Campos potenciais modificados aplicados ao controle de múltiplos robôs

Marcelo Oliveira da Silva

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura:_____

Campos potenciais modificados aplicados ao controle de múltiplos robôs

Marcelo Oliveira da Silva

Orientadora: **Profa. Dra. Roseli Aparecida Francelin Romero**

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Outubro de 2011

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Oc Oliveira da Silva, Marcelo
Campos potenciais modificados aplicados ao controle de múltiplos robôs / Marcelo Oliveira da Silva; orientadora Roseli Aparecida Francelin Romero -- São Carlos, 2011.
79 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2011.

1. planejamento de caminhos. 2. campos potenciais. 3. robótica móvel. 4. futebol de robôs. I. Aparecida Francelin Romero, Roseli, orient. II. Título.

Agradecimentos

A meu pai, minha mãe, minha irmã, meu irmão, minha avó, minha tia, minha cunhada, meu cunhado e meu sobrinho.

A minha orientadora, pois além do trabalho excepcional como minha orientadora, ser também minha protetora e, por acreditar no meu trabalho e potencial.

A meus colegas de laboratório, especialmente aqueles que passam madrugadas inteiras trabalhando junto comigo, despretensiosamente.

A todos os meus amigos e amigas, por serem quem são e, me ajudarem sempre que precisei.

A professores e funcionários do ICMC, com quem muito aprendi e, muita ajuda obtive.

A FAPESP e ao CNPq pelo apoio financeiro.

Resumo

Este trabalho aborda o problema de planejamento de caminhos em robótica móvel autônoma utilizando campos potenciais. Dentre as várias técnicas de campos potenciais para controlar robôs, encontram-se as técnica de Campos Potenciais de Khatib[1] (CP), Campo Potencial Harmônico (CPH), Campo Potencial Orientado (CPO) e Campo Potencial Localmente Orientado (CPLO). As técnicas CPH, CPO e CPLO são chamadas de técnicas baseadas em Problema de Valor de Contorno (*PVC*), pois são obtidas a partir de soluções de Equações Diferenciais Parciais (EDP) Elípticas em uma determinada condição de contorno, é obtido um sistema planejador de caminhos. Tais técnicas necessitam de uma etapa de solução de sistemas lineares, na qual se utiliza métodos iterativos, decorrentes da aplicação do método de diferenças finitas como solucionador das EDP. No presente trabalho, as técnicas de Campos Potenciais baseados em PVC foram estudadas e implementadas (usando processamento sequencial e paralelo), de modo a obter resultados de forma mais rápida e confiável. Foram utilizadas arquiteturas paralelas do tipo manycore. Finalmente, são feitas análises comparativas entre os vários métodos implementados. Todos os métodos estão prontos para serem incorporados tanto no simulador quanto nos times de robôs em desenvolvimento pelo grupo Warthog Robotics.

[1] Na literatura costuma ser chamado apenas de Campos Potenciais, o que pode gerar conflitos.

Abstract

This works details the task o path planning in autonomous mobile robots using potential fields techniques. Among potential fields techniques to control robots, there are Khatib's Potential Field[2] (KPF), Harmonic Potential Field (HPF), Oriented Potential Field (OPF) and Locally Oriented Potential Field (LOPF). The HPF, OPF and LOPF techniques are called Boundary Value Problem (BVP) based, bacause they are obtained from numerical solutions of Elliptic Partial Differential Equations (PDE) in a well-defined boundary condition.

These techniques go through a step of solving linear systems, in which is used iterative methods, that came from numerical solution of PDE.

In this work, potential fields BVP based was studied and coded (using sequential and parallel architectures), to obtain results more quickly and reliably.

And, finally, a comparative analyses of the various methods implemented are made. All methods are ready to be incorporated in the intelligent systens that are being developed by Warthog Robotics.

[2]In the literature, its common to call it as only Potential Fields, which may cause misconceptions

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. Aqueles 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 executem 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.

Em (Mackworth, 1993), foi sugerido que o ambiente de futebol de robôs é adequado para se avaliar o desempenho de técnicas de robótica móvel autônoma e áreas próximas, como por exemplo: Visão Computacional, Inteligência Artificial, Controle, Eletrônica, 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, para os quais uma grande variedade de tecnologias possam ser integradas e analisadas. Uma das vantagens oferecidas pela adoção de um problema padrão é que a avaliação do progresso de uma pesquisa pode ser claramente definida e acompanhada.

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.

Técnicas de Campos Potenciais baseadas em Problemas de Valor de Contorno, como será visto adiante, passam por uma etapa de solução de sistemas lineares, na qual se costuma utilizar algum método iterativo, como o relaxamento por Gauss-Seidel. Tais métodos são paralelizáveis e podem ficar mais rápido se assim for feito. Na última década, tem havido um crescente uso de arquiteturas do tipo *manycore* (com centenas de cores, unidades de processamento independentes), como as unidades de processamento gráfico, ou GPUs (do inglês, *Graphics Processing Unit*), em diversas áreas, mas, principalmente, na solução numérica de EDPs.

Até o início deste presente trabalho, não se encontrou na literatura trabalhos na área de planejamento de caminhos para robôs móveis que utilizassem GPUs.

1.1 Objetivos

Este trabalho teve por finalidade o estudo e aprimoramento de técnicas de planejamento de caminhos, que consistem em encontrar um caminho entre a posição atual do robô[2] e uma outra posição desejada, denominada *meta*. As

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

técnicas de planejamento de caminho abordadas neste trabalho são técnicas de *Campos Potenciais Baseados em Problema de Valor de Contorno*, em que a solução de uma *Equação Diferencial Parcial* (EDP), dada uma condição de contorno, gera um caminho.

1.2 Organização do Texto

Este texto está organizado como a seguir:

No Capítulo 2, são apresentados os principais conceitos de EDP Elípticas, bem como o Método das Diferenças Finitas, que transforma a EDP em um sistema de equações algébricas lineares, bem como os métodos de solução de sistemas lineares utilizados no decorrer deste trabalho.

No Capítulo 3, são apresentados os métodos de Campos Potenciais de Khatib, Campos Potenciais Harmônicos, Campos Potenciais Orientados e Campos Potenciais Localmente Orientados, bem como respectivos aspectos computacionais.

No Capítulo 4 são apresentados aspectos da arquitetura CUDA e de GPUs, bem como alguns detalhes da implementação dos métodos paralelizados em CUDA.

No Capítulo 5, são apresentados conceitos de robótica necessários ao entendimento do projeto. Também é apresentado o projeto de micro-robôs móveis atualmente em desenvolvido numa parceria entre o Instituto de Ciências Matemáticas e de Computação (ICMC) e a Escola de Engenharia de São Carlos (EESC)

No Capítulo 6, são apresentados os testes de desempenho dos métodos implementados neste trabalho, bem como uma análise dos resultados.

No Capítulo 7, são destacadas as principais contribuições do presente trabalho, bem como as limitações e trabalhos futuros.

CAPÍTULO

2

Equações Diferenciais Parciais Elípticas

Conforme o trabalho de (Smith, 1992) e (Ames, 1969), *Equações Diferenciais Parciais Elípticas*[1] surgem, geralmente, a partir de problemas de equilíbrio que, por sua vez, são problemas de *estado estacionário*[2], nos quais a configuração de equilíbrio P em um domínio Ω é determinado através da solução de uma equação diferencial dada pela Equação 2.1 sujeita a condições de contorno, isto é, P tem valores especificados *à priori*, na borda de Ω ($\overline{\Omega}$).

$$\mathcal{L}(P) = f \tag{2.1}$$

em que $\mathcal{L}$ é um operador diferencial elíptico.

Uma definição mais rigorosa de EDP Elípticas é encontrada no trabalho de (Gilbarg e Trudinger, 1983), em que é dito que a elipticidade de uma equação diferencial é definida como a matriz de coeficientes $[a^{ij}]$ da Equação 2.2 ser *positiva definida* no domínio Ω.

$$\mathcal{L}(P) \equiv a^{ij}(x)\mathcal{D}_{ij}P + b^{i}(x)\mathcal{D}_{i}P + c(x)P = f(x) \tag{2.2}$$

para $i, j = 1, 2$.

Uma das equações mais utilizadas é a *Equação de Laplace* (Equação 2.3) (aqui retratada em duas dimensões). (Ames, 1969) diz que a equação de Laplace modela potencial gravitacional ou eletrostático, em pontos do espaço

[1] EDP Elíptica
[2] Do inglês *Steady-State*

vazio.

$$\frac{\partial^2 P(x,y)}{\partial x^2} + \frac{\partial^2 P(x,y)}{\partial y^2} = 0 \tag{2.3}$$

Encontrar soluções explícitas de um *Problema de Valor de Contorno* (PVC) nem sempre é possível. De acordo com (Smith, 1992), muitos problemas encontrados na prática não são simples e não podem ser solucionados de forma a resultar em uma solução explícita ou, mesmo que se possua a solução explícita, ela pode ser tão complicada que se prefere utilizar uma solução numérica.

Conforme (Larsson e Thomée, 2005), um dos primeiros métodos desenvolvidos para obter soluções numéricas de EDP é o método de diferenças finitas, que consiste em procurar uma solução aproximada nos pontos de uma grade regular finita de pontos, e uma aproximação da EDP é realizada através da substituição de derivadas por diferenças finitas correspondentes. Então, isto reduz o PVC a um sistema linear de equações algébricas.

2.1 Diferenças Finitas

Em (Smith, 1992), o método das diferenças finitas é desenvolvido, em um primeiro momento, para funções de uma única variável. Posto desta forma, seja $P(x)$ com $x \in \mathcal{R}^2$. Define-se um espaçamento $\Delta x > 0$. Aproxima-se o valor de $P(x)$ quando $x = j \times \Delta x$ pelo número p_j (Equação 2.4), indexados pelo inteiro j.

$$p_j \approx P(j\Delta x) \tag{2.4}$$

Então, três aproximações comuns para a primeira derivada $\frac{dP(x)}{dx}$ são:

- Diferenças Atrasadas
$$\frac{p_j - p_{j-1}}{\Delta x} \tag{2.5}$$

- Diferenças Avançadas
$$\frac{p_{j+1} - p_j}{\Delta x} \tag{2.6}$$

- Diferenças Centradas
$$\frac{p_{j+1} - p_{j-1}}{2\Delta x} \tag{2.7}$$

Cada uma das Equações 2.5 - 2.7 é uma aproximação correta, que pode ser provada a partir da expansão de Taylor (Smith, 1992) (se garantidas as condições de regularidade apresentadas em (Guidorizzi, 2001)):

$$P(x + \Delta x) = P(x) + (\Delta x)\frac{dP(x)}{dx} + \frac{\Delta x^2}{2}\frac{d^2P(x)}{dx^2} + \frac{\Delta x^3}{6}\frac{d^3P(x)}{dx^3} + \mathcal{O}(\Delta x^4)$$

E, substituindo Δx por $-\Delta x$, obtém-se

$$P(x-\Delta x) = P(x) - (\Delta x)\frac{dP(x)}{dx} + \frac{\Delta x^2}{2}\frac{d^2P(x)}{dx^2} - \frac{\Delta x^3}{6}\frac{d^3P(x)}{dx^3} + \mathcal{O}(\Delta x^4)$$

A partir destas duas expansões, é possível deduzir que

$$\begin{aligned}
\frac{dP(x))}{dx} &= \frac{P(x) - P(x-\Delta x)}{\Delta x} + \mathcal{O}(\Delta x) \\
&= \frac{P(x+\Delta x) - P(x)}{\Delta x} + \mathcal{O}(\Delta x) \\
&= \frac{P(x+\Delta x) - P(x-\Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2)
\end{aligned}$$

Substituindo x por $j\Delta x$, pode ser visto que as Equações 2.5-2.6 são aproximações de ordem $\mathcal{O}(\Delta x)$ e a Equação 2.7 é uma aproximação de ordem $\mathcal{O}(\Delta x^2)$.

Para a segunda derivada, a aproximação mais comum é a de diferença central de segunda ordem (Equação 2.8)

$$\frac{d^2P(x)}{dx^2} \approx \frac{p_{j+1} - 2p_j + p_{j-1}}{\Delta x^2} \tag{2.8}$$

A justificativa é a mesma acima

$$\frac{d^2P(x)}{dx^2} = \frac{P(x+\Delta x) - 2P(x) + P(x-\Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^4)$$

Portanto, a aproximação da Equação 2.8 é de ordem $\mathcal{O}(\Delta x^4)$.

Para funções de duas ou mais variáveis, aplica-se o método em cada uma das variáveis

$$\begin{aligned}
\frac{\partial P(x,y)}{\partial x} &= \frac{P(x+\Delta x, y) - P(x,y)}{\Delta x} + \mathcal{O}(\Delta x) \\
&\approx \frac{p_{i,j+1} - p_{i,j}}{\Delta x}
\end{aligned}$$

ou

$$\begin{aligned}
\frac{\partial^2 P(x,y)}{\partial x^2} &= \frac{P(x+\Delta x, y) - 2P(x,y) + P(x-\Delta x, y)}{\Delta x^2} + \mathcal{O}(\Delta x^4) \\
&\approx \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2}
\end{aligned}$$

e assim por diante.

Por exemplo, a partir da Equação de Laplace (Equação 2.3) obtemos a aproximação dada pela Equação 2.9, que por sua vez origina a equação de diferen-

ças dada pela Equação 2.10.

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + \mathcal{O}(\Delta x^4) + \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta y^2} + \mathcal{O}(\Delta y^4) = 0 \tag{2.9}$$

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta y^2} = 0 \tag{2.10}$$

Se o domínio da Equação 2.10 for discretizado em uma malha quadrada, isto é, com $\Delta x = \Delta x = \delta$, obtém-se a Equação 2.11.

$$p_{i,j} = \frac{p_{i,j+1} + p_{i,j-1} + p_{i+1,j} + p_{i-1,j}}{4} \tag{2.11}$$

2.1.1 Esquemas de Discretização

Quando a EDP elíptica contém derivadas parciais de primeira ordem (como na Equação 2.12), existem várias formas para a discretização desta (de Oliveira Fortuna, 2000), como, por exemplo, centradas, upwind de primeira ordem[3], upwind de segunda ordem[4], QUICK, MSOU, HLPA e híbridos. Neste trabalho, são utilizadas as centradas e upwind.

