireccional a comportar-se como um guarda-redes. Não pudemos dar garantias que o código implementado o venha a conseguir, muito provavelmente não irá funcionar correctamente quando testado tanto em simulação, bem como com o robot real. Podemos sim dar algumas garantias da implementação do papel de guarda-redes, seus comportamentos e predicados pela simulação efectuada em MatLab.

Portanto como vimos nos resultados acima demonstrados podemos garantir que os comportamentos e escolha de comportamentos foi bem pensada e implementada, os algoritmos de cálculo da posição de seguimento da bola, direcção de intercepção da bola, direcção de passe de bola, direcção de chuto da bola para fora, e valor dos predicados dado o estado do mundo foi implementada com sucesso.

Podemos referir que este ano coincidiu com uma fase de grandes alterações a nível de toda a estrutura do projecto no qual o trabalho se inseria. Essas alterações prendem-se com a alteração da equipa dos robots existentes, para uma equipa de robots omnidireccionais, alteração da arquitectura do software e a uma mudança de simulador a que se prendia a uma necessidade de o adaptar à nova estrutura da equipa.

A equipa prepara-se para se tornar muito mais competitiva e obter melhores resultados no âmbito do estudo a que se destina. No entanto todas as alterações no sentido de a melhorar, acarretam algumas dificuldades, dificuldades essas que vieram a trazer algumas consequências em termos do andamento do trabalho realizado ao longo do ano. Ficamos em falta no sentido de não conseguirmos estar à altura de as ultrapassar de maneira a conseguir alcançar o objectivo pretendido e por essa razão o nosso trabalho está incompleto. Deixando como futuro trabalho o teste dos comportamentos criados no simulador e no robot real.

Anexo A. Sistemas de Decisão

A.1. Sistema de Decisão Reactivo

Os sistemas robóticos reactivos têm as seguintes características:

- São comportamentos como uma construção básica de blocos para acções robóticos. Um comportamento neste sistema, consiste tipicamente num par sensor/actuador, com toda a actividade sensorial a fornecer a informação necessária para satisfazer uma resposta dos motores.
- O uso de representação abstracta de conhecimento é evitado na geração de uma resposta. Sistemas puramente reactivos reagem directamente ao mundo da forma que este é sentido, evitando a necessidade de uma representação abstracta de conhecimento. Por outras palavras, o que se vê é o que se tem. Isto é particularmente valioso em mundos altamente dinâmicos e sinuosos, onde a imprevisibilidade e potenciais hostilidades estão inerentes. Construir modelos abstractos do mundo consome tempo e é um erro, uma vez que reduz a correcção com que é tomada uma acção do robot, a não ser em casos em que esse mundo é previsível.

Estes sistemas são sistemas moldáveis numa perspectiva de arquitectura de software uma vez que permite que se aumente a diversidade de comportamentos de um robot, sem que seja necessário redesenhar toda a estrutura e descartar antigos comportamentos. Esta agregação de capacidades e resultante reutilização de comportamentos é muito útil na construção de sistemas robóticos muito mais complexos.

A.2. Sistema de Decisão Deliberado

Pode decidir baseado na experiência passada e prevendo as futuras consequências dos seus actos.

O sistema pode até decidir e agir de uma forma que não poderá ser prevista pelos programadores.

Não necessita que ocorra um evento para tomar uma decisão, pode estar sempre a analisar o mundo e a tomar decisões. Pode comportar-se de maneiras diferentes dado o mesmo estado do mundo.

