Bahia2D 2008: Descrição do Time

Marco A. C. Simões*, Bruno Vinicius Silva*, Alexsandro Cerqueira† e Luciana Silva†
*Núcleo de Arquitetura de Computadores e Sistemas Operacionais(ACSO)
Universidade do Estado da Bahia(UNEB)
Salvador-BA-Brasil
msimoes@uneb.br, patucao@gmail.com
†Centro Universitário da Bahia(FIB)
Salvador-BA-Brasil
{alexsandrobomfim, lucianapvs}@gmail.com

***Resumo*—**
Este arquivo descreve o trabalho tecnico-científico da equipe Bahia2D no período 2007/2008. O Bahia2D é o atual terceiro colocado na modalidade Futebol Simulado 2D da RoboCup Brasil. Neste período a equipe trabalhou num processo de refatoração do código do agente reativo utilizado em 2007, permitindo o início de uma nova estrutura baseada num agente multicamadas concorrente. Também foi desenvolvido um processo de atualização do nível reativo do agente para ficar compatível com a versão mais atual do Simulador da 2D. O trabalho com jogadores heterogêneos, introduzido no time em 2007, foi atualizado para as inovações do simulador na sua versão mais atual. Estes trabalhos e seus resultados serão descritos neste artigo.

I. Introdução

A equipe Bahia2D iniciou suas atividades em agosto de 2006 trabalhando com agentes reativos, visando melhorar a qualidade da execução de algumas habilidades como chute a gol, posicionamento e defesas (goleiro). Já no seu primeiro ano de atividade a equipe obteve a qualificação para a RoboCup 2007 em Atlanta. No *Brazilian RoboCup Open* 2007, o Bahia2D conquistou a terceira colocação dentre as nove equipes que disputaram a competição.

O trabalho do time continuou atuando com agentes reativos até o início do ano de 2008. Nesta fase, diversas rotinas baseadas em lógica fuzzy[16] e regras de produção[13] foram desenvolvidas aperfeiçoando habilidades como: passe, cruzamentos, chute a gol, posicionamento do goleiro, reposição de bola pelo goleiro dentre outras. Os trabalhos anteriores do Bahia2D estão descritos em [15], [14]. Apesar de não ter obtido a qualificação para a RoboCup 2008, a melhoria do Bahia2D 2008 foi perceptível durante torneio de qualificação (SSIL).

Desde o final de 2007, percebemos que o desenvolvimento desetruturado das primeiras etapas gerou um código complexo e difícil de manter. Iniciamos então um processo de refatoração do nosso código-fonte visando garantir sua continuidade. Este processo está descrito na seção II.

Outra demanda observada já em 2008 foi a necessidade de adequar o código às novas versões do simulador oficial da liga 2D. O código atual é baseado no UvA Base 2003 [8] que foi desenvolvido para as versões 7 e 8 do simulador. Desde então, nenhuma atualização foi feita e boa parte do código de mais baixo nível do Bahia2D ainda depende destas implementações, por vezes obsoletas. Então iniciou-se um projeto, que acaba de ser concluído, de atualização do código para a versão 12 do simulador. Na seção III estas atualizações estão descritas.

Juntamente com a compatibilidade dos agentes jogadores para a nova versão, foi também feita a atualização do agente técnico, especialmente no que diz respeito ao modelo de escalação dos tipos heterogêneos que sofreu drásticas mudanças na versão 12. A seção IV descreve esta melhoria no time.

Por fim, foi iniciado um projeto para construção de um agente cognitivo (não mais reativo) baseado numa arquitetura de agente multicamadas concorrente, conforme descrito na seção V. Vale enfatizar que esta arquitetura está ainda na sua fase inicial de implementação e apenas uma versão inicial da mesma será testada durante esta competição.

II. Bahia2D 2007 e a Refatoração de Código

Após a participação do Bahia2D no brasileiro de 2007, ficou evidente que o processo de desenvolvimento do nosso agente precisava ser repensado. O projeto desenvolvido até então por muitas mãos, muitas vezes passageiras, começava a obter um legado indesejado, de códigos difíceis de manter, o que já havia sido provado ser um fator que dificultava extremamente a evolução do código. A produção, que sempre se intensificava durante as competições, deixava linha após linha, pontas soltas, que apesar do ganho temporário em produtividade, se mostraram a médio prazo, como era em realidade de se esperar, ser péssimo para o código, deixando-o cada vez menos maleável (*soft*) criando uma estrutura engessada para prosseguir o desenvolvimento, pairando no ar até mesmo o risco de ter que encarar um processo mais radical, voltando a reconstruir o código a partir das suas origens. Além disto, o futuro revelava novas demandas para o agente, o que requisitava mudanças estruturais.

