O Simulador 3D para Futebol de Robôs USPDS

Aluno: Diogo Oliveira de Melo
Orientadora: Profa. Dra. Roseli Aparecida Francelin Romero
ICMC/USP - São Carlos
e-mail: dmelo87@gmail.com
e-mail: rafrance@icmc.usp.br

10 de março de 2009

Sumário

1 Introdução **4**

2 Visão Geral **5**

3 Interface Gráfica **5**

4 Estratégia **6**
4.1 Campos Potenciais 7
4.1.1 Adaptação da técnica de Campos Potenciais para Futebol de Robôs 9
4.1.2 Componentes da Estratégia 11
4.1.3 Sub-Módulo Individual 11
4.1.4 Sub-Módulo Interativo para Simurosot 12
4.1.5 Sub-Módulo Interativo para Mirosot 16
4.2 Campos Potenciais Harmônicos 17
4.2.1 Métodos de Relaxação 18
4.2.2 Método de Gauss-Sidel 19
4.2.3 Método de Jacobi 21
4.3 Campos Potenciais Orientados 22
4.4 Campos Potenciais Localmente Orientados 24

5 Núcleo do Simulador **24**
5.1 Gerenciamento da Comunicação entre os Módulos 24
5.2 Engine Física 25
5.3 Detecção de Colisão 27
5.4 Tratamento de Colisões 28
5.5 Função de Tratamento de Erros 28

6 Conexão entre os Módulos **29**
6.1 Comumicação com a Estratégia 31
6.2 Comunicação com a Interface Gráfica 32

7 Grid **32**
7.1 Ferramentas e Métodos Utilizados 32
7.2 Testes e Resultados 33

8 Heurística de Avaliação de Estratégias **34**
8.1 Avaliação entre Duas Estratégias 35
8.2 Avaliação de uma Estratégia no Contexto Geral 35
8.3 Testes e Resultados 36

9 Utilização **36**
9.1 Pré-Requisitos 36
9.2 Instalar 37
9.3 Interface Gráfica - Eyes 38
9.3.1 Tipos de Câmera 38
9.3.2 Controle Sobre o Tempo de Exibição do Jogo 38
9.4 Conclusão 39

1 Introdução

Futebol de robôs é um ambiente utilizado para o desenvolvimento de pesquisas em diversas áreas. Visão computacional, aprendizado de máquina, planejamento de caminhos, controle e automação são alguns exemplos das áreas envolvidas no funcionamento de um time de futebol de robôs. Esforços empregados no desenvolvimento e otimização de um time de futebol de robôs serão refletidos em todos os outros ambientes que utilizam essas técnicas ou que possuem paradigmas simulares. A título de exemplo podemos citar a técnica de Campos Potenciais Harmônicos, primeiramente proposto em [8], é uma técnica utilizada para planejamento de caminhos. O ambiente de futebol de robôs representa uma oportunidade colocar em prática e desenvolver pesquisas importates na áreas de computação e robótica móvel.

O simulador surgiu a partir da necessidade de desenvolver estratégias de controle para times de futebol de robôs, da categoria Very Small Size para times com três jogadores. Em inumeras situações a utilização de simulação proporciona mais vantagens em relação à utilização do ambiente real. A versão 1.1 do simulador, descrita neste relatório técnico, proporciona um ambiente de simulação de qualidade e baixo custo computacional.

Tendo em mente o objetivo de desenvolver um time de futebol de robôs, existe a necessidade de testar, analisar e avaliar as estratégias construidas. Com este próposito o uso de simulação é o mais recomendado. São necessários os reesultados de vários jogos para verificar o funcionamento da estratégia ao longo de seu desenvolvimento. Certificar de que todos os robôs, mais a câmera e todo o software necessário estão funcionando é muito dispendioso. O uso de simulação nesses casos, livra o desenvolvedor de uma série de problemas.

Inicialmente foi analisada a possibilidade de utilizar algum simulador já existente, como o SServer [7] da Robocup, que é software livre e foi construído para a plataforma GNU/Linux, também há o simulador da FIRA [1], que pertence à categoria Simurosot. Entretanto, há desvantagens em relação a ambos os simuladores. O simulador da FIRA, por exemplo, só funciona na plataforma Windows, os times são compostos por cinco jogadores e o simulador não pode ser executado sem a parte gráfica [1]. O Simulador da Robocup simula futebol de robôs porém os robôs possuem geometria circular, entretanto o time a ser construído possui robôs de formato cúbico. Uma terceira possibilidade seria utilizar o Mobile Robots Simulator (MOBS) [17]. Este simulador segue as regras da categoria Simurosot e possui times com três jogadores entretanto, não é software livre e nem pode ser executado sem

[1] A importancia de ter um simulador que funcione sem visualização gráfica é a facilidade de automatizar a execução de várias simulações

a interface gráfica.

Não encontrar um simulador que atendesse a todas as necessidades levou à construção do simulador USPDS. Este simulador foi desenvolvido no Laboratório de Aprendizado de Robôs (LAR) do Instituto de Ciências Matemáticas e de Computação de São Carlos (ICMC). Este simulador possui times de três robôs, cada robô é um cubo de 75 milímetros de aresta. Este é um projeto de software livre e está disponível em http://uspds.sourceforce.net. Embora o simulador tenha sido projetado para a plataforma GNU/Linux, o ambiente de simulação oferecido é independente de linguagem de programação e sistema operacional.

2 Visão Geral

USPDS foi projetado para ser tão flexível quanto possível. O objetivo do projeto é atender pessoas que desenvolvem estratégias para futebol de robôs da categoria Very Small Size [2]. É necessário possibilitar que o usuário configure o simulador de acordo com seu caso (peso do robô, coeficiente de atrito entre o robô e o campo e etc.). Este pré-requisito fez com que o simulador fosse construído em módulos. A simulação é composta por três programas sendo executados em sincronia. Cada estratégia se conecta ao núcleo por uma conexão TCP/IP. Em cada iteração, a estratégia recebe informações sobre as condições do jogo, calcula a decisão e envia os comandos de volta para o núcleo do simulador. A interface gráfica é opcional durante a simulação, também é possível ter várias interfaces gráficas conectadas a uma simulação o que permite que várias pessoas de diferentes lugares consigam acompanhar o mesmo jogo.

3 Interface Gráfica

Durante a simulação a interface gráfica é utilizada para mostrar os dados do jogo. A interface é totalmente passiva, apenas recebe informações do simulação. Cada interface gráfica conectada ao simulador recebe o estado de cada iteração. Por estado entende-se as posições e orientações de cada robôs mais a posição da bola. A interface gráfica utiliza a linguagem de programação C++ e a biblioteca OpenGL e possibilita que o usuário assista o jogo através de quatro diferentes modos. O modo padrão mostra ao usuário a visão do espectador, que consiste de uma câmera estática sobre o campo.

[2] Uma das tarefas para uma versão futura do USPDS é extendê-lo, de modo a tornar fácil a configuração do tamnho e formato do campo assim como número de robôs

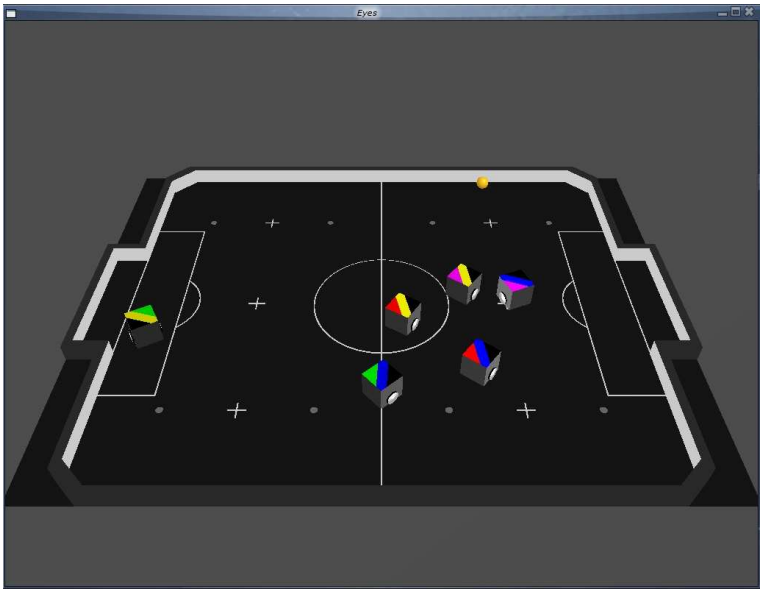

Figura 1: Interface Gráfica EYES

No segundo modo a câmera está focada em um jogador. O terceiro modo consiste na câmera embarcada ao robô. Finalmete, o quarto modo, é a visão livre, o usuário possui controle sobre a câmera e pode movimentá-la pelo campo. O Eyes (nome da interface gráfica) também possibilita controle sobre a velocidade do jogo. Isto também inclui pausar e avançar.

O Eyes é composto por duas threads. A primeira é responsável por receber dados e a segunda por mostrá-los. Toda a interação com o usuário mais a parte de OpenGL está relacionada com a segunda thread. A primeira thread mantém uma conexão aberta com o núcleo do simulador para receber os dados da simulação. Todos os dados recebidos podem ser gravados em um arquivo, no formado do simulador. A interface gráfica também possibilita exportar o jogo para o formato de vídeo AVI. A Figura 1 é uma foto da interface gráfica, mostrando um jogo.

4 Estratégia

A maioria dos times de futebol de robôs não conseguem capturar o ângulo dos robôs adversários. Por este motivo, cada time possui informações sobre a angulação apenas dos própios robôs. Além dos ângulos dos próprios robôs, cada estratégia recebe as posições, em coordenadas cartesianas, de todos os objetos do jogo: os próprios robôs, robôs adversários e a bola. Após receber

estas informações, cada time deve determinar a velocidade de cada roda de cada robô do próprio time.

O simulador possui um suporte adicional para estratégias construídas na linguagem de programação C++, que consiste dos métodos necessários para enviar e receber dados do núcleo do simulador. Também foram adicionados métodos que calculam a velocidade de cada roda para que um robô se dirija até determinado ponto do campo.

Junto ao simulador exitem duas estratégias. A primeira é baseada em Campos Potenciais (CP) e a segunda utiliza Campos Potenciais Localmente Orientados (CPLO). Entretanto, a estrátégia baseada em CP é uma adaptação de uma estratégia para a categoria Simurosot da FIRA e a estratégia baseada em CPLO é uma junção entre Campos Potenciais Harmônicos (CPH) e Campos Potenciais Orientados (CPO). Portanto, serão abordadas as quatro técnicas: CP, CPH, CPO e CPLO.