As diferenças centradas são as mais intuitivas. Utiliza a Equação 2.7 para aproximar a derivada parcial de primeira ordem, como na Equação 2.13.

$$\frac{\partial^2 P(x,y)}{\partial x^2} + f(x,y)\frac{\partial P(x,y)}{\partial x} = 0 \tag{2.12}$$

$$\frac{\partial^2 P(x,y)}{\partial x^2} + f(x,y)\frac{\partial P(x,y)}{\partial x} = 0$$

$$\equiv \quad \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + f_{i,j}\frac{p_{i,j+1} - p_{i,j-1}}{2\Delta x} = 0$$

$$\equiv \quad p_{i,j+1} - 2p_{i,j} + p_{i,j-1} + f_{i,j}\frac{\Delta x}{2}(p_{i,j+1} - p_{i,j-1}) = 0$$

$$\equiv \quad p_{i,j} = (\frac{2 + f_{i,j}\Delta x}{4})p_{i,j+1} + (\frac{2 - f_{i,j}\Delta x}{4})p_{i,j-1} \tag{2.13}$$

Já as diferenças upwind, utilizam as Equações 2.5 - 2.6, dependendo do sinal de $f(x,y)$ (ou $f_{i,j}$, discretizado). Caso o termo acompanhando a derivada parcial de primeira ordem seja positivo ($f(x,y) > 0$), utilizar a Equação 2.6,

[3]Ou simplesmente upwind
[4]Ou SOU - Second Order Upwind

senão, utilizar a Equação 2.5.

Caso $f(x,y) > 0$

$$\frac{\partial^2 P(x,y)}{\partial x^2} + f(x,y)\frac{\partial P(x,y)}{\partial x} = 0$$

$$\equiv \quad \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + f_{i,j}\frac{p_{i,j+1} - p_{i,j}}{\Delta x} = 0$$

$$\equiv \quad p_{i,j+1} - 2p_{i,j} + p_{i,j-1} + (f_{i,j}\Delta x)(p_{i,j+1} - p_{i,j}) = 0$$

$$\equiv \quad p_{i,j} = \frac{(1 + f_{i,j}\Delta x)p_{i,j+1} + p_{i,j-1}}{2 + f_{i,j}\Delta x} \tag{2.14}$$

Caso $f(x,y) < 0$

$$\frac{\partial^2 P(x,y)}{\partial x^2} + f(x,y)\frac{\partial P(x,y)}{\partial x} = 0$$

$$\equiv \quad \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + f_{i,j}\frac{p_{i,j} - p_{i,j-1}}{\Delta x} = 0$$

$$\equiv \quad p_{i,j+1} - 2p_{i,j} + p_{i,j-1} + (f_{i,j}\Delta x)(p_{i,j} - p_{i,j-1}) = 0$$

$$\equiv \quad p_{i,j} = \frac{p_{i,j+1} + (1 - f_{i,j}\Delta x)p_{i,j-1}}{2 - f_{i,j}\Delta x} \tag{2.15}$$

2.2 Solução Numérica de Sistemas Lineares

Conforme (Burden et al., 1978), sistemas lineares estão associados com muitos problemas na engenharia e nas ciências, bem como com aplicações da matemática às ciências sociais e ao estudo quantitativo de problemas de negócios e da economia. Particularmente, a solução numérica de Equações Diferenciais Parciais depende de uma etapa de solução numérica de sistemas lineares.

Métodos numéricos para solução de sistemas de quações lineares são divididos principalmente em dois grupos (Franco, 2008):

Métodos Diretos São aqueles métodos que forneceriam a solução exata, não fossem os erros de arredondamento, com um número finito de passos.

Métodos Iterativos São aqueles que permitem obter a solução de um sistema

linear com uma dada precisão, através de um processo infinito convergente.

Em (Franco, 2008), é ressaltado o fato de que os erros de arredondamento dos métodos diretos, em sistemas de grande porte, podem tornar a solução sem significado, enquanto que em métodos iterativos os erros não se acumulam. Já em (Burden et al., 1978) é dito que matrizes esparsas[5] são frequentemente resolvidas utilizando-se técnicas iterativas.

Tais características fazem dos métodos iterativos os mais apropriados para a solução numérica de sistemas lineares decorrentes de EDP.

Neste trabalho, alguns métodos iterativos serão abordados e, sempre que possível, comparar uma versão sequencial com uma versão paralela deste.

Um método iterativo consiste em uma técnica para resolver um determinado sistema linear $Ax = b$ (Equação 2.16) que a partir de uma aproximação inicial $x^{(0)}$ para a solução x, gera uma sequência de vetores $\{x^{(k)}\}_{k=1}^{\infty}$, que converge para x. Estas técnicas transformam o sistema $Ax = b$ em um sistema equivalente na forma da Equação 2.17, para alguma matriz T e algum vetor c fixos. Após o vetor inicial $x^{(0)}$ ter sido obtido, a sequência de vetores para aproximar a solução é gerada calculando-se como na Equação 2.18

$$\begin{cases} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & + & \vdots & + & \ddots & + & \vdots & = & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{cases} \tag{2.16}$$

$$x = Tx + c \tag{2.17}$$

$$x^k = Tx^{(k-1)} + c \qquad para \quad k = 1, 2, 3, \ldots \tag{2.18}$$

2.2.1 Jacobi-Richardson

A matriz A do sistema linear da Equação 2.16, pode ser decomposta como na Equação 2.19.

$$A = L + D + R \tag{2.19}$$

onde $L = (l_{ij})$ é uma matriz triangular inferior formada pela parte inferior da matriz A, $D = (d_{ij})$ é uma matriz formada pela diagonal de A e $R = (r_{ij})$ é uma matriz triangular superior formada pela parte superior da matriz A, ou seja:

$$l_{ij} = \begin{cases} a_{ij}, & se \quad i > j \\ 0, & c.c. \end{cases}$$

[5]com um grande volume de entradas iguais a zero

$$d_{ij} = \begin{cases} a_{ij}, & se \quad i = j \\ 0, & c.c. \end{cases}$$

$$r_{ij} = \begin{cases} a_{ij}, & se \quad i < j \\ 0, & c.c. \end{cases}$$

Supondo que $det(D) \neq 0$, o sistema linear original (Equação 2.16) pode ser transformado na Equação 2.20. Então, pode ser definido o processo iterativo da Equação 2.21, que é chamado de *Método de Jacobi-Richardson*. Pode ser visto que o processo iterativo da Equação 2.21 está na forma padrão da Equação 2.18, com $T = -D^{-1}(L + R)$ e $c = D^{-1}b$.

$$\begin{aligned} Ax &= b \Rightarrow \\ (L + D + R)x &= b \Rightarrow \\ Dx &= -(L + R)x + b \Rightarrow \\ x &= -D^{-1}(L + R)x + D^{-1}b \end{aligned} \tag{2.20}$$

$$x^{(k)} = -D^{-1}(L + R)x^{(k-1)} + D^{-1}b \tag{2.21}$$

Por hipótese, $a_{ii} \neq 0$, pois foi suposto que $det(D) \neq 0$. Então, antes de se decompor a matriz A em $L+D+R$, cada linha da matriz A no sistema da Equação 2.16 pode ser dividida pelo correspondente termo da diagonal principal, resultando na decomposição da Equação 2.22, onde A^* é a matriz obtida após divisão, I é a matriz identidade, L^* e R^* são as matrizes triangular inferior e superior, respectivamente, de A^*, que são dados como nas Equações 2.23 - 2.24.

$$A^* = L^* + I + R^* \tag{2.22}$$

$$l^*_{ij} = \begin{cases} a^*_{ij} = \dfrac{a_{ij}}{a_{ii}}, & se \quad i > j \\ 0, & c.c. \end{cases} \tag{2.23}$$

$$r^*_{ij} = \begin{cases} r^*_{ij} = \dfrac{a_{ij}}{a_{ii}}, & se \quad i < j \\ 0, & c.c. \end{cases} \tag{2.24}$$

Assim, o processo iterativo da Equação 2.21, pode ser reescrito como na Equação 2.25, onde b^* é como na Equação 2.26, e é o vetor b obtido após a divisão de A, anteriormente citada.

$$x^{(k)} = -(L^* + R^*)x^{(k-1)} + b^* \tag{2.25}$$

$$b_i^* = \frac{b_i}{a_{ii}}, \qquad para \quad i = 1, 2, 3, \ldots \tag{2.26}$$

Pelo processo iterativo da Equação 2.25, não é necessário calcular a matriz D^{-1}. Este processo pode ser visto expandido na Equação 2.27. Neste processo, devido a atualização simultânea de todas as equações do sistema linear, este método também é conhecido por *Método dos Deslocamentos Simultâneos* ou, simplesmente, *Método de Jacobi* (Franco, 2008).

$$\begin{cases} x_1^{(k+1)} = - a_{12}^* x_2^{(k)} - a_{13}^* x_3^{(k)} - \ldots - a_{1n}^* x_n^{(k)} + b_1^* \\ x_2^{(k+1)} = - a_{21}^* x_1^{(k)} - a_{23}^* x_3^{(k)} - \ldots - a_{2n}^* x_n^{(k)} + b_2^* \\ x_3^{(k+1)} = - a_{31}^* x_1^{(k)} - a_{32}^* x_2^{(k)} - \ldots - a_{3n}^* x_n^{(k)} + b_3^* \\ \vdots = - \vdots - \vdots - \ddots - \vdots + \vdots \\ x_n^{(k+1)} = - a_{n1}^* x_1^{(k)} - a_{n2}^* x_2^{(k)} - \ldots - a_{n,n-1}^* x_{n-1}^{(k)} + b_n^* \end{cases} \tag{2.27}$$

2.2.2 Gauss-Seidel

Seja o sistema linear $Ax = b$, tal como no método de Jacobi-Richardson, este sistema pode ser reescrito como $(L^* + I + R^*)x = b^*$ (ver Equação 2.22). Tal sistema, agora pode ser transformado como na Equação 2.28, que agora tem a forma da Equação 2.17, com $T = -(L^* + I)^{-1}R^*$ e $c = (L^* + I)^{-1}b^*$.

$$\begin{aligned} Ax &= b \Rightarrow \\ (L^* + I + R^*)x &= b^* \Rightarrow \\ (L^* + I)x &= -R^*x + b^* \Rightarrow \\ x &= -(L^* + I)^{-1}R^*x + (L^* + I)^{-1}b^* \end{aligned} \tag{2.28}$$

$$x^{(k)} = -(L^* + I)^{-1}R^*x^{(k-1)} + (L^* + I)^{-1}b^* \tag{2.29}$$

O processo iterativo da Equação 2.29 é conhecido como *Método de Gauss-Seidel*. Pela Equação 2.29, pode ser visto que o método, nesta forma, necessita do cálculo de $(L^* + I)^{-1}$. Este processo pode ser simplificado como na Equação 2.30

$$\begin{aligned} x^{(k)} &= -(L^* + I)^{-1}R^*x^{(k-1)} + (L^* + I)^{-1}b^* \Rightarrow \\ (L^* + I)x^{(k)} &= -R^*x^{(k-1)} + b^* \Rightarrow \\ x^{(k)} &= -L^*x^{(k)} - R^*x^{(k-1)} + b^* \end{aligned} \tag{2.30}$$

Pelo processo iterativo da Equação 2.30, não é necessário calcular a matriz $(L^* + I)^{-1}$. Este processo pode ser visto expandido na Equação 2.31. Este processo iterativo é também é conhecido por *Método dos Deslocamentos Sucessivos*, por utilizar componentes x_i^k no cálculo de x_j^k, para $j > i$. (Franco,

2008).

$$\begin{cases} x_1^{(k+1)} &= - & a_{12}^* x_2^{(k)} & - & a_{13}^* x_3^{(k)} & - & \dots & - & a_{1n}^* x_n^{(k)} & + & b_1^* \\ x_2^{(k+1)} &= - & a_{21}^* x_1^{(k+1)} & - & a_{23}^* x_3^{(k)} & - & \dots & - & a_{2n}^* x_n^{(k)} & + & b_2^* \\ x_3^{(k+1)} &= - & a_{31}^* x_1^{(k+1)} & - & a_{32}^* x_2^{(k+1)} & - & \dots & - & a_{3n}^* x_n^{(k)} & + & b_3^* \\ \vdots &= - & \vdots & - & \vdots & - & \ddots & - & \vdots & + \vdots & \\ x_n^{(k+1)} &= - & a_{n1}^* x_1^{(k+1)} & - & a_{n2}^* x_2^{(k+1)} & - & \dots & - & a_{n,n-1}^* x_{n-1}^{(k+1)} & + & b_n^* \end{cases} \tag{2.31}$$

2.2.3 Weighted Jacobi

O Método Weighted Jacobi consiste em uma modificação no Método de Jacobi com o objetivo de acelerar sua convergência. Tal modificação traduz-se em uma ponderação entre $x^{(k)}$ e $x_J^{(k+1)}$ (aproximação obtida a partir do método de Jacobi) tal como na Equação 2.32

$$x^{(k+1)} = x^k + \omega \left(x_J^{(k+1)} - x^k \right) \qquad \text{com } 0 < \omega < 2 \tag{2.32}$$

Realizando manipulações algébricas é possível mostrar que a forma padrão (Equação 2.17) do método WJ resulta em $T = (1-\omega)I + \omega \left(D^{-1}\left(L+R\right)\right)$ e $c = -\omega D^{-1} b$.

Se $\omega = 1$, o método WJ recai no método de Jacobi. A escolha da constante ω ótima depende das propriedades da matriz A da Equação 2.16.

2.2.4 Successive Over-Relaxation

Tal como na Sessão 2.2.3, o método de Gauss-Seidel pode ser modificado, criando o método Successive Over-Relaxation (ou simplesmente SOR), como na Equação 2.33.

$$x^{(k+1)} = (1-\omega)x^k + \omega x_{GS}^{(k+1)} \qquad \text{com } 0 < \omega < 2 \tag{2.33}$$

Realizando manipulações algébricas é possível mostrar que a forma padrão (Equação 2.17) do método SOR resulta em $T = (D - \omega L)^{-1}[(1-\omega)D + \omega R]$ e $c = \omega(D - \omega L)^{-1} b$.

Analogamente a Sessão 2.2.3, se $\omega = 1$, o método SOR recai no método de Gauss-Seidel. A escolha da constante ω ótima depende das propriedades da matriz A da Equação 2.16.

2.3 Considererações Finais

Neste capítulo, foram apresentados os principais conceitos relacionados a Equações Diferenciais Parciais Elípticas, que serão necessários para o entendimento de alguns métodos de planejamento de caminhos, a saber: Campos Potenciais Harmônicos, Campos Potenciais Orientados e Campos Potencias Localmente Orientados.

Com a EDP transformada em uma equação de diferenças em um domínio discreto, a molécula computacional (por exemplo, a Equação 2.11) pode ser aplicada em cada ponto do domínio, gerando um sistema linear.

Então, o sistema linear pode ser resolvido utilizando qualquer um dos métodos da Seção 2.2.

CAPÍTULO

3

Campos Potenciais

A idéia de imaginar forças (como na física) atuando sobre um robô, de forma a resultar em um comportamento, foi sugerida por (Khatib, 1980).