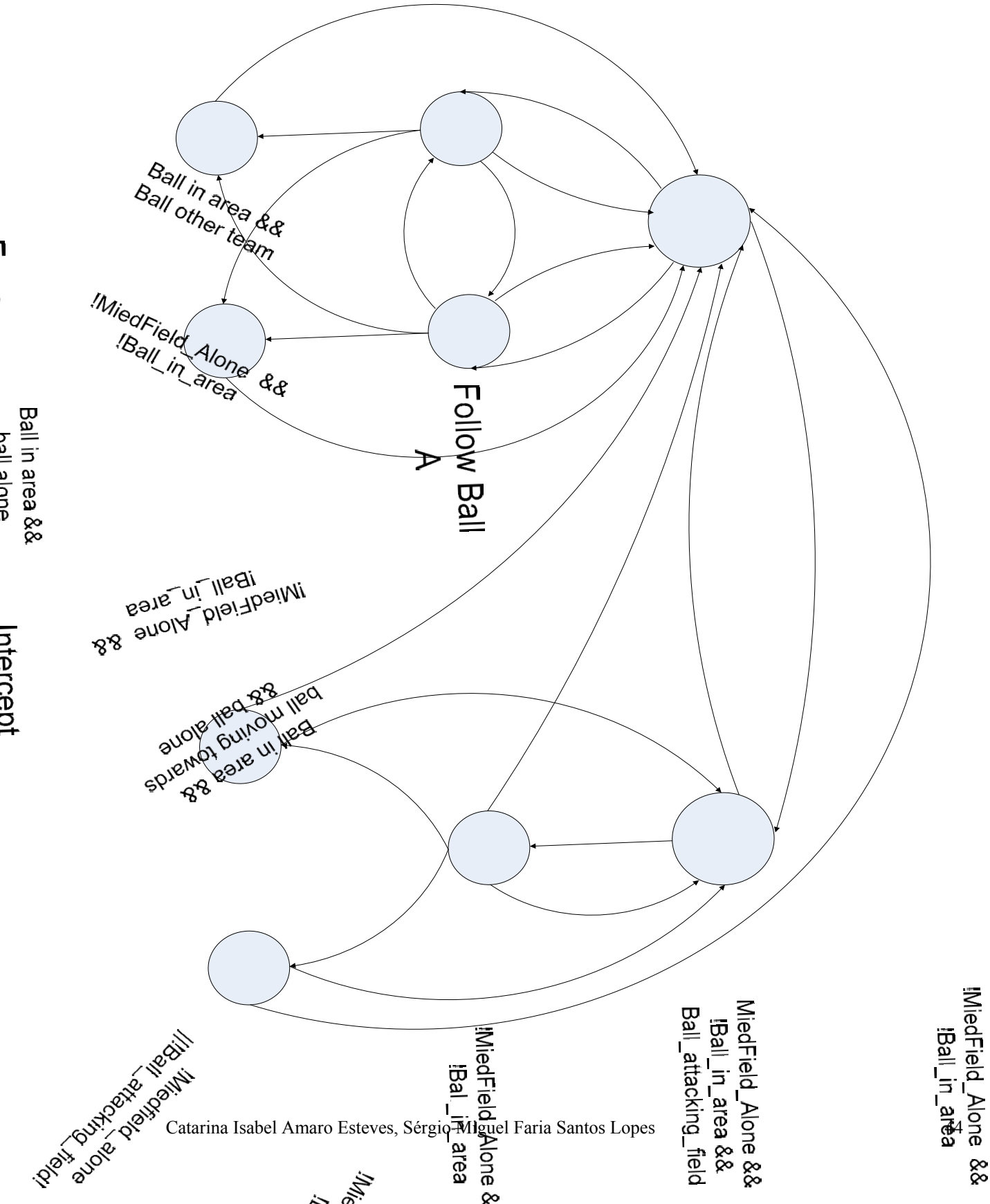
A.3. Sistemas de Decisão Híbridos

Temos o sistema de decisão reactivo e o deliberado a funcionarem em paralelo e independentemente.