4.1 Campos Potenciais

A técnica de Campos Potenciais foi proposta por Kathib [16]. Esta técnica propõe que o ambiente seja modelado como um espaço no qual os objetos promovam forças uns sobre os outros. Aplicando esta técnica ao ambiente de futebol de robôs, temos que a força resultante atuando sobre um determinado jogador (do time em que a estratégia é aplicada), deverá servir de base para determinar a trajetória que este jogador deverá seguir.

Por ser um método de baixo custo computacional, esta técnica tem sido muito utilizada em planejamento de trajetórias de robôs móveis que necessitam navegar por um ambiente desconhecidos e/ou dinâmico, evitando colisões com obstáculos. Vários pesquisadores têm apresentado propostas para melhorar a eficiência do método de Campos Potenciais.

Krogh [18] aprimorou este conceito ao considerar que a velocidade do robô deve diminuir na vizinhança dos obstáculos para diminuir o risco de colisão. Thorpe [22] aplicou o método de Campos Potenciais para pelanejamento de caminhos. Newman [20] introduziu a implementação de funções potenciais através da combinação de funções individuais de obstáculos com operadores lógicos.

Brooks [5] e Arkin [4] são os pioneiros em utilizar a técnica de Campos Potenciais de forma reativa, ou seja, para cada novo passo do robô uma nova força resultante deverá ser calculada para direcionar seu movimento. Esta característica reativa torna o método muito eficiente para robôs que realizam tarefas em ambientes dinâmicos.

Pela proposta feira por Faria [12], é necessário definir quais os pontos de atração e repulsão, no campo. Também é necessário determinar de que modo

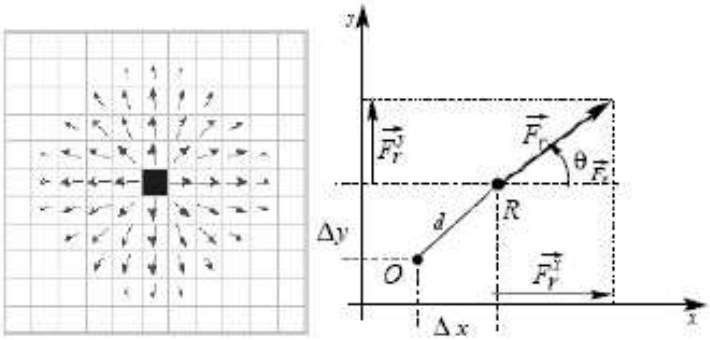

Figura 2: Campo gerado por um ponto de repulsão [12]

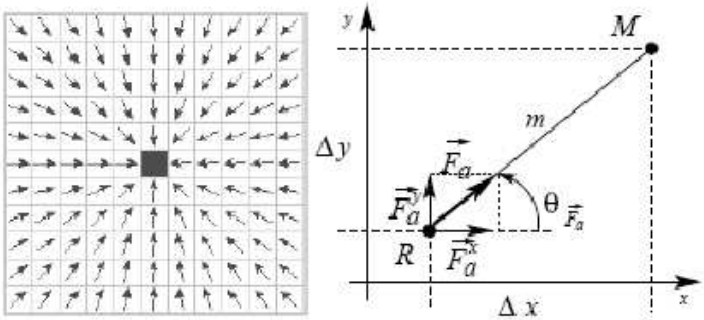

Figura 3: Campo gerado pela bola [12]

estes campos irão se propagar. O jogador atacante interpreta a bola como ponto de atração e os outros jogadores representam pontos de repulsão. A força de repulsão provocada por cada jogador é inversamente proporcional ao quadrado da distância entre o jogador que provoca a força e o jogador atacante em questão. Sendo assim, o campo potencial, gerado por um jogador, pode ser visto como ilustado na Figura 2.

O modo de propagação da força de atração é diferente. O módulo da força de atração, provocada pela bola, é constante. Foi definido desta forma porque o jogador atacante, independente de sua posição, deve ser atraído pela bola. O Campo Potencial gerado pela bola é como ilustrado na Figura 3.

Também é necessário definir constantes, que determinam o módulo relativo entre as forças exercidas sobre o jogador atacante. No caso da estratégia proposta por Faria [12], foram definidas três constantes. A primeira é usada

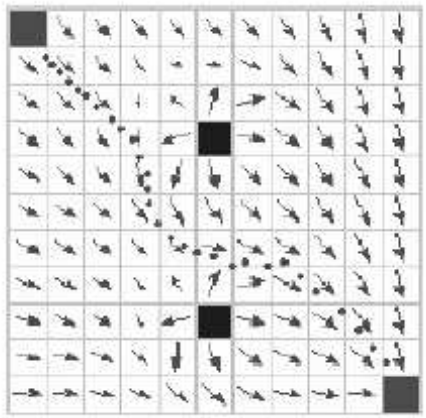

Figura 4: Campo potencial gerado por dois obstáculos e uma meta [12]

no cálculo da força gerada pelos jogadores que pertencem ao mesmo time que o jogador atacante. A segunda constante diz respeito aos jogadores do time adversário. A última constante é usada para determinar o módulo da força exercida pela bola.

Uma vez que cada força é calculada, a força resultante é obtida pela soma dos vetores das foças calculadas. A direção e o sentido, da força resultante, determinam a trajetória que o robô atacante deve seguir. A força resultante é o resultado do somatório de todas as forças que atuam sobre o jogador atacante.

A técnica de Campos Potenciais apresenta várias vantagens. Dentre estas vantagens está o baixo custo computacional e a grande eficiência em ambientes dinâmicos. Embora esta estratégia apresente um bom desempenho, Faria [12] apresenta algumas modificações que corrigem alguns erros da estratégia, até aqui descrita.

4.1.1 Adaptação da técnica de Campos Potenciais para Futebol de Robôs

Para aumentar a eficiência, Faria [12] propôs essencialmente duas alterações na aplicação da técnica de Campos Potenciais à estratégia.

A primeira modificação diz respeito ao robô artilheiro (ou atacante). Com o robô sendo atraido diretamente pela bola, foi observado que a quantidade de gols contra era grande. Para corrigir este erro, foi criada uma reta imaginária s que é determinada pelo centro do gol adversário e a bola. Sobre a reta s foi colocado o ponto P (em vermelho), que fica a uma distância i da bola,

Figura 5: Representação do comportamento do artilheiro

como ilustrado na Figura 5.

Segundo a proposta feita por Faria [12], o robô artilheiro é atraido pelo ponto P. Porém, quando o jogador estiver próximo do ponto P (na área sombreada em vermelho), o artilheiro passa a ser atraído diretamente pela bola. Essa adaptação aumenta as chances de chute certeiro ao gol adversário.

A segunda adaptação proposta por Faria [12], foi a criação de uma defesa. Para criar o comportamento da defesa, foram inseridas quatro linhas imaginárias (a, b, c e d). Estas linhas cortam o campo longitudinalmente. Uma quinta linha z foi criada. Esta última é determinada pelo centro do gol e a bola. Desta forma, cada jogador da defesa tem seu ponto de atração determinado pelo cruzamento entre os segmento z e a sua respectiva linha vertical, como mostrado na Figura 6.

Depois de ter estudado a proposta feita por Faria, são propostas algumas modificações, as quais são descritas a seguir. É importante lembrar que as alterações descritas abaixo foram implementadas tendo como base o código desenvolvido por Faria. Porém a estratégia para times com três jogadores foi implementada baseada em algumas bibliotecas criadas para o Simulador, o restante foi reimplementado do inicio.

Figura 6: Representação do comportamento da defesa

4.1.2 Componentes da Estratégia

Para manter um bom nível de organização, a Estratégia será dividida em dois sub-módulos: sub-módulo individual e sub-módulo interativo. O sub-módulo individual é composto por funcões e métodos que têm por objetivo determinar o comportamento de um jogador sem implementar táticas de equipe. Por outro lado o sub-módulo interativo visa o comportamento em grupo, isto é, a cooperação entre dois ou mais jogadores do time.

4.1.3 Sub-Módulo Individual

As funções pertencentes ao sub-módulo individual são funções básicas. Estas funções dão suporte às funções utilizadas pelo sub-módulo interativo. As funções que fazem parte do sub-módulo individual são: Velocity, Angle, Goto e GotoXY.

A função *Velocity* é a função mais básica. Como atributos, são passados o robô alvo (que é o robô considerado por essa função), a velocidade desejada na roda esquerda e a velocidade desejada na roda direita. A função faz com que o robô alvo altere as velocidades das rodas para as velocidades passadas por parâmetro.

A função *Angle* utiliza a função Velocity para fazer com que o robô alvo se posicione no ângulo passado, à função, por parâmetro. Caso o robô já esteja próximo o suficiente do ângulo desejado a função Angle pára o robô.

Para que o robô alcance o ângulo desejado, a função Angle faz com que ele gire em torno do próprio eixo.

A função Goto é semelhante a função Angle porém, faz com que o robô alcance o ângulo desejado descrevendo uma curva.

Por fim, a função GotoXY, utiliza a função Goto para guiar o robô até um determinado ponto, fornecido através de coordenadas cartesianas.

4.1.4 Sub-Módulo Interativo para Simurosot

Atualmente, três funções compõem o sub-módulo interativo da estratégia feita para Simurosot, as funções FollowBall, Defesa e Strategy. A FollowBall é usada para a parte do ataque. A função Defesa, como o próprio nome já sugere, é usada pela defesa do time, o que inclui o goleiro. A função Strategy coordena as funções que cada robô irá utilizar.

Lembrando que cada time dessa categoria tem cinco jogadores, foi decidido que dois desses jogadores farão parte do ataque e os outros três irão compor a defesa. Porém, com exceção do goleiro, essas posições não são fixas. Dependendo da situação em que o jogo se encontra, jogadores do ataque podem passar para a defesa e vice-versa. A seguir, abordaremos o modo de funcionamento de cada uma dessas três funções.

A função *FollowBall*, foi projetada para controlar dois jogadores. Entretanto, observamos que pode acontecer de dois robôs, do mesmo time, disputarem a bola, o que é indesejável.

Para resolver este problema, a função FollowBall tem a finalidade de seccionar o campo em duas áreas (separadas pela linha verde), como ilustrado na Figura 7. Quando a bola estiver na área inferior, o jogador A1 fica encarregado de buscar a bola. Ao mesmo tempo, o atacante A2, procura por uma posição propícia para brigar pela bola, caso a bola passe para a outra área. Quando a bola entra na outra área, os papéis dos atacantes são invertidos.