Foi neste momento que nossa equipe oportunamente teve seus primeiros contatos com padrões de projeto e principalmente refatoração de código. "Refatoração é uma técnica disciplinada para reestruturar um corpo de código existente,

Este trabalho é parcialmente financiado pelos programas IC/FAPESB, PICIN/UNEB e PIBIC/FIB.

alterando sua estrutura interna sem modificar seu comportamento externo"[6]. Refatorar significa produzir modificações no código, na maior parte das vezes muito pequenas, em alguns casos aparentemente inofensivas, mas que são capazes de produzir ao longo do tempo um código robusto, coeso, claro, flexível, melhorando seu desempenho e agregando características essenciais para permitir sua longevidade e garantir maior produtividade. Pode ser, deste modo, classificada dentro do processo de evolução do software, como uma modificação preventiva. Vale enfatizar, que a palavra modificação não se refere à semântica do código, que permanece inalterada. Este é um processo que sempre existiu e é natural. A novidade é a criação de um vocabulário comum para identificar os passos da refatoração e o desenvolvimento da habilidade de aplicá-los de forma criteriosa e sistemática. Surge oficialmente com a tese de Willian Opdyke [10] e é extensamente desenvolvido dentro da Extreme Programming [2]. Em [7] encontramos o necessário para começar. Nesta obra, o autor desenvolve um catálogo com 72 refatorações, explicando em detalhes como aplicá-las, seguindo os moldes de Desgin Patterns [5]. O processo de identificação dos problemas no código, é altamente dependente da intuição do programador e da sua sensibilidade para perceber que algum ponto do código "cheira mal"(*bad code smells*). As ferramentas que auxiliam a identificação de problemas, são em geral muito caras e ainda bastante ineficientes para identificar a maior parte dos "cheiros ruins", devido a dificuldade de estabelecer métricas objetivas para essas medidas como mostrado em [9]. Por outro lado, algumas ferramentas, como a IDE Eclipse oferecem módulos que auxiliam na aplicação de algumas refatorações, ainda que a atividade de identificar o problema e definir a refatoração ideal a ser aplicada seja completamente do programador. No futuro é provável que este cenário mude bastante. A refatoração pode ser seguida como uma prática diária, de modo que nunca permita a um código chegar ao estado em que seja necessário uma parada completa no desenvolvimento, e um esforço conjunto para retomar o controle. Alguns esforços de classificação tem sido feitos na tentativa de facilitar o entendimento das refatorações e dos problemas (bad code smells) que elas visam solucionar. No nosso processo, utilizamos uma classificação baseada na sugestão de [9] que tenta encontrar uma essência nos problemas, que permita reuni-los em grupos. Devido a maioria dos desenvolvedores de nossa equipe, no período da refatoração ainda serem iniciantes na tecnologia de objetos, dividimos o processo em etapas, baseadas nos grupos adaptados de [9](cujos nomes foram mantidos abaixo) para reduzir a complexidade e melhorar a eficiência em encontrar problemas. As etapas que identificamos após uma leitura inicial sobre o tema foram as seguintes:

- ETAPA 1 - *The Dispensables or Obsolete Code*
- ETAPA 2 - *The Bloaters or Fat Code*
- ETAPA 3 - *Object Orientation Abuser, The encapsulators and The couplers*
- ETAPA 4 - *The Change Preventers*
- ETAPA 5 - *Revision*

