Universidade do Estado da Bahia

Departamento de Ciências Exatas e da Terra

Colegiado de Sistemas de Informação

Automatização de Testes e Análises de Movimentos de Robôs Bípedes em Simulação 3D

Edivã Gonçalves Teodozio

Salvador-BA

2015

Edivã Gonçalves Teodozio

Automatização de Testes e Análises de Movimentos de Robôs Bípedes, em Simulação 3D

Monografia submetida ao Colegiado de Sistemas de Informação da Universidade do Estado da Bahia, como parte dos requisitos necessários para obtenção do grau de Bacharel, em Sistemas de Informação.

Área de Concentração: Sistemas de Informação
Linhas de Pesquisas: Robôs Bípedes e Agentes inteligêntes

Marcos Simões
(Orientador)

Salvador-BA
2015

Edivã Gonçalves Teodozio

Automatização de Testes e Análises de Movimentos de Robôs Bípedes, em Simulação 3D

Monografia submetida ao Colegiado de Sistemas de Informação da Universidade do Estado da Bahia como parte dos requisitos necessários para obtenção do grau de Bacharel, em Sistemas de Informação.

Aprovada em: 13/11/2015.

Prof. MSc. Marco A. C. Simões (orientador)
Universidade do Estado da Bahia (UNEB)

Prof. Ph.D Diego Gervasio Frías Suárez (Convidado)
Universidade do Estado da Bahia

Prof. Ph.D Josemar Rodrigues de Souza Professor (Convidado)
Universidade do Estado da Bahia

Resumo

O crescente interesse pelo desenvolvimento de robôs que ajudam o ser humano em tarefas simples do cotidiano até outras mais complexas, tem levado pesquisadores e empresas a investirem em pesquisas e desenvolvimento de robôs, cada vez mais avançados. Esses robôs são programados para que possam executar trabalhos rápidos, seguros, eficiente e eficazes, sem a presença do homem. Diante deste cenário surgem vários problemas que tornam tais pesquisas caras e longas. Com o propósito de tornar menos custosas as pesquisas em robótica, foram desenvolvidos sistemas de simulação que se aproximam dos robôs reais. Dessa forma, é possível investir nessas pesquisas, sem muitos custos iniciais com peças físicas, todavia com maior conhecimento humano. No entanto, os problemas de desenvolvimento tornam-se maiores, necessitando que sejam realizados vários testes, além de avaliar o desempenho desses sistemas. O presente trabalho propõe um módulo e automatização e avaliação do desempenho de robôs bípedes em ambientes simulados 3D. A validação deste projeto foi realizada utilizando coletas de informações do ambiente de simulação Simspark utilizado em conjunto com o RoboCup Soccer Server utilizados na liga de Simulação 3D da RoboCup.

Palavras-chave: Automatização, Testes, Robôs Bípedes, Avaliação, Movimentos, Simulação.

Abstract

The growing interest in the development of robots to help humans in simple everyday tasks, and also more complex ones, has led researchers and companies to focus in research and development of increasingly advanced robots. These robots are programmed so that they can perform work fast, safe, efficient and effective without the presence of man. In this scenario many problems arise leading to costly and extensive research. In order to make it less costly, many simulation systems have been developed that are close to real robots. Thus, it is possible to focus in such research without a lot of upfront costs with physical parts, but only with greater human knowledge. However, development problems become larger demanding several tests and to system evaluation. This work proposes an automation and evaluation of bipedal robot performance module for simulated 3D environments. The validation of this project was performed using data collected from Simspark simulation environment in conjunction with the RoboCup Soccer Server used in the RoboCup 3D Soccer Simulation league.

Keywords: Automation, Tests, Biped robots, evaluation, movements, simulation.

Agradecimentos

Primeiramente, agradeço a Deus, por ter me dado força, saúde e disciplina para que eu pudesse concluir mais esta etapa da minha vida.

Agradeço a Edelvilto dos Santos Teodozio, meu pai, há 10 anos falecido, e que demostrou amor, carinho, atenção sempre me apoiando, enquanto estava conosco. Sou agradecido a Aidê Gonçalves Teodozio, minha mãe, que mesmo com dificuldades de saúde, ficava às noites acordada, enquanto eu fazia estudos e trabalhos da faculdade. Agradeço a sua paciência, nas horas em que eu, com obrigações, não conseguia cumpri-las por alguma eventualidade, aconselhando-me para não desistir. Agradeço aos meus pais que proporcionaram o meu acesso ao ensino superior . Muito obrigado por terem acreditado e confiado em mim. Agradeço também a meus irmãos por estarem presentes em horas difíceis. Sou grato a todos os calouros e os colegas de turma que me orientaram em dúvidas complexas que surgiam durante cada semestre. Não posso esquecer do grupo de pesquisa **ACSO**, em especial ao colega Sérgio e meu amigo Nemuel Leal que contribuiu fortemente na reta final desta monografia.

Agradeço aos meus amigos conhecidos de escola, teatro e curso de violão, que acompanharam-me pelas redes sociais.

Meu sinceros votos também ao meu professor, Antônio Feitoza, que acompanhou a construção deste trabalho me orientando.

Agradeço ao meu orientador, professor Marcos Simões, por ter acreditado e confiado em mim. Muito obrigado pela sua atenção, disponibilidade, ensinamentos, orientações compartilhadas.

Meus agradecimentos, a todo egrégio corpo docente deste magnífico estabelecimento de ensino, a Universidade do Estado da Bahia (**UNEB**).

"Quem nunca errou nunca experimentou nada novo."

Albert Einstein

Sumário

1 Introdução 13

2 Robot World Cup (Robocup) 17
2.1 Iniciativa Robocup 17
2.1.1 RoboCup Soccer 19
2.1.2 Liga de Futebol Simulado 3D 20

3 Simulação 3D e Protocolos de Comunicação 23
3.1 Ambiente de simulação 23
3.2 SimSpark 24
3.3 Soccer Simulation League 26
3.4 Monitor de Visualização 26
3.5 Protocolo de Comunicação 28
3.5.1 Comunicação entre Servidor e Agentes 29
3.5.2 Comunicação entre Servidor e Monitor 30
3.5.3 Conteúdo da Cena Gráfica 33

4 Ferramenta *Trainer* 37
4.1 Modificação da Ferramenta *Trainer* 38
4.2 Metodologia 39
4.3 Arquitetura da ferramenta *Trainer* 41
4.3.1 Camada de Comunicação 42
4.3.2 Camada de Processamento 42
4.3.3 Camada de Análise Estatísticas 43

5 Testes, Experimentos e resultados 46
5.1 Metodologia de Testes 46
5.2 Métricas de Avaliação 47
5.3 Resultados 48

6 Conclusões e Trabalhos Futuros 53

Lista de Figuras

2.1 Campeonato da RoboCup 2013 17
2.2 Simulação 3D com esferas 20
2.3 Simulação 3D com humanóides 21
2.4 Dimensões Campo simulado 22

3.1 Arquitetura do SimSpark 25
3.2 Graus de liberdade das juntas do NAO 27
3.3 Monitor RoboViz e RCSSMonitor 28

4.1 Metodologia 39
4.2 Arquitetura da Ferramenta *Trainer* 41

5.1 BahiaRT x MagmaOffenburg 50
5.2 BahiaRT x Utaustinvillas 50
5.3 BahiaRT x MagmaOffenburg 51
5.4 BahiaRT x RoboCanes 51
5.5 BahiaRT x Futengine 51

Lista de Tabelas

2.1 Descrição do ambiente de futebol 19

3.1 Descrição das mensagens do *EnvironmentInformation* 31
3.2 Mensagens do Monitor para o Servidor 32
3.3 Abreviações de termos do Grafo da cena 34

4.1 Resumo da Partida 45
4.2 Dados das coordenadas do Agente 45

5.1 Programas e configuração de máquinas no ambiente real. 46
5.2 Tabela de times utilizado nos experimentos 47
5.3 Validação dos dados 48
5.4 Resultados das primeiras partidas. 49

Lista de Algoritmos

4.1 Algoritmo para tratamento da mensagem 43
4.2 Algoritmo para tratamento da mensagem 44
4.3 Algoritmo para tratamento da mensagem - análise sintática 45

Lista de Abreviaturas e Siglas

ACSO	*Núcleo de Arquiteturas de Computadores e Sistemas Operacionais*
BRT	*Bahia Robotics Team*
HOAP	*Humanoid for Open Architecture Platform*
ZMP	*Zero Moment Point*
SPADES	*System for Parallel Agent Discret Agente Simulation*
OpenGL	*Open Graphics Library*
ODE	*Open Dynamics Engine*

Capítulo 1

Introdução

Nos últimos anos, várias empresas e pesquisadores vêm desenvolvendo máquinas e equipamentos que visam substituir o homem em atividades de risco como desarmamento de bombas, exploração de ambientes hostis, e a condução de veículos semiautomáticos. Tais pesquisas são parte de um ramo da ciência chamado Robótica. Esta engloba tanto sistemas mecânicos, como elétricos e eletrônicos. Devido a esses fatores,a robótica torna-se um estudo multidisciplinar.

Os robôs com tais características detêm a necessidade de locomoção eficaz, em ambientes desregulares sem pavimentação. Conforme Silva e Machado (2001) a superfície terrestre tem mais de 50% do território inacessível a robôs tradicionais (com rodas ou largartas) tornando-se difícil, ou até impossível, às máquinas com rodas realazarem tarefas complexas humanas. Assim sendo, surge a necessidade do desenvolvimento de pesquisas, com robôs bípedes que trazem numerosas vantagens. Robô Bípede advém com a considerável mobilidade, versatilidade a ambientes não planos, tendo então a capacidade de superar obstáculos, garantindo relativa força, agilidade e estabilidade podendo ser um considerável campo de relativa complexidade científica (TORRES, 2006).

Neste contexto, os robôs bípedes também têm atraído a atenção da grande parcela de pesquisadores, devido ao grande oportunidade de trabalhos em diversas sub-áreas de Inteligência Articial, a saber, Sistemas Baseados em Conhecimentos, Sistemas Multiagentes, Redes Neurais Artificiais, Algorítimos Genéticos, Computação evolutivo entre outros e a possibilidade de uma melhor representação do mundo real. Desse modo, surgem grupos de

estudos e equipes de competição utilizando esses conhecimentos em robôs bípedes que vão além da simples integração da Inteligência Artificial (KRAETZSCHMAR et al., 1999).

Atualmente os robôs bípedes estão sendo desenvolvidos para atuarem em diversos campos: nas industrias, no ambiente doméstico, no auxílio a pessoas com necessidades especiais e no entretenimento. Vários protótipos têm sido produzidos pela indústria como é o caso do Asimo e o SDR-3, robôs humanoides da Honda e da Sony respectivamente. Esses robôs possuem requisitos mínimos, imprescindíveis para adaptação ao meio ambiente antrópico, dentre eles: manobrar entre obstáculos, subir e descer escadas e caminhar em terrenos irregulares (HIROSE; OGAWA, 2007).

