UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

USPDroidsSS : AMBIENTE DE SIMULAÇÃO PARA FUTEBOL DE ROBÔS DA CATEGORIA IEEE VERY SMALL SIZE – VERSÃO 2010.A

DANILO TADASHI DOI
MARCELO OLIVEIRA DA SILVA
RAPHAEL MONTANARI
WILLIAN CONSTÂNCIO DA SILVA
ROSELI APARECIDA FRANCELIN ROMERO

Nº 357

RELATÓRIOS TÉCNICOS

São Carlos – SP
Jul./2010

USPDroidsSS : Ambiente de Simulação para futebol de robôs da Categoria IEEE Very Small Size - versão 2010.A

Danilo Tadashi Doi
danilodoi@grad.icmc.usp.br

Marcelo Oliveira da Silva
msilva@icmc.usp.br

Raphael Montanari
raphaelmon@grad.icmc.usp.br

Willian Constâncio da Silva
williancs@grad.icmc.usp.br

Roseli Aparecida Francelin Romero
rafrance@icmc.usp.br

Resumo

Neste texto será apresentado o simulador USPDroidsSS (*U*SPDroids Soccer Simulator), sistema que fornece um ambiente de simulação para robôs capazes de disputar partidas de futebol robótico seguindo as diretrizes da categoria *IEEE Very Small Size*.

O simulador USPDroidsSS permite acelerar o desenvolvimento de módulos de estratégia e que estes possam ser transferidos para um sistema real sofrendo poucas alterações.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO

1

Introdução

A robótica é um dos campos de pesquisa que tem alcançado grandes desenvolvimentos nos últimos anos. Inicialmente, os robôs foram utilizados para a automação de processos de produção industrial. Esse robôs por executarem tarefas bem definidas e repetitivas não necessitavam lidar com situações imprevistas. Então, era possível programar todos as possíveis ações do agente robótico para que ele as executasse de forma satisfatória.

Porém, com o desenvolvimento tecnológico, os robôs começaram também ser utilizados para outros propósitos tais como: entretenimento (jogos, brinquedos), medicina (cirurgia), e realização de tarefas em ambientes perigosos ou de difícil acesso a humanos (espaciais, subaquáticos). Nesse contexto, surgiram os robôs móveis autônomos com a proposta de realizar uma variedade de tarefas mais complexas que seus antecessores, os robôs industriais.

A principal limitação na utilização de robôs móveis está em como controlá-los, ou seja, como criar programas capazes de operar estas máquinas complexas. A complexidade cresce, ainda mais, quando considera-se não apenas um robô, mas uma equipe de robôs para realizar uma tarefa. Para tanto são necessárias técnicas que lhes permitam interagir de forma efetiva com o ambiente. A parte essencial desta interação é o módulo de planejamento de caminhos que cada robô utiliza para decidir suas ações e encontrar seu caminho em um ambiente.

(Mackworth, 1993) sugeriu que o ambiente de futebol de robôs fosse adequado para se avaliar o desempenho de técnicas de robótica móvel autônoma e áreas próximas, como, por exemplo, Visão Computacional, Inteligência Artificial, Controle, Eletrônica e Mecânica. (Silva et al., 2009; Costa e Pegoraro, 2000; Kitano et al., 1995, 1997)

Através da adoção deste problema padrão -futebol de robôs- pode ser feita a avaliação de várias teorias, algoritmos, desempenhos e arquiteturas. Uma das vantagens oferecidas pela adoção de um problema padrão é que a avaliação do progresso de uma pesquisa pode ser claramente definido e acompanhado.

O domínio de futebol de robôs é dinâmico e imprevisível, resultando num domínio bastante complexo, que exige o uso de sistemas com alto grau de autonomia, atuando em *malha fechada*[1] e em tempo real. Diversos tópicos específicos de pesquisa são oferecidos por este domínio, incluindo, entre outros:

- Completa integração entre percepção, ação e cognição num time de múltiplos agentes robóticos;
- Definição de um conjunto de comportamentos reativos robustos para cada agente, isto é, cada agente deve ser capaz de realizar tanto ações individuais quanto colaborativas;
- Percepção em tempo real, robusta e confiável, incluindo rastreamento de múltiplos objetos em movimento.

Para o desenvolvimento de uma estratégia para um time de futebol de robôs, o uso de um ambiente de testes pode ser vantajoso. Então, a utilização de um simulador de partidas de futebol de robôs para testar a eficácia e eficiência das estratégias antes de colocá-las em prática em um ambiente real, pode gerar economias em vários aspectos. Dentre estes:

- Não há desgaste de partes mecânicas e eletrônicas dos robôs, inclusive decorrentes do uso normal, pois é comum no ambiente de futebol de robôs haver colisões entre os agentes robóticos;
- Não há consumo de baterias, seja na carga armazenada ou na vida útil;
- É "instantâneo" reposicionar os robôs, caso necessário.

Além disto, não é necessário ter seis robôs funcionando pois pode-se simular o time adversário.

Uma simulação computacional consiste em empregar conceitos matemático-físicos em computadores com o propósito de representar aspectos do mundo real. O simulador USPDroidsSS é constituído de um ambiente virtual que simula uma partida de Futebol de Robôs e a apresenta ao usuário de forma interativa.

Existem simuladores de partidas de futebol de robôs como, por exemplo, o simulador FIRA (FIRA, 2008), porém esses simuladores não oferecem o suporte necessário ao processo de teste pois foram desenvolvidos para equipes

[1] o ambiente fornece feedback dos atos do agente robótico

de outras categorias e não para a categoria Very Small Size. Como solução, a equipe USPDroids criou um simulador de código aberto compatível com os padrões IEEE da categoria Very Small Size. Ainda pode ser ressaltado que o simulador FIRA não oferece um bom ambiente para treino de algoritmos inteligentes pois, por exemplo, não permite processamento em lote e/ou em background.

Entre outras vantagens de se utilizar o simulador USPDroidsSS , ressalta-se a maior facilidade na realização de análises estatísticas.

No Capítulo 2 será feita uma breve revisão de futebol de robôs e das regras da categoria *IEEE Very Small Size*. No Capítulo 3 são apresentadas as bibliotecas utilizadas no desenvolvimento do simulador. No Capítulo 4 é apresentada a estrutura geral do simulador. No Capítulo 5 é apresentado o núcleo do simulador. No Capítulo 6 é apresentada a interface gráfica do simulador. No Capítulo 7 é apresentado o protocolo de comunicação com estratégias e detalhes internos de como funciona o gerenciamento de estratégias. No Capítulo 8 alguns resultados preliminares são apresentados. No Capítulo 9 são apresentados os trabalhos futuros e uma breve conclusão. No Apêndice A é mostrado como utilizar o simulador. E, por fim, no Apêndice B são apresentadas as estratégias básicas, que podem ser utilizadas para o desenvolvimento de estratégias mais elaboradas.

CAPÍTULO

2

Futebol de Robôs

Para desenvolver um time de robôs móveis[1] autônomos[2] capazes de disputar partidas de futebol robótico, é necessário uma abordagem multidisciplinar, combinando conhecimento de diversas áreas, como mecânica, eletrônica, controle, visão computacional, fusão de sensores em tempo real, comportamento reativo, aprendizado, planejamento em tempo real, sistemas multi-agentes, decisão estratégica e outras. No começo da década de 90, (Mackworth, 1993) apresenta o ambiente de futebol de robôs como um *benchmark* para a robótica, e desde então, diversos pesquisadores tem utilizado este ambiente como teste para suas pesquisas em robótica (Thomas e Stonier, 2002; Pereira et al., 2004). Alguns anos depois, a FIRA[3] e a RoboCup[4] surgiram como duas iniciativas de pesquisadores coreanos e japoneses, respectivamente, neste sentido.

Em 95, foi fundado um comitê organizado internacional no *Korea Advanced Institute for Science and Technology* (KAIST) para tratar do "*Micro-Robot World Cup Soccer Tournament*" (MiroSot). Em 96, em um encontro de verão, foi realizada uma prévia da MiroSot, entre os dias 29 de julho e 4 de agosto, contando com a participação de 30 equipes de 13 países. A primeira MiroSot viria a ser realizada no mesmo ano, entre os dias 9 e 12 de Novembro, que contou com a participação de 23 times de 10 países. O time *Newton*, do Newton Research Laboratories (sediado em Seattle, EUA), e o time *SOTY*, do KAIST, foram campeão e vice-campeão mundial, respectivamente. Inicialmente, a FIRA focava

[1] capazes de se deslocar pelo ambiente
[2] durante sua operação, não existe um operador humano controlando o robô
[3] `www.fira.net`
[4] `www.robocup.org`

apenas no MiroSot. Atualmente, ela conta com várias categorias, algumas são simuladas, isto é, simuladores de robôs são utilizados.