Na função *FollowBall* os jogadores atacantes são repelidos pelos outros jogadores e sentem atração pelos pontos vermelhos ilustrados na Figura 7. O ponto de atração do atacante A1 é determinado pelo segmento de reta, formado pelo fundo do gol adversário e a bola, sendo que este ponto fica a uma distância da bola, na direção oposta ao gol adversário. A distância, entre a bola e o ponto de atração, foi tomada como sendo equivalente a metade da medida da aresta do robô [3].

A escolha do ponto de atração do jogador A1 foi feita com o intuito de direcionar o ataque do jogador. No modo anterior, porposto por Faria

[3]Essa distância foi determinada empiricamente. É uma das constantes que devem ser otimizadas pelo método descrito no capítulo 6.

Figura 7: Divisão feita pela FollowBall

[12], o atacante também era atraído por um ponto colinear ao centro do gol adversário e a bola porém, o novo ponto de atração fica a uma distância menor da bola o que faz com que não seja necessário que o jogador tome uma nova atitude ao se aproximar do ponto de atração.

O atacante A2 sente atração pelo ponto cujas coordenadas são formadas pelo X da bola e o Y da lateral da "grande área". A intenção é que o jogador A2 não ataque mas, fique na posição mais estratégica possível, esperando que a bola passe para a sua área.

Para chamar a função *Defesa* é necessário passar alguns parâmetros. Parte desses parâmetros definem a área de atuação do jogador da defesa. A posição da bola, em relação à área de defesa do jogador, irá determinar a atitude desse jogador.

Na Figura 8 está ilutrado um jogador da defesa. Este jogador tem a sua área de atuação limitada pelas linhas verde e vermelha. Sendo assim, existem três partes do capo em que a bola pode estar: à frente da linha verde, entre a linha verde e a linha vermelha ou atrás da linha vermelha.

Caso a bola esteja à frente da linha verde, o jogador será atraído pelo ponto determinado pelo cruzamento entre os segmentos a e b. Isso faz com que o jogador procure estar entre a bola e o fundo do gol no qual a estratégia está atuando, servindo assim como um escudo.

Se a bola estiver entre as linhas verde e vermelha então a função de ataque (*FollowBall*) é chamada para o jogador em questão. Isso significa que, caso a

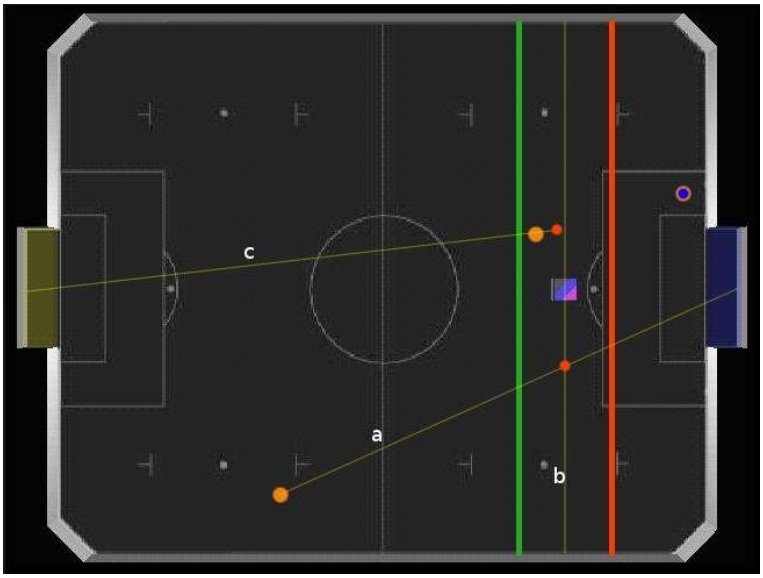

Figura 8: Área de alcance da função *Defesa*

bola esteja na área de alcance do jogador ele tentará jogar a bola na direção do gol adversário.

Caso a bola esteja atrás da linha vermelha, o jogador será repelido pela bola. O raciocínio é que, uma vez que a bola está fora do alcanve, o máximo que o jogador pode fazer é não atrapalhar. Essa medida tem se mostrado eficiente em fazer com que o jogador não atrapalhe (permitindo que a bola saia da área da defesa) porém, ineficiente para retomar sua posição adequada, quando a bola sai de trás da linha vermelha.

O goleiro também utiliza a função Defesa. Porém, a área de atuação que é determinada para ele, faz com que ele sempre fique à frente da linha verde. Essa medida induz o goleiro a ficar sempre entre o gol e a bola. É importante ressaltar que a eficiência da defesa depende das áreas de atuação que são definidas para cada jogador.

Como dito anteriormente, os jogadores (com exceção do goleiro) não possuem posições fixas. A função que cada jogador desempenha depende do Estado do jogo. A função responsável por implementar maleabilidade ao time é a Strategy.

O Estado do jogo depende da posição da bola, no campo. Existem apenas dois Estados, o Estado 1 e o Estado 2. O Estado 1 é caracterizado pelo fato da bola estar no lado do campo do time adversário. Como existem apenas dois Estados. Se o jogo não estiver no Estado 1 então estará no Estado 2.

A Figura 9 ilustra o Estado 1. Neste Estado, dois jogadores são designados

Figura 9: Representação do Estado 1

para o ataque, usando a função *FollowBall*, e os outros três usam a função *Defesa*. A estratégia montada para o Estado 1 faz com que o time ataque porém, não descuide da defesa.

A área de atuação de cada jogador é determinada pelas linhas verdes que segmentam o campo. Dentro de cada uma dessas áreas há um ponto vermelho. Cada um destes pontos representa o ponto de atração para cada jogador de sua respectiva área.

A Figura 10 representa a ação no Estado 2. Neste estado, todos os jogadores assumem função de defesa.

Quando a bola está no lado da defesa todos os jogadores passam a usar a função *Defesa*. Isso faz com que os robôs exerçam função de "bloqueadores", impedindo que a bola chegue ao nosso gol.

Da mesma forma que na Figura 9, as linhas verdes da Figura 10 dividem o campo em áreas. Para cada uma das áreas formadas, no lado da defesa, existe um jogador designado para defender a sua área.

Como visto anteriormente, quando a bola entra na área de um jogador da defesa, esse jogador passa a usar a função *FollowBall* e com isso trabalha no intuito de expulsar a bola de sua área, tentando levar a bola para o lado do adversário.

A forma dinâmica com que a Strategy faz os jogadores se movimentarem, passando de defensores para atacantes e vice-versa, se apresentou satisfatória. Porém o dinamismo usado na estratégia para times com três jogadores é mais

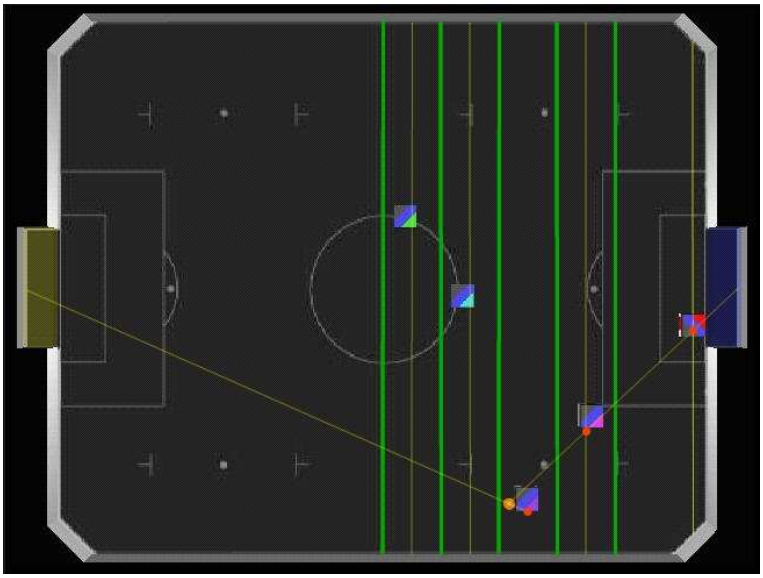

Figura 10: Representação do Estado 2

atenuado.

4.1.5 Sub-Módulo Interativo para Mirosot

Esta estratégia segue a mesma linha da estratégia para Simurosot com diferenças apenas na forma como os jogadores alteram de funções. Quando a configuração do campo está no Estado 1, dois jogadores são designados para o ataque e o terceiro jogador fica no gol. O que define quais jogadores vão para o ataque é a distância de cada um deles em relação a bola. Os dois jogadores mais próximo da bola vão para o ataque, como o ilustrado na Figura 11.

Quando o jogo está no Estado 2. Os três jogadores são designados para defesa e o que define qual deles será o goleiro é a distância em relação ao fundo do gol a ser defendido.

Em uma das estratégias o método *Defesa* possuirá o mesmo algoritmo que a defesa da Simurosot e a outra estratégia possuirá um algoritmo mais simplificado. A inteção é fazer jogos entre as duas estratégias e avaliar o desempenho de cada uma delas.

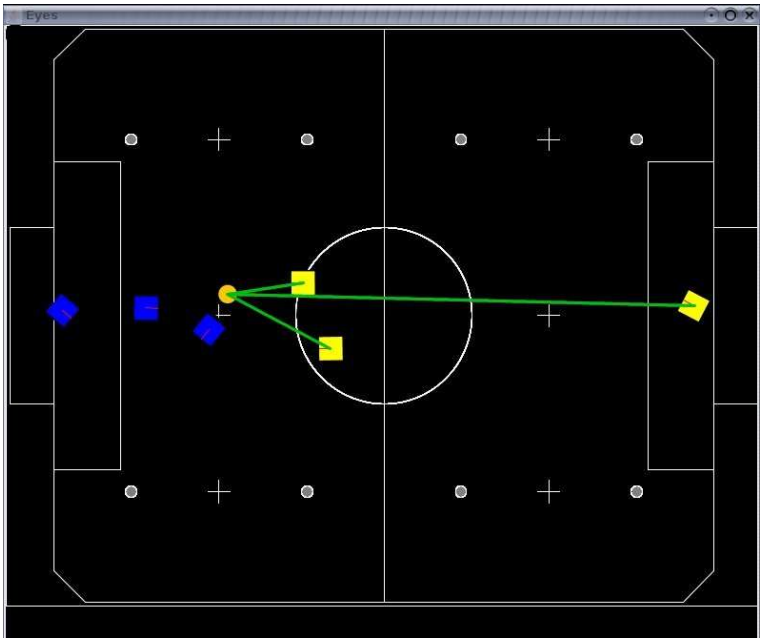

Figura 11: Representação da estratégia para Mirosot

4.2 Campos Potenciais Harmônicos

Embora a técnica de Campos Potenciais seja eficiente, esta técnica possui limitações que podem comprometer a eficiência do sistema. Um dos problemas da técnica de Campos Potenciais é a formação de mínimos locais. Conforme explicado a seguir, uma das caracteristicas do CPH (Campos Potenciais Harmônicos) é não possuir mínimos locais.