Nas últimas duas décadas, os robôs bípedes estão inseridos no mundo do futebol, no intuito de aprimorar suas características em busca de uma aproximação dos movimentos humanos e aperfeiçoamento de suas propriedades cognitivas. A lúdica idealização de um um robô jogar futebol, alimentada por pesquisadores da área de robótica, possibilita a avaliação de várias teorias, algoritmos, arquiteturas e desempenhos, validando e pondo à prova uma grande variedade de tecnologias. Desta forma, futebol de robôs está consolidado como um grande desafio padrão para a o desenvolvimento da robótica no século atua permitindo a aceleração do desenvolvimento da inteligência artificial aplicada à robótica para aplicação em domínios do mundo real.

Referente a pesquisa com robôs, não tradicionais, Heinen e Osório (2006) propuseram o sistema que realiza configuração do caminhar de robôs simulados, dotados de pernas. O caminhar é definido através da utilização de três abordagens diferentes, usando um autômato que define os ângulos de cada junta do robô, uso de *Locus Based Gait*, e o uso de uma meia elipse para a definição de movimentos dos membros durante o caminhar. O simulador foi capaz de realizar a configuração do caminhar de robôs com pernas de forma automática através de Algoritmos Genéticos. Os testes foram realizados em robôs simulados de quatro e seis patas. Foi realizado comparação de várias abordagens para indicar quais as mais apropriadas, para uma modelagem do caminhar, e quais modelos de robôs são mais eficientes.

O autor Goncalves (2011) propôs um controlador de locomoção de robôs bípedes,

produzindo uma locomoção robusta e estável. O centro de pressão, é calculado através de sensores de força colocados na sola do pé do robô. Esses sensores fornecem informações sobre a posição do centro de pressão do robô, em cada instante. Quando detectada a posição na fase da dinâmica do robô, um oscilador acoplado sincroniza padrões periódicos gerados pelo controlador. Assim, o controlador proposto é capaz de gerar, com sucesso, uma locomoção bípede estável, tanto para o robô **HOAP-2**, desenvolvido pela Fujitsu, quanto para o robô **NAO**. Ambos são humanoides autônomos, onde o primeiro é pequeno e capaz de descer escadas, escrever o próprio nome, dentre outras atividades e outro é programável desenvolvido pela empresa francesa Aldebaran Robotics. Como resultado, o autor encontrou medidas de estabilidade utilizada na avaliação do robô, através de cálculos testados com sucesso.

Todavia, as limitações do hardware dos robôs atuais geram grande impacto que limitam os avanços no desenvolvimento da **IA**. Nesse sentido, torna-se necessária a utilização de virtualização de robôs e seu ambiente. Isto permite realização de diversos experimentos, além da redução de custos e riscos, quando comparado a utilização de robôs reais. No entanto, a coleta de dados para análise torna-se mais complexa, devido a quantidade de informações do ambiente de simulação tridimensional e seus componentes gerados a cada ciclo de simulação.

Um exemplo de uma análise custosa, utilizada na simulação de futebol de robôs simulados 3D, é o procedimento de assistir à repetição de um jogo virtual para coleta de dados manualmente, através da observação de partidas com os logs, gerados durante a simulação, buscando visualmente lances importantes dos comportamentos desejados. Para um resultado sólido e uma análise estatística desses testes, sem uma ferramenta automatizada, esta abordagem manual dispende muito tempo tornando-se difícil a realização de vários testes e análise de desempenho de um robô ou de um time de robôs.

Neste contexto, o presente trabalho propõe o aperfeiçoamento de ferramentas de apoio ao uso da simulação tridimensional de robôs bípedes para análise de desempenho, viabilizando uma maior facilidade de automatização de testes e uso de métodos evolutivos de otimização e aprendizagem de máquina. Com isto, é possível diminuir o tempo gasto por pesquisadores que trabalham com robôs bípedes simulados em 3D nas etapas de treinamento e testes.

A proposta é possibilitar a análise do desempenho dos agentes e ambiente, através de relatórios gerados após cada partida, através da otimização da ferramenta *Trainer*, para gerar medidas de evolução dos bípedes, em ambiente de simulação 3D. Esta ferramenta foi desenvolvida dentro do grupo de pesquisa Núcleo de Arquitetura de Computadores e Sistemas Operacionais (**ACSO**) da Uneb e vem sendo utilizada juntamente com o simulador de futebol de robôs 3D da RoboCup - o RoboCup Soccer Server com SimSpark.

Para realização deste trabalho, a metodologia é divida em quatro etapas. Na primeira fase, é feito o estudo da ferramenta *Trainer* que deverá guardar todas as informações de posição dos bípedes, inicialmente na memoria. Na segunda fase, foram implementadas novas funcionalidades na ferramenta *Trainer* como pegar informações do todos os agentes e verificar se o existe algum robô caído. Na terceira fase, os dados coletados pelo *Trainer*, foram analisados detalhadamente para validar a correção do processo de coleta. Por fim, na quarta fase foram realizados as experimentações, testes, coletas de resultados e avaliações dos mesmos.

O restante deste trabalho está apresentado em 6 capítulos. No **capítulo 2** é apresentada a RoboCup e suas linhas de pesquisas. O **capítulo 3** descreve o ambiente de simulação 3D e seu protocolo de comunicação. O **capítulo 4** descreve todo o processo de construção da arquitetura e problemas encontrados. No **capítulo 5** são descritos os experimentos e análises de resultados do uso do *Trainer* aplicado aos testes com bípedes simulados. No **capítulo 6** são descritas as conclusões, bem como, propostas de trabalhos futuros.

Capítulo 2

Robot World Cup (Robocup)

2.1 Iniciativa Robocup

Este trabalho utilizou a plataforma da liga de futebol de robôs simulado 3D. Essa liga faz parte da RoboCup, que visa promover pesquisa e educação sobre robótica e Inteligência Artificial (**IA**) através do futebol de robôs e outros desafios científicos (KITANO et al., 1997). Devido a uma grande aderência da comunidade internacional a RoboCup em um mês após seu surgimento, ganhou proporções mundiais. Desde então, é promovido um evento anual em países diferentes. Um dos maiores eventos de robótica autônomo do mundo.

Figura 2.1: Campeonato da RoboCup 2013

Em setembro de 1993, foi anunciado ao público a iniciativa RoboCup com os regulamentos específicos. Assim, discussões e as questões técnicas sobre as organizações foram realizadas em diversas conferências e *workshops* como o Simpósio da Japanese Association for Artificial Intelligence - **JSAI** (ROBOCUP, 2015).

De quatro a oito de novembro de 1996, durante a conferência Internacional sobre inteligência Robótica e Sistemas (IROS-96), em Osaka, foi realizada a Pré-RoboCup. Na competição oito equipes competiram em uma liga de simulação e demostração de robô real. Essa foi a primeira competição usando jogos de futebol de robôs, para aprofundar as pesquisas na área de robótica. A Figura 2.1 demonstra o evento oficial da RoboCup em 2013, Holanda.

Os primeiros jogos oficiais e conferência RoboCup, foram realizados em 1997, com grande sucesso. Mais de 40 equipes participaram (robôs reais e simulação) e mais de 5.000 espectadores compareceram (ROBOCUP, 2015). Apesar da RoboCup originar-se da idéia lúdica de robôs jogarem futebol, com o amadurecimento da iniciativa, foram criadas diferentes categorias para o evento oficial:

- *RoboCup Soccer*: Robôs especializados em jogar futebol;
- *RoboCup@Home*: Robôs domésticos;
- *RoboCup Rescue*: Robôs de resgate;
- *RoboCup@Work*: Robôs industriais;
- *RoboCup Sponsored*: Fundado em 2013 trazendo desafio para aplicações de robótica na área de logística;
- *RoboCupJunior*: voltada para introduzir estudantes do ensino fundamental e médio no universo da robótica.

A principal categoria da RoboCup, é a RoboCup Soccer, onde está inserido esse trabalho.

2.1.1 RoboCup Soccer

A RoboCup pretende construir uma equipe de jogadores de futebol de robôs humanóides, em meados do século XXI, isto é, próximo a 2050, onde estes robôs serão totalmente autônomos e capazes de ganhar um jogo, em conformidade com as regras da FIFA, sobretudo contra a seleção vencedora da copa do Mundo (ROBOCUP, 2015).

O desafio gera, na comunidade, um impulso capaz de redefinir o estado da arte na área de pesquisa em IA, proporcionando importantes contribuições e metas para o estudo da robótica, no ambiente de futebol, que segundo Russel e Norvig (2003) este ambiente tem as propriedades, conforme apresentado na Tabela 2.1.

Tabela 2.1: Descrição do ambiente de futebol

Propriedade	**Futebol**
Ambiente	Dinâmico
Resultados das Ações	Estocástico
Mudança de Estado	Tempo real
Acesso a Informação	Incompleta
Controle	Distribuído

Para suprir as necessidades, na elaboração de robôs autônomos físicos, é essencial um ambiente simulador, no objetivo de realizar testes e estudos para a aproximação do ambiente real de alta complexidade. Para lidar com essa complexidade, a RoboCup decidiu criar desafios em seis ligas onde alguns delas são em ambientes simulados:

- Simulação 2D;
- Simulação 3D;
- *Small Size*;
- *Middle Size*;
- *Standard Platform*;
- Humanóide.

Cada uma destas ligas aborda diferentes aspectos, seja do design eletro-mecânico dos robôs, até questões como visão computacional, sistemas multiagentes, entre outros. Neste trabalho, nos concentraremos na liga de Simulação 3D.

2.1.2 Liga de Futebol Simulado 3D

Os primeiros campeonatos simulados foram em 2D, mas para aprimorar o estilo de visão criou-se o estilo 3D e adicionou novas funcionalidades. A primeira simulação em 3D era similar a liga de simulação 2D, como por exemplo, o chute, o jogador, o goleiro e a visão parcial do jogador em campo. Cada jogador era representado por esfera, conforme a Figura 2.2.

Figura 2.2: Simulação 3D com esferas

O sistema de simulação 3D foi implementado com mudanças na comunicação cliente-servidor em uma plataforma SPADES (System for Parallel Agent Discret Agente Simulation), responsável pela comunicação entre o cliente e o servidor (ROLLMANN, 2004). Além da plataforma SPADES, o sistema necessita de uma plataforma ODE(Open Dynamics Engine) que realiza os cálculos de interações físicas, entre os objetos e o mundo. Por fim, o OpenGL(Open Graphics Library) implementa as imagens 3D, gerando todo ambiente gráfico 3D.

O ambiente de simulação consiste de dois principais componentes: O núcleo de simulação e uma ferramenta de visualização, que permite a monitorização da simulação, em tempo de execução, ou reprodução de arquivos de log. O simulador funciona como um servidor ao qual a ferramenta de visualização e os agentes individuais, conectam-se como clientes.