Paralelamente, em 93 foi iniciado um projeto intitulado *Robot J-League*[5], objetivando a criação de uma competição de robótica. Entretanto, diversos pesquisadores de várias partes do mundo, requisitaram que a iniciativa fosse estendida para um projeto internacional. Desta forma, o projeto foi rebatizado como *Robo World Cup Initiative* (RoboCup). Durante o *Joint International Conference on Artificial Intelligence* (IJCAI-95), realizada em Montreal, Canadá, em agosto de 1995, foi divulgado que a primeira RoboCup ocorreria em conjunto com o IJCAI-97, em Nagoya. Ao mesmo tempo, foi decidido organizar a *Pré-RoboCup-96*, a fim de identificar potenciais problemas associados com a organização da RoboCup em larga escala. A pré-RoboCup-96 foi realizada durante a *International Conference on Intelligent Robots and Systems* (IROS-96), Osaka, de 4 a 8 de novembro de 1996, contando com com oito equipes. Embora em escala limitada, esta é a primeira competição de jogos de futebol de robôs. Os primeiros jogos oficiais da RoboCup foram realizados em 1997 com grande sucesso, onde participaram mais de 40 equipes. Em (Kitano et al., 1995, 1997), pode ser visto mais detalhes históricos desta iniciativa.

Ambas competições fornecem uma tarefa padrão para a pesquisa em robótica móvel autônoma em tempo real, no qual os agentes necessitam colaborar para resolver problemas dinâmicos. O objetivo final delas é que em futuro próximo seja possível partidas de futebol entre robôs e humanos. Como diz (Kitano et al., 1997), problemas-padrão em Inteligência Artificial (IA) são a força motriz da pesquisa em IA. Um destes problemas-padrão mais conhecidos é o xadrez.

Ainda em (Kitano et al., 1997), o futebol de robôs é apresentado como um ambiente multiagente. Durante um jogo, existem dois times competindo, com o objetivo de vencer o adversário. O time oponente pode ser visto como obstáculo dinâmico do ambiente, que irão atrapalhar a realização da meta. Para atingir a meta (ganhar o jogo), cada time necessita marcar gols, que pode ser visto como uma sub-meta. Para atingir esta meta, cada agente precisa ser rápido e ter um comportamento flexível e cooperativo, levando em conta situações globais e locais. Um time também pode ser visto como esquema de planejamento cooperativo e distribuído em tempo real.

2.1 Especificação da categoria IEEE Very Small-Size

A categoria *IEEE Very Small Size* é a única categoria em nível nacional que permite o uso de micro-robôs móveis. Ela é equivalente a MiroSot, diferenciando-

[5] *J-League* é o nome da liga profissional de futebol japonês

se em dois aspectos principais: tamanho do campo e quantidade de robôs por time.

A categoria de futebol de robôs IEEE Very Small-Size é apresentada em (VerySmall, 2008) e suas regras definem o campo, a bola, os robôs, o ambiente onde deve ser realizado o jogo, como o sistema deverá se estruturar, as formas de pontuação e as possíveis faltas e suas penalidades. Pode-se ver a equipe USPDroids na Figura 2.1.

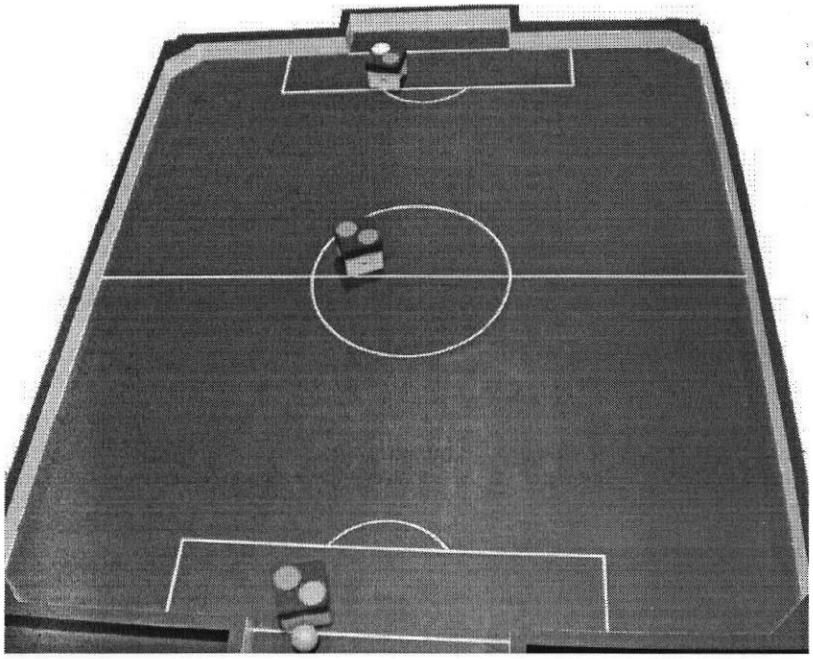

Figura 2.1: Time de Futebol de Robôs USPDroids

2.1.1 Estrutura do Sistema

Um único computador pode ser utilizado para cada time. Ele fica responsável pelo sistema de visão e, possivelmente, o de estratégia. As informações devem ser enviadas de forma sem fio para os robôs. Cada time deve ter três robôs.

Para identificar os robôs pode-se utilizar um sistema de visão. O sensor utilizado, p. ex. uma câmera, deverá estar sobre o próprio campo, podendo estar sobre o círculo central, de forma que não precise ser movido quando o time mudar de lado. O sensor deve estar a uma altura de, no mínimo, 2 metros.

O sistema deve ser controlado autonomamente, i.e. não se pode controlar os robôs manualmente durante a partida. Cada time tem direito a dois intervalos de dois minutos, devendo solicitá-los ao juiz.

2.1.2 Campo

O campo é feito de uma chapa de madeira com $1500mm$ x $1300mm$, na cor preta, fosca. Suas laterais devem medir $50mm$ de altura com $25mm$ de espessura, com a parte superior preta e as laterais brancas. Nos cantos do campo deve haver triângulos isósceles de $70mm$ x $70mm$, com cores definidas da mesma forma que as laterais. Haverá dois gols de $100mm$ x $400mm$, posicionados cada um no centro de cada lado menor ($1300mm$) do campo. A textura do campo deve ser equivalente ao de uma mesa de ping-pong comum.

O campo deverá possuir algumas marcações, conforme pode ser visto na figura 2.2. Todas as marcações deverão ser em branco com $3mm$ de espessura, exceto se especificado de outra forma, e são as seguintes:

- Um círculo central de $200mm$ de raio;
- Uma linha central dividindo o campo em dois retângulos de $750mm$ x $1300mm$;
- Um retângulo indicando a área de cada gol com $150mm$ x $700mm$, posicionados na frente de cada gol;
- Um arco no lado mais próximo ao centro do campo de cada área de gol, com raio de $100mm$ e $50mm$ no eixo perpendicular ao gol;
- Quatro marcações de bola livre em forma de cruz, posicionadas a $375mm$ x $250mm$ de cada canto do campo. Estas marcações devem ser na cor cinza;
- Duas marcações de pênalti/tiro livre em forma de cruz, posicionadas a $375mm$ de cada gol, no centro do lado menor do campo (a $650mm$ de cada parede de $1500mm$);
- Dois círculos indicando a posição que os robôs devem ser colocados em situação de bola livre, posicionados simetricamente a $200mm$ de cada marcação de bola livre.

2.1.3 Robôs e Bola

Os robôs (Figura 2.3) devem ser de, no máximo, $75mm$ x $75mm$ x $75mm$, desconsiderando antenas de rádiofreqüência. A parte superior dos robôs não deve possuir as cores laranja, branca ou cinza e não deve possuir mais de duas cores diferentes de preto.

Cada time deve possuir uma região com a cor primária (azul ou amarelo) que identifica o time, na parte superior do robô. Essa região deve conter

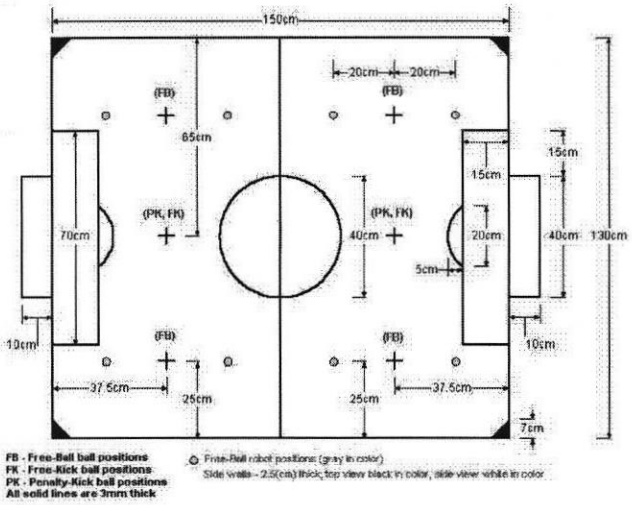

Figura 2.2: O campo. (VerySmall, 2008)

um quadrado de $35mm$ de lado ou um círculo de $40mm$ de diâmetro. Pode-se utilizar uma cor secundária para a identificação de cada robô, porém esta não deve ser igual à cor primária do time adversário.