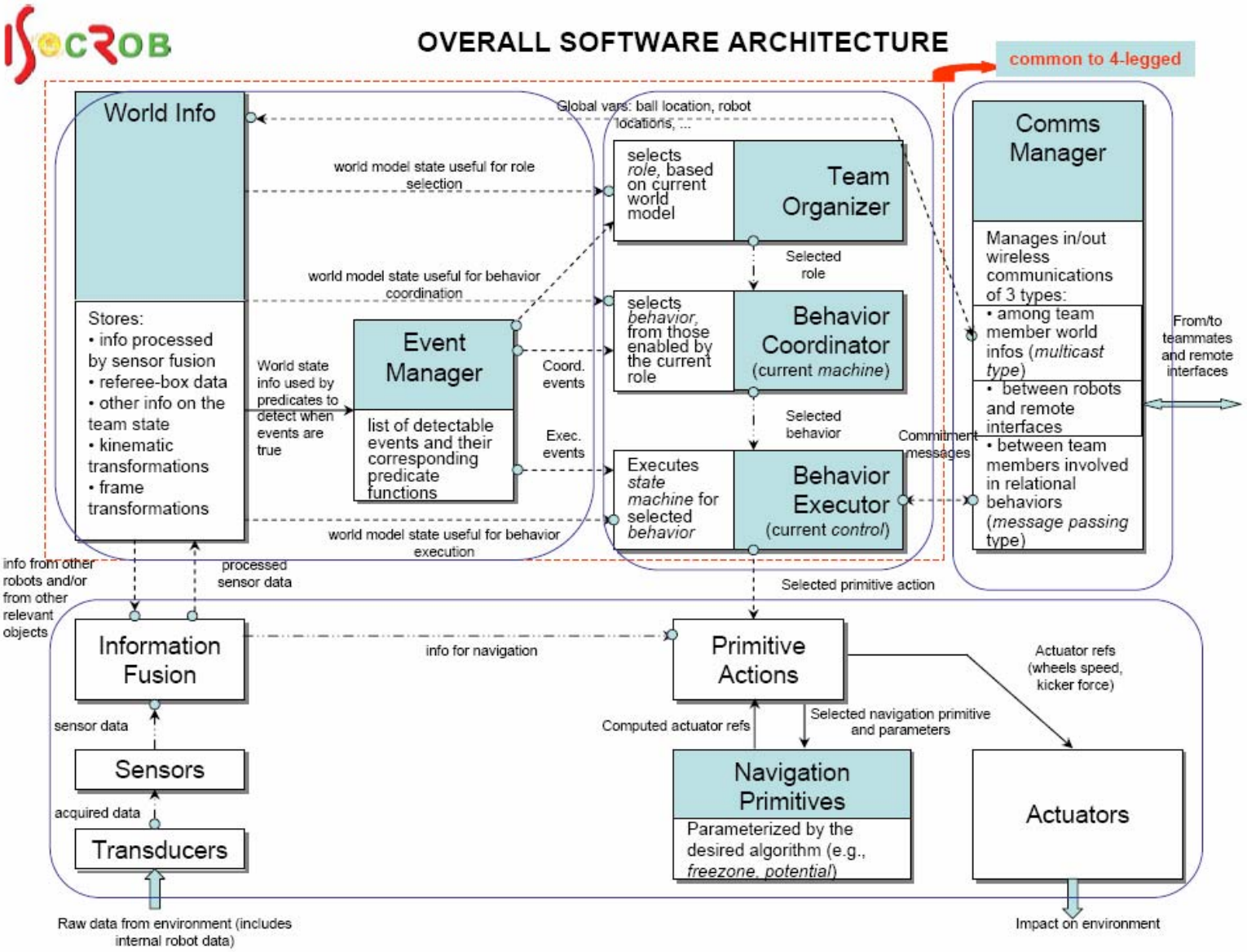
Este sistema usa normalmente as decisões tomadas pelo sistema de decisão deliberado, mas quando este sistema demora demasiado tempo a decidir ou algum evento inesperado ocorre usa as decisões tomadas pelo sistema reactivo. Embora essa decisão não seja tão boa como a tomada pelo sistema deliberado é melhor do que ficar sem fazer nada à espera de uma decisão.

A razão pela qual é preferível o uso de um sistema híbrido em vez de um sistema deliberado é porque uma decisão gerada pelo sistema deliberado poderá demorar demasiado tempo.