A liga de simulação 3D utiliza esses principais componentes para montar o ambiente de jogo. Dessa forma, a liga possui características e desafios no objetivo de criar humanóides próximos da realidade. Cada jogo possui 22 humanóides (Figura 2.3), isto é, agentes formando dois times, onze de cada lado, seguindo o padrão de escalação de um futebol real.

Figura 2.3: Simulação 3D com humanóides
(SIMPARK, 2012)

O campo simulado possui uma dimensão de 30 por 20 metros, que corresponde a 35% do campo da FIFA. A área central tem um raio de 2 metros, cada baliza 2,1 por 0,6 metros e uma altura de 0,8 metros, a grande área tem 3,9 por 1,8 metros e por fim a bola tem 0,04 metros de raio e um peso de 26 gramas. Em cada canto existe uma bandeira diferente que permite o agente se localizar no campo. Essas possuem códigos os quais as distinguem (Figura 2.4).

Para que o jogo possa rodar de modo eficiente, faz-se necessário a utilização de três a quatro computadores, com boa capacidade de memória e processamento. Isso devido a quantidade de processos e *threads* de cada time, além do simulador e o monitor, carregando todo o ambiente de futebol.

Cada partida de simulação possui 300 ciclos que equivalem em média a cinco minutos. O

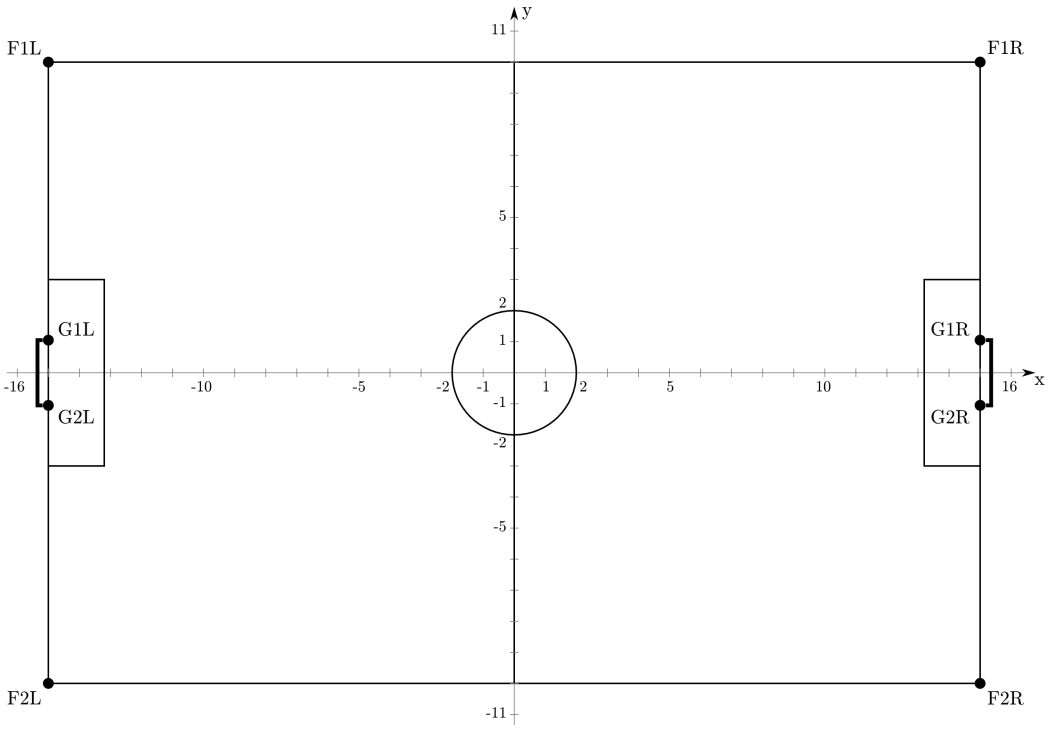

Figura 2.4: Dimensões Campo simulado
(SIMPARK, 2012)

vencedor de cada partida possui geralmente jogadores que apresentam bom desempenho durante o jogo, bom equilíbrio, bom chute, estratégias de jogo e passes, além de outras habilidades e ao final da liga, aquele time que acumular mais pontos corridos por cada vitória, será o vencedor da liga de simulação 3D.

Capítulo 3

Simulação 3D e Protocolos de Comunicação

Neste capítulo, detalhes da simulação 3D são abordados, para o entendimento do leitor sobre o ambiente de simulação SimSpark, o *Soccer Simulation League* e o protocolo de comunicação da troca de mensagens entre cliente-servidor, são conceitos fundamentais para compreensão geral do trabalho.

3.1 Ambiente de simulação

O sistema de simulação segue os conceitos de Russel e Norvig (2003), que define um agente inteligente como entidade autônoma, que percebe seu ambiente através de sensores e age no mesmo, através de atuadores maximizando o seu desempenho.

Exemplos de ambientes de simulação, são softwares que simulam modelos da vida real, sendo eles, acontecimentos proporcionados pela natureza (catástrofes naturais, cósmicas) ou um objeto criado pelo homem (aviões e carros) definindo agentes para perceber e agir em seu ambiente.

Simuladores no mundo da robótica tornam-se imprescindíveis para investigação científica e desenvolvimento nesta área proprocionando vantagens em relação ao uso de robôs reais pelo

seu baixo custo, diferentemente dos simples manipuladores e pequenos humanóides que, por sua vez, para desenvolvimento inicial de uma pesquisa, possuem altos custos (NICOLAS, 2005). Seguem algumas das vantagens de simuladores na robótica.

- Custo relativamente baixo, em comparação a uso de robô real.
- Facilidade de testar novos algoritmos.
- Possibilidade de adicionar e remover componentes diferentes.
- Todos o testes podem ser realizados sem danificar o robô físico.
- Possibilita a proximidade realística, com certa facilidade de transição, para o modelo real.

Baseado em diversas pesquisas, com simulação 3D que se aplicam no mundo real e o aperfeiçoamento da mesma, tornou-se possível a criação de modelos computacionais que representam simulações físicas, as quais proporcionam interações realísticas. O SimSpark é um dos simuladores mais utilizado para pesquisas em robótica, principalmente na RoboCup (SIMPARK, 2012).

3.2 SimSpark

O SimSpark é um sistema de simulação genérica, para sistemas multi-agente, em ambientes tridimensionais. O mesmo simula vários tipos de situações, desde pequenas experiências físicas, até um jogo de futebol. Atualmente, é usado como simulador oficial, na Iniciativa RoboCup 3D *Soccer Simulation League* (BOEDECKER; ASADA, 2008).

O simulador SimSpark é baseado no *Spark*, simulador genérico de física muiti-agente, para ambientes tridimensionais. A estrutura do SimSpark basea-se em três componentes principais apresentados na Figura 3.1.

i :**Motor de simulação física**

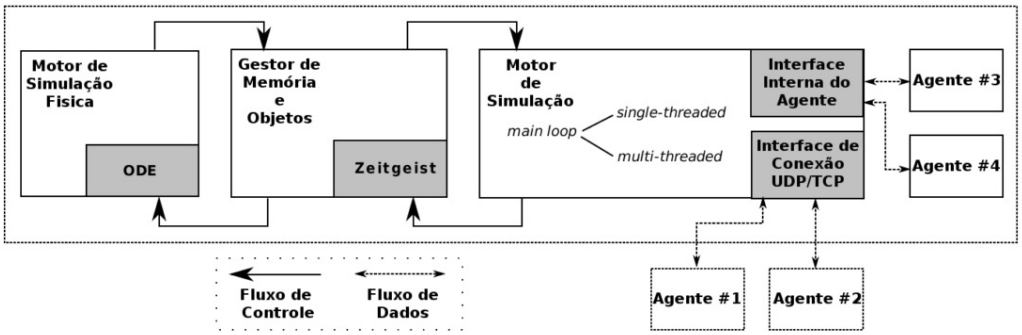

Figura 3.1: Arquitetura do SimSpark
(SIMOES et al., 2011)

ii : **Gestor de memória e objeto**

iii : **Motor de simulação**

O motor de símulação física é responsável por detectar colisões, atualização de posições e velocidade de objetos simulados, utilizando a **ODE** como sua principal biblioteca (SMITH et al., 2005). O simulador SimSpark, em sua implementação atual, usa *Spark* para simulações de dinâmica, fisicamente realista.

O gestor de memória e objetos ou mesmo *Zeitgeist* é o componente central do sistema, responsável por acesso a todos os serviços do simulador (KÖGLER, 2003). Existem três tipos de objetos gerenciados pelo *Zeitgeist*: Objetos representando o cenário atual, objetos encapsulados para funcionalidade central do simulador e fábricas para a criação de novos objetos, com base em mensagem de textos.

O *Zeitgeist* é divido principalmente em dois conceitos chaves: o primeiro é a implementação de variante do padrão de fábrica reflexiva, que permite a instanciação baseada em fábrica de objetos, em tempo de execução, durante o armazenamento de informações sobre a criação de novos objetos. O segundo conceito-chave é da organização das fábricas que criam objetos, em sistema de arquivos virtuais (SIMPARK, 2012).

O motor de simulação é responsável pelo controle do loop principal do SimSpark e da interação entre ele e os agentes. Esse componente utiliza o *middleware* **SPADES** (CHICK et al., 2003). O **SPADES**, sistema de distribuição de simulação, mantém o controle dos eventos entre os agentes e o simulador, sendo responsável pela comunicação entre o cliente

e servidor.

3.3 Soccer Simulation League

Na competição de futebol 3D é necessário o modulo *rcssserver3d* que controla toda competição, fazendo a simulação realística de um jogo de futebol. Esse módulo possui o SimSpark como mecanismo de simulação subjacente. Assim, a plataforma se esforça para reproduzir desafios de programação de software (SIMPARK, 2012).

Cada jogador é controlado por um agente de software autônomo e independente que recebe informações do servidor. Com isso, o agente deve decidir que ações executar para maximizar seus objetivos.

O servidor envia o estado do jogo para cada agente e como resposta os mesmos enviam comandos que controlam o movimento de seu corpo. Cada agente funciona como cérebro de um jogador, porém deve obedecer aos protocolos estabelecidos pelo *Soccer Server* não podendo se comunicar diretamente com outro agente, apenas através do servidor que impõe certas restrições sobre distância e quantidade de informação que é enviada (SIMPARK, 2012).

O modelo do robô utilizado, atualmente, nas competições simuladas 3D é o **NAO** (Figura 3.2). Este robô possui 22 motores posicionados nas juntas oferecendo igual quantidade de graus de liberdade.

A maioria das regras do jogo são básicas, julgadas de forma automática. No entanto, situações envolvendo comportamento desleal, necessita de um árbitro humano que deverá intervir através de um monitor.