Um robô dentro da área de gol do próprio time será considerado goleiro. O goleiro pode ser trocado dinamicamente durante a partida, dentre os robôs que estiverem no campo. Nenhum robô pode encobrir a bola, na vista superior, em mais de 30%, excetuando-se o goleiro.

A bola deve ser uma bola de golfe laranja, com $42,7mm$ de diâmetro e $46g$ de massa.

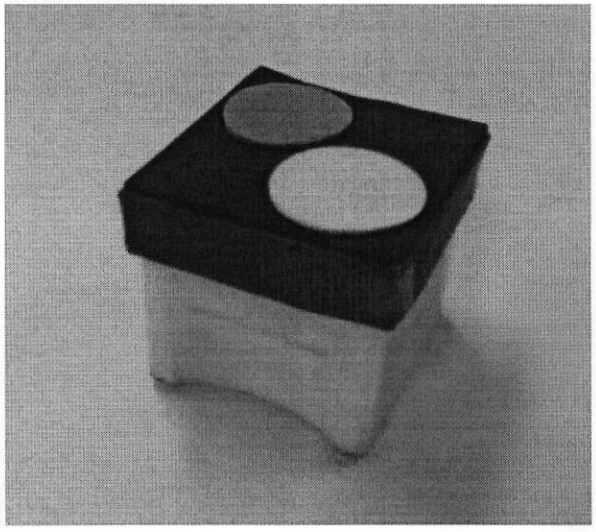

Figura 2.3: Robô construído no LAR-ICMC-USP

2.1.4 Jogo

Cada jogo terá uma duração de dez minutos, divididos em dois tempos de cinco minutos. Os times trocam de lado ao final do primeiro tempo. A posse de bola ao início da partida será definida por sorteio.

O time que estiver com a posse para a saída de bola pode posicionar seus robôs em qualquer posição dentro de seu lado do campo. O time que não estiver com a posse de bola pode posicionar seus robôs em qualquer posição de seu lado do campo, exceto dentro do círculo central. Após a ocorrência de um gol executa-se novamente a saída de bola com posse para o time que sofreu o gol.

Computar-se-á gol quando a bola ultrapassar inteiramente a linha de um dos gols, sendo o gol computado ao time contrário àquele que teve a bola entrando em seu gol. Declara-se vencedor o time que marcar mais gols.

Existem quatro situações de jogo excepcionais que são Bola Livre, Tiro Livre, Tiro de Meta e Pênalti.

- **Bola Livre**

  A bola é posicionada na posição de bola livre apropriada. Os robôs que estiverem a menos de $200mm$ do ponto de bola livre deverão ser movidos em direção ao próprio lado do campo. Um robô de cada time é posicionado no local apropriado, no ponto mais próximo ao seu campo.

- **Tiro Livre**

  A bola é posicionada na posição de tiro livre apropriada. O robô do time que irá cobrar o tiro livre pode ser posicionado de frente para a bola e os outros deverão ser posicionados no lado do campo do próprio time. Os robôs defensores devem ser posicionados em contato com a linha da área do gol, e o goleiro deve permanecer dentro da área do gol.

- **Tiro de Meta**

  A bola é posicionada em qualquer ponto da área de gol de onde será cobrado o tiro de meta. Somente o goleiro pode permanecer na área do gol.

- **Pênalti**

  A bola é posicionada na posição de pênalti apropriada. O robô do time que irá cobrar o tiro livre deve ser posicionado de frente para a bola. O goleiro adversário deve ter um dos lados encostado na linha do gol. Todos outros robôs deverão ser posicionados no lado do campo contrário de onde será cobrado o pênalti.

Todas estas situações terminam quando o juiz apita, permitindo aos robôs moverem-se livremente.

2.1.5 Faltas

Define-se falta quando ocorrer uma das seguintes situações:

- Um robô atingir um robô adversário de forma que influencie o andamento da partida, ou seja, potencialmente danoso ao robô adversário. Penalidade: Tiro livre para o time adversário.
- Um robô empurrar o goleiro adversário com a bola em direção ao gol ou empurrar o goleiro diretamente. Penalidade: Tiro de meta para o time adversário.
- Um time estiver com mais de um robô mais de 50% dentro da área adversária. Penalidade: Tiro de meta para o time adversário.
- Um time estiver com mais de um robô dentro da área do próprio gol e a bola também estiver dentro de sua área de gol. Penalidade: Pênalti para o time adversário.
- Se um robô diferente do goleiro prender a bola em si mesmo de forma que nenhum outro robô seja capaz de movê-la. Penalidade: Tiro livre para o adversário, se a infração ocorrer fora da área de gol, ou Pênalti, caso contrário.
- Se a bola permanecer mais de 10 segundos dentro da área de gol, com o goleiro em sua posse, sem disputá-la. Penalidade: Pênalti para o time adversário.
- Bloquear o goleiro adversário dentro de sua respectiva área. Penalidade: Tiro livre para o time adversário.

CAPÍTULO

3

Ambientes de Desenvolvimento

No processo de desenvolvimento do simulador USPDroidsSS algumas bibliotecas foram utilizadas, objetivando clareza de código, portabilidade e menor tempo de implementação. As bibliotecas utilizadas foram:

- Qt para comunicação, interface gráfica e gráficos 3D;
- ODE para a criação de modelos robóticos e o tratamento de suas forças físicas.

A seguir, uma breve descrição destas bibliotecas será dada.

3.1 Qt

Para um simulador, é conveniente que haja alguma forma de interação com o usuário, para que seja possível alguma interferência na simulação ou um ajuste de alguma configuração do simulador, assim como uma visualização do jogo em tempo real para avaliação de comportamentos específicos da estratégia. Além disso, o simulador precisa ter alguma maneira de se comunicar com as estratégias. A framework Qt oferece ferramentas que facilitam o desenvolvimento de tais propriedades. A Qt(Nokia Corporation, 2010) é uma framework multiplataforma de código aberto dentro da licença GPL criada pela empresa Trolltech (e atualmente pertencente à Nokia), que oferece diversos módulos e bibliotecas de classe para C++. Códigos escritos dentro deste framework possuem grande portabilidade: não é necessária a adaptação para o uso em sistemas operacionais distintos. Dentre os módulos fornecidos pelo Qt, o USPDroidsSS utiliza quatro: QtCore, QtGui, QtNetwork e QtOpenGL. A versão do Qt utilizada para a criação do USPDroidsSS foi a 4.5.

3.1.1 QtCore

O QtCore contém as funcionalidades centrais que são relacionadas à interface gráfica, sendo essencial para todos os outros módulos. Dentro dele, existem também as classes responsáveis por tratamento de threads.

3.1.2 QtGui

O QtGui é responsável pela interface gráfica, que permite a interação do usuário com o simulador, oferecendo a geração de objetos de interação como botões e barra de menu, detecção e manipulação de eventos de interação (como eventos de mouse ou teclado), gerenciamento de exibição de texto, entre outras funcionalidades.

3.1.3 QtNetwork

O QtNetwork oferece classes que lidam com propriedades de conexão de rede cliente-servidor: através dela, ocorre o gerenciamento da conexão através de sockets das estratégias e do simulador. No simulador, é utilizada a classe QUdpSocket para tais tarefas porque o protocolo de conexão escolhido foi o UDP.

3.1.4 QtOpenGL

Este módulo possui classes que facilitam o uso da API OpenGL, que renderiza gráficos em 3D, em aplicações Qt. Há um widget neste módulo que facilita a integração entre a interface gráfica e o OpenGL, abrindo um buffer de visualização para a API dentro dessa interface gráfica. Tal módulo é utilizado para a exibição da partida no simulador.

3.2 ODE

A simulação física é a parte mais importante de um simulador de futebol, por ser responsável pela dinâmica dos corpos rígidos e pela interação entre os robôs e as demais entidades do ambiente. Simular a física ajuda a fazer os objetos virtuais se comportarem da maneira que esperamos no mundo real.

Engines físicas são programas que executam estas simulações. O desenvolvimento de uma engine física, porém, é bem complexo. Existem bibliotecas de código aberto e de livre distribuição disponíveis na Internet que podem reduzir significativamente o custo e o tempo de produção do simulador. Uma das mais conhecidas, e a escolhida para o USPDroidsSS , é a Open Dynamics Engine (Smith, 2010), amplamente utilizada em vários jogos profissionais e

aplicações científicas. Existem muitos simuladores, como (Browning e Tryzelaar, 2003), (Tikhanoff et al., 2008), (Project, 2010), e (Ltd., 2010), que utilizam essa biblioteca para a simulação de corpos rígidos.