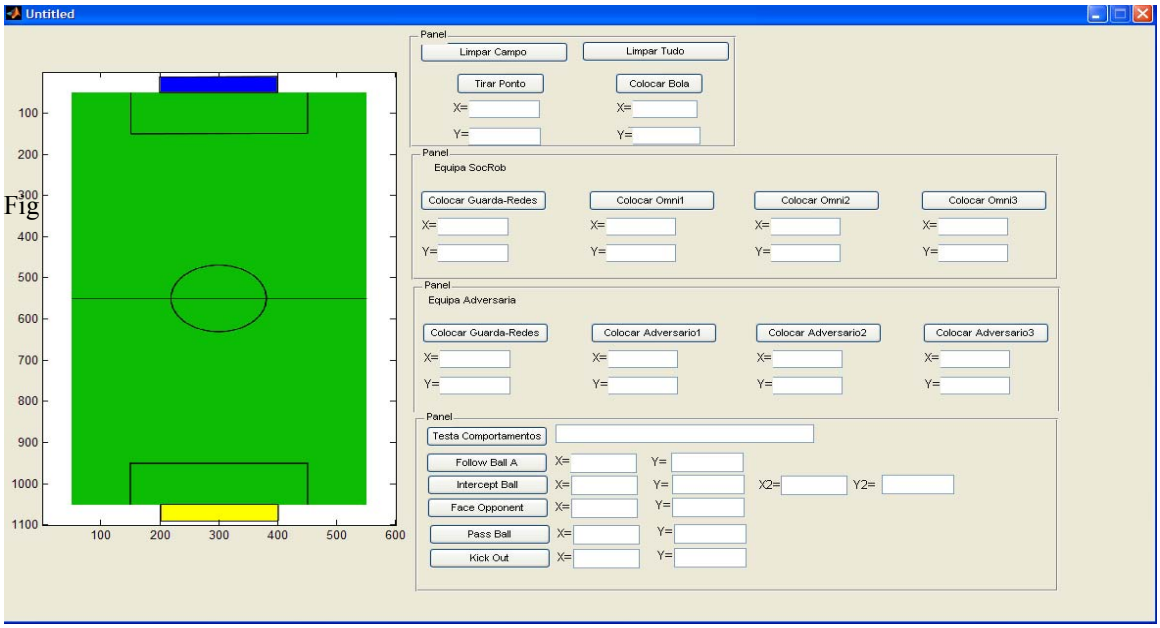
Anexo B. Ciclo de Execução



Anexo C. Arquitectura de Software da equipa ISocRob



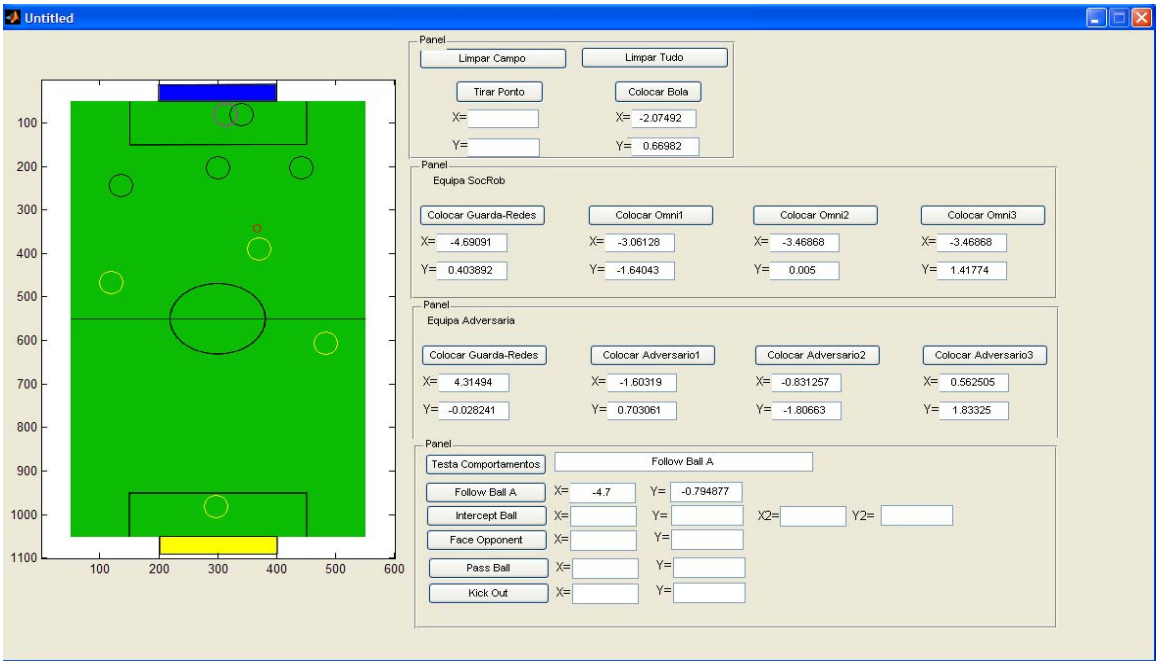
Anexo D – Situações de jogo que resultam em diferentes comportamentos.



Fig

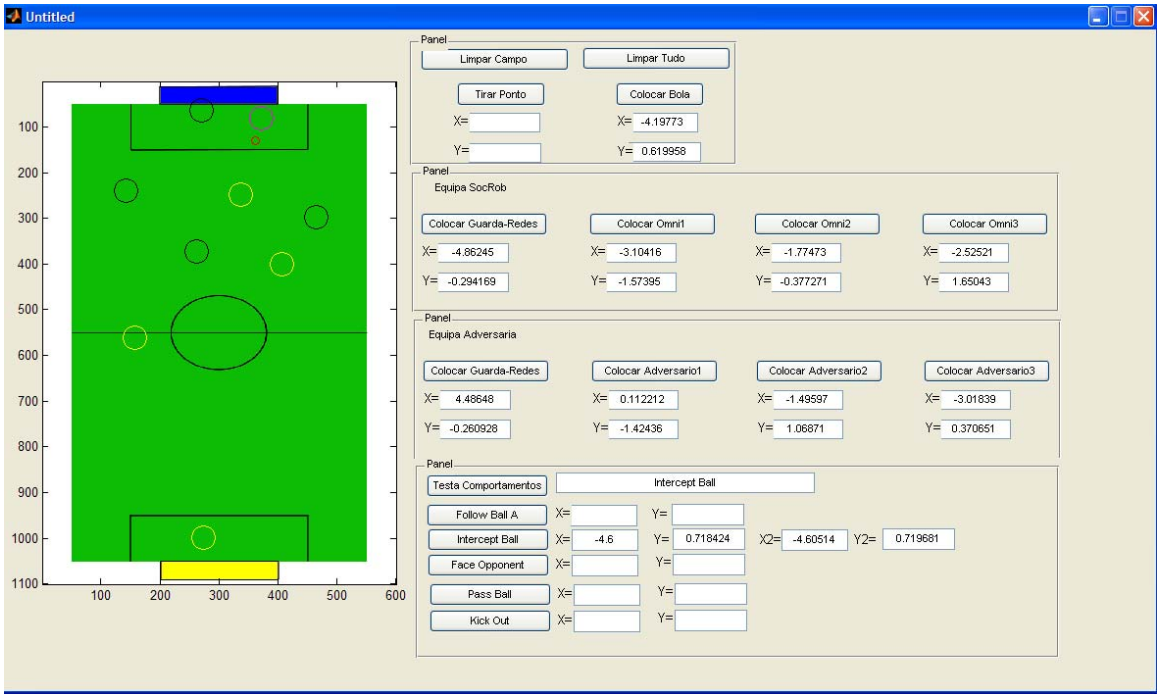
(a)