3.4 Monitor de Visualização

O monitor SimSpark é responsável por renderizar toda simulação. Ele conecta-se a execução do servidor e recebe continuamente o fluxo de atualizações que descrevem o estado do jogo

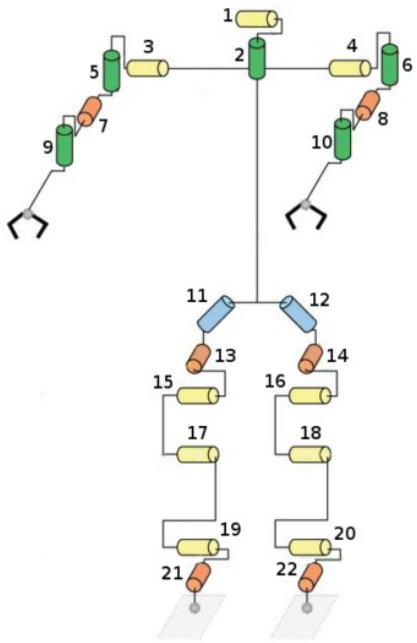

Figura 3.2: Graus de liberdade das juntas do NAO
(SIMPARK, 2012)

(SIMPARK, 2012).

O modelo de fluxo de dados que o servidor envia para o monitor, é chamado de Formato de Monitor, uma linguagem personalizável, usada para descrever o estado da simulação. Além disso, cada formato de monitor pode fornecer um mecanismo para transferir um estado específico adicional ao jogo. Para a simulação de futebol, o estado atual de jogo e gols marcados até o momento podem ser considerados como um exemplo.

O próprio monitor, isto é, o cliente, utiliza *plugins* que são destinados a analisar o estado do jogo e exibí-lo, a exemplo, a (Figura 3.3a) que apresenta *PLAYMODE* com valor PlayOn que indica jogo em andamento e o placar do jogo, nos cantos superiores da tela. O monitor também pode reproduzir o jogo a partir de arquivos gravados em log, sevindo como replay da partida (SIMPARK, 2012). Os logs possuem informações de todo cenário parcial, ou completo, enviado pelo servidor.

O *rcssmonitor* é monitor interno que faz parte do servidor SimSpark. Ele é ativado no script de instalação *simspark.rb*, permitindo a visualização do jogo processado no servidor. Esse monitor apresenta uma menor interatividade, como também, uma visualização menos realística, tendo baixo jogo de iluminação e textura do ambiente comparado ao da Figura

3.3b.

(a) RCSSMonitor (b) RoboViz

Figura 3.3: Monitor RoboViz e RCSSMonitor
(ROBOVIZ, 2012)

Diferente do *rcssmonitor* o *Roboviz* é um monitor interativo (desenvolvido em JAVA por *Justin Stoecker* em colaboração com o grupo RoboCup da *University of Miami's Department of Computer Science*), para analisar e apoiar o desenvolvimento de agentes para liga de simulação 3D. O Roboviz facilita a visualização do comportamento em tempo real dos agentes para fornecer análise de alto nível, gerando imagens próximas da realidade como mostra a Figura 3.3b (STOECKER; VISSER, 2012).

Sejam quaisquer um dos dois monitores, os mesmos interagem com o servidor, em tempo real, no propósito de enviar mensagens para mover a posição de um agente, bola ou modificar o modo de reprodução do jogo.

Atualmente, o *Roboviz* é utilizado pelos times da RoboCup na liga de simulação 3D do campeonado mundial de futebol de robôs, permitindo visualizar novos comportamentos dos agentes durante uma competição oficial, ou testes em tempo real. Essa visualização é através de dados coletados de acordo com protocolos de comunicação.

3.5 Protocolo de Comunicação

Os atuadores e sensores são simulados no SimSpark utilizando troca de mensagens com agentes. A mensagem é o protocolo de comunicação baseada na estrutura de dados *S-expressions*, conforme Quadro 3.1, ou seja, abreviação de expressões símbolicas, criadas

na linguagem de programação **Lisp**, utilizadas para códigos e dados (BUSTAMANTE, 2008).

O uso da *S-expressions* se dá, devido ao fato de ser o formato de dados de fácil compreensão e análise, sobretudo uma sintaxe compacta, proporcionando legibilidade na compreensão para seres humanos, possuindo também menor tráfego de informação na rede (SIMOES et al., 2011).

3.5.1 Comunicação entre Servidor e Agentes

O servidor expõe uma interface de rede para todos os agentes, na porta TCP 3100 por padrão. Quando um agente conecta-se com servidor, primeiro envia uma mensagem *CreateEffector*, seguido por outra mensagem *InitEffector* (SIMPARK, 2012). O *CreateEffector* contém informações, com todos os parâmentros do jogador identificando unicamente, a posição no ambiente de cada um deles, no objetivo de se criar uma instância de cada agente, em seguida, o *InitEffector* contém informações de seu time e camisa, para a identificação de qual time pertence e o seu número.

Por conseguinte, após o processo de conexão, o servidor envia mensagens para grupos de agentes. Essas, contêm informações sobre posições de articulação, modelo robótico, mensagens ouvidas, objetos vistos, entre outros. As mensagens enviadas dependem do modelo criado para o agente. Em resposta as informações de percepção, o agente pode influenciar a simulação enviando mensagens efetuadoras que executam tarefas como mover articulações do modelo robótico, para chutar, caminhar, ou girar (SIMPARK, 2012).

Cada ciclo de mensagem é de 20ms. Caso o tempo de interação entre o agente e o servidor seja superior ao ciclo estimado, haverá muitas atualizações no motor de simulação física representado na arquitura do SimSpark (Figura 3.1), uma vez que o servidor não possui a interação direta com os agentes.

3.5.2 Comunicação entre Servidor e Monitor

O servidor se comunica com os monitores através da interface de rede na porta TCP 3200 por padrão. Essa porta permite que processos externos sejam periodicamente notificados, sobre o estado da simulação. Além disso, uma vez conectado ao servidor, o monitor pode enviar várias mensagens de comandos que têm a função de mover a posição do agente, isso pode ocorrer em um campeonato, quando o juiz humano deve inteferir no jogo, pois dois jogadores de diferentes times colidem-se não podendo mudar de posição. Esses comandos não estão disponíveis para os agentes diretamente, nem são permitidos durante a competição de futebol, exceto em casos especiais, como citado acima (SIMPARK, 2012).

Sempre que um usuário conecta o monitor ao servidor SimSpark para visualizar o jogo, o monitor recebe a informação da seguinte ordem:

1. O servidor envia uma mensagem de informação do ambiente e do estado do jogo, seguido do grafo de cena completa.
2. O servidor mantém o envio de mensagens de estado de jogo parcial, seguidas de um grafo de cena total ou parcial, dependendo do que foi atualizado no ambiente.

A taxa na qual o servidor envia mensagens para o monitor é de 40 ms, conforme definido no arquivo de configuração *spark.rb*, localizado no diretório de instalação

Mensagens Enviadas do Servidor para o Monitor

A mensagem do servidor contendo informações sobre o ambiente tem a seguinte estrutura:

Quadro 3.1: Estrutura total da Mensagem do servidor

((<EnvironmentInformation>)(<SceneGraphHeader>)(<SceneGraph>))

Onde *EnvironmentInformation* representa as mensagens de dimensões do campo e do gol, distância entre agentes, tempo da partida, placar, estado do jogo entre outras informações sobre o ambiente conforme o Quadro 3.1 e as descrições da mensagens na Tabela 3.1.

SceneGraphHeader recebe informações que indicam se a construção da cena é parcial, ou total, por último, o *SceneGraph* contém informações sobre toda a cena gráfica com as posições dos objetos.

Quadro 3.2: Mensagem do Ambiente

((FieldLength 18)(FieldWidth 12)(FieldHeight 40) (GoalWidth 2.1)(GoalDepth 0.6)(GoalHeight 0.8) (FreeKickDistance 1.3)(WaitBeforeKickOff 2) (AgentRadius 0.4)(BallRadius 0.042)(BallMass 0.026) (RuleGoalPauseTime 3)(RuleKickInPauseTime 1)(RuleHalfTime 300) (play_modes BeforeKickOff KickOff_Left KickOff_Right PlayOn KickIn_Left KickIn_Right corner_kick_left corner_kick_right goal_kick_left goal_kick_right offside_left offside_right GameOver Goal_Left Goal_Right free_kick_left free_kick_right)) (time 0)(score_left 0)(score_right 0)(play_mode 0)

Tabela 3.1: Descrição das mensagens do *EnvironmentInformation*

Mensagem	**Descrição**
FieldLength, FieldWidth e *FieldHeight*	Dimensões do campo de futebol
GoalWidth, *GoalDepth*	Dimensões dos gols
FreeKickDistance	Distância que os agentes opostos têm que se posicionar quando um jogador realiza uma cobrança de falta
WaitBeforeKickOff	Tempo em que o servidor aguarda antes de iniciar o jogo automaticamente
AgentRadius	Raio de cada agente
BallRadius e *BallMass*	Raio e massa da bola respectivamenteg
RuleGoalPauseTime	Tempo que o servidor espera após um gol marcado antes de mudar o *PlayMode*
RuleKickInPauseTime	Tempo que o servidor espera após a bola sair do campo antes de mudar para o chute no modo de reprodução
RuleHalfTime	Comprimento de um intervalo
play_modes	Lista as diferentes play_modes da simulação de futebol
time	Duração do jogo
half	Tempo do jogo (1^o e 2^o tempo)
score_right e *score_left*	Pontuação do time do lado direito e esquerdo
play_mode	Estado do jogo

Quadro 3.3: Mensagem do cabeçalho - cena gráfica

(RSG 0 1)

No Quadro 3.3 o **RSG** (*Ruby Scene Graph*) indica que a mensagem do grafo de cena possui a descrição completa do ambiente, ao contrário do **RDS** (*Ruby Diff Scene*) que é uma descrição parcial. O **RDS** é composto de nós vazios que representam a estrutura do grafo de cena e algumas informações atualizadas, sobre os nós, que mudaram recentemente.

Por fim, o *SceneGraph* é uma das informações que o servidor passa para o monitor, as quais contêm as posições das traves, das linhas do campo, da bola e dos agentes. O *SceneGraph* será mais aprofundado, em outro subtópico, por ser um dos focos deste trabalho.

Mensagens Enviadas do Monitor para Servidor

O monitor pode servir de árbito durante a competição, fazendo cumprir as regras do jogo ainda não realizada pela simulação (SIMPARK, 2012). Na Tabela 3.2 pode ser observadas algumas mensagens enviada do monitor para o servidor.

Tabela 3.2: Mensagens do Monitor para o Servidor

Função	**Sintaxe do comando**
Mover Agente	(agent (unum <num>) (team <team>) (pos <x> <y> <z>) (rot <x> <y> <z>))
Mover Bola	(ball (pos <x> <y> <z>)(vel <x> <y> <z>)
Mudar *Playmode*	(playMode <playmode>)
Reposicionar Agente	(repos (unum <num>) (team <team>))

Na sintaxe do comando da função **Mover Agente** o *unum* é o número do agente; *team* especifica a equipe da esquerda ou direita; *pos* indica a posição para onde o agente será movido; *rot* é movimento de rotação em relação ao seu eixo, x, y, z são valores absolutos em coordenadas de campo, a exemplo, $(0, 0)$ que é o centro do campo.