As vantagens da ODE são sua boa documentação, excelente tratamento de colisões com objetos muito rápidos, facilidade de uso, e grande quantidade de recursos disponíveis, como colisões com modelos.

A ODE modela os corpos rígidos a partir de primitivas e juntas. Por padrão existem as seguintes primitivas: Esferas, Caixas, Planos, Cilindros, Raios e Malhas de triângulos. Ligando as primitivas através de juntas podem-se obter elementos mais complexos, permitindo a modelagem de objetos como robôs com rodas.

Pode-se criar mundos de simulação independentes, com espaços, que são grupos de objetos, que interagem entre si. O conceito de espaços torna mais rápido a detecção da existência de colisões, pois não há necessidade de verificar colisão entre objetos de espaços que não se estão se interceptando. Os mundos de simulação possuem parâmetros como gravidade e redução de erro padrão para as juntas.

Uma vantagem da ODE é ela não impor qualquer restrição de unidade de medida para os números utilizados, ou seja, a posição (1,0,0) pode significar tanto 1km como 1mm na direção x. Isso permite a modelagem de sistemas "grandes", como a física de um avião, ou de sistemas "pequenos", como a simulação do movimento de uma formiga. Uma pequena dificuldade inerente a essa liberdade é que é necessário manter-se coerente com as unidades para obter simulações satisfatórias. Isso pode se tornar complexo quando se utilizam unidades não primitivas, como força ou torque, portanto é recomendado que se utilize as unidades do Sistema Internacional (SI), tal qual neste trabalho.

A modelagem matemática do sistema físico (os objetos, as juntas e forças entre eles) é feita através de um sistema de Equações Diferenciais Ordinárias. Para obter o resultado da simulação é necessário resolver-se numericamente o sistema. Para isso, a ODE utiliza um integrador de primeira ordem "semi-implícito"(as forças geradas pelas juntas/restrições são calculadas por um integrador implícito, e as forças aplicadas pelo usuário por um integrador explícito). Os integradores de primeira ordem podem não ser extremamente rápidos e/ou precisos, mas a simulação é feita de forma rápida o bastante e a utilização de pequenos passos de tempo fornece bons resultados.

CAPÍTULO

4

Simulador USPDroidsSS

O Simulador USPDroidsSS foi projetado para simular o ambiente de uma partida da categoria de futebol de robôs *IEEE Very Small Size*. (Silva et al., 2009) Esta é a única categoria em nível nacional que permite o uso de micro robôs móveis e é equivalente à *MiroSot*, porém, com duas diferenças: o tamanho do campo e o número de robôs por time. Dessa forma, o ambiente (Figura 4.1) desta categoria possui os seguintes módulos:

- Visão: responsável pela localização dos robôs e classificação destes, diferenciando entre robôs do mesmo time e adversários;
- Estratégia: responsável pelo sistema autônomo de tomada de decisão;
- Eletrônica: responsável pelo processamento de informações recebidas do rádio e controle dos motores;
- Mecânica: responsável pela parte física do robô (carcaça, rodas, engrenagens, etc).

O Simulador USPDroidsSS deve simular os seguintes módulos: visão, eletrônica e mecânica (Figura 4.2). Ou seja, o USPDroidsSS irá substituir estes módulos, e dessa forma, será mais fácil de criar, testar, avaliar e melhorar as estratégias; os robôs reais não sofrerão desgastes; e evitaremos gastos desnecessários com bateria. Outra vantagem é que o desenvolvimento da estratégia não fica dependente dos outros módulos, e poderá ser desenvolvido até mesmo se o robô real não estiver pronto. Sendo assim, o USPDroidsSS deve se comunicar com uma estratégia e lidar com as informações recebidas como se fosse o ambiente real.

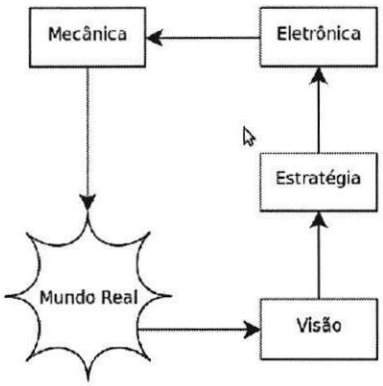

Figura 4.1: Estrutura real do time USPDroids

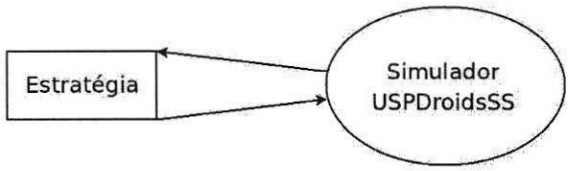

Figura 4.2: Estrutura do Simulador USPDroidsSS

O resto deste capítulo irá explicar a estrutura geral do simulador, suas funcionalidades e a arquitetura utilizada.

4.1 Características e Funcionalidades

Para suprir todas as necessidades impostas por uma partida e tornar-se uma ferramenta de fácil utilização, o USPDroidsSS possui as seguintes características:

- Uma Interface Gráfica, que disponibiliza todos os recursos disponíveis e exibe o nome de cada estratégia, o placar da partida, o tempo decorrido e uma lista dos eventos que ocorreram na utilização do simulador. Também exibirá a partida em tempo real, em um ambiente 3D, permitindo que o usuário acompanhe a partida e, a partir disso, possa avaliar as estratégias, verificando o comportamento destas. O visualizador 3D também oferece a opção de ativação, rotação e zoom do campo, permitindo que o usuário possa escolher se deseja, ou não, assistir a partida e caso deseje, que assista a partida com o ângulo e aproximação desejados;

- Os objetos (robôs, bola, campo) são modelados para manter as proporções do ambiente real. É realizado o tratamento de colisões entre eles.

- Todos os robôs são posicionados em posições pré-determinadas, porém, a cada reposicionamento realizado, estas posições irão variar um pouco.

Isso fará com que ocorram mais variações dentro de uma partida e entre partidas diferentes.

- Um sistema de comunicação, no qual o simulador recebe os dados da estratégia e valida esta mensagem, assim como também envia dados, checando o que deve ser enviado a cada estratégia. Vale destacar que o que será enviado a uma estratégia é diferente da mensagem que será enviada a outra.

- É possível mudar as portas utilizadas pelo USPDroidsSS para se comunicar com as estratégias. Sendo assim, existem quatro portas padrões, porém, fica a cargo do desenvolvedor da estratégia escolher quais portas utilizar.

- Após as estratégias estabelecerem sua conexão com o simulador, o USPDroidsSS validará a possibilidade de início da partida (enquanto as duas estratégias não estiverem conectadas, não será permitido iniciar a partida).

- Há a possibilidade de pausar uma partida, desconectar uma estratégia, e em seguida conectar outra, sem finalizar a partida, ou seja, o usuário pode testar estratégias diferentes em uma mesma partida. Além disso, é possível iniciar uma nova partida sem a necessidade de fechar o simulador, ou seja, o usuário pode parar a partida a qualquer momento e começar um novo jogo, ou após o jogo terminar, conectar novas estratégias e iniciar a nova partida.

4.2 Arquitetura do USPDroidsSS

Para que o simulador se comporte de forma próxima ao ambiente real, seja capaz de oferecer todas as funcionalidades citadas acima, e se torne fácil de atualizar em versões futuras, o USPDroidsSS foi dividido em três módulos: Unidade de Interface Gráfica, Kernel e Comunicação.

A arquitetura mostrada na Figura 4.3 foi desenvolvida para implementar as funções e módulos descritos acima. Nesta figura, cada módulo é representado por um conjunto de classes. A Unidade de Interface Gráfica é gerenciada pelo Visualizador, o Kernel pelo GerenciadorJogo, e a comunicação entre o simulador e as estratégias pela Comunicação. Com essa estrutura, cada módulo do simulador é desenvolvido de forma separada dos outros módulos, e assim, cada um destes irá se preocupar apenas com as suas próprias funcionalidades. Porém, logicamente é necessário que exista comunicação entre estes módulos para que o simulador funcione adequadamente. Os próximos capítulos visam explicar detalhadamente cada um destes módulos.