Tais técnicas são conhecidas como Campos Potenciais por construirem uma *função potencial*, na qual obstáculos tem potenciais repulsivos e metas tem potenciais atrativos. Seguindo a direção do gradiente descendente, é sempre encontrada uma região de baixo potencial, isto é, longe de obstáculos e cada vez mais próximo da meta. Algumas técnicas podem gerar mínimos locais espúrios (uma meta é um mínimo local não-espúrio) na função potencial, que pode acabar comprometendo a eficácia da técnica, de forma que, mesmo que exista um caminho até a meta, este não seja encontrado.

Neste trabalho serão abordadas três destas técnicas: Campos Potenciais Harmônicos, Campos Potenciais Orientados e Campos Potenciais Localmente Orientados.

Para facilitar a compreensão destes métodos, antes será apresentada uma técnica de campos potenciais mais antiga e simples, denominada *Campos Potenciais de Khatib*.

3.1 Campos Potenciais de Khatib

Neste método, obstáculos exercem forças repulsivas e a meta aplica uma força atrativa sobre o robô e, com estas calcula-se a força resultante sobre o robô.

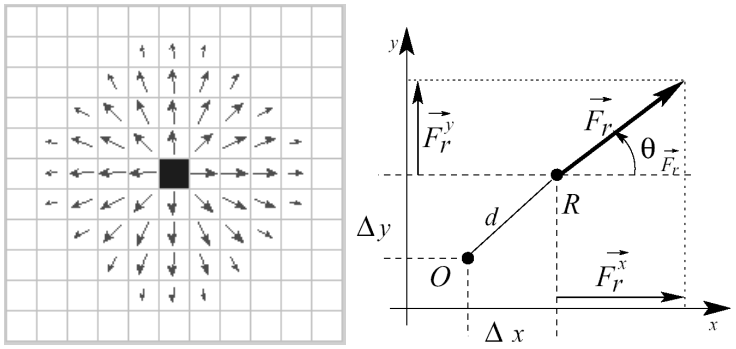
Tendo em vista que este é um método de baixo custo computacional (pois necessita apenas somar vetores), esta técnica tem sido muito utilizada em

planejamento de trajetórias de robôs móveis que necessitam navegar por ambientes desconhecidos e/ou dinâmicos evitando colisões com obstáculos.

Até meados da década de 80, o método de campos potenciais era utilizado com um modelo conhecido de mundo, i.e.,com formas geométricas pré-definidas representando obstáculos ou, então, o caminho era gerado *a priori*. No entanto, (Brooks, 1986; Arkin, 1989) foram os pioneiros em utilizar a técnica de campos potenciais de forma reativa, ou seja, para cada novo passo do robô, uma nova força resultante seria calculada para direcionar seu movimento. Esta característica reativa tornou o método muito eficiente para robôs realizarem tarefas em ambientes desconhecidos e/ou com dinamismo. Dentro deste paradigma, a cada instante de tempo, sensores captam o estado do ambiente e então uma força resultante é calculada para direcionar a trajetória do robô.

A seguir, será mostrado como (Faria, 2006) realizou os cálculos das forças de repulsão, atração à meta e da força resultante, embora outros modelos de forças vetoriais possam ser criados para representar outros tipos de comportamentos.

FORÇA DE REPULSÃO Na Figura 3.1(a) mostra-se que a força de repulsão decai ao se afastar de um obstáculo, pois a força de repulsão ($\overrightarrow{F_r}$) é definida como um vetor cujo módulo é inversamente proporcional ao quadrado da distância (d) entre o robô (R) e objeto observado (O) (Figura 3.1(b)). O vetor



(a) Força repulsiva na vizinhança de um obstáculo

(b) Força repulsiva do objeto O sobre um robô localizado em R

Figura 3.1: Força Repulsiva(Faria, 2006)

$\overrightarrow{F_r}$ também pode ser representado por suas componentes: módulo e direção, apresentados na Equação (3.1).

$$|\overrightarrow{F_r}| = \frac{Q}{d^2} \quad e \quad \theta_{\overrightarrow{F_r}} = \arctan\left(\frac{-\Delta y}{-\Delta x}\right) \tag{3.1}$$

sendo que: Q representa um escalar constante de repulsão; $\arctan$ é uma função na qual se considera os sinais de Δx e Δy para fornecer o ângulo correto

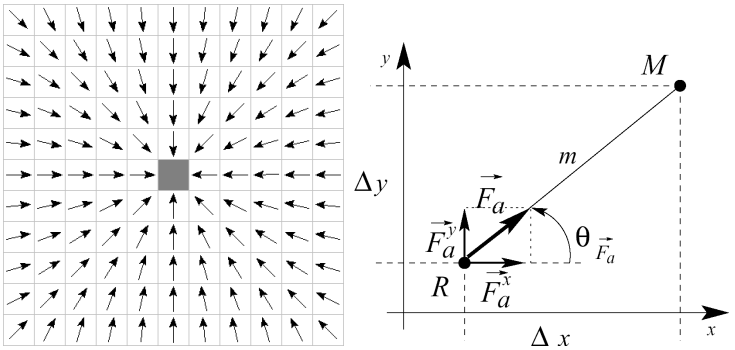
cuja tangente seja $(-\Delta y/-\Delta x)$; e os sinais negativos de Δx e Δy fornecem uma direção oposta ao objeto detectado (O).

Para calcular a força de repulsão para vários obstáculos, deve-se fazer o somatório dos vetores das forças de repulsão geradas, como mostrado na Equação.(3.2).

$$\overrightarrow{F_R} = \sum_i \left(\overrightarrow{F_r}\right)_i \tag{3.2}$$

onde $\left(\overrightarrow{F_r}\right)_i$ é a força de repulsão agindo sobre o robô devido ao objeto i.

FORÇA DE ATRAÇÃO Na Figura 3.2(a) e Figura 3.2(b) mostra-se o campo potencial de atração, no qual deve-se considerar $|\overrightarrow{F_a}|$ constante para que o agente seja atraído pela meta mesmo estando distante da mesma (Equação (3.3)).



(a) Força atrativa constante

(b) Força atrativa entre o robô R e a meta M

Figura 3.2: Força Atrativa (Faria, 2006)

$$|\overrightarrow{F_a}| = C \quad e \quad \theta_{\overrightarrow{F_a}} = \left(\frac{\Delta y}{\Delta x}\right) \tag{3.3}$$

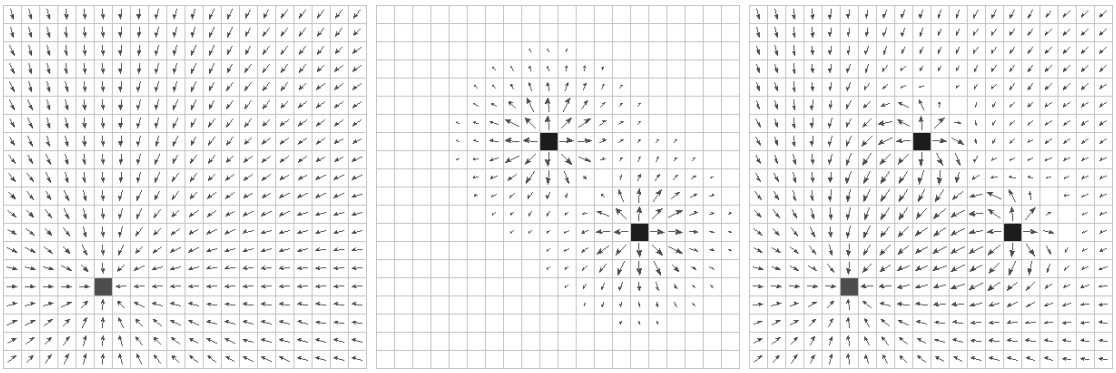
no qual C representa um escalar constante de atração. Note que Δx e Δy diferem da equação (3.1) por não estarem acompanhados do sinal negativo. Como resultado, a direção do vetor força de atração será na direção da meta.

FORÇA RESULTANTE Ao realizar a soma dos vetores força de atração e força de repulsão obtém-se a força resultante dada pela equação (3.4).

$$\overrightarrow{F} = \overrightarrow{F_R} + \overrightarrow{F_a} \tag{3.4}$$

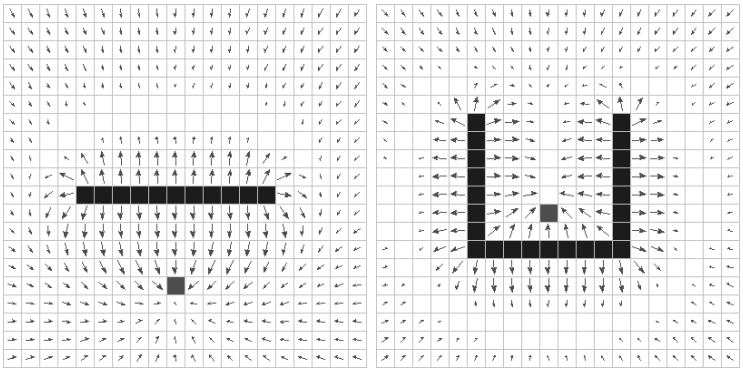
Para ilustrar a superposição dos campos de atração e repulsão, mostra-se na Figura 3.3(a) o campo de atração gerado pela meta, na Figura 3.3(b) o campo de repulsão gerado por dois obstáculos e na Figura 3.3(c) a soma dos campos de atração e de repulsão.

Porém, nem sempre há garantias de se chegar até a meta, ainda que exista um caminho, como pode ser visto nas Figuras 3.4 (a) (d).

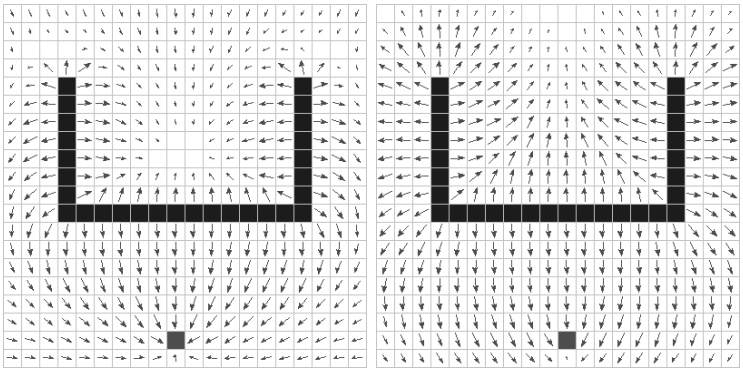


(a) Potencial atrativo da meta (b) Potencial repulsivo de dois obstáculos (c) Soma destes dois campos

Figura 3.3: Força Resultante(Faria, 2006)



(a) Ambiente 1 (b) Ambiente 2



(c) Ambiente 3 - Situaçao 1 (d) Ambiente 3 - Situação 2

Figura 3.4: Mínimos locais gerados pela anulação das forças de atração e repulsão.(Faria, 2006)

3.2 Campos Potenciais Harmônicos

Os sistemas de planejamento que utilizam campos potenciais têm sido bastante utilizados (Pacheco e Costa, 2002; Gomes e Campos, 2001; Faria e Romero, 2000; Borenstein e Koren, 1989, 1991; Brooks, 1986; Krogh e Thorpe, 1986; Newman e Hogan, 1987; Krogh, 1984), embora este método possua limitações, que possam diminuir a eficiência do sistema.

O problema de mínimos locais foi solucionado no trabalhos de (Connolly et al., 1990), com a utilização de funções harmônicas para o cálculo do campo potencial de ambientes nos quais as posições de paredes, objetos e metas sejam conhecidas.

$$\nabla^2 P = 0 \qquad \text{para } P : \mathbb{R}^2 \mapsto \mathbb{R} \tag{3.5}$$

As funções harmônicas utilizadas são soluções para a equação de Laplace (Equação 3.5) (Figueiredo, 2005; Strauss, 1992; Courant e Hilbert, 1962), sendo que estas possuem propriedades muito úteis ao desenvolvimento de sistemas de controle robótico, resultando em um método com as seguintes características (Faria, 2006):

Integridade - o método fornece um sistema de planejamento *completo*, ou seja, é garantido encontrar um caminho livre entre dois pontos, caso ele exista.

Robustez - é capaz de lidar com a presença de obstáculos não conhecidos a priori.

Então, pode ser definido um PVC na região de atuação do robô utilizando a condição de Dirichlet(Gilbarg e Trudinger, 1983; Smith, 1992; Figueiredo, 2005; Strauss, 1992) com, potencial alto (normalmente 1) para obstáculos e potencial baixo para a meta (normalmente 0).

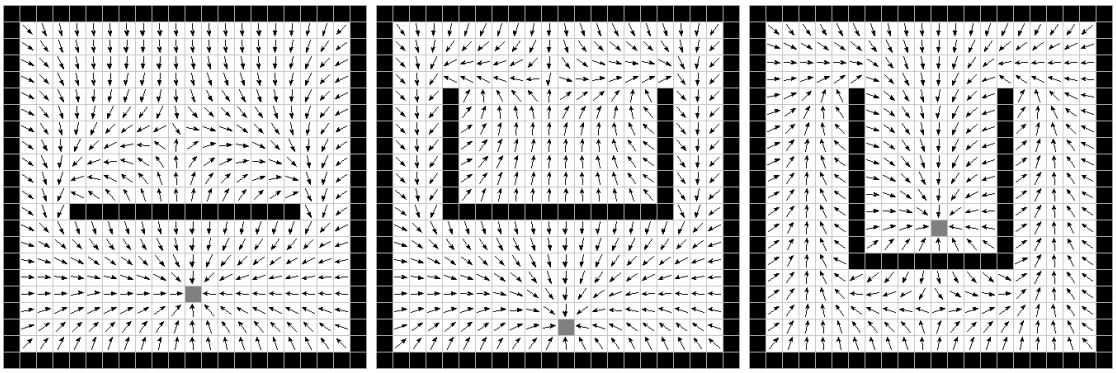
A partir de uma solução numérica da Equação 3.5, são extraídas as *linhas de força*, com base no *gradiente descendente*(Connolly et al., 1990; Connolly e Grupen, 1993) do potencial, que direcionam o robô para sua meta, desviando de obstáculos. Sendo assim, dada a posição de um robô em uma determinada célula (i, j), a direção a ser seguida Φ pode ser calculada como:

$$\Phi = \arctan(p_{i-1,j} - p_{i+1,j}, p_{i,j-1} - p_{i,j+1}) \tag{3.6}$$

na qual, Φ pertence ao intervalo $[-\pi, \pi]$.