Começamos cada etapa estudando sobre o grupo de problemas que atacaríamos, treinando nossos "narizes"para encontrar os "cheiros ruins"no código. Em seguida, dividimos o trabalho para a fase de busca dos erros, realizando, quando necessário, este trabalho em duplas de modo a gerar discussão e aprofundamento sobre o processo. Terminada a busca, estudamos as refatorações associadas aos problemas encontrados, possibilitado pelo modo como Fowler estrutura o seu catálogo, e abríamos as discussões para determinar as alterações que faríamos no código. Nesta etapa, é fundamental a importância de desenvolver um conjunto de testes de unidade sólidos, que garantam que a semântica do código permaneça inalterado, após as reestruturações internas. Uma boa referência sobre testes pode ser encontrada em [1]. A própria ideologia da refatoração favorece o programador neste instante, já que as alterações propostas são individualmente pequenas, além de existir a descrição detalhada de cada passo que deve ser seguido para que uma refatoração seja feita com sucesso. Isso não reduz em nada a importância dos testes, que diminuem, por vezes drasticamente, o tempo gasto para encontrar um bug. Num ambiente como o de futebol de robôs, onde muitas variáveis atuam simultaneamente, um erro inserido no código não se reflete de modo aparente sobre o comportamento do agente, mas estes pequenos erros podem, assim como uma série de pequenas restruturações transformam o código para melhor, piorar drasticamente o comportamento do agente e serão, por estarem espalhadas e serem individualmente pequenos, muito difíceis de serem rastreados. Validada as alterações da etapa em questão, passavamos a seguinte e repetiamos o processo. Um ganho indireto essencial do trabalho desenvolvido foi a ampliação do conhecimento de todos os componentes da equipe sobre o código e funcionamento do agente, principalmente por trabalharmos sobre um código pré-existente (UvABase 2003), onde inclusive funcionalidades não utilizadas puderam ser descobertas. Tentamos registrar esse conhecimento, seguindo a filosofia de passar para o código as compreensões que se obtem do mesmo, e que "qualquer tolo consegue escrever código que um computador entenda, bons programadores escrevem códigos que humanos consigam entender"[7]. A tarefa serviu também para detectar alguns bugs importantes do agente UvATrilearn e o surgimento de idéias sobre melhorias, ambas as coisas para serem desenvolvidas após a atividade de refatoração. Na etapa 5, de revisão, reestruturamos parte da documentação do código, estabelecemos um conjunto de convenções de codificação para o software, e geramos a versão final do time resultante da atividade. Esta versão, reduzida em cerca de 16% em linhas de código, já que muitos dos nossos problemas encontravam-se na etapa 1 do processo (código obsoleto, que em geral se soluciona por meio da exclusão), se mostrou na prática muito mais fácil de ser utilizada e mantida. O código também ganhou muito em modularização, encapsulando funcionalidades do agente, a exemplo do cruzamento, em um único trecho de código ou conjunto de funções, o que isola na medida do possível o resto do código dos efeitos de mudanças nestes trechos, diminuindo o acoplamento. Desenvolver um

agente inteligente jogador de futebol robusto, só é possível acompanhado de boas práticas de programação e engenharia de software.

III. Atualização para o simulador 12.x

Desde o German open 2008, as competições oficiais utilizam alguma versão do servidor 12 para a 2D. No evento citado, participamos da competição, porém apenas as modificações mínimas necessárias para que o agente funcionasse no servidor foram feitas. Para os times baseados no UvA, é necessário (ou útil) fazer as seguintes alterações:

- No arquivo mainCoach.cpp e main.cpp, alterar o valor da variável dVersion para 12.0;
- No arquivo ServerSettings.cpp, alterar o valor de iPlayerTypes para 18;
- No arquivo SoccerTypes.h, alterar o valor de `#define` MAX_HETERO_PLAYERS para 18;

A principal alteração refere-se aos jogadores heterogêneos e a sua utilização, maiores informações podem ser encontradas na sessão IV. As mudanças realizadas para a versão 12, nos fizeram voltar a atenção para todas as mudanças realizadas no servidor da 2D desde as versões sobre as quais o UvA foi desenvolvido, as versões 7 e 8. As mais relevantes são (os números indicam a versão a partir da qual a mundança descrita foi incorporada):

8.03 Fullstate - É possível em jogos não-oficiais ativar esta funcionalidade. Os agentes recebem informações sem ruído, além das informações visuais normais. Excelente para debug.

8.03 PointTo - Novo comando que permite aos jogadores apontar para uma determinada região do campo.

8.04 Tackle - Permite que os jogadores realizem um "carrinho". O UvABase possui uma implementação básica para o comando.

8.04 Hear e say - Tornou-se possível para os agentes se comunicarem. O UvABase utiliza algumas trocas de mensagens, provavelmente remanescente do time original para melhorar a eficácia em evitar impedimentos.