Na próxima linha da Tabela 3.2 a sintaxe da função **Mover Bola** constitui-se de *vel*, que é a velocidade que a bola terá após ser movida, para a posição indicada pelo parâmetro *pos*.

Na mesma tabela a função **Mudar *Playmode*** define modos de jogo, como inicio ou fim.

Reposicionar Agente é uma função que contém dois parâmetros: *unum* parâmento onde é informado o número do jogador e *team* o time que pode ser *Right* ou *Left*

3.5.3 Conteúdo da Cena Gráfica

Este trabalho usou a mesma conexão do monitor, para coletar os dados sobre o jogo, sendo assim, todas mensagens recebidas são as mesmas que o monitor visualiza. Dessa forma, vamos detalhar cada parte do conteúdo da cena gráfica, enviada do servidor para o monitor.

O grafo da cena é uma estrutura que organiza a representação lógica e espacial da cena gráfica. No SimSpark, o gráfico da cena é uma árvore com o nó raiz definido na origem <$0; 0; 0$>. Cada nó representa um objeto tendo zero ou mais filhos, a exemplo do nó do agente, onde seus nós filhos compõem cada parte do corpo. O valor da posição e da rotação de cada parte do corpo é obtido pela multiplicação das matrizes de transformação (SIMPARK, 2012).

Quando o servidor envia a mensagem contendo informação do ambiente, cabeçalho da cena gráfica e o grafo da cena gráfica conforme o Quadro 3.1, a parte da mensagem para construção de objetos da cena, é o grafo da cena. É nesse grafo que contém os nós raiz dos objetos, seus nós filhos e outros. É essencial as abreviações de termos para descrever o meio ambiente, no objetivo de poupar largura banda na rede. Esses termos são mostrados na Tabela 3.3 (BUSTAMANTE, 2008).

É importante mencionar que a mensagem responsável por construir o grafo, recebida pelo monitor, representa uma atualização para o grafo da cena. Quando um objeto na cena não modifica suas informações, o servidor envia o grafo para o monitor omitindo as informações do nó que não está sendo atualizado. Em outras palavras, a estrutura do grafo da cena permanece inalterada e esses nós (*nd*) serão vazios.

Existem dois tipos de nós: Nó de base (Quadro 3.4) e de transformação (Quadro 3.5).

Quadro 3.4: Mensagem de nó base

(*nd BN <contents>*)

Tabela 3.3: Abreviações de termos do Grafo da cena

Abreviatura	**Descrição**
nd	Nós BN e TRF
BN	Nó base dos seguintes tipos: *SMN, sSc, sMat, RSG, RDS*
TRF	Nó de transformação com o nó filho *SLT*
SLT	Definir transformação local do nó folha
SMN	Estátística da malha nó
sSc	Define escala nó folha
sMat	Define material nó folha
RSG	Estado completo
RDS	Estado de Atualização

Cada tipo de nó que não é o nó de transformação, o nó de malha estático, ou o nó de luz, é considerado o nó de base (incluindo o nó da raiz). Nó de base não têm descrição na mensagem, eles são simplesmente usados para preservar a estrutura do grafo de cena. Eles podem conter qualquer número de nós filhos.

Quadro 3.5: Mensagem do nó de transformação

```
[nx ox ax Px]
[ny oy ay Px]
[nz oz az Pz]
[ 0  0  0  1]
```

O nó transformado, representa a matriz de transformação homogênea 4x4, que é comumente usada em robótica e geometria computacional para representar transformações geométricas. A matriz de transformação define como mapear pontos de um espaço de coordenadas, para outro espaço de coordenadas, através da definição de tradução, de rotação e o fator de escala. O fator de escala considera 1, para a maioria das aplicações (SIMPARK, 2012).

O Quadro 3.5 apresenta vetores n, o e a representam normal, orientação e abordagem, respectivamente. Esses três vetores representam um quadro de referência no espaço tridimensional, possibilita a criação de um objeto no espaço.

O SimSpark representa a matriz de transformação do Quadro 3.5 em uma linha do seguinte

modo: *(nd TRF (SLT nx ny nz 0 ox oy oz 0 ax ay az 0 Px Py Pz 1))* onde. *SLT* representa um nó folha, *TRF* nó de transformação.

Geometria dos Nós

Os nós *StaticMash* e o *SMN* descrevem formas de objetos, eles especificam sua dimensão, sua textura e não têm nós filho (são folhas). O *Light* especifica o jogo de iluminação dos objetos na cena.

O *StaticMash* define a malha a ser carregada a partir do arquivo *.obl*.

Quadro 3.7: Mensagem nó de Malha

(nd StaticMesh (load <model>) (sSc <x> <y> <z>) (setVisible 1) (setTransparent) (resetMaterials <material-list>))

Onde, *model* é o caminho para o arquivo *.obj*; *sSc* define a escala do objeto; *setVisible* (opcional) carrega uma *flag* de bit que indica se o objeto é visível ou não; *resetMaterials* define a lista de materiais utilizados. O Quadro 3.8 apresenta um exemplo de uma mensagem de nó de malha.

Quadro 3.8: Mensagem nó de Malha

(nd StaticMesh (load models/rlowerarm.obj) (sSc 0.05 0.05 0.05)(resetMaterials matLeft naowhite)) (nd StaticMesh (load models/naohead.obj) (sSc 0.1 0.1 0.1)(resetMaterials matLeft naoblack naogrey naowhite))

SMN, tipo de malha predefinida:

Quadro 3.9: Mensagem nó de Malha SMN

(nd SMN (load <type> <params>) (sSc <x> <y> <z>) (setVisible 1) (setTransparent) (sMat <material-name>))

Onde, *type* deve ser um dos seguintes: *StdUnitBox StdUnitCylinder*, em que *params* detém dois valores: comprimento e raio.
StdUnitSphere StdCapsule com *params* (não é usado no rcssserver3d 0.6.3, mas está

disponível para outras simulações SimSpark); *sMat* especifica o nome do material a ser aplicado ao objeto.

O nó de luz descreve a maneira na qual a luz difusa, ambiente e especular afetam o objeto. A estrutura é apresentada no Quadro 3.2:

Quadro 3.10: Mensagem Nó de Luz

(nd Light (setDiffuse x y z w) (setAmbient x y z w) (setSpecular x y z w))

Onde, <x; y; z> é o vetor que define a direção de luz e w é o fator de escala; *setDiffuse* é a luz que vem de uma direção, atinge a superfície e é refletida em todas as direções; assim, apresenta a mesma intensidade independente do local no qual o observador está posicionado; *setAmbient* é o resultado da luz refletida no ambiente; é a luz que vem de todas as direções. *setSpecular* é a luz que vem de uma direção e tende a ser refletida numa única outra direção.

Capítulo 4

Ferramenta *Trainer*

Na área da robótica existem diversos pesquisadores desenvolvendo novas funcionalidades para robôs bípedes ou até melhorando as que existem. Os competidores das equipes da RoboCup são exemplos desses tipos de pesquisadores que desenvolvem robôs bípedes para jogar futebol. Para que esses agentes tenham um melhor desempenho, se torna necessário a análise estatística sobre como o agente se comporta na partida e assim podendo aperfeiçoar as características dos agentes de certo time.

Dessa forma, nota-se que é preciso uma ferramenta para analisar de forma automática a maneira com que os agentes se comportam na partida. Visto que, análises, testes e resultados coletados manualmente tornam-se trabalho dispendioso e em muitos casos imprecisos e incoerentes. Devido a esse fato, entre outros trabalho, a monografia de Silva (2014) foca na ferramenta de análise de agentes de futebol, buscando coletar dados para gerar estatística de desempenho.

Em busca de analisar o desempenho de robôs bípedes simulados em 3D, este trabalho foca no desenvolvimento de novas funcionalidades da ferramenta *Trainer*. Essa ferramenta é utilizada para coletar dados dos agente durante o jogo, permitindo pesquisadores desenvolverem novas caracteristas de acordo com o desempenho dos agentes no propósito de evoluir as habilidades dos agentes, durante a partida.

Nesse sentido, Soares (2013) desenvolveu um modelo para que os agentes possam traçar

melhores trajetórias e desviar-se de obstáculos para passar por seus adversários sem colidir com os mesmo. Na realização dos testes, foi necessária a utilização da versão anterior do *Trainer*, que apresentava alguns limites, como por exemplo, a coleta de dados para todos os agentes da mesma partida. Este trabalho permite que o *Trainer* referido possa coletar dados de todos os agentes da mesma partida, comprovados com teste, experimentos e resultados relevantes.

Ferreira et al. (2013) em seu trabalho, criaram funcionalidades para comportamentos de robôs bípedes baseado na Inteligência Artificial que define o movimento de juntas no objetivo de desenvolver um chute omnidirecional em tempo real. Desse modo, o humanóide adapta-se às condições dinâmincas do jogo, sem necessitar de movimentos pré-programados. Os resultados obtidos provaram que os comportamentos do agente são executados de forma mais precisa e eficiente.

4.1 Modificação da Ferramenta *Trainer*

O foco desta pesquisa foi a coleta confiável dos dados, através da mensagem enviada do servidor para o monitor, essa mensagem tem uma estrutura conforme o Quadro 3.1. Assim sendo, a mensagem do *EnvironmentInformation* fornece os dados conforme a Tabela 3.1. Por conseguinte, esses dados são tratados gerando informações úteis.

Além disso, *SceneGraphHeader* fornece o dado necessário para ser tomada a decisão. Caso esse dado seja para construção da cena completa, será coletada a posição da bola e de cada jogador com o seu número para identificar unicamente cada agente e posteriormente gerar estatísticas de forma individual. Caso seja para construção da cena parcial, então é utilizado a mensagem do *SceneGraph* apenas para coletar as posições dos objetos (bola ou agente) de forma dinâmica, para gerar estatísticas ao final do jogo.

Portanto essa coleta de dados se torna extremamente importante para gerar análises estatísticas, com a extensão da ferramenta *Trainer*, a qual é capaz de avaliar o desempenho de todos agentes que estiverem em uma partida de uma só vez e retornar as informações sobre quantidades de quedas, por agente, tempo que permaneceu caído, somatório do tempo

que permaneceu caído, média de queda dos agentes por time e por jogo. E ao final, é gerado relatórios.

Essa extensão da ferramenta motiva trabalhos como os supra citados a avaliar desempenho de seus agentes, após aplicações de novas características, pois, a partir da coleta de dados, além da funcionalidade já implementada (análise de quedas simples de um agente), é possível obter qualquer informação sobre os agentes que poderão ser desenvolvidas por pesquisadores.