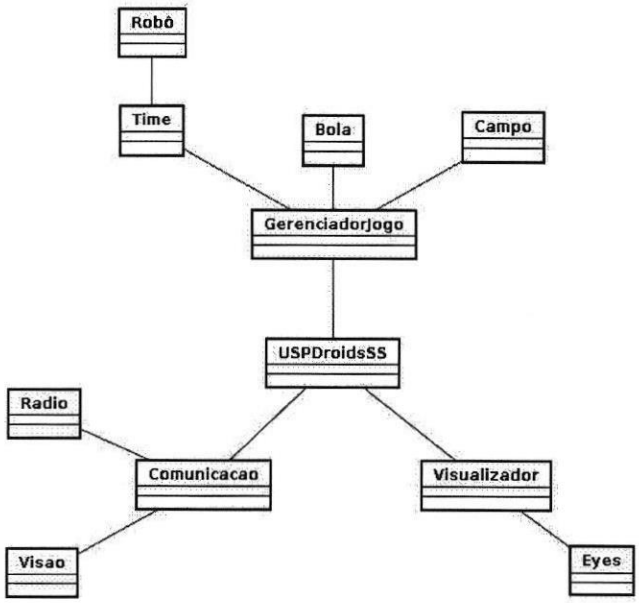

Figura 4.3: Estrutura de software do Simulador USPDroidsSS

CAPÍTULO

5

Ambiente de Simulação

5.1 Kernel

O Kernel é o núcleo do simulador (a parte responsável por fazer a simulação física). Ele possui os modelos dos objetos simulados, aplica as entradas fornecidas e retorna o novo estado do sistema simulado. O Kernel é gerenciado pelo GerenciadorJogo.

O Kernel é composto pelas classes Campo, Bola e Time, sendo que esta última contem a classe Robô. Campo (ver 5.1.1) é responsável pelo elementos de simulação dos objetos estáticos, i.e., o campo. Bola (ver 5.1.2) é, como o próprio nome diz, responsável pelo modelo de simulação da bola. Time possui 3 objetos da classe Robô (ver 5.1.3), que modelam cada robô.

O GerenciadorJogo é responsável por chamar a simulação de corpos rígidos, realizada pela ODE, sempre que o simulador necessitar de novos dados, atribuindo as velocidades necessárias para os robôs e obtendo as novas posições. Esse comportamento é descrito em 5.1.4.

5.1.1 Construindo o ambiente de simulação

O primeiro passo é a criação de um mundo dinâmico da ODE, onde os objetos vão interagir. No início da partida, a única força existente é a da gravidade. Após a configuração dos parâmetros necessários, é possível anexar os objetos ao mundo. Um pequeno algoritmo pode ser visto em Algoritmo 1.

Basicamente, o campo de futebol é um conjunto de caixas interconectadas. O campo é um objeto estático (ou seja, tem posição fixa), então colisões não são capazes de movê-lo, como se sua massa fosse "infinita". O modelo do

campo segue a descrição das regras oficiais da categoria IEEE Very Small, que pode ser visto na Seção 2.1.2.

5.1.2 Bola

O objeto mais simples é a bola; seu corpo rígido é modelado como uma esfera simples de $21.35mm$ de raio e $46gramas$ uniformemente distribuídos na esfera. Sua posição inicial é o centro do campo; ela está parada no início da partida e possui um parâmetro de atrito associado.

5.1.3 Robôs

Os robôs (Figura 5.1) são um pouco mais complexos, pois são compostos de três corpos rígidos simples: um cubo de $7.0cm$ localizado no centro do robô, e as rodas, que são cilindros com raio de $2.5cm$ e profundidade de $0.25cm$, que ficam a uma distância de $7.0cm$ entre si, e de $3.5cm$ em relação ao centro do robô. Para conectar estes três corpos rígidos foram utilizados juntas, que também são utilizadas para representar os motores do robô. Elas podem receber uma velocidade desejada e torque, modelando de maneira fiel um motor real, como podemos ver na Seção 8.1.2.

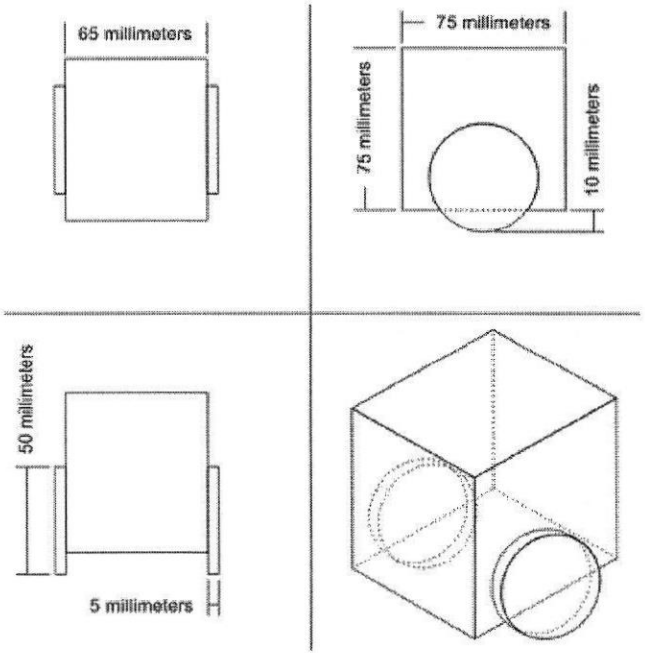

Figura 5.1: Design técnico do modelo do robô

No mundo real, nem sempre a velocidade que é enviada para um robô é a velocidade que ele irá manter, devido a diversos fatores, como atrito, derrapagens, baixa carga na bateria, imperfeições no campo ou no sistema de controle. Todos esses ruídos exercem grande influência no comportamento de

um robô. Atualmente não é modelado nenhum tipo de ruído no simulador, ou seja, a velocidade atribuída a um robô é exatamente a que ele irá manter. Isso é vantajoso pois permite o teste apenas das estratégias, sem considerar os erros induzidos pelo ruído. É também interessante o teste com ruídos para analisar o comportamento de uma estratégia sob a influência deste, e tal aspecto será modelado futuramente no simulador.

5.1.4 A Simulação

Após adicionar todos os objetos ao mundo dinâmico da ODE e configurar os parâmetros do sistema, podemos simular as interações entre eles, feitas através de passos de simulação. Inicialmente, procuramos por colisões. Caso ocorram, são geradas as forças de colisão, incluindo atrito e reação. Em seguida, resolvemos o sistema (modelado por Equações Diferenciais Ordinárias) por um passo. Este passo é o tempo de simulação, de valor padrão de $16ms$, o equivalente a $60Hz$. Após isso, teremos as novas posições dos objetos considerando que se passou o tempo do passo de simulação, neste caso após $16ms$. Isso se repete até que o tempo decorrido de simulação seja 10 minutos, que é o tempo de uma partida.

Algoritmo 1 Simulação

Requer obstáculos e posições da meta

Garante uma função potencial na qual, a partir de qualquer ponto que não esteja nas fronteiras do campo, se a gradiente descendente é seguida, a meta será alcançada.

1: Cria um ambiente dinâmico.
2: Cria corpos no ambiente dinâmico.
3: Configura o estado (posição, ângulo) de cada objeto no ambiente.
4: Cria juntas no ambiente dinâmico.
5: Anexa as juntas aos corpos.
6: Configura os parâmetros das juntas.
7: Cria um ambiente de colisão e uma geometria de colisão (quando aplicável).
8: Cria um grupo de juntas para unir as juntas de contato.
9: **repita**
10: Aplica forças aos corpos (quando necessário).
11: Ajusta os parâmetros das juntas (quando necessário).
12: Chama a detecção de colisões.
13: Cria uma junta de contato para cada ponto de colisão, e a coloca no grupo de juntas de contato.
14: Termina um passo de execução.
15: Remove todas as juntas do grupo de juntas de contato.
16: **até** "10 minutos de simulação tenham passado"
17: Destrói a dinâmica e os ambientes de colisão.

CAPÍTULO

6

Visualização

6.1 Interface Gráfica

O simulador prove uma interface para construir o ambiente, que chamaremos de GUI (Graphic User Interface, ou Interface Gráfica do Usuário). O módulo da GUI gera um ambiente que facilita a utilização dos recursos do simulador, além de permitir que o usuário possa assistir ao jogo simulado (Figura 6.1). O visualizador 3D possui apenas um tipo de câmera que proporciona a visão focada sobre o centro do campo, mas é possível rotacionar o ângulo de visão e ampliar ou reduzir o zoom. As operações são feitas usando o mouse ou atalhos do teclado (+ ou ? para zoom). São oferecidas ao usuário as seguintes funções:

- Inicia a partida (somente quando as estratégias de cada um dos dois times estão conectadas);
- Pausar o jogo (se a partida já está em andamento);
- Parar a partida (se ela já começou), desconectando as estratégias envolvidas e fazendo o simulador aguardar novas conexões de estratégias;
- Uma opção para realocar os robôs em suas posições iniciais;
- Desconectar uma das estratégias (desde que o jogo esteja pausado);
- Trocar as portas de comunicação padrões;
- Opção de tela cheia;

- Limpar a lista de eventos;
- Visualizar ou não a partida;
- Rotacionar ou zoom no ambiente 3D;

Figura 6.1: Simulador USPDroidsSS

Na Figura 4.3, podemos ver que este módulo também é composto pela classe Eyes: a classe Visualizador é responsável por gerar toda a interface gráfica e gerenciar as funções mencionadas acima; a classe Eyes responde pela geração da visualização 3D em tempo real usando a API OpenGL. Com essa divisão, torna-se fácil ativar ou não a visualização 3D e portanto economizar custos computacionais. A comunicação entre este módulo e a classe Simulator (Figura 4.3) é concluída pelo mecanismo de sinal/slot oferecido pela Qt (Nokia Corporation, 2010).