8.04 AttentionTo - Possibilita um jogador ouvir prioritariamente as mensagens de say de um jogador específico.

9.1.0 Keepaway - Modalidade de jogo keepaway, mais conhecido por aqui como bobinho. Divertido e útil para testes.

9.3.0 Penalty automático - Série de mudanças que tornaram praticamente toda a cobrança de penaltys automatizada e parametrizável.

12.0.0 Informação sobre colisão adicionada ao final da mensagem de sense_body. Comparando a colisão que o UvA detecta visualmente e a nova informação enviada pelo servidor, notamos que o UvA acerta em média 63% das vezes.

12.0.0 Novo modo síncrono.

12.0.0 Mudanças no funcionamento do tackle.

12.0.0 Modificado o modelo de ruído do chute.

12.0.0 Modificado o modelo de ruído para objetos móveis.

Algumas regras do jogo também foram alteradas à proporção que as novas versões do simulador foram sendo publicadas. É fundamental atualizar o agente para atender às novas regras, evitando cometer penalidades desnecessárias durante a partida. As principais mudanças nas regras são:

8.04 Um jogador não pode mais chutar a bola para si mesmo em um free kick. Erro cometido comumente pelo UvABase.

8.04 Recuo de bola para o goleiro passaram a ser punidos, quando este executa um catch.

9.2.1 Impedimentos deixaram de ocorrer em laterais, tiros-de-meta e escanteios.

9.3.4 Proibida substituições durante os penaltys.

11.0.3 Proibido que jogadores chutem a bola, quando for tiro-de-meta para o adversário.

Além de diversas modificações relacionadas aos valores dos parâmetros, jogadores heterôgeneos ou diretamente com o técnico que serão detalhadas na seção IV. Atualmente estamos terminando de fazer uma atualização mais robusta do nosso agente para a versão 12 do servidor que será utilizada no Latino Americano.

IV. Jogadores Heterogêneos e Técnico compatível com versão 12.x

Com a atualização para o simulador 12.x, as regras relacionadas com a escolha dos tipos heterogêneos mudaram. Nos simuladores anteriores o número de tipos disponíveis era de 7 portanto dois ou mais jogadores teriam o mesmo tipo heterogêneo. No simulador atual este número aumentou para 18 e agora é obrigatório que cada jogador tenha seu próprio tipo, os jogadores que não são definidos pelo técnico ou que tem um tipo já escolhido anteriormente para outro jogador irão receber um tipo heterogêneo arbitrário indicado pelo simulador. O agente técnico do Bahia2D realizava as escolhas para a escalação do time transformando os valores dos parâmetros de cada tipo em um número na faixa de 0 a 100 e multiplicando por um peso específico, desta forma os tipos com maior prioridade, ou seja, com os maiores valores, seriam os escolhidos. Este processo ainda se baseava na regra dos servidores anteriores ao 12.x e por isso dois ou mais jogadores compartilhavam o mesmo tipo. O processo de escalação precisaria ser modificado para atender as novas regras do simulador mais atual.

Então o técnico compatível com a versão 12.x teria uma nova implementação para o processo de escalação. Para isto foi utilizado um sistema de produção (sistema baseado em regras de produção) composto por uma base de fatos, uma base de regras (arquivo editado pelo usuário) e um motor de inferência. O motor de inferência é baseado no motor da Expert-Coop++[3], uma biblioteca de classes, em linguagem de programação C++, que oferece um modelo de arquitetura para o agente cognitivo, chamado Agente Autônomo Concorrente [4]. O motor de inferência controlará as atividades do sistema em ciclos correspondentes às fases de: unificação (onde é realizada a seleção de regras que satisfazem à situação atual), resolução de conflitos (escolhe e ordena as regras que serão utilizadas dentre as selecionadas anteriormente) e ação (execução da regra). Ele utilizará as bases de fatos e regras no processo de inferência de onde

surgirão novos fatos para serem adicionados na sua base. A figura 1 ilustra esta estrutura.