4.2 Metodologia

Esta seção tem como objetivo a apresentação de cada fase da metodologia. A metodologia de desenvolvimento está divida em quatro fases, contendo duas atividades cada uma. Tanto as fases como as atividades, devem ser seguidas em sequência lógica, de forma que, se o leitor pretender reproduzir, ou melhorar esse trabalho, tenha mais aproveitamento do que já foi implementado. A Figura 4.1 apresenta a metodologia de desenvolvimento.

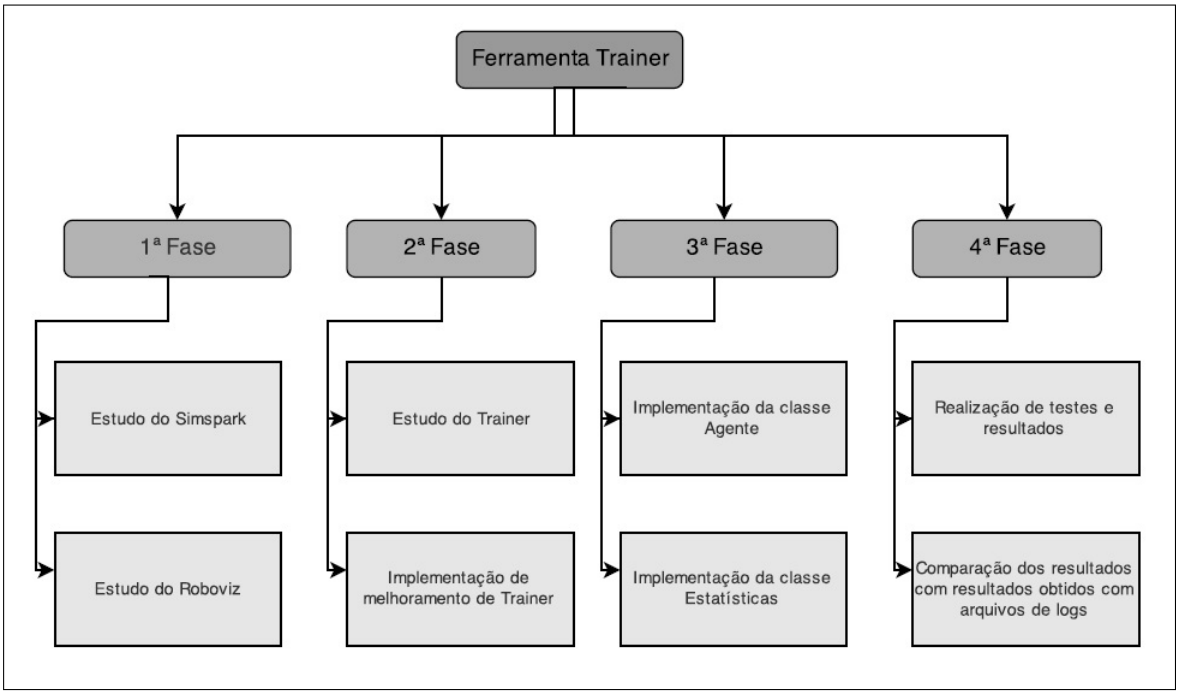

Figura 4.1: Metodologia

A **primeira fase** consistiu-se no estudo do servidor Simspark e no monitor Roboviz. A primeira atividade teve como objetivo entender qual o tipo de mensagem, o servidor envia para o monitor, além de entender o ambiente de simulação. Já a segunda atividade realizou o estudo do monitor roboviz. O mesmo recebe mensagem completa, ou parcial, da cena enviada pelo servidor e através de renderização, mostra graficamente, o ambiente em 3D. Para isso, o roboviz interpreta as mensagens enviadas pelo servidor e monta a árvore com objetos, mostrando os gráficos da cena.

Na **segunda fase** foi realizado o estudo do *Trainer* visando verificar quais as funcionalidades estavam implementadas e o que realmente precisava de melhorias. Na primeira atividade, verificou-se que o *Trainer* não estava conseguindo mover os jogadores, devido ao formato da mensagem enviada errada para o servidor, além de ser verificado porque o mesmo não conseguia pegar dados de todos os agentes em campo.

Na segunda atividade, foi criado o método para divisão da mensagem em três blocos, além da implementação de três novos métodos, o primeiro para verificar, se mensagem é parcial ou total; o segundo método para o tratamento da mensagem, caso ela seja parcial e o terceiro caso ela seja total.

Já na **terceira fase** foram desenvolvidas as classes Agente e Estatística. A primeira atividade consistiu na especificação de requisitos, modelagem e implementação da classe que tem como base guardar todas as informações sobre os agentes, como posição de cada parte do corpo. A segunda atividade especificou-se requisitos da classe Estatística, bem como, a realização da modelagem e implementação. Essa classe tem como papel principal guardar o vetor de agentes, com dados de cada ciclo, por mensagem recebida durante o jogo. Após o final da partida a classe Estatistica tem a funcionalidade de analisar, gerar estatística sobre desempenho dos agentes e também gerar arquivos que contêm posições das partes do corpo de cada agente, bem como, dados sobre o ambiente de simulação.

Por fim, na **quarta fase** realiza-se teste e comparação de resultados. A primeira atividade foram realizados teste para verificar a eficácia da coleta de dados. Segunda atividade foram realizadas comparações de resultadoes obtidos em tempo de execução com resultados obtidos a partir de arquivos de log.

A definição das fases, suas etapas de busca proporcionar ao leitor o entendimento a cerca do planejamento do projeto e todo o escopo de trabalho.

4.3 Arquitetura da ferramenta *Trainer*

A arquitetura do modelo apresenta os componentes que foram criados, ou melhorados, mostrando o fluxo de execução e a interconexão entre os mesmos.

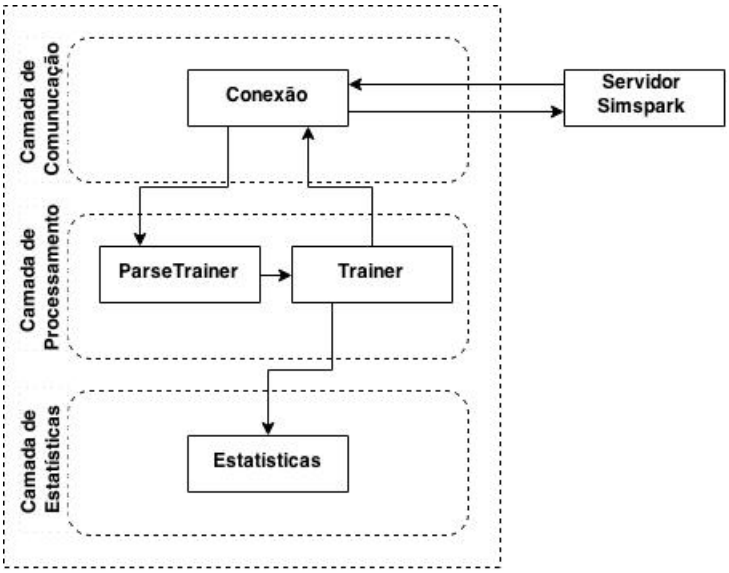

Figura 4.2: Arquitetura da Ferramenta *Trainer*

Inicialmente a ferramenta *Trainer* conecta-se com o servidor SimSpark, através da camada de comunicação utilizando *Socket* com o protocolo *TCP/IP*, para receber e enviar mensagens para o servidor. Após a conexão, essa camada se comunica com a camada de processamento no objetivo de enviar a mensagem recebida para ser tratada pela classe *ParseTrainer*, sendo assim é a vez da classe *Trainer* receber a mensagem e instanciar os objetos simulados pelo servidor: Agentes e bola, além de guardar as informaçoes sobre o ambiente. Na sequência a camada de processamento envia informações dos objetos instanciados para a camada de estatística a cada ciclo, essa, guarda todas as informações durante jogo e no final gera um arquivo *.txt* com resumo da partida e uma pasta contendo arquivos *.xls* para cada agente e bola, com todos dados de movimentos do agente e da bola. A figura 4.2 mostra arquitetura do projeto extendido.

4.3.1 Camada de Comunicação

A camada de comunicação consiste no módulo de conexão que recebe a mensagem enviada pelo servidor, e envia para a camada de processamento. Um exemplo dessa mensagem encontra-se no Quadro 3.2. O módulo de processamento também pode enviar mensagens para o servidor utilizando a camada de comunicação. Isso ocorre quando o Trainer necessita influenciar no ambiente.

4.3.2 Camada de Processamento

A camada de processamento caracteriza-se como a camada central, por possuir dois módulos que realiza o tratamento de toda mensagem. O *ParseTrainer* que faz análise léxica da mensagem e o *Trainer* que faz a análise sintática.

Os algoritmos responsáveis pela análise léxica da classe *ParseTrainer*, estão descritos no Algoritmo 4.1 e 4.2 e o do *Trainer* responsável pela análise sintática é mostrado no Algoritmo 4.3. No 4.1 a mensagem recebida é divida pelo método *EXpress()* da classe *ParseTrainer* em três partes conforme Quadro 3.1.

Quando mensagem é completa, o primeiro *token* é identificado como $FieldLength$ como mostra o Quadro 3.2 e quando é parcial o primeiro token é o $time$ (Quadro 4.1)

Quadro 4.1: Mensagem parcial

((time 600.003)(team_left TinManBots)(half 2)(score_left 0)(score_right 0)(play_mode 12))

O próximo passo é a identificação dos tokens dessa mensagem, pelo algoritmo 4.2 das mensagens recebidas.

Depois disto, na classe *Trainer*, é realizada a análise sintática. O método *mensagemParseTrainer()* coleta os tokens iniciais para verificar se a mensagem é completa ou parcial, além de atualizar o estado do jogo com dados do ambiente, isto é, tempo do jogo, *playMode*, *score* e lado de cada time. Depois ele chama o método *atualizacaoAgentes()* ou *atualizaPosicoes()* caso a mensagem seja completa ou parcial respectivamente. Também é instanciado a classe *Estatistica* que guardará os dados durante todo o jogo.

Algoritmo 4.1 Algoritmo para tratamento da mensagem

```
1:    std::string ParserTrainer::EXpress(){
2:      std::string builder;
3:      int openBraceCount = 0;
4:       for(int i = 0;i<expression.length(); i++){
5:            char c = getChar();
6:            if(c=='('){
7:               builder+= c;
8:               openBraceCount++;
9:             }else if(c==')'){
10:              builder+=c;
11:              openBraceCount--;
12:              if(openBraceCount!=0)
13:                builder+=' ';
14:             } else if (isblank(c)) {
15:             if(openBraceCount!=0)
16:              builder+=' ';
17:         }else {
18:           builder+=c;
19:         }
20:         if(openBraceCount==0 && (!isblank(c))){
21:           if(index==expression.length()){
22:            return builder;
23:         }
24:      }
25:   }
```

O método *atualizaPosicoes()* é responsável por coletar apenas as posições de cada agente e bola que foi alterada.