O visualizador 3D utiliza os recursos da QtOpenGL, via QGLWidget, para abrir um buffer de exibição onde é possível usar a API OpenGL para renderizar os gráficos 3D. O módulo de visualização recebe dados da classe Simulador e transmite para a classe Eyes desenhar a imagem na tela. Os dados recebidos pela classe Eyes são as posições cartesianas na dimensão R^3 e o ângulo de rotação em relação aos três planos dessa dimensão. Com este modelo, a responsabilidade do OpenGL é renderizar o ambiente virtual e exibir dentro da GUI. A taxa de atualização desta interface é de $60Hz$, a mesma com que o Kernel obtém novos dados de posicionamento.

CAPÍTULO

7

Comunicação

O Simulador USPDroidsSS precisa se comunicar com as estratégias em tempo real, ou seja, necessita de uma comunicação rápida e eficiente. Para esta comunicação não depender do computador que está rodando, a utilização de Sockets foi requerida. Além disso, utilizar Sockets traz várias outras vantagens, pois a linguagem para desenvolver as estratégias não fica pré-estabelecida, pois pode-se utilizar qualquer tipo de linguagem, desde que essa possua suporte para realizar uma comunicação via Socket. Outra vantagem é que se torna viável rodar o simulador e as estratégias em computadores distintos.

Os tópicos deste capítulo visam explicar detalhadamente como a comunicação é realizada, dissertando uma visão geral da comunicação, os módulos criados, o protocolo utilizado, as mensagens que são enviadas e recebidas do simulador, etc.

7.1 Protocolo de Comunicação

Para um simulador se comunicar com alguma estratégia em tempo real, é necessário que esta comunicação seja rápida e eficiente, porém confiável o suficiente em relação à entrega de pacotes. Para suprir essas características, o protocolo UDP (User Datagram Protocol) foi escolhido pois:

- é um protocolo adequado para o fluxo de dados em tempo real;
- se houver perda de pacotes entre o simulador e a estratégia, não afetará tanto o desempenho da partida, pois é mais importante garantir que não

ocorram grandes atrasos do que garantir que as mensagens não serão perdidas.

Todas estas características fazem com que este protocolo satisfaça as necessidades impostas por um simulador de tempo real. Além disso, a biblioteca Qt possui suporte para Sockets e para comunicação via protocolo UDP, ou seja, independente do Sistema Operacional utilizado, a comunicação será realizada sem precisar de nenhuma modificação.

7.2 Comunicação com as estratégias

A comunicação entre as estratégias e o simulador é feita através de quatro Sockets (Figura 7.1), todas utilizando o protocolo UDP. Cada par de Sockets se comunicará com uma estratégia, sendo que um Socket recebe os dados e o outro envia estes.

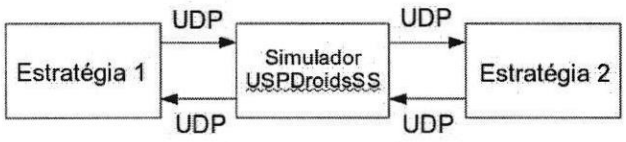

Figura 7.1: Comunicação entre o Simulador USPDroidsSS e as estratégias

Dessa forma, existem dois módulos de comunicação com as estratégias, que neste texto chamaremos de Comunicação via Rádio e Comunicação com a Visão. O primeiro destes é responsável receber as mensagens de uma estratégia, enquanto o outro é responsável por enviar. Esta divisão foi feita por causa da estrutura que a equipe USPDroids utiliza atualmente, pois cada estratégia envia mensagens para os robôs através de uma comunicação via Rádio e recebe as mensagens do módulo da Visão. Sendo assim, uma estratégia criada para rodar no simulador poderá ser executada da mesma forma no sistema real, sem haver nenhuma modificação de código.

Na Estrutura do Simulador (Figura 4.3), pode-se ver que estes módulos são dados por "Radio"e "Visao". Estes módulos são gerenciados pelo módulo "Comunicacao". Ou seja, este módulo é responsável por gerenciar a comunicação com as duas estratégias, tanto no recebimento como no envio de mensagens.

Para uma estratégia se comunicar com o simulador, é necessário haver um método para estabelecer esta conexão. No caso, utilizamos um "hand-shake", ou seja, a estratégia envia uma mensagem solicitando o estabelecimento de uma conexão; quando o simulador receber esta mensagem, vai enviar à estratégia que a solicitação foi efetuada; e assim, ambos identificaram com quem

irão conversar e ficam aguardando o início do jogo. Além disso, neste "handshake"também é enviado o nome da estratégia, sendo assim, é possível visualizar na interface gráfica do simulador (em cima do placar da estratégia) o nome do time. Após o estabelecimento das conexões, e o início do jogo, a estratégia e o simulador devem trocar mensagens entre si, utilizando mensagens que indicam a posição dos robôs no campo (Comunicação com a Visão) ou a velocidade de rodas dos robôs (Comunicação via Rádio).

Cada um dos sockets citados anteriormente utilizam uma porta diferente, pois como cada um destes executa uma função diferente, reservamos uma porta para cada função. Inicialmente o simulador utiliza o IP de localhost e as portas com valores entre 26000 e 26003, porém, pela interface gráfica, é possível alterar estes valores.

Com isso, o Simulador USPDroidsSS se preocupa com:

- o estabelecimento de conexões, passando também o nome da estratégia para ser exibido na GUI;
- as trocas de mensagens entre o simulador e as estratégias após o jogo ser iniciado;
- uma opção para pausar o jogo, ou seja, as estratégias continuam conectadas, porém, não trocam mensagens com o simulador. Isto evita um gasto desnecessário com processamento, memória e rede;
- uma opção para desconectar todas as estratégias e fazer com que o simulador fique aguardando novas estratégias se conectarem, ou seja, não é necessário fechar o simulador para iniciar outra partida com novas estratégias;
- e uma opção para desconectar uma estratégia específica, sem desconectar a outra, fazendo com que o simulador fique aguardando uma nova estratégia. Sendo assim, é possível trocar uma estratégia no meio de uma partida, sem precisar finalizar esta.

Para auxiliar os usuários do simulador, um log de todo o sistema de comunicação é gerado, armazenando cada mensagem que está sendo enviada e recebida. Este log também identifica de quem o simulador recebeu uma mensagem ou para quem o simulador enviou. As sessões seguintes explicam como funciona cada um dos módulos citados.

7.3 Comunicação com a Visão

Este módulo é responsável pelo estabelecimento de conexão com uma estratégia; o envio de mensagens para as estratégias contento as posições atuais

(x e y) da bola, dos robôs do time e dos robôs adversário, assim como o ângulo dos robôs do time; e também por receber o nome da estratégia que está se conectando, que será exibida em seguida na GUI.

Em relação ao estabelecimento de conexão, a Simulador espera a estratégia enviar a mensagem "STRING DE BOAS VINDAS". Enquanto esta mensagem não for recebida, o USPDroidsSS fica aguardando a estratégia, e após receber, valida esta comunicação para poder iniciar a partida.

Para receber o nome da estratégia, após a conexão ser estabelecida, a próxima mensagem que possuir o início com "Eu sou "será a identificação do nome da estratégia. Ou seja, uma estratégia deve enviar, por exemplo, "Eu sou USPDroids", para que o USPDroidsSS reconheça que o nome da estratégia é "USPDroids".

Em relação às mensagens que devem ser trocadas entre a Estratégia e o Simulador após o início da partida, este módulo monta as mensagens com as seguintes características:

- O primeiro caractere deve identificar o tipo de mensagem, esta identificação será feita através da letra 'O';
- Cada campo da mensagem será separado por uma vírgula;
- Os três primeiros campos após o caractere de validação serão a posição (x,y) da bola, e um caractere de validação das informações sobre a bola (que será '0' ou '1');
- Os próximos itens serão identificados pela posição (x,y) de um robô, seguido ângulo e consequentemente de um caractere de validação para estas informações (que também deverá conter '0' ou '1'). Os três primeiros robôs identificados serão dos robôs do time. Os próximos três serão dos robôs adversários, e sendo assim, o campo de ângulo será desconsiderado e não utilizaremos esta informação;
- O valor de 'x' está compreendido entre -75 e 75. Já 'y' está compreendido entre -65 e 65. Enquanto o ângulo varia entre 0 e 360. Todos estes valores são valores inteiros.