Fig. 1. Estrutura do Motor de Inferência do ExpertCoop++

O tipo heterogêneo será classificado de acordo com as expectativas descritas no arquivo de regras. Quando todos os tipos estiverem classificados será feito uma nova inferência desta vez para decidir qual tipo servirá para um determinado jogador. O processo realizado para definir as expectativas descritas no arquivo de regras é dividido em três partes: primeiro foi necessário estabelecer um valor mínimo desejável para cada parâmetro dos tipos heterogêneos e quais características mais se encaixavam em determinados jogadores, depois foi realizada a procura de uma semente de randomização que gerasse o maior número de tipos cujos parâmetros fossem maiores ou iguais aos valores mínimos estabelecidos. Na terceira e última fase foram feitos os testes com a semente escolhida para definir o quanto estes valores influenciavam no desempenho do jogador durante o jogo. Mais de 200 testes foram executados contra vários times de futebol simulado 2D. Alguns resultados foram animadores, foi constatado um aumento no número de chutes a gol e interceptações de bola, os atacantes conseguiam acelerar mais rápido que os adversários e consequentemente alcançavam a bola antes deles. Outros resultados não foram tão bons assim, o número de passes errados dos jogadores da defesa aumentou, o que mostrou que uma das características priorizada para os atacantes, a precisão, também era importante para os defensores. Desta forma a proposta inicial foi modificada para atender às novas expectativas, novos testes foram realizados para confirmar a proposta atual e como resultado foi constatado que o número de passes errados decresceu. Baseando-se nestes resultados foram criadas as regras que classificam os tipos heterogêneos e que escolhem qual jogador terá o tipo em questão.

V. O PROJETO DO AGENTE CONCORRENTE MULTICAMADAS

O agente concorrente com multicamadas de aprendizado [4] surgiu da necessidade de tornar o time de futebol de robôs simulados do Bahia2D mais inteligente e mais eficiente nas suas tomadas de decisões. Este agente é composto por uma base de conhecimento e um motor de inferência que serão responsáveis por tratar as percepções de ambiente do agente em mais alto nível. Este agente foi desenvolvido baseando-se em publicações feitas pelos times FURG [12] da Universidade Federal do Rio Grande e pelo MecaTeam2008[11] da Universidade Federal da Bahia.

Existem hoje no time Bahia2D três bases de conhecimento e raciocínio do agente. O nível cognitivo, o nível instintivo e o nível reativo. O nível cognitivo é responsável por determinar as metas locais e globais dos agentes. Este é o nível de maior grau de raciocínio do agente. Suas decisões não implicarão de imediato no nível reativo, ou seja, a meta enviada por este nível será tratada em mais baixo nível pelo nível instintivo e este nível sim, trará impacto sobre o comportamento esperado do agente para alcançar a meta.

O nível instintivo é responsável pelo reconhecimento dos diferentes estados do ambiente e por notificações de controle de jogo, ou seja, é responsável por perceber o ambiente e tratá-lo a partir das informações recebidas. Este também é responsável por atualizar as informações a serem passadas ao nível cognitivo na forma de informações simbólicas e que irão compor a base das decisões futuras de novas metas locais da camada superior, além de tratar a meta local enviada pelo nível cognitivo, planejar ações abstratas para esta meta e a partir desta, escolher o comportamento selecionado a ser executado pelo agente no jogo.

O comportamento selecionado será tratado pelo nível mais baixo do agente, o nível reativo que transformará este comportamento em uma ação. A esquemática do agente com multicamadas é baseada na arquitetura do time da FURG, conforme ilustrado na figura 2.

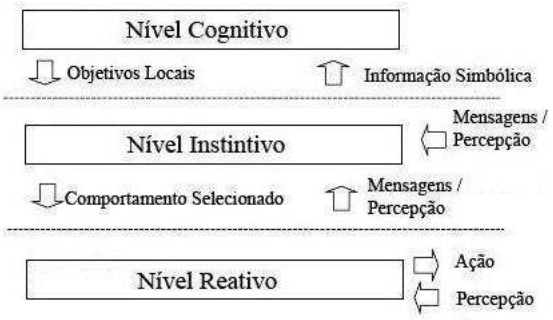

Fig. 2. Esquema do Agente Multicamadas Concorrente

É observado que o processamento é feito de forma bidirecional. O nível instintivo é o nível intermediário de raciocínio do agente e não é tão dependente do nível cognitivo, ou seja, a partir de percepções recebidas do ambiente, podem ser planejadas novas ações abstratas, e a partir daí enviar o comportamento selecionado ao nível reativo do agente. Para esta situação pode ser admitido também o envio de uma informação simbólica ao nível cognitivo a partir destas percepções. O nível reativo assim como o nível instintivo pode receber percepções do ambiente, retornar ações a serem executadas ou ainda enviar esta informação para o nível instintivo tratá-la em mais alto nível. A comunicação entre os níveis é feita pelo uso de threads. A comunicação entre os threads é feita através de mailbox, ou memória compartilhada

de buffer. Nesta arquitetura o mailbox foi aplicado para garantir endereçamento indireto, ou seja, todas as mensagens podem ser endereçadas por um thread e retiradas por outro garantindo assim ao máximo o assincronismo entre os threads. Este modelo está sendo implementado e o time que participará do *Latin American RoboCup Open* 2008 deve apresentar apenas uma implementação inicial do mesmo.