As soluções da equação de Laplace não possuem mínimos e máximos locais no interior de um conjunto (Gilbarg e Trudinger, 1983; Figueiredo, 2005; Strauss, 1992), e assim, caso exista um caminho, o objetivo é sempre alcançado seguindo-se as linhas de força. Esta propriedade pode ser observada na Figura 3.5, na qual é mostrado o CPH para três ambientes distintos, idênticos aos ambientes das Figuras 3.4(a)-(d).

Utilizando o método das diferenças finitas, apresentado na Seção 2.1, a equação de Laplace pode ser discretizada como na Equação 2.11.



(a) Ambiente 1 (b) Ambiente 2 (c) Ambiente 3

Figura 3.5: Exemplo de Campos Potenciais Harmônicos (CPH), o qual não apresenta mínimos locais.(Faria, 2006)

3.2.1 Pseudocódigo

O Algoritmo 1 mostra o pseudocódigo do Gauss Seidel como iterador do CPH, em versão sequencial. Alguns procedimentos, como *getPotential* e *setPotential*, bem como algumas estrutuas, como o *Ambiente* e, que no caso do CPH, contém as posições dos obstáculos, robôs e meta, dimensões das grades, uma grade de potenciais, uma grade de ocupação e etc., são omitidos aqui por simplicidade. Já o Algoritmo 2, *getNeighborhood*, por não ser o algoritmo trivial de busca por vizinhos, é detalhado a seguir.

O Algoritmo 1 consiste em substituir, sistematicamente, o potencial (valor armazenado numa determinada célula da grade) por uma média aritmética dos seus vizinhos. Com este novo potencial, é obtida uma medida de distância, que poderá ser usada como um critério de convergência, pelo método, função ou procedimento que estiver chamando o Algoritmo 1.

Baseando-se em algumas propriedades da implementação feita neste trabalho, o algoritmo para encontrar os vizinhos de uma determinada célula pode ser otimizado (menos comparações são feitas) e fica como no Algoritmo 2.

3.3 Campos Potenciais Orientados

As soluções da equação de Laplace (Equação 3.5) garantem que a geração dos campos potenciais não irão apresentar mínimos locais, no entanto, este é apenas um caso particular de um conjunto maior de funções que também apresentam esta característica.

Em (Prestes, 2003), é apresentada uma família de soluções de uma EDP elíptica (Equação 3.7), que também não apresenta mínimos locais.

$$\nabla^2 P + \epsilon \langle \nabla P, v \rangle = 0 \tag{3.7}$$

Algoritmo 1 Iterator Gauss Seidel do CPH

requer Um ambiente e para CPH

garante Uma iteração do método de Gauss-Seidel, em CPU, aplicado ao grid contido em e

```
1: procedimento CPH_GS(e)
2:     error ← 0
3:     j ← 0
4:     repita
5:         i ← 0
6:         repita
7:             se FreeCell = getOccupancy(e, j, i) então
8:                 oldPotential ← getPotential(e, j, i)
9:                 (top, bottom, left, right) ← getNeighborhood(e, j, i)  ▷ Algoritmo 2
10:                newPotential ← (right + left + top + bottom)/4
11:                error ← error + (newPotential − oldPotential)^2
12:                setPotential(e, j, i, newPotential)
13:            fim se
14:            i ← i + 1
15:        até i > getHeight(e)
16:        j ← j + 1
17:    até j > getWidth(e)
18:    retorna error
19: fim procedimento
```

com $\epsilon \in \mathbb{R}$, $v \in \mathbb{R}^2$ e $||v|| = 1$.

Pode-se observar que a equação de Laplace (Equação 3.5) é um caso especial da Equação 3.7, quando $\epsilon = 0$.

Devido a influência do vetor v na orientação do campo gradiente, esta abordagem utilizando a Equação (3.7) foi denominada no trabalho de (Faria, 2006) como Campos Potenciais Orientados [1] (*CPO*). Na Figura 3.6 é mostrado o campo potencial gerado utilizando-se a equação (3.7) e a influência recebida pela direção do vetor v.

Já nas Figuras 3.7(a) a 3.7(d), observa-se o comportamento que valores diferentes de ϵ podem provocar no campo potencial. A variável ϵ pode ser vista como a taxa de *influência* do vetor v sobre o campo potencial.

Na Figura 3.7(e) comparam-se as trajetórias seguidas com $0 \leq \epsilon \leq 1.5$, e é possível verificar que, quanto maior o valor de ϵ, mais a trajetória se aproxima da direção do vetor v. Contudo, o valor de ϵ não pode crescer indefinidamente, a depender da discretização utilizada, como será mostrado posteriormente. Podemos observar que na Figura 3.7(e), o uso de $\epsilon = 4$ mostra um campo instável que não garante uma trajetória até a meta.

[1] Oriented Potential Field (*OPF*)

Algoritmo 2 Obtém vizinhança

requer Um ambiente e contendo um grid
requer Uma posição (j, i) no ambiente e
garante A vizinhança de Manhattan da célula (j, i)

```
1: procedimento GETNEIGHBORHOOD(e, j, i)
2:     se i = 0 então
3:         bottom ← maxPotential
4:         top ← getPotential(e, j, i + 1)
5:     senão
6:         se i + 1 = height(e) então
7:             bottom ← getPotential(e, j, i − 1)
8:             top ← maxPotential
9:         senão
10:            bottom ← getPotential(e, j, i − 1)
11:            top ← getPotential(e, j, i + 1)
12:        fim se
13:    fim se
14:    se j = 0 então
15:        left ← maxPotential
16:        right ← getPotential(e, j + 1, i)
17:    senão
18:        se j + 1 = width(e) então
19:            left ← getPotential(e, j − 1, i)
20:            right ← maxPotential
21:        senão
22:            left ← getPotential(e, j − 1, i)
23:            right ← getPotential(e, j + 1, i)
24:        fim se
25:    fim se
26:    retorna (top, bottom, left, right)
27: fim procedimento
```

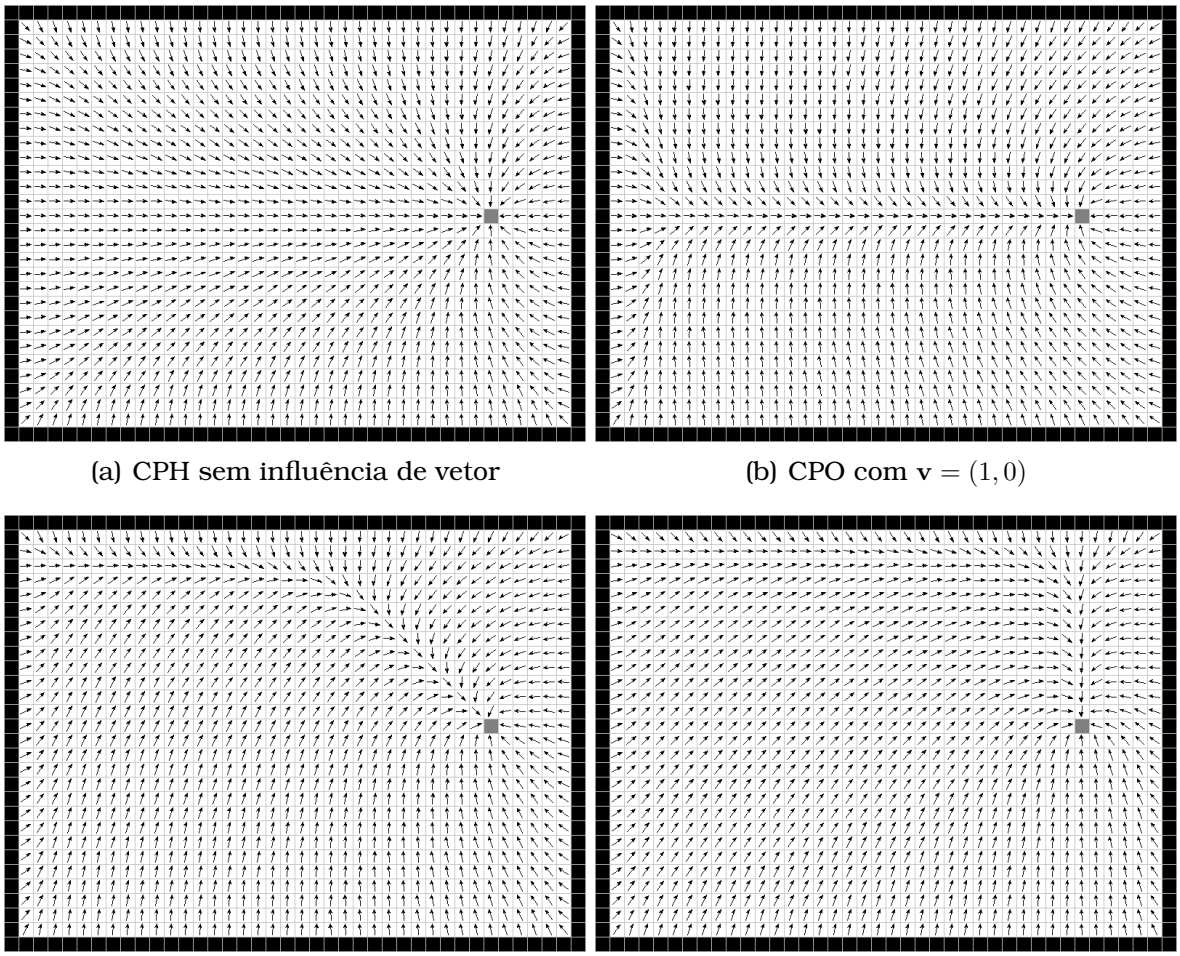
3.3.1 Solução Numérica por Diferenças Centradas

Utilizando diferenças centradas, pode ser mostrado que a Equação 3.7 discretizada fica como na Equação 3.8.

$$\begin{aligned} &\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta y^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} \\ &+\frac{\epsilon}{2}\left[\frac{p_{i+1,j} - p_{i-1,j}}{\Delta y} v_y + \frac{p_{i,j+1} - p_{i,j-1}}{\Delta x} v_x\right] = 0 \end{aligned} \tag{3.8}$$

onde $\epsilon \in \mathbb{R}$ e, $v = (v_x, v_y) \in \mathbb{R}^2$ e $||v|| = 1$.

Como em (da Silva et al., 2008), assumindo que se está trabalhando em uma malha quadrada, temos que $h = \Delta x = \Delta y$. Ainda, definindo $\lambda = \frac{\epsilon h}{2}$, encontra-se a molécula computacional da Equação 3.9, que é uma generali-



(a) CPH sem influência de vetor

(b) CPO com $\mathbf{v} = (1, 0)$

(c) CPO com $\mathbf{v} = (1, 1)$

(d) CPO com $\mathbf{v} = (0, 1)$

Figura 3.6: Influência do vetor v no campo potencial

zação[2] da Equação 2.11.

$$p_{i,j} = \frac{1}{4}\left[(1 + \lambda v_y)p_{i+1,j} + (1 - \lambda v_y)p_{i-1,j} + (1 + \lambda v_x)p_{i,j+1} + (1 - \lambda v_x)p_{i,j-1}\right] \tag{3.9}$$

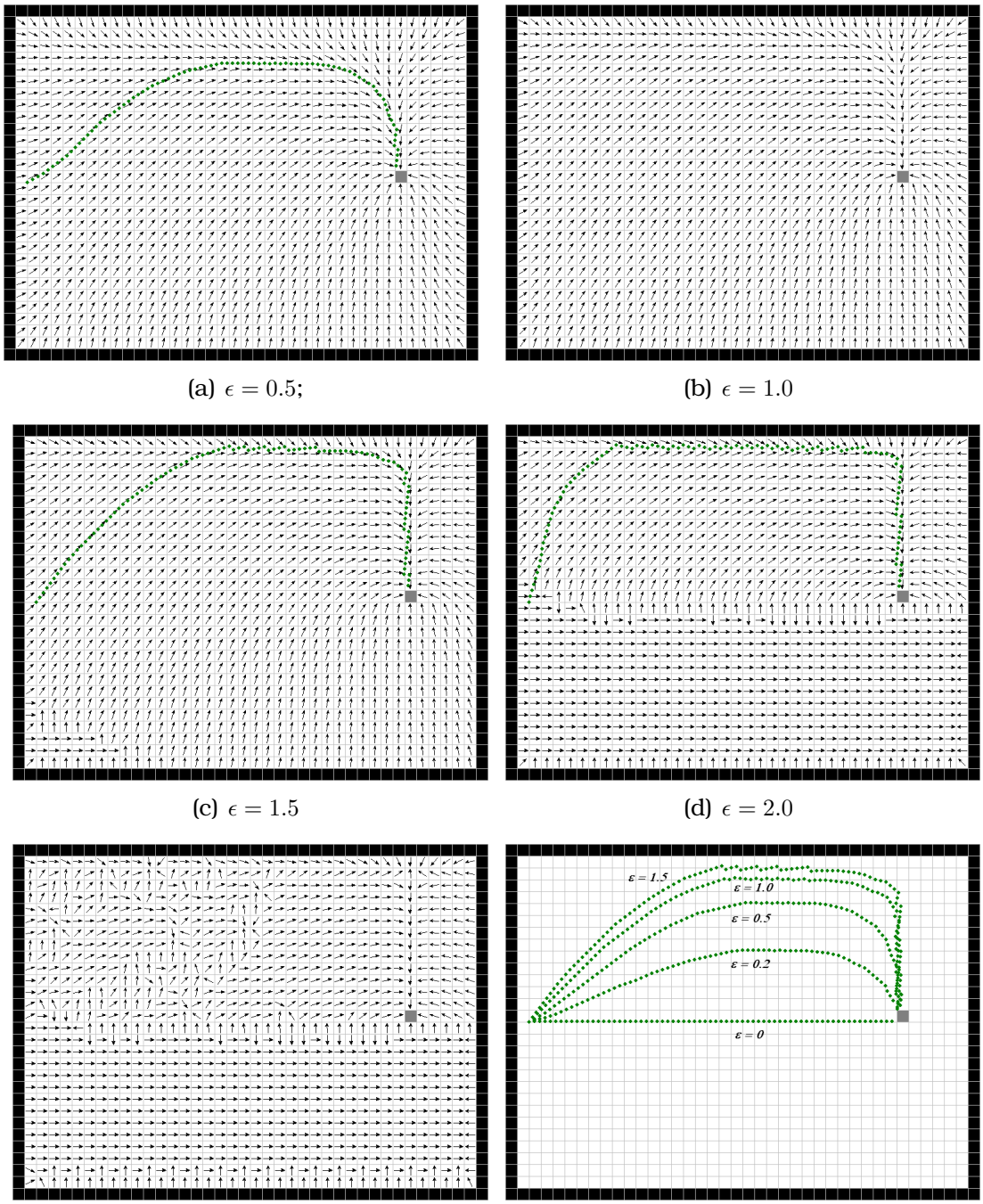
Tal discretização impõe um limite em λ, como mostrado em (da Silva et al., 2008). Isto é, conforme maior é ϵ, menor precisa ser o espaçamento da malha.