Dessa forma, somam-se ao todo vinte e oito campos separados por vírgulas (vinte e sete vírgulas). Sendo assim, um exemplo desta mensagem pode ser dado por: "O,0,0,1,-74,0,270,1,-18,-25,270,1,-25,19,253,1,75,0,0,1,10,-21,0,1,15,17,0,1", e esta mensagem diz que:

- Todas as informações estão válidas, pois temos a identificação de 'O' no primeiro caractere e de '1' nos caracteres para validar a bola e os robôs.

- A bola está em (x,y) = (0,0);
- Os robôs do time possuem os valores de:
  - Robô 1: (x,y,ângulo) = (-74,0,270);
  - Robô 2: (x,y,ângulo) = (-18,-25,270);
  - Robô 3: (x,y,ângulo) = (-25,19,253).
- Os robôs do time adversário possuem os valores de:
  - Robô 1: (x,y) = (75,0);
  - Robô 2: (x,y) = (10,-21);
  - Robô 3: (x,y) = (15,17).

7.4 Comunicação via Rádio

Este módulo é responsável pelo estabelecimento de conexão com uma estratégia e o recebimento das mensagens que indicam as velocidades de rodas de cada robô. Ou seja, ao receber uma mensagem, esta deverá ser quebrada e lida adequadamente, e em seguida, repassar para o Kernel executar a movimentação dos robôs.

Neste módulo, o estabelecimento de conexão é feito da mesma forma que o outro. Sendo assim, o USPDroidsSS espera a Estratégia enviar a mensagem "STRING DE BOAS VINDAS"para validar a conexão.

As mensagens que chegam da estratégia possuem a seguinte característica:

- Nesta mensagem, não existe caractere de identificação ou validação;
- Os dois primeiros valores se referem ao Robô 1, os dois próximo ao Robô 2, e consequentemente, o último par ao Robô 3;
- A ordem de cada par de mensagens é de primeiro a roda direita seguido da roda esquerda;
- A escala de cada valor deve estar entre 0 e 1, inclusive estes. Utilizamos estes valores visando 0 quando a roda precisar estar parada, e 1 quando a roda precisar estar com a velocidade máxima permitida;
- Separamos cada valor por um espaço em branco;
- Valores positivos e negativos se referem ao lado que a rotação da roda deve ser feita.

Sendo assim, temos ao todo 6 valores, separados por 5 espaços em branco. Um exemplo para esta mensagem poderia ser "0.056 -0.048 0 0 0.022 0.006", e esta mensagem possui as seguintes características:

- Robô 1: (velocidade da roda direita,velocidade da roda esquerda) = (0.056,-0.048);
- Robô 2: (velocidade da roda direita,velocidade da roda esquerda) = (0,0);
- Robô 3: (velocidade da roda direita,velocidade da roda esquerda) = (0.022, 0.006).

CAPÍTULO

8

Resultados e Discussões

Nesta capítulo discutiremos sobre os testes realizados para a validação do Simulador USPDroidsSS.

8.1 Testes Preliminares

Foram realizados dois tipos de teste no Simulador USPDroidsSS : sem viés e com verossimilhança do motor.

8.1.1 Sem Viés

O teste para mostrar que o simulador não é enviesado é importante para o teste de estratégias pois evita o favorecimento de uma das estratégias, o que resulta em testes comparativos não confiáveis. Para isso, colocamos uma estratégia enfrentando a si mesma e esperamos que o resultado fosse um empate. As estratégias utilizadas são baseadas em Campos Potenciais Localmente Orientados (Silva et al., 2009) e possuíam duas configurações: dois defensores e um atacante (Estratégias 1 e 2) ou um goleiro, um defensor e um atacante (Estratégias 3 e 4). Foram simulados 262 jogos, e os resultados podem ser vistos nas Tabelas 8.1 e 8.2.

Como podemos observar nas Tabelas 8.1 - 8.2, as estratégias obtiveram resultados similares . Portanto, temos indícios de que o simulador é não-enviesado.

Métrica	Valor
Jogos	159
Gols da Estratégia 1	58
Gols da Estratégia 2	64
Vitórias da Estratégia 1	37
Vitórias da Estratégia 2	47
Empates	75
Maior número de gols em um jogo da Estratégia 1	3
Maior número de gols em um jogo da Estratégia 2	2
Diferença de Gols da Estratégia 1	-6
Diferença de Gols da Estratégia 2	6
Proporção Gols Feitos / Gols Sofridos da Estratégia 1	0.90625
Proporção Gols Feitos / Gols Sofridos da Estratégia 2	1.10345
Maior diferença de gols a favor da Estratégia 1	3
Maior diferença de gols a favor da Estratégia 2	2
Diferença de gols média da Estratégia 1	-0.0375
Diferença de gols média da Estratégia 2	0.0375

Tabela 8.1: Resultados dos jogos Estratégia 1 x Estratégia 2.

Métrica	Valor
Jogos	103
Gols da Estratégia 3	46
Gols da Estratégia 4	48
Vitórias da Estratégia 3	26
Vitórias da Estratégia 4	28
Empates	49
Maior número de gols em um jogo da Estratégia 3	3
Maior número de gols em um jogo da Estratégia 4	3
Diferença de Gols da Estratégia 3	-2
Diferença de Gols da Estratégia 4	2
Proporção Gols Feitos / Gols Sofridos da Estratégia 3	0.958333
Proporção Gols Feitos / Gols Sofridos da Estratégia 4	1.04348
Maior diferença de gols a favor da Estratégia 3	3
Maior diferença de gols a favor da Estratégia 4	3
Diferença de gols média da Estratégia 3	-0.0192308
Diferença de gols média da Estratégia 4	0.0192308

Tabela 8.2: Resultados dos jogos Estratégia 3 x Estratégia 4.

8.1.2 Verossimilhança

A verossimilhança do motor é importante em um simulador porque torna as simulações mais próximas à realidade, já que os motores simulados agem como os motores reais. Sendo assim, podemos comparar o ganho de velocidade do motor simulador e com o ganho ideal de velocidade do motor real, como descrito em seu manual. Nós utilizamos o motor 2224 006 SR Faulhaber para comparação, e os resultados podem ser vistos em Figura 8.1. Pode ser visto que ambos possuem comportamento similar.

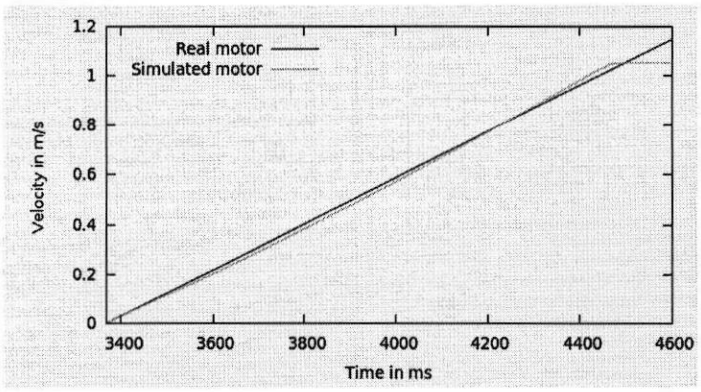

Figura 8.1: Comparação de ganho de velocidade entre motores reais e simulados.

Portanto, as reações dos motores simulados estão de fato próximas das reações dos motores reais, o que concede maior confiabilidade aos resultados das simulações.

CAPÍTULO

9

Conclusão e Trabalhos Futuros

Considerando as características requeridas para o simulador mencionadas anteriormente (Capítulo 1), assim como a análise feita, atingimos resultados satisfatórios com o simulador, pois este permite a aceleração do desenvolvimento e teste de novas estratégias (Doi et al., 2010), que é seu objetivo principal. Porém, ainda existem funcionalidades cuja implementação ainda será realizada, visando melhorias em usabilidade e qualidade. Sendo assim, temos que:

- Permitir testes rápidos, ou seja, executar a simulação de forma tão rápida quanto as estratégias suportarem, para minimizar o gasto de tempo com testes;
- Programar um árbitro para identificar situações de falta definidas na seção 2.1.5;
- Realizar testes de funcionamento para garantir confiabilidade dos resultados obtidos pelo simulador;
- Criar um módulo de análise estatística de experimentos em lote conduzidos pelo simulador;
- Melhoramento da interface gráfica, permitindo que vários clientes possam ver o mesmo jogo e que eles possam gravar vídeos.

9.1 Agradecimentos

Os autores agradecem o apoio financeiro da FAPESP e CNPq.

APÊNDICE

A

Tutorial de Uso

Nesta seção, é apresentado um breve tutorial de uso do Simulador USP-DroidsSS , incluindo como: iniciar o programa, conectar as estratégias, iniciar uma partida, entre outras funcionalidades.