VI. Considerações Finais

Atualmente o Bahia2D está numa fase de validação da versão mais atual do código, completamente compatível com a versão 12.x do simulador oficial. Será validado em uma competição oficial pela primeira vez este código atualizado, além do uso mais eficiente dos tipos heterogêneos já compatível com o novo modelo do simulador 12.x. Algumas rotinas já testadas durante a qualificação (SSIL) para a RoboCup 2008, serão aperfeiçoadas e testadas agora com um nível reativo básico mais consistente com o simulador.

Há uma boa expectativa de melhorias como consequência direta do uso eficiente dos jogadores heterogêneos. Os resultados e logs desta competição servirão como fase de coleta inicial de dados para sedimentar os requisitos já definidos para a nova arquitetura multicamadas concorrente. Como principal trabalho futuro, está a implementação de agentes cognitivos, além de um agente técnico compatível com o ambiente dinâmico do jogo, capaz de intervir durante a partida de forma a influenciar positivamente no comportamento do time.

References

[1] "junit home page," http://www.junit.org.

[2] K. Beck, *eXtreme Programming eXplained: Embrace Change*. Addison-Wesley, 2000.

[3] B. G. G. E. M. N. e. S. L. R. Costa, A. L. d., "Expertcoop++:ambiente para desenvolvimento de sistemas multiagente," in *IV ENIA Encontro Nacional de Inteligência Artificial. XXIII Congresso da Sociedade Brasileira de Computação*. SBC, 2003.

[4] A. L. da Costa and G. Bittencourt, *IJCAI'99, Third International Workshop in RoboCup*, ser. Lecture Notes in Artificial Intelligence, 1999, ch. From a Concurrent Architecture to a Concurrent Autonomous Agents Architecture, pp. 85–90.

[5] R. J. e. J. V. Erich Gamma, Richard Helm, *Gangue dos Quatro - Padrões de projeto, Soluções reutilizáveis de software orientado a objetos*. Bookman, 2000.

[6] M. Fowler, "Refactoring home page," http://www.refactoring.com.

[7] ——, *Refatoração - Aperfeiçoando o projeto de código existente*. Bookman, 2004.

[8] J. Kok, N. Vlassis, and F. Groen, "Uva trilearn 2003 team description," in *CD RoboCup 2003*. Padua, Italy: Springer-Verlag, Ju;y 2003.

[9] M. Mäntylä, "Bad smells in software - a taxonomy and an empirical study," Master's thesis, Helsinky University of Technology, 2003.

[10] W. F. Opdyke, "Refactoring object-oriented frameworks," Ph.D. dissertation, University of Illinois at Urbana Campaign, 1992.

[11] C. PINELLI, O. V. SANTANA JR, and A. C. L. COSTA, "Mecateam 2008: Programação concorrente para futebol de robôs simulado," in *VIII ERBASE (Escola Regional de Computação Bahia-Alagoas-Sergipe)*, 2008.

[12] V. F. PIO, N. WIHIBI, N. M. E. GONÇALVES, and C. S. S. BOTELHO, "Arquitetura furgbol aplicada ao soccerserver," in *EnRI III Encontro de Robótica*. SBC, 2006.

[13] S. Russel and P. Norvig, *Artificial Intelligence*, 2nd ed. Prentice-Hall, 2002.

[14] M. A. C. Simões, H. G. Aragão, V. Souza, S. Viegas, H. da L. Silva, and J. Meyer, "Bahia2d team description," in *CD Robocup 2007*. Atlanta, EUA: Springer-Verlag, July 2007.

[15] M. A. C. Simões, H. Silva, J. Meyer, J. Oliveira, D. Cruz, L. Pessoa, H. Aragão, and R. Lima, "Bahia 2d: Descrição do time," in *Anais do Simpósio Brasileiro de Automação Inteligente (SBAI)*. Florianópolis, SC: Soceidade Brasileira de Automática (SBA), 2007.

[16] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.