O método *atualizacaoAgentes()* é responsável por coletar a quantidade de agentes em campo, suas posições iniciais, qual o lado de cada agente e sua numeração. Instanciando cada um com suas informações em um vetor de agentes, sendo um vetor para cada time.

4.3.3 Camada de Análise Estatísticas

A camada de estatística tem por objetivo coletar todos os dados referentes aos agentes, bola e estados do jogo para gerar relatórios e estatísticas. Essa camada também é responsável por gravar o relatório das posições dos agentes e da bola em aquivos *.xls* e gerar resumo da partida. Após a modificação do *Trainer*, é possível gerar diversas informações, para que seja realizada uma análise estatística, como: informações sobre quedas dos agentes (tempo de queda, soma dos tempos de queda, total de quedas, média do total de quedas), todas

Algoritmo 4.2 Algoritmo para tratamento da mensagem

```
1: std::string ParserTrainer::take() {
2:         int openBraceCount = 0;
3:         skipWhiteSpaces();
4:
5:         std::string builder;
6:
7:         while (true) {
8:                 char c = getChar();
9:
10:                if (c == '\0')
11:                        return builder;
12:
13:                if (c == ')') {
14:                        if (openBraceCount == 0) {
15:                                pushChar();
16:                                return builder;
17:                        }
18:
19:                        --openBraceCount;
20:
21:                        if (openBraceCount == 0) {
22:                                builder+=c;
23:                                return builder;
24:                        }
25:                } else if (c == '(') {
26:                        ++openBraceCount;
27:                } else if (isblank(c)) {
28:                        if (openBraceCount == 0)
29:                                return builder;
30:                }
31:
32:                builder+=c;
33:        }
34: }
```

as posições referente as partes do corpo do agente (cabeça, tronco, braço, antebraço, coxa, perna e pé), como também, as posições da bola durante todo ciclo do jogo.

Análise de Desempenho

O desempenho dos agentes é realizado com base em relatórios, gerados pelo *Trainer* dentro de um arquivo *.txt* e *.xls*. Na Tabela 4.1 é demostrado alguns dados do relatório de uma partida. As quedas são classificadas como queda simples, porque é utilizado apenas a variável Z que corresponde a posição da cabeça.

Algoritmo 4.3 Algoritmo para tratamento da mensagem - análise sintática

```
1: void Trainer::mensagemParseTrainer(std::string content){
2:     ...
3:
4:    if(token=="time"){
5:          ...
6:
7:         atualizarPosicoes(noAux,i, mLastUpdateTime);
8:
9:     }else if(token=="FieldLength"){
10:         ...
11:
12:             atualizacaoAgentes(noAux,i,mLastUpdateTime );
13:         }
14:
15: }
```

Tabela 4.1: Resumo da Partida

Legenda	**Valor**
Mensagens completas	1
Mensagens parciais	7477
Ciclos	300
Quantidade total de quedas	190
Somatório da duração de cada queda	201,266

Outra saída gerada pelo *Trainer*, é uma pasta com arquivos *.xls* para cada agente da partida, contendo coordenadas de cada parte do agente, mais um arquivo *.xls* da bola. Esses dados são guardados na memória a cada ciclo.

Tabela 4.2: Dados das coordenadas do Agente

TEMPO (CICLO)	**HEAD**	**P_X**	**P_Y**	**P_Z**	**P_E**
0,02		-0,0136844	-0,999854	-0,0102093	1

A Tabela 4.2 possui os valores *P_X*, *P_Y*, *P_Z* que correspondem as coordenadas, posição da cabeça do agente e *P_E*, a escala do ponto. O *Trainer* também gera saídas para abordagem, orientação e normal, formando a matriz de quatro dimensões que é renderizada no roboviz citada no Quadro 3.5.

Capítulo 5

Testes, Experimentos e resultados

5.1 Metodologia de Testes

A metodologia de testes realizou-se no ambiente adequado para a competição. Foi necessário montar e configurar todo ambiente, seguindo a instalação de acordo com a Tabela 5.1 para realização dos teste e validação.

Tabela 5.1: Programas e configuração de máquinas no ambiente real.

Programas	**Configuração de maquinas**
Simspark e *Trainer*	Computador Core i7, RAM-4GB, HD-1TB
Roboviz	Computador Core i3, RAM-4GB, HD-1TB
Time A X Time B	Notebook DELL Core i7, RAM-8GB, HD-1TB

Para testes e validações, foram utilizados quatro times (Tabela 5.2) da liga de simulação 3D que participaram do campeonato mundial de futebol de Robôs, RoboCup 2013. Os mesmos jogaram contra o BahiaRT do **ACSO**, laboratório de pesquisas na área robótica da Universidade Estadual da Bahia - **UNEB** onde foram realizados os testes. Foram realizadas 40 partidas, sendo dez com cada time e cada time com onze humanóides. A cada rodada, os times eram trocados de lado.

Tabela 5.2: Tabela de times utilizado nos experimentos

Equipe	**Universidade**
Utaustinvilla	*University of Texas at Austin*
RoboCanes	*University of Miami*
MagmaOffenburg	*Offenburg University of Applied Sciences*
HFutEngine	*School of Computer and Information Hefei University of Technology*

5.2 Métricas de Avaliação

A validação da ferramenta *Trainer* foi comprovada, através da análise dos resultados gerados. A primeira análise é realizada após o término da partida, todavia os dados obtidos durante a partida, pelo *Trainer*, é confrontados de forma manual, com as informações contidas no arquivo *.log* que é gerado pelo monitor. A segunda análise segue os mesmos passos de avaliação, diferenciando-se por ser de forma automática. Essas análises tem o objetivo de avaliar a consistência dos dados da ferramenta *Trainer*, para que futuros pesquisadores possam gerar estatística mais avançadas dos agentes.

A princípio, para que sejam verificados as posições dos agentes, é necessário que a ferramenta *Trainer* reconheça-os. Desse modo, para essas duas análises de validação, a ferramenta *Trainer* se comunica com o servidor, após os 22 agentes estiverem em campo, sendo dois times de onze jogadores. A ferramenta dá início a partida, iniciando o recebimento de mensagens completas e parciais do servidor, baseado nessas mensagens, o *Trainer* coleta os dados e gera para cada agente um arquivo *.xls*.

Os 22 arquivos dos agentes gerados, são compostos por linhas, em cada linha contém um instante (ciclo) da partida, seguido pelas posições de todas partes do corpo do agente e dele, possibilitando identificar unicamente as posições de cada agente, em cada instante de tempo da partida.

Além disso, para realizar a análise de validação, é necessário o log que é gerado pelo monitor. Esse log é finalizado, após o término do jogo; o mesmo serve para informar todos os acontecimentos que ocorreram durante o jogo e possibilita uma reprodução exata da partida, ou seja, o replay.

Devido a geração dos arquivos: *.xls* e *.log* respectivamente pela ferramenta *Trainer* e monitor, agora é possível confrontar os dados. Na primera análise, essa comparação é realizada da seguinte maneira: Para cada linha de cada arquivo *.xls* busca-se, manualmente, no arquivo *.log* comparar as posições das partes dos corpo de cada agente no certo ciclo da partida. Assim sendo, essa análise é repetida sequencialmente, nas primeiras 10 linhas do arquivo *.xls*. Esse processo manual foi bastante dispendioso, pois, o arquivo *.log* possue os dados brutos, dificultando encontrar os dados para comparação.

A segunda análise é realizada de forma automática, onde o *Trainer*, após a geração dos arquivos *.xls* de cada agente que foram coletados diretamente do servidor, durante a partida, é submetido a uma nova execução, utilizando dessa vez o arquivo .log dessa mesma partida, gerado pelo monitor. Então, o *Trainer* gera outro conjunto de arquivos *.xls*. Por fim, para cada linha de cada arquivo *.xls* gerado diretamente pelo servidor, compara-se com cada linha de cada arquivo gerado, baseado no *.log* gerado pelo monitor.

5.3 Resultados

Após os experimentos manuais e automáticos, percebe-se que a ferramenta *Trainer* agiu de modo eficaz, em cada ciclo no reconhecimento dos 22 robôs e suas posições, conforme a Tabela 5.3.

Tabela 5.3: Validação dos dados

Arquivos	**Log**
$.xls_1$	Ok
$.xls_2$	Ok
$.xls_3$	Ok
$.xls_4$	Ok
$.xls_5$	Ok
...	Ok
$.xls_{22}$	Ok

A partir da análise do resultado e a coerência dos dados afirmados pelos resultados do *Trainer*, elaborou-se a simples estatística de quedas de robôs, para comprovar que com

base nos dados fornecidos pela ferramenta, é possível realizar diferentes estatísticas. Vale a pena citar algumas estatísticas possíveis para serem implementadas, por exemplo, velocidade de movimentação dos robôs, quantidade de chutes que um agente realiza, passes corretos, precisão dos chutes, entre outros.

Assim, constrói-se uma estatística com base nos dados coletados pelo *Trainer*, desenvolvendo uma estatística de quedas de agentes, baseada na posição da cabeça. Essa estatística é realizada durante a partida. Quando um agente cai, o valor do vetor da posição da cabeça, fica menor que 0.2 identificando uma queda, com isso, é possível extrair as seguintes informações geradas, em um arquivo *.txt*: Quantidade de quedas, tempo total de queda do time *left* e *Right*, total de quedas do jogo, média de quedas, entre outras.

Para geração de estatística de quedas, foram escolhidas algumas equipes que sempre participam do campeonato de futebol na liga 3D, sendo que os binários (executáveis) são da RoboCup 2013, realizada em *Eindhoven - Holanda*. É preciso ressaltar que todas as partidas foram realizadas da seguinte forma: Todas as equipes jogaram contra a BahiaRT, pois este trabalho surgiu do grupo. Após a elaboração do confronto, foram alcançados os resultados a seguir

As dez partidas do BahiaRT contra o Magma, tiveram os resultados apresentados na Tabela 5.4.

Tabela 5.4: Resultados das primeiras partidas.

Partidas	**Times**	**Tempo(ciclos)**	**Média de Quedas**	**Desvio Padrão**
10	BahiaRT x MAGMA	300	222	+-92,74966307

A análise mostrada na Tabela 5.4 demonstra que os agentes ainda possui o nível alto de quedas, no entanto análises mais aprofundadas para saber a causa, não foram implementadas. Sendo assim, não é possível jugar porque acontecem tantas quedas. Podendo as mesma serem causadas por colisões, desequilibrio ou métricas para executar alguma ação, por exemplo, chutar ou dar um passe de bola.

Na Figura 5.1 é possível observar a quantidade de quedas simples do jogo, entre o time BahiaRT x MagmaOffenburg.