As funções harmônicas foram primeiramente utilizadas para controle de robôs por Connolly [9]. De acordo com Connolly, uma função $p : \Omega \rightarrow R$, com $\Omega \subset R^n$, é uma função Harmônica, se a Equação 1 for setisfeita.

$$\nabla^2 p = \sum_{i=1}^{n} \frac{\partial p^2}{\partial x_i^2} = 0 \tag{1}$$

Embora os robôs sejam tridimensionais, eles se movem em um plano. Dessa forma $\Omega \subset R^2$. A Equação 1 pode ser reescrita na forma da Equação 2.

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0 \tag{2}$$

Conforme descrito por Faria [11], para resolver equações diferenciais par-

ciais existem métodos numéricos como Gauss-Seidel (GS), Sucessiva Sobre-Relaxações (SOR) e o método de Jacobi. Aplicar um método de relaxação em uma malha resolverá o sistema numericamente.

4.2.1 Métodos de Relaxação

Alguns métodos de relaxamento foram implementados. Com o objetivo de fazer uma análise comparativa entre os métodos de relaxamento, será necessário definir as dimensões da malha[4]. O campo de futebol de robôs mede $1,7mx1,3m$. Cada robô, possui aresta de, aproximadamente, $7,5cm$. Para discretizar o campo, usaremos uma malha retangular quadriculada. Esta malha, possuirá o formato retangular do campo de futebol de robôs e será composta por quadrados de $1cm$ de lado. Uma vez determinada que cada quadrado possui $2,5cm$ de lado, temos que, nossa malha retangular possuirá um total de $170x130$ células.

Foram analisadas três opções: método de Jacobi, método de Gauss-Siedel e SOR (Successive Over Relaxion). O Método de Jacobi possui a característica de necessitar de duas malhas para o seu cálculo. A primeira malha é utilizada para guardar a i-ésima iteração e, durante o cálculo da iteração $i+1$, esta malha é utilizada apenas para leitura. A iteração $i+1$ é calculada e armazenada na segunda malha. Por ser um método iterativo, a iteração $i+1$ necessita apenas do resultado da itareção i. A vantagem deste método é o fato dele ser altamante paralelizável. Como descrito adiante, paralelização será essencial para executar esta aplicação em tempo real.

O método SOR, assim como o de Gauss-siedel necessita alocar espaço para apenas uma malha, o que representa uma vantagem em relação ao método de Jacobi. Além desta, o método SOR possui a vantagem de necessitar de um menor número de iterações para atingir a convergência. Entretanto, o fato das iterações intermediárias do SOR não terem a garantia de serem utilizáveis torna o uso deste método impróprio, devido às necessidades da aplicação. Por esta ser uma aplicação que deve ser processada em tempo real, caso a convergência demande mais tempo do que o limite, para ser calculada, será necessário encerrar as iterações e utilizar a malha gerada até o momento. Este é um caso em que será necessário perder em precisão para ganhar em tempo.

Abaixo está descrita a implementação feita através da relaxação pelo método de Gauss-Siedel.

[4]Definir a priori as dimensões da malha é necessário pois a velocidade de convergência dos métodos de relaxamento dependem das configurações da malha

Categoria do Ponto	Valor Inicial
Meta	0.0
Vazio	0.5
Obstáculo	1.0

Tabela 1: Valores iniciais das células, na malha

4.2.2 Método de Gauss-Sidel

O relaxamento utiliza o operador de Laplace [19], mostrado na Equação 3

$$L_{s1} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3}$$

Utilizando o método de Gauss-Siedel com o escolhido operador de Laplace temos que a célula $p_{i,j}$ da malha é obtida de forma iterativa, segundo a Equação 4.

$$p_{i,j}^{t+1} = \frac{1}{4} \times (p_{i-1,j}^{t+1} + p_{i,j-1}^{t+1} + p_{i,j+1}^{t} + p_{i+1,j}^{t}) \tag{4}$$

Cada iteração percorre todas as células da malha. Quanto maior o número de iterações, mais precisos são os valores da malha. É necessário definir o número de iterações pelo qual a malha irá passar. Para monitorar a velocidade de convergência da malha podemos utilizar o erro quadrático. O erro quadrático e_q é dado pela Equação 5.

$$e_q = \sum_{i=1}^{m} \sum_{j=1}^{n} (p_{i,j}^{t} - p_{i,j}^{t-1})^2 \tag{5}$$

Em que m e n significam o número de linhas e colunas, respectivamente, da matriz que representa a malha. Utilizaremos e_q como critério de parada para as iterações. Quando o erro quadrático for menor do que determinada constante, o programa pára de fazer iterações sobre a malha.

Lembrando que a malha representa as dimensões de um campo de futebol de robôs, foram fixados obstáculos nos pontos $(2cm, 2cm)$ e no ponto $(100cm, 100cm)$ e uma meta no ponto $(50cm, 50cm)$. A configuração inicial do campo é dada pela Tabela 1.

O primeiro experimento necessitou de 3902 iterações para que e_q convergisse para um valor da ordem de 10^{-6}. Uma vez que estamos utilizando ponto flutuante de precisão simples, escolhemos, como limiar para o erro quadrático, o limite de representação do ponto flutuante. A Figura 12 é uma representação gráfica da malha ao final das iterações.

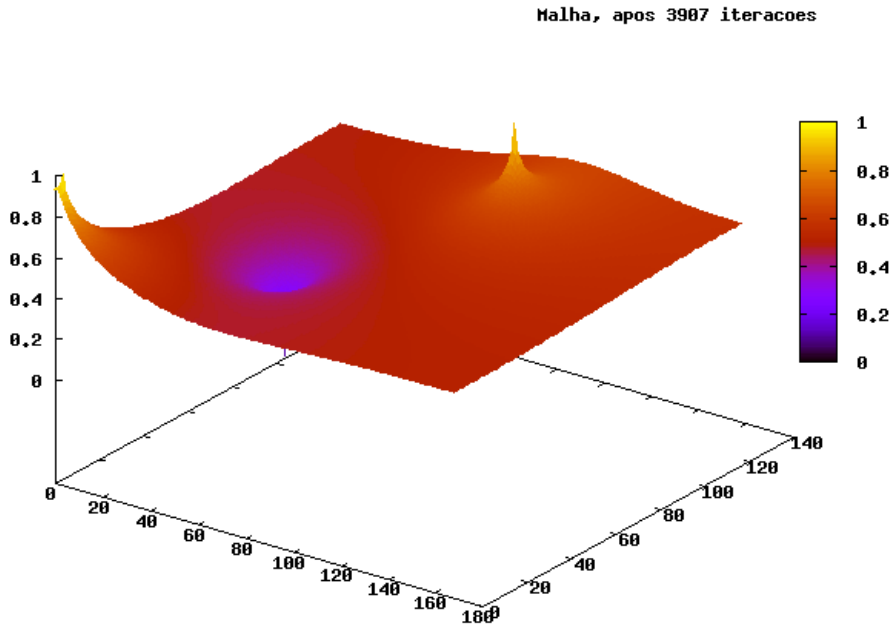

Figura 12: Malha gerada utilizando o operador de laplace L_{s1}.

L_{s2}, definido pela Equação 6, é um outro operador de Laplace.

$$L_{s2} = \begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix} \tag{6}$$

A definição de $p_{i,j}$ utilizando o operador L_{s2} é obtida do mesmo modo que para o operador L_s1. Utilizando o operador L_{s2}, o erro quadrático alcançou 10^{-6} em 4048 iterações. Foram realizados experimentos com outras configurações de malhas. Em todos os experimentos, a relaxação utilizando L_{s2} foi mais lenta do que utilizando o operador L_{s1}. Com estes resultados experimentais, podemos concluir que o operador L_{s1} leva o sistema à convergência mais rápido do que o operador L_{s2}.

Ao implementar a estratégia utilizando CPH dois fatos importantes foram observados. O primeiro fato é que a necessidade de muitas iterações ocorre apenas no inicio do jogo, ao fazer a primeira relaxação. Depois da primeira relaxação são necessárias, em média, apenas 10 iterações. O segundo fato aferido diz respeito a tempo gasto por iteração. O tempo gasto por iteração é de $7ms$. Este tempo é proibitivo, seriam necessários $70ms$[5] por quadro para executar o CPH. Para melhorar o tempo de execução, a malha passou a ter a dimensão de $85x65$ células. Esta redução fez o tempo de execução, por iteração, cair para $1,62ms$. Este tempo ainda é muito alto. Por este motivo foi decidido fazer a relaxação utilizando paralelização. O método de Jacobi, por ser paralilizável, foi escolhido para substituir o de Gauss-Siedel.

4.2.3 Método de Jacobi

O método de Jacobi é extremamente parecido com o método de Gauss-Siedel. Matematicamente, a diferença entre os dois métodos está na equação de relaxação. A Equação 7 mostra a relaxação para o método de Jacobi. É possível observar que, para executar esté método serão necessárias duas matrizes. A primeira guarda a malha no estado atual e a segunda é utilizada para guardar o resultado da iteração seguinte.

$$p_{i,j}^{t+1} = \frac{1}{4} \times (p_{i-1,j}^{t} + p_{i,j-1}^{t} + p_{i,j+1}^{t} + p_{i+1,j}^{t}) \tag{7}$$

A implementação paralela foi feita através da biblioteca OpenMP [2]. Com o intuito de obter o melhor desempenho possível é necessário levar em consideração as características da arquitetura do computador utilizado. O

[5]Todos os sistemas que compõem o futebol de robôs tem um total de $33ms$ para serem executados

processador é um Intel Core 2 Quad modelo Q6600 2.4GHz com 8MB de memória Cache.

Cada célula da malha ocupa 28 bytes. Isso faz com que toda a malha necessite de aproximadamente 150KB. Logo, o grid pode ser facilmente alocado na memória cache, o que provê um considerável ganho de desempenho.

A biblioteca OpenMP é composta por uma interface que permite gerenciar o paralelismo da aplicação de modo simples. Após implementar a versão paralela da relaxação, utilizando o método de Jacobi, o tempo de processamento necesário por iteração desceu para $0,45ms$ aproximadamente. Este tempo consiste em um speedup absoluto de $3,6$ e desempenho de 0.9. Mais informações sobre o cálculo de speedup e rendimento de uma aplicação paralela podem ser encontradas em [3].