Fig D.1.
(a) Apresentação da Interface



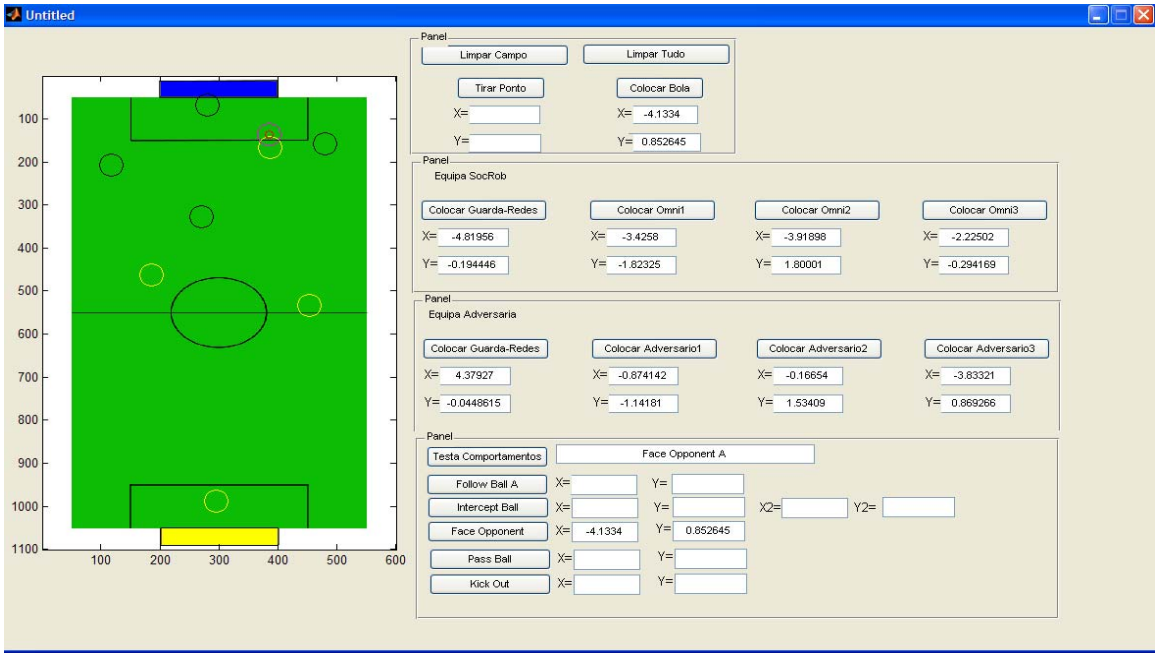
(a)

Fig D.2.
(a) Situação de Follow Ball



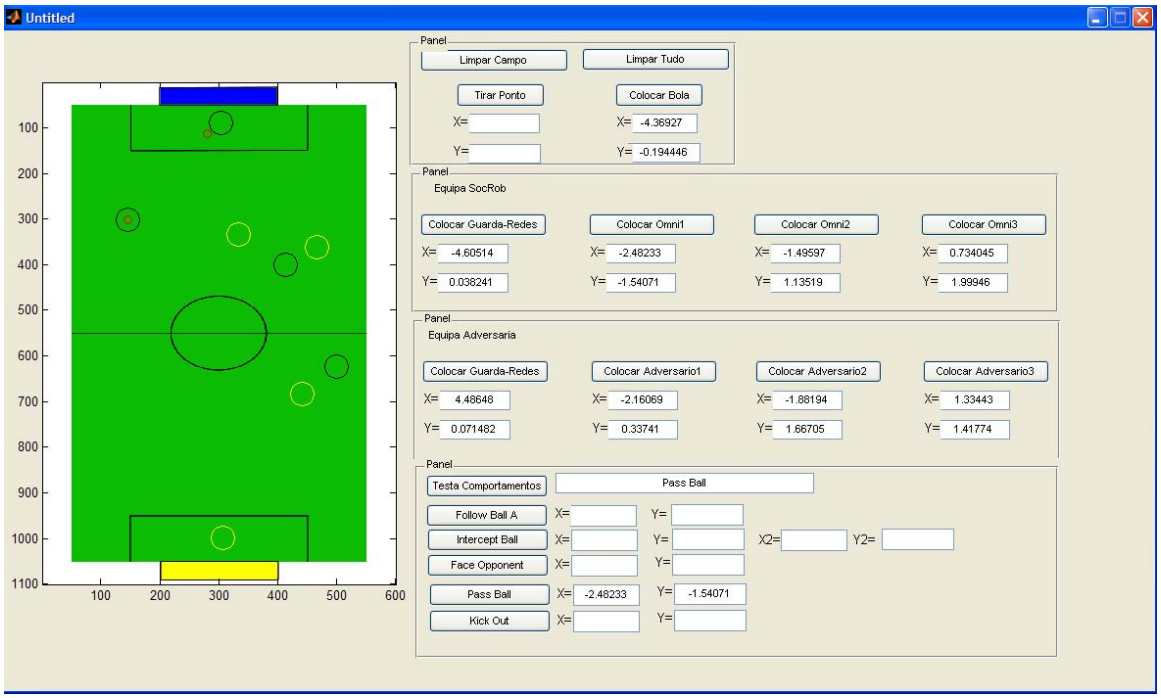
(a)