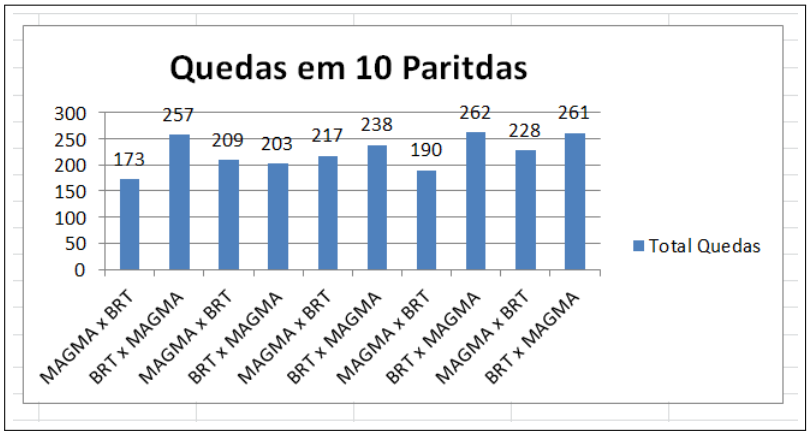

Figura 5.1: BahiaRT x MagmaOffenburg

São gerados pelo *Trainer*, dados específicos para cada time. Os resultados dos mesmos, são apresentados a seguir. Os times trocaram o lado do campo, a cada 300 ciclos.

Resultados de quedas simples, nas partidas:

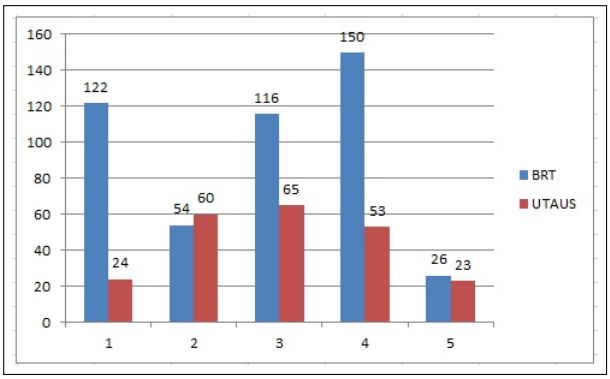

Figura 5.2: BahiaRT x Utaustinvillas

Percebe-se, na Figura 5.2, que a quantidade de quedas do time BahiaRT é maior que a do adversário, equipe que é considerada a melhor da RoboCup. Nas partidas seguintes, é mostrado nos gráficos que os times caem mais que o BahiaRT. Esses são considerados inferiores ao analisado na Figura 5.2.

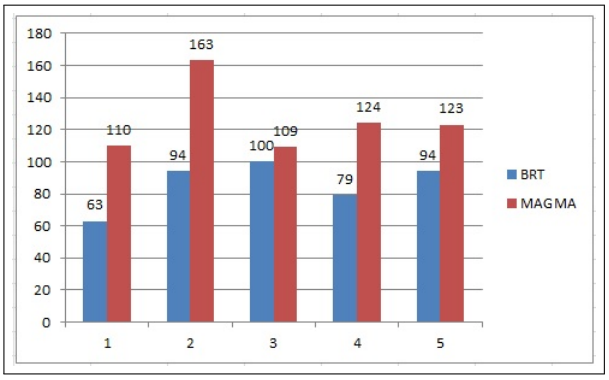

Figura 5.3: BahiaRT x MagmaOffenburg

Observe na Figura 5.3 que o time *MagmaOffenburg*, em três partidas, chega a cair quase o dobro do que o time BahiaRT. No entanto, contra o *Utaustinvillas* o BahiaRT cai bem mais que o dobro.

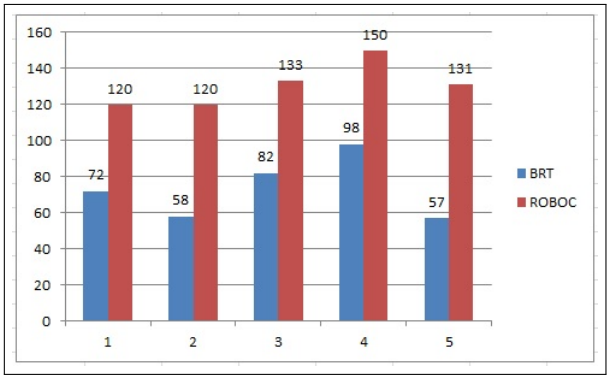

Figura 5.4: BahiaRT x RoboCanes

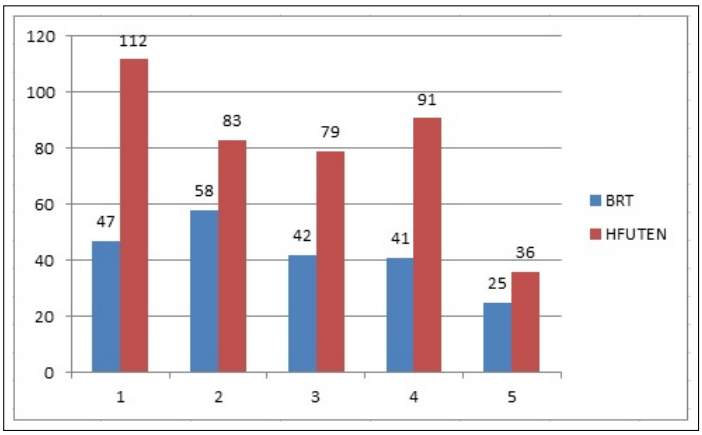

Figura 5.5: BahiaRT x Futengine

Este trabalho, baseado em experimentos, resultados e análises buscou a automatização da

coleta de dados e geração de estatísticas para análise de desempenho de times de futebol de robôs em simulação 3D.

Capítulo 6

Conclusões e Trabalhos Futuros

Tendo em vista os aspectos observados como também as melhorias implementadas na ferramenta (coleta de dados, geração de análises e de relatórios), percebemos que é possível fazer várias análises e levantamentos estatísticos que possibilitem uma avaliação completa de desempenho dos agentes na partida. Sendo assim os pesquisadores poderão evoluir seus agentes tendo maior feedback sobre aplicação de novas métricas.

No início dos testes, percebeu-se que a arquitetura do socket, no projeto base, não estava estruturada para outros cenários, o que provocou limitação, no projeto atual, não sendo possível coletar dados com o *Trainer*, em máquina diferente da máquina do servidor. Mas mesmo com as dificuldades, foi possível concluir que: a extensão deste projeto trouxe dados e avaliação de desempenho importantes para o grupo de pesquisa que testou o modelo. O objetivo foi criar uma arquitetura que analisasse movimentos dos agentes, gerando relatório de seu desempenho, como também, possibilitar automatização dos testes.

A análise de resultados de forma ágil e confiável é uma das maiores dificuldades no trabalho com robôs bípedes, especialmente, quando o foco de pesquisa é direcionado a otimização de movimentos. O arcabouço estruturado na ferramenta *Trainer* através deste trabalho, fornece a base não somente para análises estatísticas sofisticadas como também, para o uso dos dados coletados para suprir funções de *fitness* e avaliação em métodos de otimização baseados em computação evolucionária ou aprendizagem por reforço. Como trabalhos futuros, citamos a resolução de problemas que foram identificados durante o desenvolvimento e testes. Além

da criação de novos métodos para analises mais profundas, como, verificar quanto tempo um agente teve posse de bola, a causa da queda, a velocidade média, entre outras que não foram implementadas. Neste processo deve ser agregada ao projeto uma classe abstrata extensível que permita facilmente adicionar qualquer estatística que seja necessária. Outro trabalho futuro relevante é a integração com otimizadores e mecanismos de aprendizagem de máquina. Por fim, sugerimos a criação de uma interface gráfica para facilitar a criação de cenários de testes que devem ser repetidos diversas vezes, facilitando o uso da ferramenta, bem como a extração das estatísticas necessárias.

Referências Bibliográficas

BOEDECKER, J.; ASADA, M. Simspark - concepts and application in the robocup 3d soccer simulation league. 2008.

BUSTAMANTE, C. Simspark monitor protocol. 2008.

CHICK, S. et al. Spades -a distributed agent simulation environment with software-in-the-loop execution. 2003.

FERREIRA, R. P. G. et al. Desenvolvimento de um chuto omnidirecional para um robô humanoide. 2013.

GONCALVES, A. Locomocao de bípede. 2011.

HEINEN, M.; OSÓRIO, F. Evoluccão do caminhar de robôs móveis simulados utilizando algoritmos genéticos. 2006.

HIROSE, M.; OGAWA, K. Honda humanoid robots development. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, The Royal Society, v. 365, n. 1850, p. 11–19, 2007.

KITANO, H. et al. Robocup: The robot world cup initiative. In: ACM. *Proceedings of the first international conference on Autonomous agents*. [S.l.], 1997. p. 340–347.

KÖGLER, M. Simulation and visualization of agents in 3d environments. In: . [S.l.: s.n.], 2003.

KRAETZSCHMAR, G. K. et al. The ulm sparrows: Research into sensorimotor integration, agency, learning, and multiagent cooperation. In: *RoboCup-98: Robot Soccer World Cup II*. [S.l.]: Springer, 1999. p. 452–457.

NICOLAS, J. Artificial evolution of controllers based on non-linear oscillators for bipedal locomotion. *Master's thesis, EPFL, winter*, v. 2006, 2005.

ROBOCUP. *A Brief History of Robocup*. 2015. Disponível em: <http://www.robocup.org/about-robocup/a-brief-history-of-robocup>. Acesso em: 10 de Maio de 2015.

ROBOVIZ, C. *Rovoviz*. 2012. Disponível em: <https://sites.google.com/site/umroboviz/features>. Acesso em: 09 de Novembro de 2014.

ROLLMANN, M. Spark - a generic simulator. In: . [S.l.: s.n.], 2004.

RUSSEL, S. J.; NORVIG, P. *Inteligencia Artificial*. [S.l.]: Pearson Education, 2003.

SILVA, B. V. Mineração de dados para análise quantitativa de chutes a gol em um ambiente de simulação de futebol de robôs em duas dimensões. 2014.

SILVA, M. S.; MACHADO, J. T. Sistemas robóticos de locomoção-estado da arte. *ISEP-Instituto Superior de Engenharia do Porto, Departamento de Engenharia Electrotécnica, Porto, Portugal*, 2001.

SIMOES, M. et al. Virtualizac ao de robôs. 2011.

SIMPARK, C. *SIMPARK Sourceforge*. 2012. Disponível em: <http://simspark.sourceforge.net/wiki/index.php/Network_Protocol>. Acesso em: 09 de Setembro de 2014.

SMITH, R. et al. Open dynamics engine. 2005.

SOARES, A. dos S. Planejamento e mapeamento de trajetórias em tempo real para navegac ao de robôs humanóides em ambiente simulado 3d. 2013.

STOECKER, J.; VISSER, U. Roboviz: Programmable visualization for simulated soccer. In: *RoboCup 2011: Robot Soccer World Cup XV*. [S.l.]: Springer, 2012. p. 282–293.

TORRES, S. O. do A. *Avaliacao de protótipo Mecatrônoco de Locomaccão Bípede*. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, COPPE, 2006.