Utilizando os resultados obtidos é possível manipular um CPH com o custo computacional de $4,5ms$, a cada quadro. O tempo de $4,5ms$ é considerado baixo, entretanto é necessário alocar um CPH para cada jogador, o que resultaria em $13,5ms$ por quadro, que é um tempo alto. Uma vez que a visão computacional consome $20ms$, a estratégia e o controle possuem, no máximo, $13ms$ para serem executados.

Uma possível solução para este problema é utilizar a teoria de Campos Potencias Localmente Orientados (CPLO), proposta por Faria [13]. Na seção seguinte estão descritos os resultados obtidos no estudo do Campos Potencial Orientados (CPO), que é a base do CPLO.

4.3 Campos Potenciais Orientados

A técnica CPO foi proposta por Prestes em [21]. Está técnica consiste em adicionar um vetor de influência sobre a malha com o intuito de direcionar o movimento do robô. A Figura 13 ilustra a influência provocada por diferentes vetores. É possível observar que o vetor de influência força o caminho alcançar a meta com um ângulo próximo ao ângulo do vetor de influência.

A diferença matemática entre CPH e CPO está na equação do relaxamento das células. A Equação 8 representa o relaxamento das células no CPO, utilizando o método de Jacobi. O CPO adiciona um termo a equação do CPH, este termo é responsável por produzir o efeito de orientação na malha.

$$p_{i,j}^{t+1} = \frac{1}{4}(p_{i-1,j}^{t}+p_{i,j-1}^{t}+p_{i,j+1}^{t}+p_{i+1,j}^{t})+\frac{1}{8}[(p_{i+1,j}^{t}-p_{i-1,j}^{t})vx+(p^t i,j+1-p^t i,j-1vy)] \tag{8}$$

O CPO é muito vantajoso, por exemplo, para fazer com que o jogador atacante direcione o seu ataque para o gol adversário ou também para im-

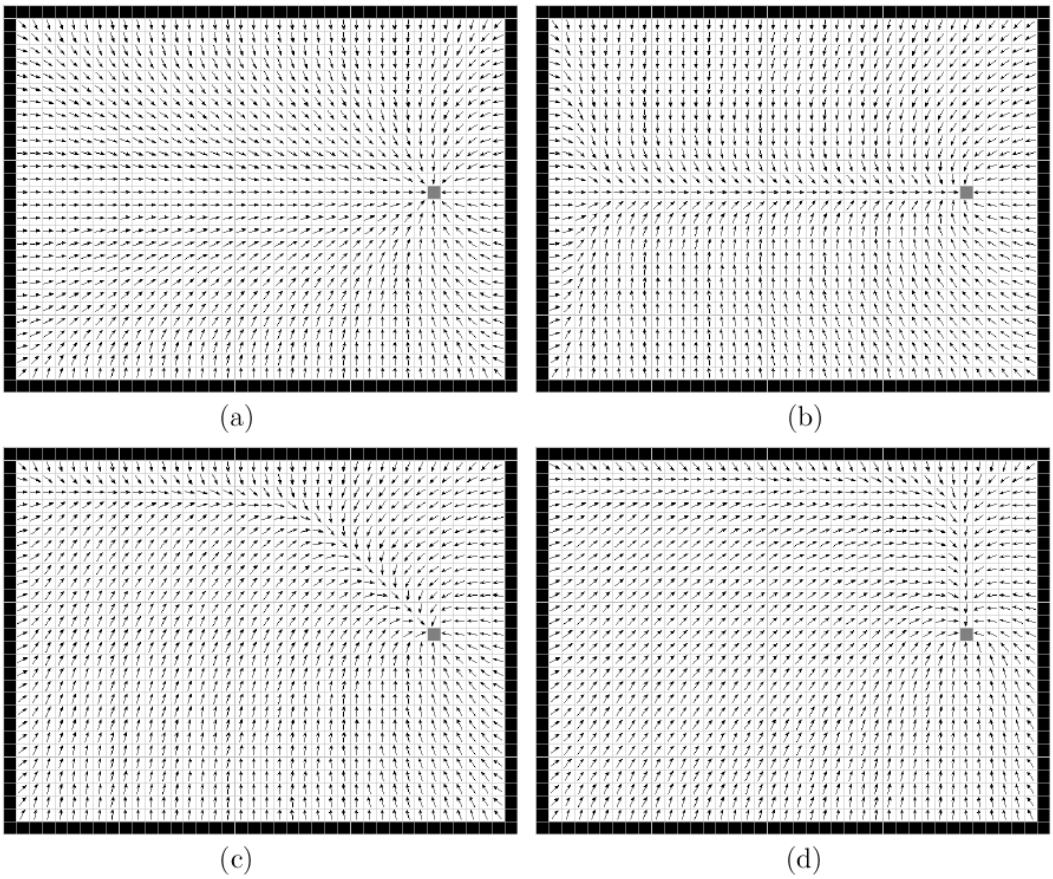

Figura 13: Influência do vetor v no campo potencial. (a) CPH sem influência do vetor; (b) CPO com $v = (1, 0)$; (c) CPO com $v = (1, 1)$; (d) CPO com $v = (0, 1)$

pedir que o goleiro abandone a área do gol. Entretanto, o CPO só consegue mapear um tipo de comportamento por malha. Ainda com o CPO é necessário utilizar três malhas para o controle dos três jogadores. O CPLO provê uma solução para controle de multiplos robôs em uma única malha.

4.4 Campos Potenciais Localmente Orientados

A teoria de CPLO proposta em [13] apresenta uma solução de baixo custo computacional para o controle de múltiplos robôs. O CPLO é basicamente uma junção entre o CPH e múltiplos CPOs. A cada robô é associado um vetor e duas constantes λ e ϵ. O algoritmo para relaxar a malha cosiste em percorrer todos as células da malha. Para cada célula encontra-se o robô mais próximo. Se a distância até o robô mais próximo for maior do que λ, é utilizada a Equação 3, caso contrário, utilização a Equação 8.

O algoritmo de estratégia ACIn, proposto em [10], que utiliza CPLO foi implementado e também faz parte do USPDS. Posteriormente, serão realizados testes no simulador para verificar o desempenho da estratégia utilizando CPLO em relação a estratégia que utiliza CP.

5 Núcleo do Simulador

O núcleo do simulador possui duas tarefas: se comunicar com os demais módulos e calcular as novas posições e ângulos dos objetos a cada iteração. A seguir está descrito o modo como o simulador realiza estas duas tarefas.

5.1 Gerenciamento da Comunicação entre os Módulos

Existem, no mínimo, quatro threads no núcleo do simulador. Três delas destinam-se a comunicação. Destas três, duas são utilizadas para comunicação com as estratégias, uma thread para cada estratégia. A terceira thread é utilizada para comunicação com interface gráfica. Toda vez que uma interface gráfica se conecta ao simulador, é por meio desta terceira thread. O papel desta thread é criar uma nova thread que converse com a interface gráfica, toda vez que uma interface gráfica se conectar a ela. Desta forma, o simulador é capaz de se conectar a várias interfaces gráficas. A quarta thread é reponsável por executar a Engine física.

A estratégia e o simulador se comunicam através de duas conexões TCP/IP. A escolha por dois sockets ao invés de apenas um se deve ao fato de muitos times de futebol de robôs utilizarem computadores diferentes para processar a visão, estratégia e comandos de baixo nível do robô. Lembrando que o

simulador desenvolve o papel do sistema de visão e controle de baixo nível, durante as simulações, dois sockets são suficientes para estabelecer a comunicação entre o simulador e uma estratégias.

O simulador possui uma porta fixa para esperar por conexões da interface gráfica, a porta 4321. A thread responsável por lidar com a comunicação abre esta porta e espera por conexões. Quando uma conexão é estabelecida, a porta é trocada, é criado uma nova thread e a comunicação continua na nova thread com a porta que acabou de ser criada. A thread original (responsável por esperar conexões na porta 4321) está novamente livre para aguardar por novas interfaces gráficas.

5.2 Engine Física

A Engine Física é o coração do simulador. É nesta parte que são calculados os deslocamentos dos objetos, ocorrencias de gols e etc. Basicamente, a Engine física tem como entrada o que definiremos como cena atual do jogo. Uma cena é o conjunto das posições, velocidades, acelerações, ângulos, velocidades angulares, acelerações ângulares de todos os objetos (apenas a bola não possui referencia a ângulo) e etc no instante de tempo T. O resultado do processamento é uma nova cena do jogo, referente ao instante de tempo $T + \Delta T$, em que ΔT é o passo de tempo utilizado. Desta forma, para chegar ao resultado de um jogo inteiro, é necessário realizar várias iterações até que a soma dos Δt se equivalha ao tempo total de jogo.

Existem três tipos de objetos: campo, bola e robô. O campo é considerado como um conjunto de paredes. A bola é uma esfera perfeita. O terceiro tipo de objeto é o robô em formato cúbico de duas rodas. A complexidade do modelo físico adotado determina a qualidade da simulação e também o custo computacional para processar o modelo. Utilizar um modelo muito realista aumenta a qualidade da simulação mas também aumenta o custo computacional necessário. Um modelo físico pobre, por outro lado, produz uma simulação de qualidade insatisfatória mas com consumo computacional baixo.

Embora o simulador se comunique com as estratégias a cada $33ms$ de simulação, o passo de tempo ΔT utilizado dentro do simulador é menor. A questão é que se ΔT tiver um valor grande demais, pode acontecer de colisões não serem detectadas. Para explicar melhor vamos pegar um caso em que um robô e a bola estão em rota de colisão. Em uma das cenas os objetos estão para colidir. Então esta cena é passada para a Engine física, que não detectará nenhuma colisão pois nenhuma colisão ocorreu até este instante. Como não há colisão na cena a Engine física vai calcular os novos atributos dos objetos e gerar a cena seguinte. Se ΔT for grande demais, a nova cena

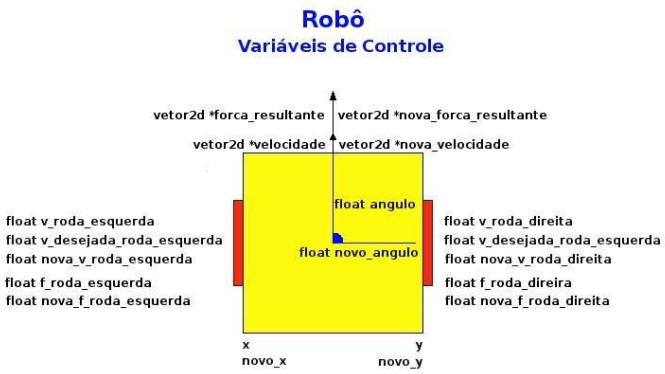

Figura 14: Atributos da classe robo

terá os dois objetos em posição tal como se eles fossem fantasma.... [escrever melhor essa pichorra]

Para construir a Engine foram criadas duas classes principais: a classe campo e a classe objeto. Além dessas duas foram criadas as classes robo e bola. Estas duas últimas herdam parte de seus atributos da classe objeto. A priori a classe campo também era uma classe filha da objeto. Porém, como o campo não possui atributos como velocidade ou aceleração, optou-se por separar a classe campo da objeto.