Então, o pseudocódigo fica como no Algoritmo 3. Quando $\epsilon < 2$, observa-se no Algoritmo 3 que o novo pontencial de uma determinada célula é uma média ponderada dos seus vizinhos. Tal ponderação leva em conta o vetor v e o seu módulo ϵ.

3.3.2 Solução Numérica por Diferenças Up-Wind

Neste trabalho, foi feito um estudo sobre o uso do esquema upwind para a discretização do CPO, com a finalidade de ter um sistema de planejamento de

[2]como sugerido anteriormente, tomando $\epsilon = 0$, implica em $\lambda = 0$



(a) $\epsilon = 0.5$;

(b) $\epsilon = 1.0$

(c) $\epsilon = 1.5$

(d) $\epsilon = 2.0$

(e) $\epsilon = 4.0$

(f) comparação entre as trajetórias seguidas de 3.7(a) a 3.7(d)

Figura 3.7: Várias taxas de influência ϵ em um campo potencial com $\mathbf{v} = (0, 1)$

caminhos com menos limitações, que será detalhado nesta sub-seção.

A Equação 3.7 possui duas derivadas de primeira ordem, portanto, utilizar diferenças centradas gerará uma molécula computacional diferente daquela resultante das diferenças upwind.

Utilizando diferenças upwind, com $\Delta x = \Delta y = h$, podem ser definidos quatro casos:

Algoritmo 3 Iterator Gauss Seidel usando diferenças centradas do CPO

requer Um ambiente e para CPO

garante Uma iteração do método de Gauss-Seidel, em CPU, aplicado ao grid contido em e

```
1: procedimento CPO_GS_C(e)
2:     error ← 0
3:     v.y ← sin(getVector(e))
4:     v.x ← cos(getVector(e))
5:     λ ← getEpsilon(e)/2
6:     j ← 0
7:     repita
8:         i ← 0
9:         repita
10:            se FreeCell = getOccupancy(e, j, i) então
11:                oldPotential ← getPotential(e, j, i)
12:                (top, bottom, left, right) ← getNeighborhood(e, j, i) ▷ Algoritmo 2
13:                newPotential ← ((1 + λv.x)right + (1 − λv.x)left + (1 + λv.y)top +
   (1 − λv.x)bottom)/4
14:                error ← error + (newPotential − oldPotential)^2
15:                setPotential(e, j, i, newPotential)
16:            fim se
17:            i ← i + 1
18:        até i > getHeight(e)
19:        j ← j + 1
20:    até j > getWidth(e)
21:    retorna error
22: fim procedimento
```

Quando $v_x(x,y) > 0$ **e** $v_y(x,y) > 0$

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2} + \frac{p_{i+i,j} - 2p_{i,j} + p_{i-1,j}}{h^2}$$

$$+\epsilon v_x \frac{p_{i,j+1} - p_{i,j}}{h} + \epsilon v_y \frac{p_{i+1,j} - p_{i,j}}{h} = 0$$

Então, fatorando a expressão acima em p:

$$\left(\frac{1}{h^2} + \frac{\epsilon v_x}{h}\right) p_{i,j+1} + \left(\frac{1}{h^2} + \frac{\epsilon v_y}{h}\right) p_{i+1,j} + \left(\frac{1}{h^2}\right) p_{i,j-1} + \left(\frac{1}{h^2}\right) p_{i-1,j}$$

$$+ \left(\frac{-4}{h^2} - \frac{\epsilon v_y}{h} - \frac{\epsilon v_x}{h}\right) p_{i,j} = 0$$

E, isolando $p_{i,j}$:

$$p_{i,j} = \frac{(1 + h\epsilon v_x)\, p_{i,j+1} + (1 + h\epsilon v_y)\, p_{i+1,j} + p_{i,j-1} + p_{i-1,j}}{4 + h\epsilon\,(v_y + v_x)}$$

Quando $v_x(x,y) > 0$ **e** $v_y(x,y) < 0$

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2} + \frac{p_{i+i,j} - 2p_{i,j} + p_{i-1,j}}{h^2}$$

$$+\epsilon v_x \frac{p_{i,j+1} - p_{i,j}}{h} + \epsilon v_y \frac{p_{i,j} - p_{i-1,j}}{h} = 0$$

Então, fatorando a expressão acima em p:

$$\left(\frac{1}{h^2} + \frac{\epsilon v_x}{h}\right) p_{i,j+1} + \left(\frac{1}{h^2}\right) p_{i+1,j} + \left(\frac{1}{h^2}\right) p_{i,j-1} + \left(\frac{1}{h^2} - \epsilon v_y\right) p_{i-1,j}$$

$$+ \left(\frac{-4}{h^2} + \frac{\epsilon v_y}{h} - \frac{\epsilon v_x}{h}\right) p_{i,j} = 0$$

E, isolando $p_{i,j}$:

$$p_{i,j} = \frac{\left(1 + h\epsilon v_x\right) p_{i,j+1} + p_{i+1,j} + p_{i,j-1} + \left(1 - h\epsilon v_y\right) p_{i-1,j}}{4 + h\epsilon\left(-v_y + v_x\right)}$$

Quando $v_x(x,y) < 0$ **e** $v_y(x,y) > 0$

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2} + \frac{p_{i+i,j} - 2p_{i,j} + p_{i-1,j}}{h^2}$$

$$+\epsilon v_x \frac{p_{i,j} - p_{i,j-1}}{h} + \epsilon v_y \frac{p_{i+1,j} - p_{i,j}}{h} = 0$$

Então, fatorando a expressão acima em p:

$$\left(\frac{1}{h^2}\right) p_{i,j+1} + \left(\frac{1}{h^2} + \frac{\epsilon v_y}{h}\right) p_{i+1,j} + \left(\frac{1}{h^2} - \frac{\epsilon v_x}{h}\right) p_{i,j-1} + \left(\frac{1}{h^2}\right) p_{i-1,j}$$

$$+ \left(\frac{-4}{h^2} - \frac{\epsilon v_y}{h} + \frac{\epsilon v_x}{h}\right) p_{i,j} = 0$$

E, isolando $p_{i,j}$:

$$p_{i,j} = \frac{p_{i,j+1} + \left(1 + h\epsilon v_y\right) p_{i+1,j} + \left(1 - h\epsilon v_x\right) p_{i,j-1} + p_{i-1,j}}{4 + h\epsilon\left(v_y - v_x\right)}$$

Quando $v_x(x,y) < 0$ **e** $v_y(x,y) < 0$

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2} + \frac{p_{i+i,j} - 2p_{i,j} + p_{i-1,j}}{h^2}$$

$$+\epsilon v_x \frac{p_{i,j} - p_{i,j-1}}{h} + \epsilon v_y \frac{p_{i,j} - p_{i-1,j}}{h} = 0$$

Então, fatorando a expressão acima em p:

$$\left(\frac{1}{h^2}\right) p_{i,j+1} + \left(\frac{1}{h^2}\right) p_{i+1,j} + \left(\frac{1}{h^2} - \frac{\epsilon v_x}{h}\right) p_{i,j-1} + \left(\frac{1}{h^2} - \frac{\epsilon v_y}{h}\right) p_{i-1,j}$$
$$+ \left(\frac{-4}{h^2} + \frac{\epsilon v_y}{h} + \frac{\epsilon v_x}{h}\right) p_{i,j} = 0$$

E, isolando $p_{i,j}$:

$$p_{i,j} = \frac{p_{i,j+1} + p_{i+1,j} + (1 - h\epsilon v_x)\, p_{i,j-1} + (1 - h\epsilon v_y)\, p_{i-1,j}}{4 + h\epsilon\,(-v_y - v_x)}$$

Para um caso geral:

$$\begin{aligned} v_x(x,y)\frac{\partial}{\partial x}P(x,y) &= |v_x|\frac{p_{i,j+sgn(v_x)} - p_{i,j}}{h} \\ v_y(x,y)\frac{\partial}{\partial y}P(x,y) &= |v_y|\frac{p_{i+sgn(v_y),j} - p_{i,j}}{h} \end{aligned} \tag{3.10}$$

onde

$$sgn(x) = \begin{cases} +1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$$

Isto é,

$$p_{i,j} = \frac{(1 + h\epsilon|v_x|)\, p_{i,j+sgn(v_x)} + (1 + h\epsilon|v_y|)\, p_{i+sgn(v_y),j} + p_{i,j-sgn(v_x)} + p_{i-sgn(v_y),j}}{4 + h\epsilon\,(|v_y| + |v_x|)} \tag{3.11}$$

Então, o pseudocódigo fica como no Algoritmo 4. Pode ser observado que independentemente do valor de ϵ, o novo potencial de uma determinada célula sempre será uma média ponderada entre os seus vizinhos.

Algoritmo 4 Iterator Gauss Seidel usando diferenças upwind do CPO

requer Um ambiente e para CPO

garante Uma iteração do método de Gauss-Seidel, em CPU, aplicado ao grid contido em e

```
 1: procedimento CPO_GS_UP(e)
 2:     error ← 0
 3:     v.y ← sin(getVector(e))
 4:     v.x ← cos(getVector(e))
 5:     ro.y ← 1+ getEpsilon(e) * abs(sin(v.y))
 6:     ro.x ← 1+ getEpsilon(e) * abs(cos(v.x))
 7:     den = 2 + ro.x + ro.y
 8:     j ← 0
 9:     repita
10:         i ← 0
11:         repita
12:             se FreeCell = getOccupancy(e, j, i) então
13:                 oldPotential ← getPotential(e, j, i)
14:                 (top, bottom, left, right) ← getNeighborhood(e, j, i) ▷ Algoritmo 2
15:                 se v.y > 0 então
16:                     se v.x > 0 então
17:                         newPotential ← (ro.x * right + left + ro.y * top + bottom)/den
18:                     senão
19:                         newPotential ← (right + ro.x * left + ro.y * top + bottom)/den
20:                     fim se
21:                 senão
22:                     se v.x > 0 então
23:                         newPotential ← (ro.x * right + left + top + ro.y * bottom)/den
24:                     senão
25:                         newPotential ← (right + ro.x * left + top + ro.y * bottom)/den
26:                     fim se
27:                 fim se
28:                 error ← error + (newPotential − oldPotential)^2
29:                 setPotential(e, j, i, newPotential)
30:             fim se
31:             i ← i + 1
32:         até i > getHeight(e)
33:         j ← j + 1
34:     até j > getWidth(e)
35:     retorna error
36: fim procedimento
```

3.4 Campos Potenciais Localmente Orientados

A técnica de Campos Potenciais Localmente Orientados[3] (*CPLO*), foi proposta em (Faria, 2006) e, consiste, basicamente, de uma modificação do CPO[4].

O objetivo do CPLO é conduzir múltiplos agentes para a meta, através de

[3]Locally Oriented Potential Field (*LOPF*)

[4]que, por sua vez, é uma modificação do CPH

diferentes direções (vetores), em um mesmo grid. Desta forma, vários robôs podem seguir diferentes trajetórias, usando diferentes vetores, que os conduzem até a meta. Isto é obtido aplicando a Equação 3.7 somente em uma região (denominada *área de influência*) em torno de cada robô. Normalmente, esta região é definida como um círculo de raio fixo com centro na posição do robô. E, para as outras regiões, é aplicada a Equação 3.5.

O CPLO pode ser descrito como na Equação 3.12.

$$\nabla^2 P(x,y) + \epsilon(x,y) \langle \nabla P(x,y), v(x,y) \rangle = 0 \tag{3.12}$$

onde

$$\epsilon(x,y) = \begin{cases} \epsilon_k & \text{com } \epsilon_k \in \mathbb{R}, \text{ caso } (x,y) \text{ esteja} \\ & \text{na área de influência do robô } k \\ 0 & \text{caso contrário} \end{cases}$$

e,

$$v(x,y) = \begin{cases} v_k & \text{com } v_k \in \mathbb{R}^2, \text{ e } ||v_k|| = 1, \text{ caso } (x,y) \text{ esteja} \\ & \text{na área de influência do robô } k \\ \overrightarrow{0} & \text{caso contrário} \end{cases}$$

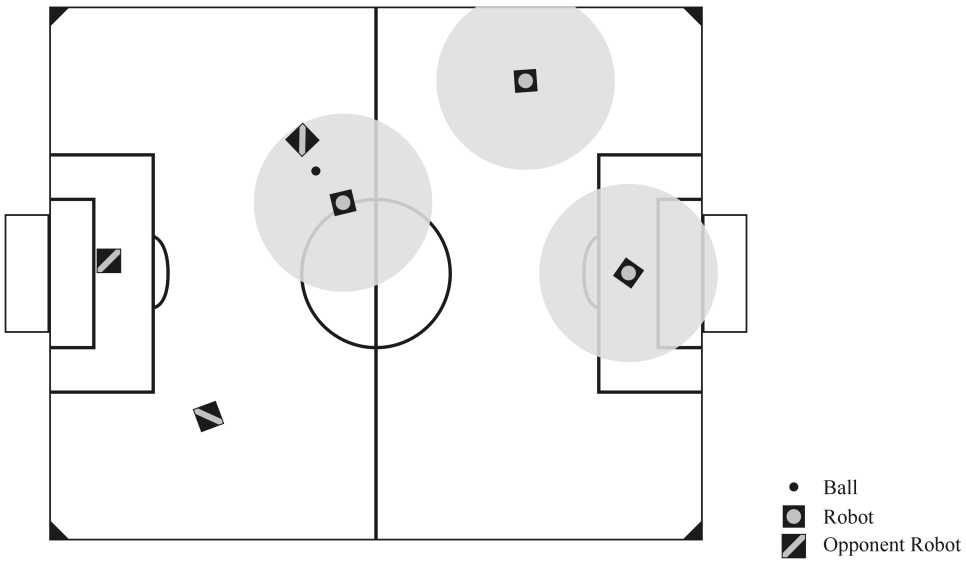


Figura 3.8: Uma situação que pode ocorrer dentro do ambiente de futebol de robôs. As áreas cinzas indicam as regiões de influência de cada robô (da Silva et al., 2008)

Pode ser visto pela Equação 3.12 que o CPLO mescla o CPO e o CPH, em uma mesma malha.