Fig D.3.
(a) Situação de Intercept Ball



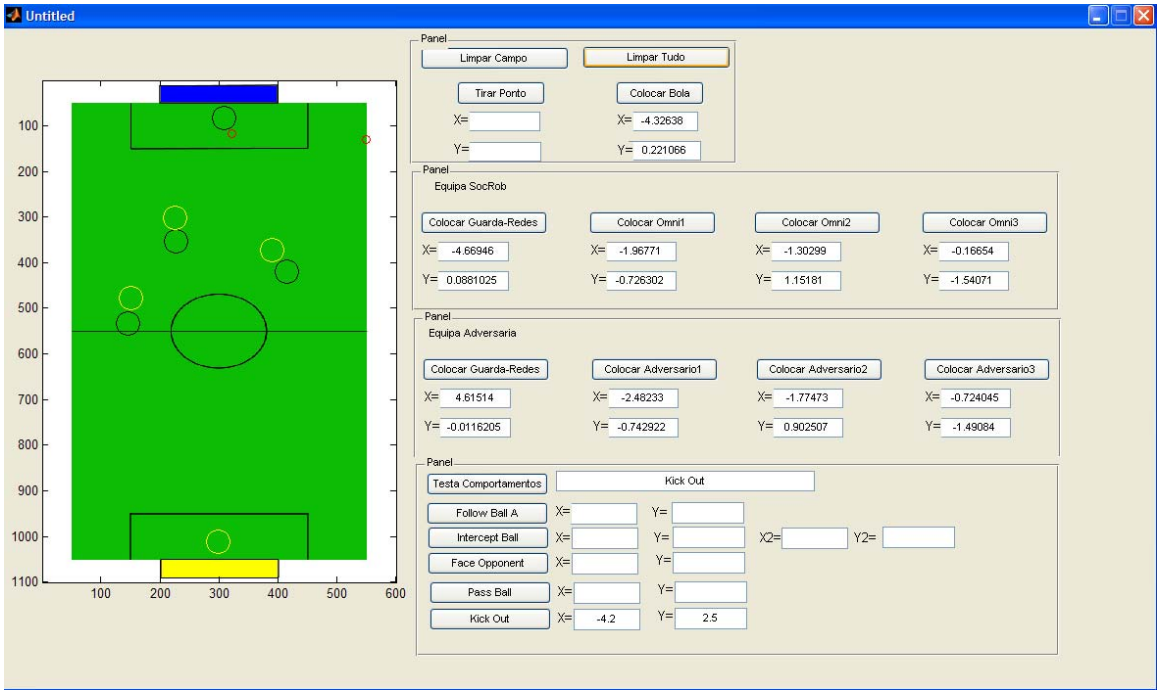
(a)

Fig D.4.
(a) Situação de Face Opponent



(a)