Os atributos de cada objeto são basicamente: posição, velocidade e força resultante. Além disso, para os robôs, tem os atributos torque, velocidade angular e ângulo. A cada período Δt de tempo todos os atributos são calculados novamente. Como os atributos de um objeto podem depender dos atributos de outros objetos (no caso de colisões) foi determinado que para cada atributo existe um outro atributo relativo ao próximo instante de tempo. Por exemplo, existem os atributo forca_resultante e nova_forca_resultante. Os dois atributos têm o mesmo significado porém, se referem a instantes de tempo diferentes. O primeiro (forca_resultante) se refere ao tempo T (atual), enquanto o atributo nova_forca_resultante se refere à força resultante que atua sobre o centro de massa do objeto no instante de tempo T + Δt.

Foram criados atributos secundários para os robôs. Atributos como v_roda_direita, que descreve a velocidade da roda esquerda. Esses atributos secundários tem são utilizados para o calculo dos outros atributos.

Para lidar com grandezas vetorias como força e velocidade foi necessário criar uma classe de vetores nomeada vetor2d. Foram implementados métodos para soma, subtação e multiplicação (vetorial e escalar) entre os vetores. Além disso também foi implementado um método para decomposição de

vetores.

A Engine permanece em em loop infinito. No estágio em que o Simulador está ainda não foram implementados métodos para reconhecer faltas, gols, fim de jogo e etc. O algoritmo que o Simulador segue para recalcular os atributos dos objetos é o descrito abaixo:

- Identificar colisões;
- tratar colisões;
- calcular força resultante e torque;
- calcular velocidade e velocidade angular;
- calcular nova posição e novo ângulo.

5.3 Detecção de Colisão

No simulador, estão modelados quatro tipos de colisões. Estas colisões são entre robô e campo, robô e bola, robô e robô, bola e campo.

A verificação de colisão entre campo e bola, por exemplo, é dividida em 16 verificações. Uma para cada parede do campo. Como a altura das paredes do campo é maior do que o raio da bola, podemos fazer uma modelagem bidimensional sem perder a exatidão. Neste caso a parede é modelada como um segmento de reta e a bola como uma circunferência. Para calcular a distância entre este dois objetos utiliza-se a Equação 9

$$d = \begin{vmatrix} b_x & b_y & 1 \\ P1_x & P1_y & 1 \\ P2_x & P2_y & 1 \end{vmatrix} \times \frac{1}{2} \tag{9}$$

O tempo, na simulação, é discreto, a colisão é detecteda quando os limites dos objetos se interseptam e formam uma região comum. Para evitar o perigo de um objeto passar pelo outro sem que a colisão seja detectada é necessário estimar a velocidade máxima a qual um objeto pode chegar. Baseado nessa velocidade é possível o tempo ideal de cada iteração. Assumindo que a velocidade máxima não irá ultrapassar $10m/s$, escolhemos $\Delta t = 1,111...$ ms. Isso significa que o simulador calcula todas as variáveis 900 vezes por segundo. Exitem dois motivos para se escolher exatamente o número 900. O primeiro é o fato de 900 ser um múltiplo de 30, que é o número de vezes ao qual o simulador se comunica com as estratégias por segundo. O segundo motivo é que dessa forma, nenhum objeto se move mais do que $1,111$ cm entre uma iteração e outra.

As colisões foram classificadas em quatro tipos: robô e campo, robô e robô, robô e bola e, finalmente, bola e campo. Colisões entre o robô e campo são detectadas quando o robô colide com alguma borda do campo. Nesta verificação, robô e campo são considerados objetos bidimensionais. Para detectar colisões entre dois robôs o simulador simplesmente verifica se cada um dos quatro cantos do robô está dentro do segundo robô. Para calcular se um ponto está dentro de um quadrado verificamos soma das áreas $S = \Delta EAB + \Delta EBC + \Delta ECD + \Delta EDA$, de acordo com a Figura 15. Se S for maior do que a área do robô significa que o ponto E está fora do robô e, portanto, não houve colisão com esta aresta.

Considere a Figura 16. Vamos definir $d_r = \frac{r_r}{cos(\alpha)}$ e $d_b = \frac{r_b}{cos(\alpha)}$. A distância limite $d_{limite} = d_r + d_b$, será usada para determinar se houve ou não colisão entre robô e bola.Só haverá colisão se d_{limite} for maior do que a distância entre o robô e a bola.

5.4 Tratamento de Colisões

O tratamento de colisões influi de modo significativo na qualidade da simulação. Entretando, modelos muito complexos são computacionalmente caros e modelos simples produzem simulações com qualidade inferior à admissível. Algumas simplificações podem ser feitas sem que a qualidade seja afetada, Por exemplo, uma vez que o campo tem uma posição fixa, após toda colisão com algum objeto não é necessário recalcular a posição do campo. Para todas as colisões, os objetos são tratados como sendo bidimensionais. Esta simplificação pode ser feita baseado no fato de que os robôs sempre estarão tocando o campo.

Uma vez que cada tipo de objeto é tratado de modo diferente, é necessário descrever um algoritmo para cada possível colisão. O primeiro passo é identificar o ponto de impácto, calcular as forças repulsivas que agem sobre os objetos. Após isso é necessário dividir essas forças em torque e força que age sobre o centro de massa do robô. Desta forma, é possível calcular a aceleração linear e ângular do robô [6].

5.5 Função de Tratamento de Erros

Para alguns tipos de colisões é difícil identificar o ponto em que ocorreu o impácto. Quando robôs batem no campo, um lado inteiro do robô pode estar em contato com a parede. A parte de tratamento de colisão não considera todas as possibilidades. Isso tornaria o modelo complexo demais. A solução

[6] Não é considerado a rotação da bola

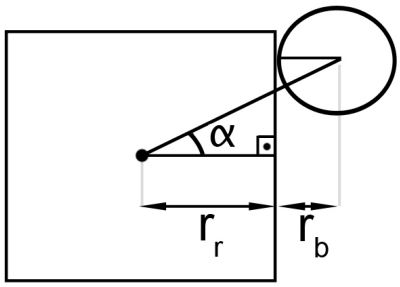

Figura 15: Verificar se um ponto está dentro de um quadrado

para este problema é criar uma função de tratamento de erros que corrige a situação toda vez que o tratamento de erros gera um resultado nitidamente errado. Basicamente, o que esta função faz é verificar novamente se os objetos estão em colisão e separá-los. Está função é executada logo após a função do tratamento de colisões, logo, se alguma colisão for detectada, significa que o tratamento de colisões falhou.

A função de tratamento de erros separa os objetos dois a dois, a medida que as colisões são detectadas. Esta separação occore sobre a reta determinada pelo centro de massa dos dois objetos em colisão. Após separar todos os objetos, verifica-se novamente se eles estão em colisão. Este procedimento é repetido até que nenhuma colisão é encontrada.

Esta função de tratamento de colisões não se baseia em nenhum modelo físico e, embora necessária, denigre a qualidade da simulação. Felizmente, durante os testes realizados, verificou-se que menos de 1% das situações necessitam desta função.

6 Conexão entre os Módulos

Toda as comunicação utilizam o protocolo TCP/IP e o USPDS é o servidor. Existem dois tipos de comunicação. O primeiro tipo é entre o simulador e a estratégia, descrito na seção 6.1 e o segundo tipo é entre o simulador e o visualizador do jogo, descrito na seção 6.2.

A interface de comunicação com os módulos deve prover comunicação entre entre entre o Kernel, as estratégias e a interface gráfica. É importante lembrar que o Simulador funciona independente da interface gráfica estar ou não funcionando mas, é necessário ter as duas estratégias conectadas.

Conforme o representado na Figura 17, a comunicação entre o Simulador

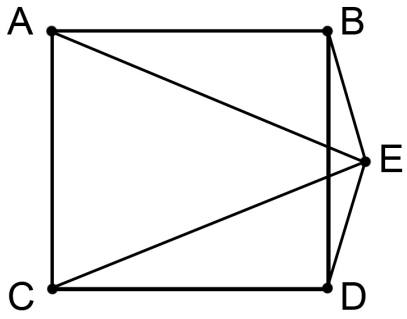

Figura 16: Colisão entre bola e robô

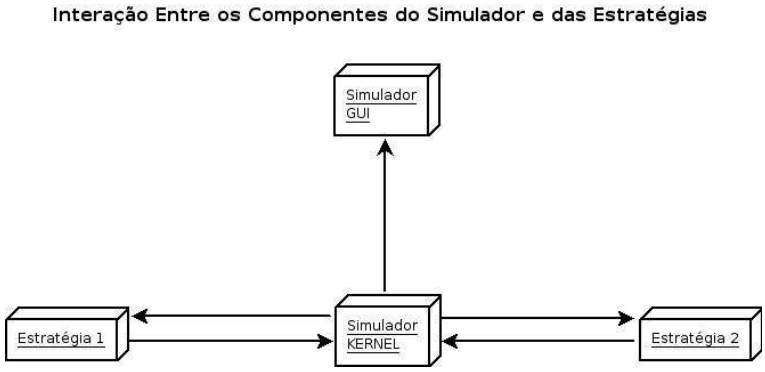

Figura 17: Representação da interface entre os módulos

e a interface gráfica é unidirecinal, ocorre apenas do Kernel para a GUI. Isso ocorre porquê a única função da interface gráfica é apenas mostrar o jogo. Todas as interações do usuario com o Simulador ocorrem antes de iniciar o jogo, por modo texto. Mas isso não impede que a GUI seja interativa. Para o GUI especificada no projeto de Computação Gráfica espera-se que ela amazene o jogo a medida que recebe os dados do Simulador para que seja possível manipular o tempo de apresentação do jogo [7]. Também faz parte da proposta da GUI que o usuário seja capaz de escolher de qual ângulo ele deseja ver o jogo.