A.1 Início

Para iniciar o programa, tanto em ambiente Windows quanto em Linux, basta rodar o executável gerado pelo Qt ou compilar todo o projeto utilizando o QtCreator. Após uma imagem de inicialização, a interface gráfica será exibida. Para fechar o simulador, vá em Simulador/Sair, ou utilize o atalho Ctrl+Q.

A.2 Começando uma partida

Para começar uma partida, você deve antes conectar duas estratégias ao simulador: basta iniciar as duas estratégias, uma associada ao "time azul"e outra ao "time amarelo". As estratégias devem estabelecer a conexão utilizando as portas padrões, ou então, o usuário deve alterar as portas de conexão para cada estratégia (em Conexões/Portas). A lista de eventos ao lado da visualização do campo indicará algumas mensagens que podem indicar que o simulador está aguardando por novas conexões, ou quando uma estratégia conseguiu se conectar com o simulador.

Após o simulador se conectar a duas estratégias, é possível iniciar a partida em Simulador/Iniciar, ou através do atalho Ctrl+N. Após o término do tempo de jogo, as duas estratégias serão desconectadas, e o simulador aguardará a

conexão de novas estratégias. Caso queira forçar o final da partida, vá em Simulador/Parar, ou alternativamente use o atalho Ctrl+W.

A.3 Opções

Nesta seção, apresentaremos algumas opções que podem ser escolhidas pelo usuário ao utilizar o Simulador USPDroidsSS.

A.3.1 Opções de Conexão

Caso haja ao menos uma estratégia conectada, e a partida não esteja em andamento, o usuário pode desconectar uma das estratégias através da opção Conexões/Desconectar time Azul ou Conexões/Desconectar time Amarelo, dependendo da estratégia que se deseja desconectar. É possível pausar a partida em andamento, desconectar uma ou as duas estratégias (utilizando as opções citadas acima) e, em seguida, conectar novas estratégias, dando continuidade ao jogo.

As portas de conexão das estratégias podem ser reconfiguradas em Conexão/Portas. As portas de Rádio são as portas correspondentes à passagem de informações da estratégia para o simulador, enquanto que as portas de Visão são as utilizadas para o envio de dados do simulador para as estratégias.

A.3.2 Pausar

A qualquer momento de execução da partida, você pode pausar temporariamente o jogo através de Partida/Pausar, ou através do atalho Ctrl+P. Você pode continuar o jogo a partir do comando de início de jogo. Vale destacar novamente que a troca de estratégias só é permitida quando o jogo estiver pausado.

A.3.3 Reposicionamento Automático de Robôs

Caso o usuário observe, durante a simulação, que alguma anomalia que interfere no andamento do jogo, ou simplesmente quer que os robôs e a bola se reposicionem em uma posição de início de jogo, o usuário pode utilizar o comando Simulador/Reposicionar robôs ou o atalho Ctrl+R.

A.3.4 Opções de Exibição

Se o usuário deseja ativar o visualizador de jogo (ou desativá-lo), ele pode utilizar a opção Exibir/Ativar Visualizador. Para utilizar a exibição em Tela

Cheia (ou sair da mesma), vá em Exibir/Tela cheia, ou aperte F11. Em Exibir/Limpar lista de eventos, a lista de eventos é reiniciada.

Também é possível rotacionar e aproximar-se o campo. Para isso, basta que o usuário posicione o mouse em cima do ambiente do campo e utilize o clique esquerdo + arrastar para rotacionar, ou o scroll para aproximar/afastar.

APÊNDICE

B

Estratégias Básicas

Este apêndice visa explicar a Estratégia Padrão e o Módulo do Joystick, que se referem ao API desenvolvido pelo grupo para facilitar a utilização do Simulador USPDroidsSS , e também de um módulo criado, que se conecta ao simulador como se fosse uma estratégia normal, mas no qual o usuário, através de um joystick, controla os robôs.

B.1 Estratégia Padrão

Para facilitar na utilização do Simulador USPDroidsSS e no desenvolvimento de novas estratégias, desenvolvemos uma API com esse propósito. Essa API, chamada de EstrategiaBase, é uma estratégia sem nenhuma funcionalidade de jogo, porém, possui várias funcionalidades de baixo nível implementado.

A principal preocupação da EstrategiaBase é a comunicação, ou seja, os usuários podem utilizar esta API de forma que não precisem se preocupar com a comunicação entre o simulador e a estratégia. Sendo assim, esta API já disponibilizará recursos para receber e interpretar esta mensagem, oferecendo ao usuário as informações (armazenando em variáveis) que o simulador enviou; assim como recursos para envio de mensagens, no qual uma mensagem será montada com a velocidade de rodas dos robôs e será enviado ao simulador.

Além disso, esta API oferece um modelo de organização de classes que pode ser utilizado pelo usuário.

B.2 Módulo do Joystick

Outra função possível do simulador é controlar os robôs manualmente via um joystick. Este módulo foi construído à parte do simulador, mas a integração do joystick é a mesma das estratégias, ou seja, o software do controle se conecta como um cliente UDP/IP ao simulador e passa a enviar os comandos recebidos do joystick.

Os comandos são feitos apertando os botões direcionais digitais, ou analógicos para ter mais controle dos robôs. Ao apertar para cima no joystick, o robô se locomove para frente, enquanto ao apertar para baixo faz com que o robô se locomova para trás. As direções esquerda e direita do botão direcional fazem o robô girar em seu eixo.

É possível controlar um robô por joystick. Caso estejam disponíveis três joysticks, cada um deles irá controlar um robô.

A vantagem de possuir um sistema de controle por joystick é que um jogador humano pode criar situações para testar as jogadas desenvolvidas pela estratégia. Além disso, é muito útil para fazer demonstrações ao público nas atividades de cultura e extensão da pesquisa em robótica, pois a interação com a pesquisa gera interesse ao publico.

Para manter a portabilidade do simulador, o módulo do joystick foi programado usando a biblioteca Simple DirectMedia (Lantinga, 2010) para acessar os recursos de baixo nível do computador, porque desta forma a SDL fornece uma interface genérica para acessar os dispositivos em diversos ambientes operacionais.

Bibliografia

Browning, B. e Tryzelaar, E. (2003). Ubersim: A multi-robot simulator for robot soccer. Em *In Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Costa, A. H. e Pegoraro, R. (2000). Construindo robôs autônomos para partidas de futebol: O time guaraná. *SBA Controle e Automação*.

Doi, D., Calvo, R., e Romero, R. (2010). Sistema de formação de robôs inteligente baseado em otimização por enxame de partículas aplicado a futebol de robôs. Em *XVIII Congresso Brasileiro de Automática*.

FIRA (2008). *www.fira.net*. FIRA. visitado em Fevereiro/2008.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., e Osawa, E. (1995). Robocup: The robot world cup initiative. Em *Proc. of IJCAI-95 Workshop on Entertainment and AI/Alife*, Montreal.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., e Matsubara, H. (1997). Robocup: A challenge problem for ai and robotics. *AI Magazine*, 18(1):73–85.

Lantinga, S. (2010). Simple directmedia layer. http://www.libsdl.org/.

Ltd., C. (2010). Webots simulator. http://www.cyberbotics.com/.

Mackworth, A. K. (1993). On seeing robots. Em Basu, A. e World, X. L., editors, *Computer Vision: Systems, Theory, and Applications*, pages 1–13. World Scientific Press, Singapore.

Nokia Corporation (2010). Qt - a cross-platform application and ui framework. http://qt.nokia.com/.

Pereira, G. A. S., Torres, F. B., e Campos, M. F. M. (2004). Desenvolvimento de robôs holonômicos de baixo custo para o estudo de robótica móvel. Em *Anais do XV Congresso Brasileiro de Automática (CBA'04)*, Gramado, RS.

Project, T. P. (2010). Stage. http://sourceforge.net/projects/playerstage/.

Silva, M. O., Ribeiro, M. V. F., Gaspar, L. S., Silva, W. C., Montanari, R., e Romero, R. A. F. (2009). O sistema do time de futebol de robôs uspdroids. Team Description Paper on CBR2009.

Smith, R. (2010). Open dynamics engine. http://ode.org/.

Thomas, P. e Stonier, R. (2002). Fuzzy control in robot-soccer, evolutionary learning in the first layer of control. Em Callaos, N., Bica, M., e Sanchez, M., editors, *6TH WORLD MULTICONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS, VOL XII, PROCEEDINGS - INDUSTRIAL SYSTEMS AND ENGINEERING II*, pages 181–186. INT INST INFORMATICS & SYSTEMICS.

Tikhanoff, V., Fitzpatrick, P., Nori, F., Natale, L., Metta, G., e Cangelosi, A. (2008). The icub humanoid robot simulator. Em *IROS Workshop on Robot Simulators*.

VerySmall (2008). *Rules for IEEE Very Small Competition*. IEEE, 1.0 edition. These rules are based on the Mirosot FIRA robot soccer category and they are specific for the IEEE Latin American Students Robotics Competition Very Small Size category.