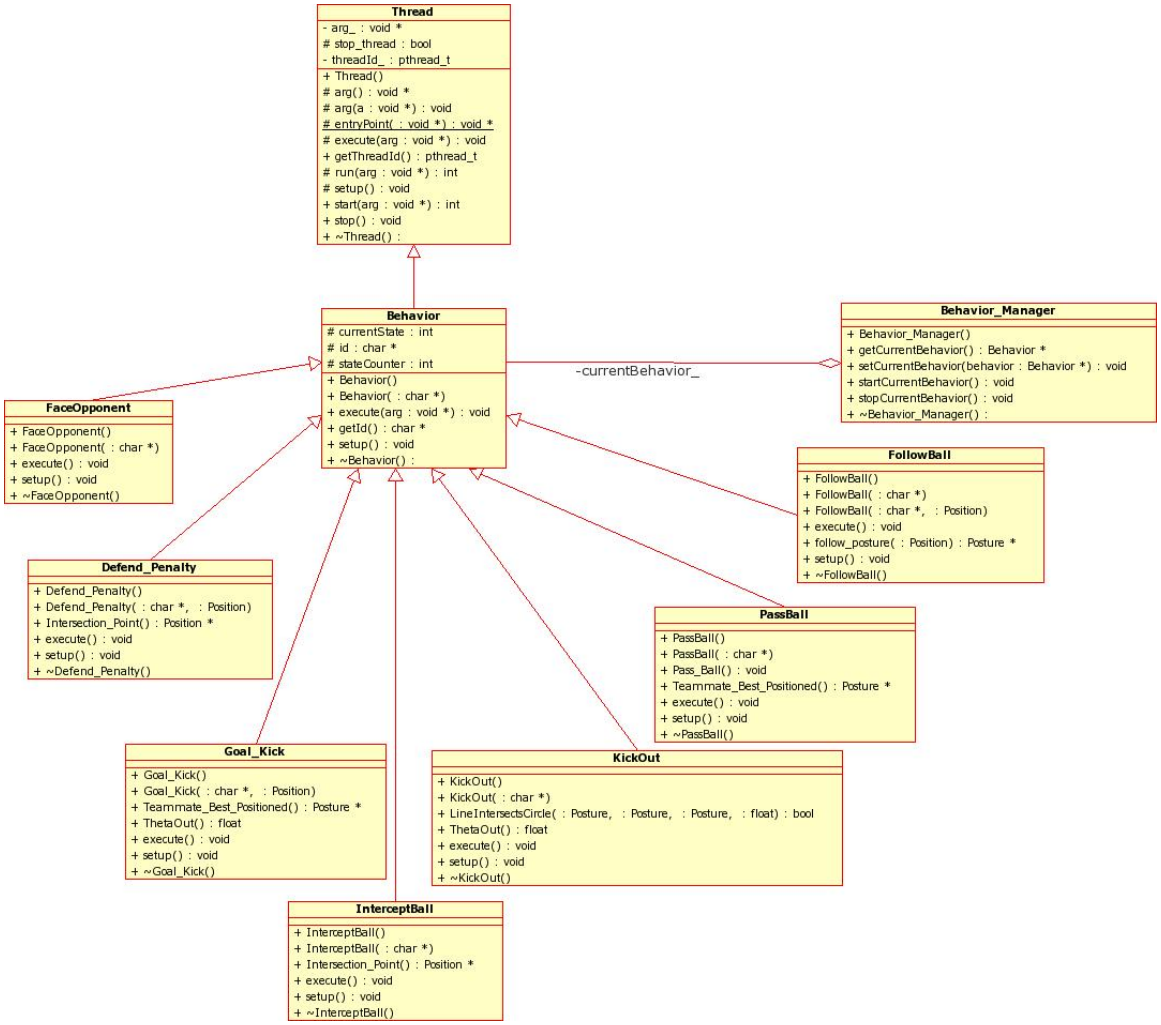
Fig D.5.
(a) Situação de Pass Ball



(a)