A comunicação do Kernel com as estratrégias ocorre em ambos os sentidos. Cada estratégia recebe informações sobre o jogo a cada 33ms, processa essas informações e retorna as velocidades que as rodas de cada um dos jogadores do seu time deve ter. O Kernel não fica esperando por uma resposta de nenhuma das estratégias sendo que se uma estratégia ficar um ciclo sem enviar dados, o Kernel utiliza as informações enviadas no quadro mais recente.

Cada uma das threads possuem temporizadores. Trechos de código encarregados de sincronizar as thread. No estágio atual do Simulador, se esses trechos não existissem o Kernel processaria informações a uma velocidade maior do que a de tempo real de forma tal que as informações passadas para as estratégias seriam inúteis.

6.1 Comumicação com a Estratégia

Existem dois tipos de comunicação entre o simulador e a estratégia. No primeiro tipo os dados são enviados do simulador para a estratégia e no segundo é da estratégia para o simulador. O time 1 recebe dados pela porta 4322 e o time 2 pela porta 4324. Os dados são enviados na forma de um array de float: b_x, b_y, $pa[1]_x$, $pa[1]_y$, $pa[1]_a$, $pa[2]_x$, $pa[2]_y$, $pa[2]_a$, $pa[3]_x$, $pa[3]_y$, $pa[3]_a$, $pb[1]_x$, $pb[1]_y$, $pb[2]_x$, $pb[2]_y$, $pb[3]_x$, $pb[3]_y$. $pa[1]_x$, representa a coordenada x do robô 1 do time a. Os times são representados por a e b, time 1 e time 2, respectivamene. A numeração dos robôs vai de 1 a 3. Os tipos de coordenadas são x para eixo X do plano cartesiano, y eixo Y e a ângulo do robô. Os números são pontos flutuantes de precisão simples, de acordo com o padrão IEEE 754. Observe que o tamanho, em bytes de cada número é constante assim como a quantidade de números transmitidos. O tamanho deste tipo de mensagem é sempre 68 bytes.

As mensagens enviadas da estratégia para o simulador devem ser do tipo: $v[1]_x$, $v[1]_y$, $v[2]_x$, $v[2]_y$, $v[3]_x$, $v[3]_y$. $v[1]_x$, representa a componente X do

[7] Será possível escolher a velocidade com a qual o jogo é mostrado, e também será possível escolher a partir de qual ponto a animação deve começar.

vetor de velocidade do robô 1. Da mesma forma que a mensagem anterior, está mensagem é enviada como um array de pontos flutuantes de precisão simples. E o tamanho da mensagem é fixo e igual a 24 bytes. Por padrão, o time 1 utiliza a porta 4323 para enviar este tipo de mensagem e o time 2 utiliza a porta 4325.

6.2 Comunicação com a Interface Gráfica

Qualquer cliente que queira abrir uma conexão para receber dados sobre o jogo deve se conectar a porta 4321. Os dados são transmitidos unicamente no sentido de servidor para cliente. A cada instante de tempo é enviado um array de pontos flutuantes que representam b_x, b_y, $pa[1]_x$, $pa[1]_y$, $pa[1]_a$, $pa[2]_x$, $pa[2]_y$, $pa[2]_a$, $pa[3]_x$, $pa[3]_y$, $pa[3]_a$, $pb[1]_x$, $pb[1]_y$, $pb[1]_a$, $pb[2]_x$, $pb[2]_y$, $pb[2]_a$, $pb[3]_x$, $pb[3]_y$, $pb[3]_a$

7 Grid

Uma das vantagens do simulador é a possibilidade de avaliar estratégias baseado em uma análise estatísticas de repetidos jogos. Um jogo tem a duração de 10 minutos e não consegue fornecer dados suficientes para avaliar o desempenho de uma estratégia em um ambiente tão caótico. Com base nisso, surgiu a necessidade de fazer o Grid. O papel do Grid é gerenciar a execução de uma grande quantidade de jogos, distribuindo-os entre vários computadores.

Todo o sistema foi construído utilizando o sistema operacional GNU/Linux e faz parte da versão 1.1 do simulador. Durante o desenvolvimento e na fase de testes foram utilizados seis computadores. Entretanto, o Grid foi construído de forma tal a permitir que este número indefinido de computadores heterogêneos sejam utilizados para executar diversas simulações. Durante uma seção de testes, oito máquinas foram utilizadas para compor o Grid. A aplicação funcionou conforme o previsto em todos os testes, o que leva a crer que o limite máximo de computadores que o Grid suporta gerenciar está bem acima de oito.

7.1 Ferramentas e Métodos Utilizados

A princípio qualquer computador conectado a internet pode fazer parte do Grid, desde que seja acessível via SSH (Secure Shell) [23] e esteja executando o sistema operacional GNU/Linux. O computador central (o mestre) se conecta aos outros computadores usando o protocolo SSH. A partir do ponto em que

a conexão foi estabelecida, o mestre envia uma cópia do simulador para o escravo. Recebida a cópia, o escravo aguarda por ordens do mestre.

O mestre utiliza as bibliotecas OMP e PThread para gerenciar threads. No inicio da execução, o mestre cria um conjunto de threads. As tarefas são divididas entre as threads. O algoritmo básico seguido por cada thread é simples: para cada tarefa a ser executada, procure o computador menos ocupado e envie a tarefa a ele. Entretanto, quando o computador mais desocupado é encontrado, também é verificado se ele já não está sobrecarregado. Se o computador estiver sobrecarregado, a thread espera por 30 segundos e realiza uma nova pesquisa. Repete este procedimento até encontrar um computador disponível para executar a tarefa. Outro detalhe importante é que quando uma tarefa é enviada a um computador remoto, nada garante que este escravo executará a função até o final. Instabilidade na rede ou queda de energia, por exemplo, podem fazer com que um escravo receba a tarefa mas não a cumpra. Para se previnir desse tipo de fatalidade, a thread não passa para a próxima tarefa até se certificar de que a tarefa corrente tenha sido finalizada e o resultado recebido do escravo. Caso algo errado ocorra com a execução da tarefa, a thread envia a tarefa a outra máquina que possa executá-la.

Para medir o quanto um computador está sobrecarregado é utilizado o conceito "load average"do Linux. O "load average"é a soma dos processos na fila de execução com os processos já sendo executados pela máquina. O Grid utiliza o "load average"correspondentes aos últimos cinco minutos do sistema. Esta medida é considerada muito comum, simples e útil em aplicações para sistemas Linux/UNIX.

Para realizar as conexões utilizando ssh é necessário criar um par de chaves RSA [15] ou DSA [6] para permitir que a conexão aconteça sem que seja necessário interação com o usuário. O mestre mantem, em um arquivo de texto, uma lista dos computadores que podem ser acessados pelo Grid. O formato do arquivo conf/hosts é [usuario]@[hostname], por exemplo, *dmelo@lar1*. Em que *dmelo* é o nome do usuário e *lar1* é o nome pelo qual a máquina é acessível pela rede. O nome também pode ser substituido pelo IP. O Grid verifica constantemente quais máquinas estão disponíveis para planejar a distribuição de tarefas.

7.2 Testes e Resultados

O tempo que cada escravo consome para executar operações de comunicação com o mestre representam menos de 2% do tempo total de processamento do escravo. Isso significa que a performance do Grid é praticamente proporcional ao poder computacional empregrado para executá-lo. Durante a execução do

Grid, o mestre tenta manter o "load average"próximo a 6. Isso significa que máquinas com grande poder de processamento executarão mais simulações em paralelo do que máquinas com menor desempenho.

Em testes feitos com um computador que utiliza um processador Intel Core 2 Quad 2.0GHz Q6600, com 4 Gbytes de memória RAM, constatou-se que uma simulação consome, em média, 7 minutos. Cinco simuladores executanto ao mesmo tempo, consumiram aproximadamente 10 minutos para serem finalizados. Para mais do que cinco processos foi constatado que o desempenho (tempo médio por simulação) aumentava. A explicação do fato de que cinco simuladores processando concorrentemente consomem 2 minutos cada, em média, enquanto apenas um simulador consome 7 minutos é simples. Uma vez que foi utilizado um processador com quatro núcleos, apenas um simulador processando não era o suficiente para ocupar nem um núcleo por completo. Isso porque o gargalo durante esta simulação era a comunicação entre o simulador e as estratégias. A medida que o número de simulações foi sendo incrementado, os núcleos começaram a ser ocupados até que, utilizando cinco simuladores, foi atingido o total aproveitamento do processador. A partir de cinco simulações executando simultâneamente, espera-se que o desempenho diminua por conta do aumento da frequência de chaveamento entre os processos que o processador terá que fazer.

Um outro teste realizado em um computador com processador Intel Pentium 4 2.8GHz e 2 GBytes de memória RAM, teve como resultado que o ideal para esta máquina é executar apenas um simulador por vez. No caso desta máquina, com um simulador, o processador foi totalmente ocupado. Ao tentar executar dois processadores ou mais, a "load average"da máquina fica muito alto e o desempenho em processar os simuladores diminiu devido à constante troca de processos.

Como o Grid chega a executar várias simulações em uma mesma máquina ao mesmo tempo, ele não utiliza as portas padrões do simulador. Para cada simulação são escolhidas cinco portas aleatórias, entre 1025 e 65535. Após escolher as portas elas são testadas. Depois de conseguir cinco portas aptas a serem utilizadas, a ordem de executar a simulação utilizando estas portas é enviada para o mestre.

8 Heurística de Avaliação de Estratégias

Estratégias baseadas em Campos Potenciais (CP), Campos Potenciais Orientados (CPO) e Campos Potenciais Localmente Orientados (CPLO) existem constantes de ajuste que determinam aspéctos sobre o comportamento dos robôs. O ajuste destas constantes é feito de modo empírico e impreciso. Ter

uma função de avaliação que determine o quão eficiente é uma estratégia torna possível a utilização de técnicas de otimização.

8.1 Avaliação entre Duas Estratégias

É importante ressaltar que ainda que um time A sejá superior a um time B, não é necessariamente verdade que A ganharia todas as partidas jogadas contra B. O fato do time A ser melhor que o time B implica apenas que a probabilidade de A ganhar de B é maior do que a probabilidade de B ganhar de A. Outro fato importante que deve ser levado em consideração é que se um time A é considerado melhor do que um time B e o time B é considerado maior do que um time C então não é necessariamente verdade que A é melhor do que C. Entretanto, para a heurística proposta é importante comparar a performance de um time baseado em um segundo time.

Para calcular a probabilidade de um time A ganhar uma partida de um time B, vamos nos basear no Teorema do Limite Central [14], que diz que a soma de muitas variáveis aleatórias independentes com a mesma distribuição de probabilidade tende à Distribuição Normal. A idéia básica é simular diversos jogos entre os times A e B, afim de calcular a probabilidade de vitória.