3.4.1 Solução Numérica por Diferenças Centradas

Utilizando diferenças centradas e seguindo o mesmo raciocínio da Seção 3.3, pode ser encontrada a Equação 3.13.

$$p_{i,j} = \frac{1}{4}\left[(1+\lambda_{i,j}v_{x;i,j})p_{i+1,j} + (1-\lambda_{i,j}v_{x;i,j})p_{i-1,j}\right. \\ \left. +(1+\lambda_{i,j}v_{y;i,j})p_{i,j+1} + (1-\lambda_{i,j}v_{y;i,j})p_{i,j-1}\right] \tag{3.13}$$

Onde

$$\begin{aligned} \lambda_{i,j} = \frac{h\epsilon_{i,j}}{2} &\approx \frac{h\epsilon(x,y)}{2} \\ v_{x;i,j} &\approx v_x(x,y) \\ v_{y;i,j} &\approx v_y(x,y) \end{aligned}$$

Então, o pseudocódigo do Gauss Seidel utilizando diferenças centradas fica como no Algoritmo 5.

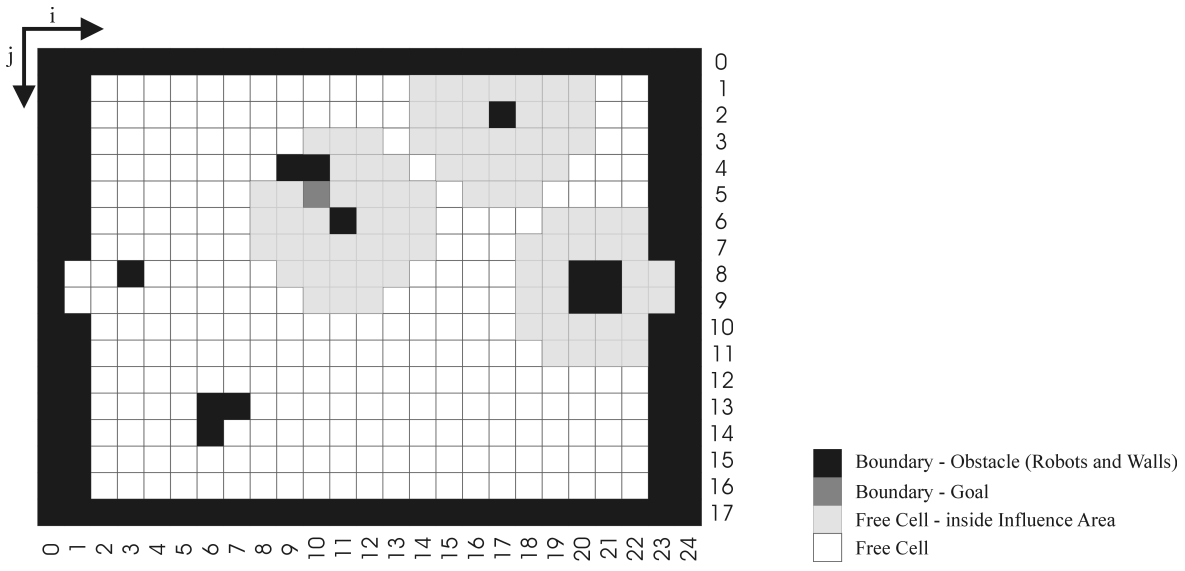


Figura 3.9: Um possível PVC, resultante da situação mostrada na Figura 3.8 (da Silva et al., 2008)

Algoritmo 5 Iterator Gauss Seidel usando diferenças centradas do CPLO

requer Um ambiente e para CPLO

garante Uma iteração do método de Gauss-Seidel, em CPU, aplicado ao grid contido em e

1: **procedimento** CPLO_GS_C(e)
2: $error \leftarrow 0$
3: $j \leftarrow 0$
4: **repita**
5: $i \leftarrow 0$
6: **repita**
7: **se** $FreeCell =$ getOccupancy(e, j, i) **então**
8: $oldPotential \leftarrow$ getPotential(e, j, i)
9: $(top, bottom, left, right) \leftarrow$ getNeighborhood(e, j, i) ▷ Algoritmo 2
10: $k \leftarrow 0$
11: $lessDist \leftarrow \infty$
12: $k_{lessDist} \leftarrow -1$
13: **repita**
14: $(x_r, y_r) \leftarrow$ getRobotPosition(e, k);
15: $dist \leftarrow \sqrt{(x_r - j)^2 + (y_r - i)^2}$
16: **se** $dist < lessDist$ **então**
17: $lessDist \leftarrow dist$
18: $k_{lessDist} \leftarrow k$
19: **fim se**
20: $k \leftarrow k + 1$
21: **até** $k >$ getRobotListSize(e)
22: **se** lessDist < getRadiusOfInfluence(e) **então**
23: $v.y \leftarrow$ sin(getVector(e, $k_{lessDist}$))
24: $v.x \leftarrow$ cos(getVector(e, $k_{lessDist}$))
25: $\lambda \leftarrow$ getEpsilon(e, $k_{lessDist}$)/2
26: $newPotential \leftarrow ((1+\lambda v.x)right + (1-\lambda v.x)left + (1+\lambda v.y)top + (1-\lambda v.x)bottom)/4$
27: **senão**
28: $newPotential \leftarrow (right + left + top + bottom)/4$
29: **fim se**
30: $error \leftarrow error + (newPotential - oldPotential)^2$
31: setPotential(e, j, i, $newPotential$)
32: **fim se**
33: $i \leftarrow i + 1$
34: **até** $i >$ getHeight(e)
35: $j \leftarrow j + 1$
36: **até** $j >$ getWidth(e)
37: **retorna** $error$
38: **fim procedimento**

3.4.2 Solução Numérica por Diferenças Upwind

Utilizando diferenças upwind e seguindo o mesmo raciocínio da Seção 3.3, pode ser encontrada a Equação 3.14.

$$p_{i,j} = \frac{1}{4 + h\epsilon_{i,j}\left(|v_{y;i,j}| + |v_{x;i,j}|\right)} \Big[\left(1 + h\epsilon_{i,j}|v_{x;i,j}|\right) p_{i,j+sgn(v_{x;i,j})} + \left(1 + h\epsilon_{i,j}|v_{y;i,j}|\right) p_{i+sgn(v_{y;i,j}),j} + p_{i,j-sgn(v_{x;i,j})} + p_{i-sgn(v_{y;i,j}),j} \Big] \tag{3.14}$$

Então, o pseudocódigo do Gauss Seidel utilizando diferenças upwind é similar ao apresentado no Algoritmo 5, bastando substituir o trecho de código que começa na linha 22 e vai até a linha 29, pelo Algoritmo 6.

Algoritmo 6 Iterator Gauss Seidel usando diferenças upwind do CPLO

```
se lessDist < getRadiusOfInfluence(e) então
    v.y ← sin(getVector(e, k_lessDist))
    v.x ← cos(getVector(e, k_lessDist))
    ro.y ← 1+ getEpsilon(e, k_lessDist) * abs(sin(v.y))
    ro.x ← 1+ getEpsilon(e, k_lessDist) * abs(cos(v.x))
    den = 2 + ro.x + ro.y
    se v.y > 0 então
        se v.x > 0 então
            newPotential ← (ro.x * right + left + ro.y * top + bottom)/den
        senão
            newPotential ← (right + ro.x * left + ro.y * top + bottom)/den
        fim se
    senão
        se v.x > 0 então
            newPotential ← (ro.x * right + left + top + ro.y * bottom)/den
        senão
            newPotential ← (right + ro.x * left + top + ro.y * bottom)/den
        fim se
    fim se
senão
    newPotential ← (right + left + top + bottom)/4
fim se
```

3.5 Estabilidade dos Métodos Numéricos aplicados ao CPLO, CPO e CPH

Nesta seção será avaliado a estabilidade dos métodos de Jacobi-Richardson e Gauss-Seidel (Burden et al., 1978; Franco, 2008; Ortega, 1972) aplicados aos sistemas lineares resultantes dos métodos CPH, CPO e CPLO, através de diferenças centradas e upwind. Embora a prova será feita apenas para o

CPLO, pode ser vista que ela é valida para o CPH e para o CPO, pois ambos são casos especiais do CPLO. Basta ver que quando $\epsilon(x, y) \equiv 0$ e $v(x, y) \equiv \overrightarrow{0}$, o CPLO se reduz ao CPH e, quando $\epsilon(x, y) = c \in \mathbb{R}$ e $v(x, y) = \overrightarrow{w} \in \mathbb{R}^2$ o CPLO se reduz ao CPO.

Na Sub-seção 3.5.1, serão apresentados detalhes da prova da convergência condicional das diferenças centradas para o CPLO e, na Sub-seção 3.5.2, de forma sucinta, a prova da convergência incondicional das diferenças upwind para o CPLO. As definições e teoremas utilizadas nestas Sub-seções encontram-se no Apêndice A

3.5.1 Diferenças Centradas

Sem perda de generalização, na Equação 3.13, pode ser suposto que $h = 1$ [5], obtendo-se a Equação 3.15, proposta em (Faria, 2006) .

$$\begin{aligned} p_{i,j} = \frac{1}{4}\left(p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1}\right) + \\ \frac{\epsilon_{i,j}}{8}\left([p_{i+1,j} + p_{i-1,j}]v_{x;i,j} + [p_{i,j+1} + p_{i,j-1}]v_{y;i,j}\right) \end{aligned} \tag{3.15}$$

Suponha que $P : \Omega \subseteq \Re^2 \to \Re$ é como na Equação 3.12 e que Ω é discretizado, com a propriedade de ser conectado por caminhos. Ao aplicar o método das diferenças finitas para discretizar a Equação 3.12, o método de Gauss-Seidel e o método de Jacobi Richardson convergem se aplicados ao sistema linear resultante.

Sem perda de generalização, usando o exemplo da Figura 3.9, para cada célula livre é aplicada a Equação 3.13, então, obtém-se a Equação 3.16.

$$\begin{aligned} p_{2,1} - \tfrac{1}{4}\left(p_{3,1} + p_{2,2}\right) &= \tfrac{1}{4}\left(p_{2,0} + p_{1,1}\right) \\ p_{3,1} - \tfrac{1}{4}\left(p_{2,1} + p_{4,1} + p_{3,2}\right) &= \tfrac{1}{4}\left(p_{3,0}\right) \\ &\vdots \end{aligned} \tag{3.16}$$

Observe que as células $p_{i,j}$ do lado direito da igualdade fazem parte da condição de contorno, ou seja, valores conhecidos, enquanto o lado esquerdo é composto por células livres, ou seja, variáveis.

Para atualizar as células, pode ser adotada a orientação da esquerda para a direita, de cima para baixo, tal como na Figura 3.9, para facilitar a notação,

[5] porque sempre é possível encontrar um fator de escala adequado durante o processo de discretização do ambiente

podemos considerar as variáveis como na Equação 3.17

$$\begin{array}{llllll} x_1 = p_{2,1}, & x_2 = p_{3,1} & , \ldots , & x_{21} = p_{22,1}, & x_{22} = p_{2,2} & , \ldots , \\ x_{36} = p_{16,2}, & x_{37} = p_{18,2} & , \ldots , & x_{41} = p_{22,2}, & x_{42} = p_{2,3} & , \ldots \end{array} \tag{3.17}$$

Observe que $p_{17,2}$ não é uma variável (já que faz parte da condição de contorno). Então, a Equação 3.16 pode ser vista como na Equação 3.18

$$\begin{bmatrix} 1 & -\frac{1}{4} & 0 & \ldots & 0 & -\frac{1}{4} & 0 & \ldots \\ -\frac{1}{4} & 1 & -\frac{1}{4} & \ldots & 0 & 0 & -\frac{1}{4} & \ldots \\ 0 & -\frac{1}{4} & 1 & \ldots & 0 & 0 & 0 & \ldots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots \\ 0 & 0 & 0 & \ldots & 1 & 0 & 0 & \ldots \\ -\frac{1}{4} & 0 & 0 & \ldots & 0 & 1 & -\frac{1}{4} & \ldots \\ 0 & -\frac{1}{4} & 0 & \ldots & 0 & -\frac{1}{4} & 1 & \ldots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{21} \\ x_{22} \\ x_{23} \\ \vdots \end{bmatrix} = \begin{bmatrix} \frac{1}{4}\left(p_{2,0} + p_{1,1}\right) \\ \frac{1}{4}\left(p_{3,1}\right) \\ \frac{1}{4}\left(p_{4,1}\right) \\ \vdots \\ \frac{1}{4}\left(p_{22,0} + p_{23,1}\right) \\ \frac{1}{4}\left(p_{1,2}\right) \\ 0 \\ \vdots \end{bmatrix} \tag{3.18}$$

ou, simplesmente $Ax = b$, onde $x \in \Re^n$ é o vetor formado pelos pontos $p_{i,j}$ pertencentes ao interior do domínio discretizado Ω, $A_{n,n} \in \mathcal{L}(\Re^n)$ é uma matriz que fornece relações de vizinhança entre os pontos $p_{i,j}$ e $b \in \Re^n$ é soma, dividida por 4, dos vizinhos (dos pontos $p_{i,j}$) que pertencem ao contorno do domínio.

Algumas proprieades deste sistema são importantes, como a seguir.

Suponha que q seja a linha do sistema linear relacionada ao ponto $p_{i,j}$, então $a_{qq} = 1$ e, dois casos principais podem ser considerados:

- $p_{i,j}$ está na região de influência do k-ésimo robô.
  - Se $p_{i,j}$ não tem vizinhança contida no contorno, então $b_q = 0$ e $\sum_{\substack{r=0 \\ r \neq q}}^{n} |a_{qr}| = \frac{1}{4}\left(|1 + \lambda v_{k_x}| + |1 - \lambda v_{k_x}| + |1 + \lambda v_{k_y}| + |1 - \lambda v_{k_y}|\right)$;
  - Se $p_{i,j}$ tem somente um vizinho satisfazendo a condição de contorno. Supondo que este vizinho seja $p_{i+1,j}$, então $b_q = \frac{1}{4} p_{i+1,j}$ e $\sum_{\substack{r=0 \\ r \neq q}}^{n} |a_{qr}| = \frac{1}{4}\left(|1 - \lambda v_{k_x}| + |1 + \lambda v_{k_y}| + |1 - \lambda v_{k_y}|\right)$;
- $p_{i,j}$ não está na região de influência de qualquer um dos robôs.
  - Se $p_{i,j}$ não tem vizinhança contida no contorno, então $b_q = 0$ e $\sum_{\substack{r=0 \\ r \neq q}}^{n} |a_{qr}| = 1$;
  - Se $p_{i,j}$ tem somente um vizinho satisfazendo a condição de contorno. Supondo que este vizinho seja $p_{i+1,j}$, então $b_q = \frac{1}{4} p_{i+1,j}$ e $\sum_{\substack{r=0 \\ r \neq q}}^{n} |a_{qr}| = \frac{3}{4}$;

Analogamente, o mesmo pode ser feito para outros casos (dois, três ou quatro vizinhos[6] no contorno).

Considerando o que foi dito anteriormente, se o ponto $p_{i,j}$ não está na região de influência de qualquer um dos robôs, a linha q da matriz A satisfaz a condição $|a_{qq}| \geq \sum_{\substack{r=0 \\ r \neq q}}^{n} |a_{qr}|$. E, caso $p_{i,j}$ tenha algum vizinho no contorno, a inequação é estrita.