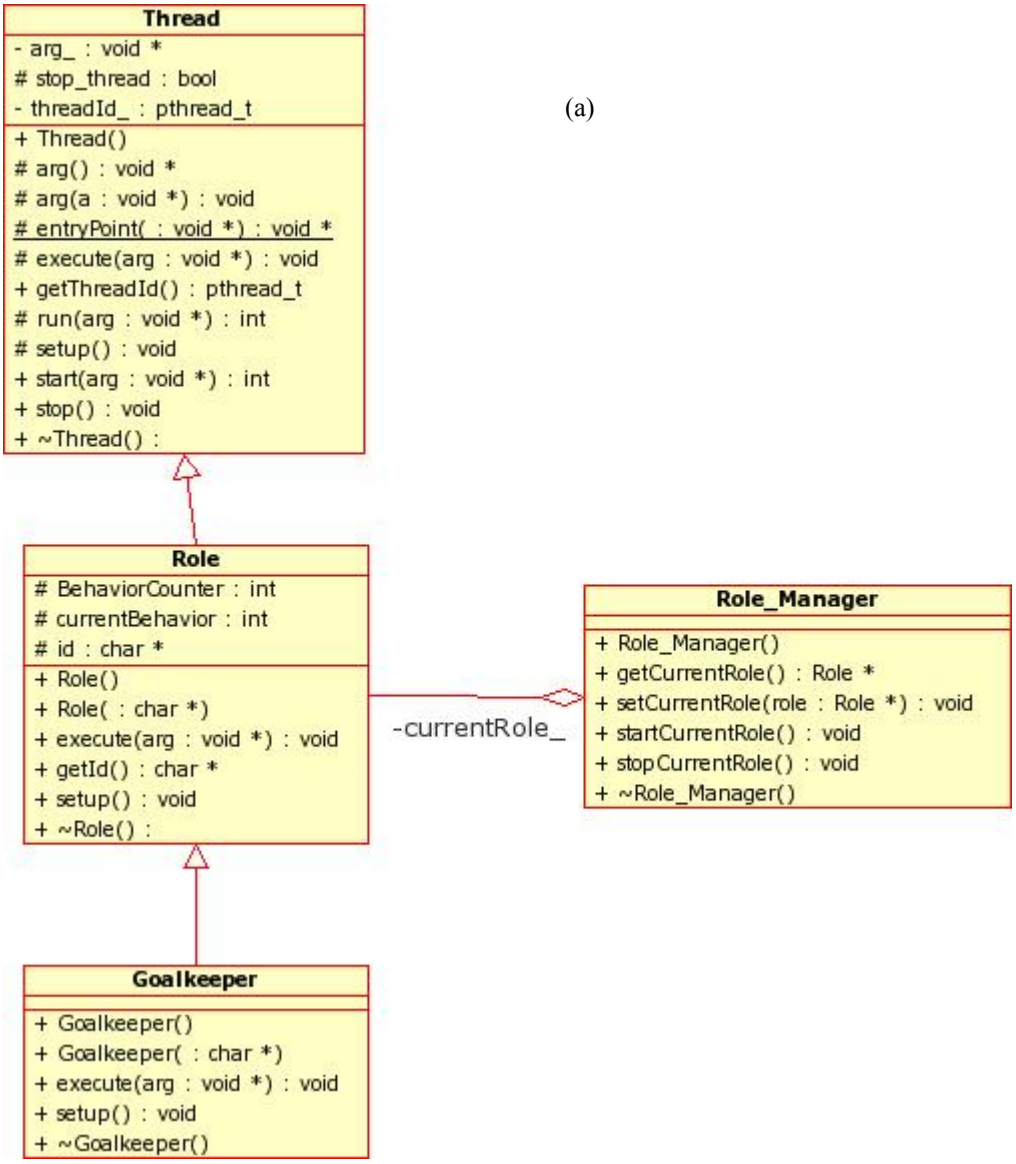
Fig D.5.
(a) Situação de Kick Out

Anexo E – UML



(a)

Fig E.1.
(a) UML referente aos comportamentos criados.



(a)

(a)

Fig E.2.
(a) UML referente ao papel do guarda-redes

Referências

[1] *Behavior-Based Robotics* – Ronald C. Arkin, 1998, The MIT Press

[2] *SocRob Goalkeeper* – Hans H. Lausen, Jakob Bro Nielsen, Michael Schioldan Nielsen, ISR/IST and Dept. of Control Engineering, IES, Aalborg University, Denmark.

[3] "SocRob Architecture" – Pedro Lima

[4] "SubVersion" – João Pavão

[5] *Logic Based Hybrid Decision System for a Multi-Robot Team* – Vasco Pires, Miguel Arroz, Luís Custódio, 2004, 8th Conference on Intelligent Autonomous Systems, Amsterdam, The Netherlands.

[6] *Logic Based Distribution Decision System for a Multi-Robot Team* – Miguel Arroz, Vasco Pires, Luís Custódio, 2003, Actas do Encontro Cientifico do Robotica 2003 - Festival Nacional de Robótica.

[7] *Near-optimal dynamic trajectory generation and control of an omnidireccional vehicle* – Tamás Kalmár-Nagy, Raffaello D'Andrea, Pritam Ganguly, 2003.

[8] *ISocRob Status November 2004* – Pedro Lima, Luís Custódio, Pedro Pinheiro, Hugo Costelha, Gonçalo Neto, Vasco Pires, Miguel Arroz, Bob Vecht, 2004.

[9] *Multi-sensor Navigation for Soccer Robots* – Carlos F. Marques, Pedro U. Lima, 2002, RoboCup 2001 Book, Springer-Verlag, Berlin, July 2002.

[10] *A Modified Potential Fields Method for Robot Navigation Applied to Dribbling in Robotic Soccer* – Bruno D. Damas, Pedro U. Lima, Luís M. Custódio, 2002, Proc. of the RoboCup 2002 Symposium.

[11] *WebotsTM: Professional Mobile Robot Simulation*, Michel, O. / Cyberbotics Ltd, 2004, pp. 39-42, International Journal of Advanced Robotic Systems, Volume 1 Number 1 (2004).