8.2 Avaliação de uma Estratégia no Contexto Geral

O problema de se avaliar a estratégia A em relação a apenas uma outra estratégia B é que na melhor das hipóteses chegariamos a uma estratégia especializada em vencer a estratégia B. Para calcular a qualidade de uma estratégia em um contexto geral seria necessário realizar várias partidas entre a estratégia em questão e cada possível estratégia adversária. Felizmente, uma amostra com algumas estratégias já deve retornar um resultado aceitável sobre a qualidade da estratégia em um contexto geral.

Vamos denotar por S_n um conjunto constituido por n estratégias distintas [8]. O vetor n dimensional V representa o conjunto de probabilidades de vitória, de tal forma que a probabilidade de A vencer o time s_i é igual a v_i, para todo $i|1 \leq i \leq n$. O time A é considerado melhor do que o conjunto S se, e somente se, $\frac{\sum_{i=1}^{n} v_i}{n} > \frac{1}{2}$. Logo, quanto maior a média dos elementos do vetor V, melhor é time, em relação ao conjunto S.

A heurística de avaliação consiste em comparar a estratégia em questão com um conjunto de estratégias. Este método não é determinístico. Ou-

[8]Duas estratégias são consideradas distintas se produzem padrões de comportamento diferentes sobre os jogadores

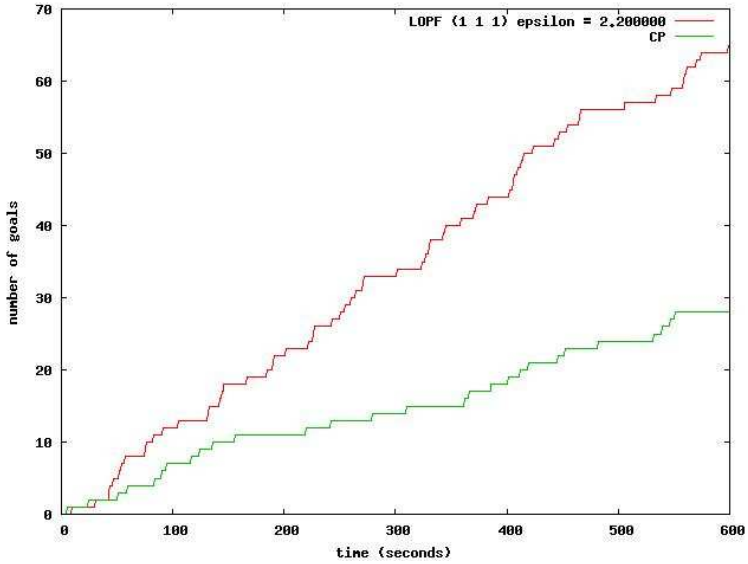

Figura 18: Resumo de um Jogo

tro problema é que a precisão deste método é inversamente proporcional ao tempo computacional necessário para processar o problema.

8.3 Testes e Resultados

Afim de obter dados sobre as simulações foi necessário instrumentar o código da estratégia de modo tal que a cada jogo fosse gerado um arquivo de texto representando contendo os instantes de tempo em que os gols ocorreram e um gráfico para melhor visualizar o resultado. A Figura 18 é um exemplo de um arquivo gerado após uma simulação.

9 Utilização

9.1 Pré-Requisitos

Para compilar e executar o USPDS são necessários:

- Sistema operacional GNU/Linux
- GCC 4.2.0 ou superior
- G++ 4.2.0 ou superior

- make
- libxvidcore4
- libxvidcore4-dev
- freeglut3
- freeglut3-dev
- xlibmesa-gl
- xlibmesa-gl-dev
- xlibmesa-glu
- libglu1-mesa-dev
- tar
- xterm [9]

9.2 Instalar

O modo de instalação é simples. Tenha o arquivo compactado do simulador no diretório corrente. Para descompactar use:

tar -zxvf uspdsXXX.tar.gz

Após isto, basta entrar na pasta e instalar:

cd uspds/

sh configure.sh

make

O comando *make install* não está funcionando corretamente no simulador. Deste modo, após instalar, para executar o simulador é necessarío entrar na pasta e utilizar o arquivo *fast_exec.sh*

sh fast_exec.sh

Este script irá inicializar o simulador com as duas estratégias contidas mais a interface gráfica. Para executar o simulador manualmente, utilizando suas próprias estratégia é necessário seguir os seguintes passos:

./bin/simulator_main

xterm -e ./bin/eyes localhost 4321

./bin/strategy localhost 1

./bin/strategy_pf2 localhost 2

[9] Xterm é o emulador de console recomendado para ser utilizado junto à interface gráfica Eyes

Se tudo ocorrer dentro do normal, após estes passos o simulador estará executando com a exibição do jogo na interface gráfica Eyes. Caso seja necessário matar algum processo referente ao simulador use o script terminar.sh

sh terminar.sh

9.3 Interface Gráfica - Eyes

O Eyes possui comandos para alterar o tipo de câmera, zoom e controlar o tempo de exibição do jogo. Todos os comandos são inseridos como teclas de atalho. abaixo está a lista dos atalhos para os modos de câmera.

9.3.1 Tipos de Câmera

- *f* - Modo de câmera free;
- *c* - Modo de câmera Chase;
- *o* - Modo de câmera Onboard;
- *F*, *C*, *O* - Modo de câmera padrão;

A câmera padão proporciona a visão sobre o campo. É permitido que a câmera se aproxime ou se afaste, utilizando as teclas + e -, respecticamente. Também pode-se transladar a câmera ao redor do campo utilizando as teclas *z*, no sentido anti-horário, ou *Z*, sentido horário. As teclas *x* e *X* também podem ser utilizadas para transladar a câmera em um eixo determinado pelo centro dos gols.

No modo de câmera Free, o usuário pode navegar pelo campo alterando a posição da câmera e alterando o ponto para o qual ela está apontada. Os comandos frente, traz, direita e esquerda são ativados pelas teclas *u*, *j*, *h* e *k*, respectivamente. Para subir e descer a câmera, os atalhos são *y* e *b*, respectivamente.

Ao ativar o modo Chase, a câmera fica centrada em um dos robôs. Para alterar o robô focado pela câmera utiliza a tecla *c*. A câmera pode se aproximar do robô através da tecla + ou se afastar, utilizando a tecla -.

A câmera Onboard proporciona a visão de uma câmera embarcada no robô. Para trocar de robô utilize a tecla *o*.

9.3.2 Controle Sobre o Tempo de Exibição do Jogo

O controle do tempo de exibição do jogo permite ao usuário adiantar, atrasar e parar o jogo, através das teclas . (ponto), , (virgula) e *p*, respectivamente.

9.4 Conclusão

O projeto já possui a versão 1.0 que é a primeira versão estável. O simulador oferece um ambiente realístico e com custo computacional aceitável para o desenvolvimento de estratégias nesta plataforma. A possibilidade de executar simulações sem a interface gráfica mostrou-se extremamente útil nos casos em que é necessário executar várias simulações. É importante ressaltar que o projeto continua em desenvolvimento para correção erros, adição de novas funcionalidades e melhoria das já existentes. Pretende-se, para a versão 2.0 do simulador, dar suporte a mais categorias de futebol de robôs.

Referências

[1] http://www.fira.net/soccer/simurosot/overview.htm.

[2] http://www.openmp.org.

[3] George Karypis Ananth Grama, Anshul Gupta and Vipin Kumar. *Introduction to Parallel Computing*. Addison Wesley, 2003.

[4] R. C. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, 4(8):92–112, 1989.

[5] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[6] James H. Burrows. Digital signature standard (dss). Technical report, Federal Information Processing Standards Publications (FIPS PUBS), May 1994.

[7] M. Chen and E. Foroughi. *RocoCup Soccer Server, Users Manual*, 2002.

[8] C. I. Connolly and R. A. Grupen. On the application of harmonic functions to robotics. *Journal of Robotic Systems*, 10:931–946, 1993.

[9] Christopher I. Connolly. Aplication of harmonic functions to robotics. *Proceedings of the 1992 IEEE International Symposium on*, 1992.

[10] G. FARIA. Tese de doutorado, icmc. page 92, 2006.

[11] G. Faria. *Uma arquitetura de controle inteligente para múltiplos robós*. PhD thesis, Instituto de Ciências Matemáticas e de Computação - USP, 2006.

[12] G. Faria and R. A. F. Romero. Estratégia para futebol de robôss baseada em campos potenciais. *I EnRi 2004, (CD-ROM), Anais do SBC 2004 XXIV Congresso da Sociedade Brasileira de Computação ISBN 85-88442-93-0*, 2004.

[13] G. Faria, R. A. F. Romero, E. Prestes, and M. A. P. Idiart. Comparing harmonic functions and potential fields in the trajectory control of mobile robots. *IEEE Conference on Robotics, Automation and Mechatronics, Singapore*, 1:762–767, 2004a.

[14] W. Feller. *Teoria das Probabilidades e Suas Palicações*. Edgar Blucher, São Paulo, 1976.

[15] B. Kaliski and J. Staddon. Pkcs 1: Rsa cryptography specifications. Request For Comments, Internet Engineering Task Force, October 1998.

[16] O. Kathib. Real-time obstacle avoidance for manipulators and mobile robots. *IEEE International Conference on Robotics and Automation, St. Louis, Missouri*, pages 500–505, 1985.

[17] G. Klancar. http://msc.fe.uni-lj.si/PublicWWW/Klancar/RobotSimulator.html.

[18] B. H. Krogh. A generalized potential field approach to obstacle avoidance. *International Robotics Research Conference Bethlehem, Pennsylvania*, 1986.

[19] Fujiki Morii. Distortion analysis on discrete laplacian operators by introducing random images. *Image and Graphics, 2004. Proceedings. Third International Conference on*, 18-20 Dec. 2004.

[20] W. S. Newman and N. Hogan. High speed robot control and obtacle avoidance. *Proceeding of the 1987 IEEE International, Raleigh, North Carolina*, pages 14–24, 1987.

[21] Edson Prestes. *Navegação Exploratória Baseada no Problema do Valor de Contorno*. PhD thesis, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, 2003.

[22] C. Thorpe. Path relaxation: Path planning for a mobile robot. *Technical Report CMU-RI-TR-84-05, Robotics Institute, Carnegie Mellon University, Pittsburg, PA*, 1984.

[23] T. Ylonen. The secure shell (ssh) authentication protocol. Request For Comments, Internet Engineering Task Force, January 2006.