Por outro lado, caso $p_{i,j}$ esteja na região de influência do k-ésimo robô, para que a matriz seja diagonalmente dominante, é preciso impor que $|a_{qq}| > \sum_{\substack{r=0 \\ r \neq q}}^{n} |a_{qr}|$. isto é

- $\big(|1 + \lambda v_{k_x}| + |1 - \lambda v_{k_x}| + |1 + \lambda v_{k_y}| + |1 - \lambda v_{k_y}|\big) < 4$, caso $p_{i,j}$ não tenha vizinhos contidos no contorno.
- $\big(|1 - \lambda v_{k_x}| + |1 + \lambda v_{k_y}| + |1 - \lambda v_{k_y}|\big) < 4$, caso $p_{i+1,j}$ esteja no contorno;

Novamente, de forma análoga, o mesmo pode ser feito para outros casos (dois, três ou quatro vizinhos[7] no contorno).

Analisando o pior caso, que é quando $p_{i,j}$ não tem vizinhos contidos no contorno, e $v_k = (1, 0)$, $v_k = (0, 1)$, $v_k = (-1, 0)$ ou $v_k = (0, -1)$, obtém-se a Equação 3.19.

$$|1 - \lambda| + |1 + \lambda| < 2 \tag{3.19}$$

Resolvendo a Equação 3.19 em função de λ, obtém-se

$$|\lambda| < 1$$

O grafo associado a matriz A é fortemente conectado pois, por hipótese, a discretização de Ω preserva a propriedade do domínio ser conexo por caminhos. Levando em consideração uma da aplicação do método das diferenças finitas é que se o ponto $p_{r,s}$ é vizinho do ponto $p_{i,j}$ então $p_{i,j}$ é vizinho de $p_{r,s}$, e disto se conclui que o grafo associado é não-dirigido.

Ainda, se for imposto que $|\lambda| < 1$, de acordo com a Definição 2, a matriz A é diagonalmente dominante e irredutível.

Então, o Teorema 2 garante a convergência do método de Gauss-Seidel e do método de Jacobi-Richardson aplicado ao sistema da Equação 3.18

Tal raciocínio pode ser transposto para a Equação 3.15 impondo $|\epsilon| < 2$, dado que é equivalente a impor $|\lambda| < 1$.

Pode ser visto que

$$|\lambda| = \frac{|\epsilon|}{2} < \frac{2}{2} = 1$$

então,

$$|\lambda| < 1$$

[6]vizinhança de Manhattan (Krause, 1986)

3.5.2 Diferenças Upwind

Sendo feitas as mesmas considerações da Sub-seção 3.5.1, será feita análise do pior caso, que é o mesmo anterior, isto é,$v_k = (1, 0)$, $v_k = (0, 1)$, $v_k = (-1, 0)$ ou $v_k = (0, -1)$. E, supondo, sem perda de generalização[7] que $\epsilon_{i,j} > 0$

$$|1| + |1| + |1| + |1 + h\epsilon_{i,j}| \leq |4 + h\epsilon_{i,j}| \tag{3.20}$$

A desigualdade da Equação 3.20 verifica-se sempre. Caso a célula analisada tenha ao menos um vizinho no contorno, haverá uma desigualdade (similar a da Equação 3.20), porém estrita.

Como existe ao menos uma célula com vizinhos no contorno, pode-se dizer que a matriz de iteração resultante da aplicação das diferenças upwind é diagonalmente dominante e irredutível (ou estritamente diagonalmente dominante num caso degenerado em que todas as células contém ao menos um vizinho no contorno).

Então, a aplicação do método de Jacobi-Richardson ou o método de Gauss-Seidel, utilizando as difereças upwind, para resolver o PVC do CPLO, é incondicionalmente convergente. Isto é, não existe a limitação $|\lambda| < 1$ (ou $|\epsilon| < 2$, a depender da formulação do problema).

3.6 Análise dos métodos apresentados

A aplicação da técnica de Campos Potenciais de Khatib à navegação autônoma de robôs móveis pode apresentar algumas vantagens e desvantagens. Entre as vantagens destacam-se (Faria, 2006):

1. o custo computacional bastante baixo se comparado a métodos no qual se utilizam mapas;

2. o sistema é reativo, ou seja, nenhuma modificação precisa ser feita para tratar os obstáculos dinâmicos pois o cálculo é refeito para cada nova posição do robô;

3. é um sistema modular no qual as forças podem ser calculadas em paralelo o que facilita implementações em hardware.

Como desvantagem tem-se que o método não realiza exploração de ambientes, pois não guarda informações ou modelos do ambiente. Uma outra desvantagem se refere as forças de atração e de repulsão, pois estas podem se anular em determinados pontos, gerando *mínimos locais*. Na Figura 3.4(a),(b)

[7] Pois bastaria trocar o sinal de $\epsilon_{i,j}$ e multiplicar v_k por -1 caso $\epsilon_{i,j} < 0$

e (c) pode-se observar a presença de mínimos locais representados por células em branco que não contém um vetor indicando a direção a ser seguida.

O problema de mínimos locais também pode gerar armadilhas no ambiente. Na Figura 3.4(b) mostra-se uma destas armadilhas no qual um robô que se aproxime pela parte superior do campo pode ser direcionado para um local do qual não há saída. Já na Figura 3.4(c) o método não consegue gerar um caminho até a meta pois esta tem sua força de atração anulada pelas paredes que estão próximas.

Um outro problema do método é encontrar valores para as constantes de atração (K) e de repulsão (Q) de forma a maximizar a eficiência do robô para um determinado ambiente. Isto pode ser observado ao aumentar em 4 vezes o valor de (Q) para o ambiente apresentado na Figura 3.4(b) obtendo-se o campo mostrado na Figura 3.4(d). Pode ser notado que o problema da armadilha foi solucionado no entanto a meta teve seu campo anulado pelo campo gerado pelos obstáculos próximos a ela, e portanto, não existe um caminho que leve a meta.

O método CPH, soluciona estes problemas do CP de Khatib e, têm sido utilizado para os mais variados fins, não somente para o problema de planejamento de caminhos para robôs. No trabalho de (Masoud, 2006), dois algoritmos são propostos para resolver o problema de encontrar o caminho mínimo entre dois vértices de um *grafo ponderado*(Tenenbaum et al., 1995). E, no trabalho de (Shi et al., 2007), é descrito um método para navegação autônoma de navios.

O método CPO também compartilha da propriedade do CPH de não possuír mínimos locais. Como visto nas Figuras 3.7(a)-(f), o módulo de ϵ pode ser interpretado como a *intensidade de influência* de um *vetor orientador*.

Em (Faria, 2006), é descrito um *Sistema Baseado em Regras* que, junto ao CPLO, é capaz de controlar um time de futebol de robôs de modo eficiente, com comportamentos como, por exemplo, o de atacante, goleiro ou defesa. Assim como no CPO, para que a solução numérica baseada em diferenças centradas tenha garantia de ser consistente e estável, é necessário impor que $|\lambda| < 1$ (ou, equivalentemente, $|\epsilon| < 2$)(da Silva et al., 2008).

Porém, conforme mostrado na Sub-seção 3.5.2, é possível utilizar a discretização upwind e, obter um método numérico incondicionalmente convergente. De acordo com o que é dito em (Silva et al., 2010a), as limitações apontadas por (Faria, 2006) são provenientes da discretização utilizada nas equações que regem as técnicas de campos potenciais baseadas em problemas de valor de contorno e, não das equações em si que, em sua definição, são contínuas.

Embora o CPLO permita controlar vários robôs utilizando uma mesma malha, isto não quer dizer que ele seja melhor. Se consideramos que cada agente

tem uma meta específica e única, o uso de um grid por agente é o ideal.

No caso do futebol de robôs, por exemplo, a posição da bola não é a meta de todos os robôs de um mesmo time pois, se assim fosse, haveria uma competição entre os agentes.

Com o uso do CPO ou CPH é possível definir metas específicas para cada agente, evitando que exista competição: Enquanto o agente com papel de goleiro tente defender o gol, o agente com papel de atacante poderia se posicionar para uma jogada de contra-ataque.

Tanto o CPH, CPO como o CPLO podem ser utilizados como técnicas de planejamento de caminhos em outros ambientes e aplicações que não sejam o de futebol de robôs. Por exemplo, numa tarefa de exploração e mapeamento de ambientes em um ambiente pouco estruturado.(Prestes, 2003)

3.7 Considerações Finais

Neste capítulo foram apresentadas as técnicas de campos potenciais mais utilizadas, os campos potenciais de Khatib e os campos potenciais harmônicos, bem como duas variações que adicionam funcionalidades aos campos potenciais harmônicos, os campos potenciais orientados e o campos potenciais localmente orientados.

Entre as possíveis aplicações das técnicas descritas neste capítulo, há as diversas categorias das competições de futebol de robôs, descritas no Capítulo 5.

CAPÍTULO

4

Arquiteturas Paralelas

Em (Moore, 1965), Gordon Moore apresentou o que hoje conhecemos como Lei de Moore: *"The number of transistors incorporated in a chip will approximately double every 24 months."*[1]. Podemos observar que os computadores estão em constante evolução, e continuam seguindo a Lei de Moore, desde 1965.

Atualmente, temos microprocessadores com bilhões de transistores em uma área de poucos centímetros quadrados. Novos processos são capazes de produzir transistores com $22nm$ de comprimento, apenas 200 vezes maior que o raio atômico de um átomo de Silício, elemento químico utilizado na fabricação destes.

Conforme (Sutter, 2005; Post, 2011), até 2004 a frequência de operação dos microprocessadores aumentava uma ordem de grandeza por década. Porém, em 2004, devido a propriedades eletrônicas da tecnologia de fabricação de chips de silício, tornou-se extremamente difícil aumentar a frequência de operação sem elevar o consumo de energia a níveis inaceitáveis. Desde 2004, os processadores não quebraram a barreira dos $4GHz$.

Frequências de operação mais altas de operação, como $5GHz$ ou $6GHz$ podem ser obtidas através de overclocking, porém, é necessário uso de nitrogênio líquido a -196^oC (ou $77K$) no sistema de refrigeração. No site da AMD[2] encontra-se um experimento com hélio líquido a -270^oC (ou $4K$), no qual um processador atingiu a marca de $7GHz$. Mas, tal uso além de diminuir a vida útil do computador, é extremamente caro, chegando a custar algumas vezes mais o preço do mesmo.

Então, pode ser visto que não é mais válida a interpretação Lei de Mo-

[1] O número de transistores em um chip dobrarão a cada 24 meses

[2] http://links.amd.com/LightSpeed

ore ser da frequência de operação de um microprocessador aumentar exponencialmente. A partir de 2004 fabricantes de microprocessadores passaram a aumentar a quantidade de núcleos (cores) em cada processador, com isto mantendo a Lei de Moore. Isto levou ao que atualmente chamamos de arquiteturas *multicore* (com algumas dezenas de núcleos) e *manycore* (com centenas ou milhares de núcleos).

Os microprocessadores da Intel, que foram utilizados neste trabalho, podem ser vistos como multicore, por conterem menos de 10 núcleos por processador. Já os microprocessadores da NVidia(Nvidia, 2011) podém conter centenas de núcleos.

Até 2004, as aplicações rodavam mais rapidamente, ano após ano, sem que houvesse mudanças ou com mudanças pouco significativas no código. Um programa desenvolvido para funcionar sequencialmente, não roda duas vezes mais rápido em um processador com dois cores. Para isto, é preciso estabelecer explicitamente que porções do código serão paralelizadas.

Ainda em (Post, 2011), ele descreve as implicações de um estudo do *National Research Concil (NRC)* do uso destas arquiteturas:

- Códigos massivamente paralelos são mais complexos que os seus equivalentes sequenciais, e as ferramentas de progração disponíveis são rudimentares;
- Modificar aplicações existentes para explorar estas novas arquiteturas será acompanhado de enorme esforço e pode nem ser possível, dado que nem todo código sequencial pode ser paralelizado;
- As arquiteturas para computadores massivamente paralelos continuam evoluindo rapidamente, então programadores estarão tentando atingir um alvo em movimento.

Em (Flynn, 1972), estabeleceu uma taxonomia que, apesar de incompleta, é utilizada até hoje devido a facilidade de compreensão. São quatro os tipos de computadores:

SISD De *Single Instruction, Single Data*. É o computador clássico, serial ou sequencial, a depender do autor.

Apenas uma única operação é executada em um determinado ciclo, bem como apenas uma fonte de dados. É o tipo de computador mais comum atualmente. (Barney, 2011) O fluxo de execução de um computador com arquitetura SISD pode ser visto na Figura 4.1.

SIMD De *Single Instruction, Multiple Data*. É um tipo de computador pararelo, no qual todas as unidade de processamento tem a mesma instrução.

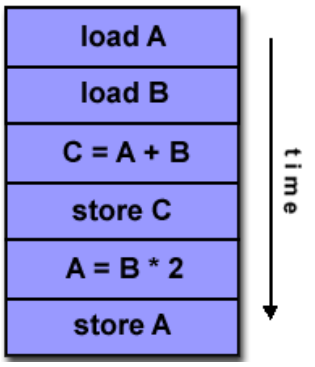


Figura 4.1: SISD: Single Instruction, Single Data (Barney, 2011)

Porém, cada unidade pode processar dados diferentes. Computadores modernos que possuem unidades de processamento gráfico (Graphics Processing Units - GPU) dedicadas, utilizam instruções do tipo SIMD. (Barney, 2011) O fluxo de execução de um computador com arquitetura SIMD pode ser visto na Figura 4.2.

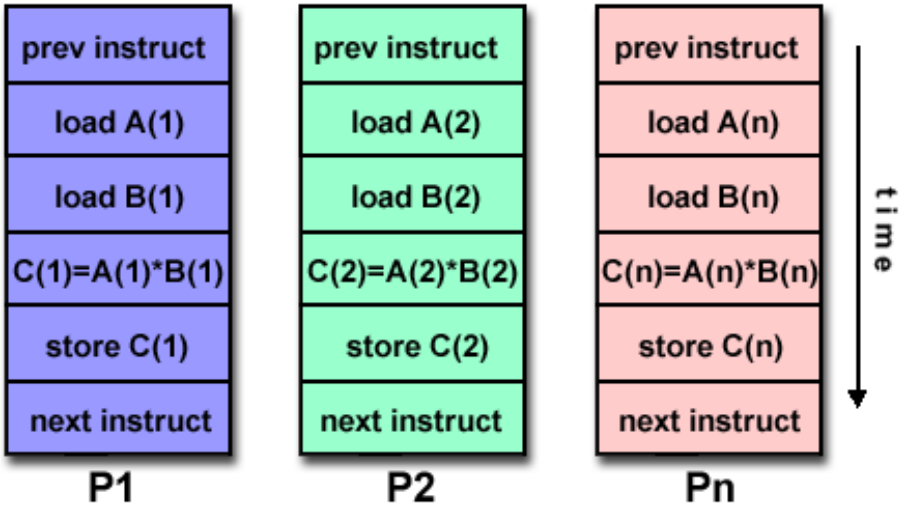


Figura 4.2: SIMD: Single Instruction, Multiple Data (Barney, 2011)

MISD De *Multiple Instruction, Single Data*. É um tipo de computador pararelo, no qual cada unidade de processamento pode ter uma sequência de instruções única, porém, há uma única fonte de dados. Em (Barney, 2011) é dito que poucos computadores com esta arquitetura foram construídos.

O fluxo de execução de um computador com arquitetura MISD pode ser visto na Figura 4.3.

MIMD De *Multiple Instruction, Multiple Data*. É um tipo de computador pararelo, no qual cada unidade de processamento pode ter uma sequência de instruções e fonte de dados únicas. Em (Barney, 2011), supercomputadores, grids e clusters, e computadores multicore são exemplos desta arquitetura.

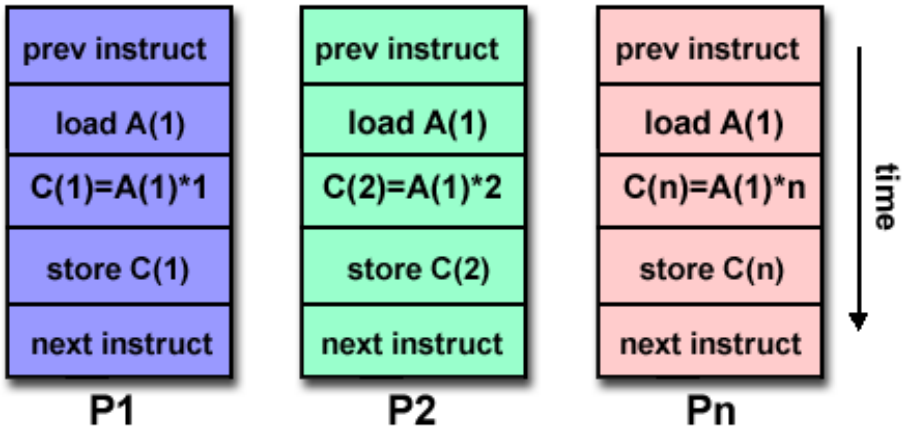


Figura 4.3: MISD: Multiple Instruction, Single Data (Barney, 2011)

O fluxo de execução de um computador com arquitetura MIMD pode ser visto na Figura 4.4.

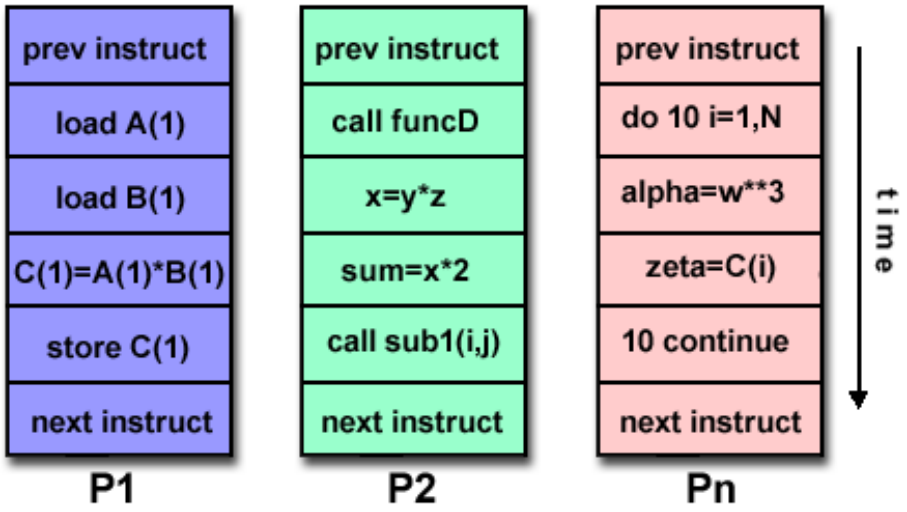


Figura 4.4: MIMD: Multiple Instruction, Multiple Data (Barney, 2011)

Uma das maneiras mais comuns de se medir o desempenho de uma arquitetura paralela sobre uma arquitetura sequencial é calculando o *speedup* S, como na Equação 4.1

$$S = \frac{T_s}{T_p} \tag{4.1}$$

onde T_s é o tempo de execução do aplicativo numa arquitetura sequencial e T_p é o tempo de execução do mesmo aplicativo numa arquitetura paralela.

4.1 CUDA

O primeiro passo para o surgimento das GPGPU (General-Purpose computation on Graphics Processing Units[3]), também grafado como GP²U, foi o sur-

[3]Programação de propósito geral em unidades de processamento gráfico

gimento de GPUs que possibilitavam a reprogramação de *shaders* do *pipeline gráfico*. Shaders podem ser vistos como pequenos funções que são chamadas para realizar operações de renderização, transformações geométricas e outras operações gráficas.

O principal tipo de shader é o *pixel shader*, que executa operações em cada um dos pixels de um objeto e devolve a cor deste pixel. Para gerar a cor de saída, o pixel shader pode utilizar várias informações como, por exemplo: posição XY na tela, posição XYZ na cena, posição na textura, direção do vetor normal a superfície e etc.

Um exemplo de como vários shaders podem ser aplicados ao mesmo objeto pode ser visto na Figura 4.5. O shader A sempre retorna a mesma cor. O shader B varia a cor baseando-se na coordenada Y. O shader C gera a cor baseando-se nas coordenadas tridimensionais do objeto dentro da cena. O shader D gera tons de cinza proporcionais ao coseno do ângulo entre o vetor normal à superfície e o vetor de iluminação ("iluminação difusa"). O shader E utiliza um modelo de iluminação mais complexo e adiciona textura ao objeto e, o shader F intruduz uma perturbação no vertor normal.

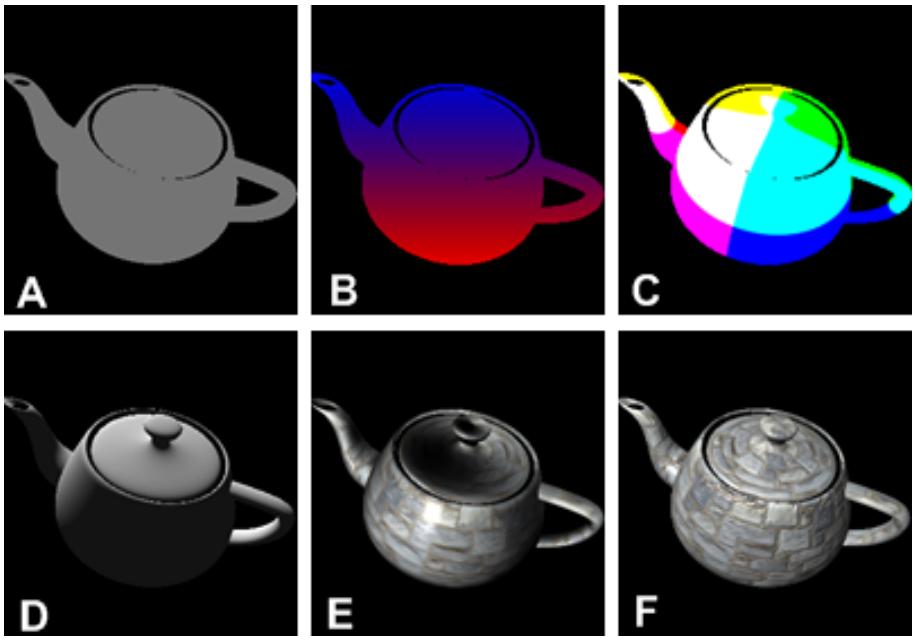


Figura 4.5: Diversos pixel shaders aplicados a um bule (Ostrovsky, 2011)

Um outro pixel shader apresentado em (Ostrovsky, 2011) é um *operador de desfocagem*[4], como na Figura 4.6. Este shader é uma operação em um grid (imagem) que leva em conta pontos vizinhos (pixels vizinhos). Abstraindo este conceito, é possível aplicar um pixel shader praticamente identico ao anterior, num grid, que resolve a equação do calor, como na Figura 4.1.

Uma vez demonstrada a capacidade das GPUs de resolvem problemas com

[4]blur operator

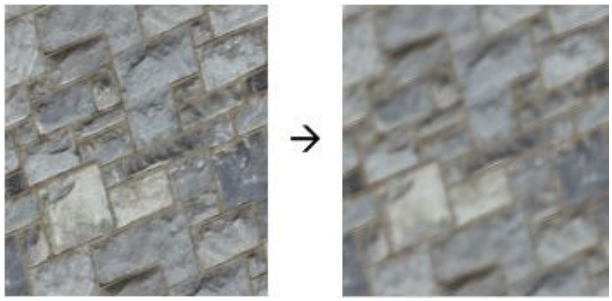


Figura 4.6: Pixel shader que aplica uma transformação numa imagem(Ostrovsky, 2011)

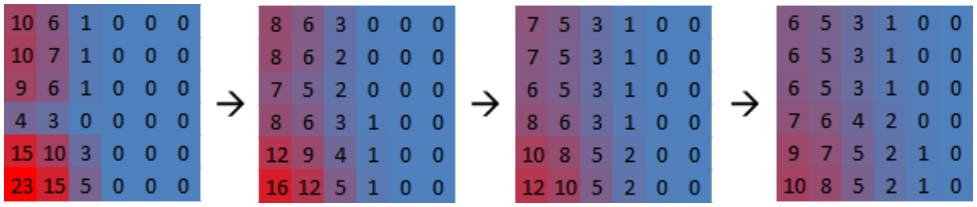


Figura 4.7: Pixel shader que aplica uma transformação num grid(Ostrovsky, 2011)

alto custo computacional (como encontrar a solução de uma Equação Diferencial Parcial), fabricantes de GPUs passaram a fabricar GPUs para computação de propósito geral (GPGPU). Isto se deu com a criação do conceito de kernel, que é uma generalização de um pixel shader:

- Pixel shaders sempre atuam em um ambiente bi-dimensional, enquanto um kernel pode trabalhar com espaços de dimensão N;
- Pixel shaders sempre retornam uma única cor. Kernels podem retornar um número arbitrário de dados;
- Pixel shaders operam apenas com números ponto-flutuantes de 32 bits (single float). Kernels, além disto, suportam o uso de números inteiros e ponto-flutuantes de 64 bits (double float).
- Pixel shaders lêem dados de texturas, porém kernels podem ler dados de praticamente qualquer lugar da memória da GPU.

O controle de fluxo de operação de aplicativos modernos costumam ser complexos. Em (Ostrovsky, 2011), como exemplo, cita aplicações web, nas quais é necessário a CPU comunicar-se com vários dispositivos, manter diversas estruturas de dados, posição de diversos elementos gráficos e etc. Não só são tarefas distintas, como eles dependem uma das outras de muitas maneiras. Já o controle de fluxo de um pixel shader é muito mais simples, pois consiste em um rotina pequena, chamada milhões (senão bilhões) de vezes por segundo, cada vez com uma entrada diferente. Então, um hardware otimizado

para rodar um pixel shader será diferente de um hardware otimizado para visualizar páginas web. Isto pode ser observado na Figura 4.8. Uma GPU destina a maior parte dos transistores para as *unidades lógico-aritméticas* (ULA)[5], enquanto que uma CPU tem um controle de fluxo muito mais complexo, e memória cache maior.

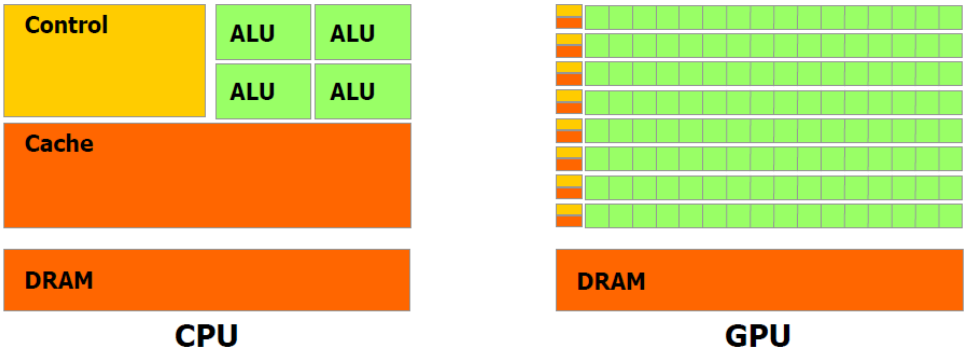


Figura 4.8: Alocação de transistores em uma CPU e em uma GPU(Nvidia, 2011)

Essa diferença entre CPU e GPU pode ser melhor acompanha nas Figuras 4.9 e 4.10. Na Figura 4.9 pode ser visto que uma GPU comercial, acessível a um usuário doméstico, é capaz de atigir mais de 1.5 teraflops por segundo, isto é, 130 vezes mais rápido que o Deep Blue, supercomputador que em 1997 demonstrou a capacidade de uma máquina vencer um grande mestre em partidas de xadrez.

Em 2006 surgiu a *Compute Unified Device Architecture*, ou simplesmente *CUDA*, que é uma arquitetura que facilita o desenvolvimento de aplicações voltadas a chipsets[6] da Nvidia. Entre as diversas possibilidades apresentadas pela Nvidia, há o *C for CUDA*, que é uma extensão da linguagem C que pode gerar programas que rodam na GPU.

Uma GPU pode ter centenas de núcleos, mas, normalmente, eles não podem se comunicar diretamente com todos. Dentro das placas da Nvidia, os cores, chamados de *stream processors* (SP) são agrupados em multicores denominados *stream multiprocessors* (SMP), como na Figura 4.11. Os primeiros chipsets programáveis (G80(Nvidia, 2011)) tinham 8 SMPs com 16 SPs cada um, totalizando 128 SPs. Os chipsets mais novos possuem 32 SMPs com 32 SPs cada um, totalizando 1024 SPs.

Esta organização física induz particularidades no desenvolvimento do software, como por exemplo com quais SPs um SPs em particular pode cooperar (apenas os SPs que estiverem no mesmo SMP), ou qual memória tem a menor latência.

[5]ALU - Arithmetic Logic Unit

[6]Atualmente a Nvidia fornece unidades "GPGPU reais" (série Tesla), isto é, unidades que sequer tem saída para monitores, servindo apenas para progração paralela.

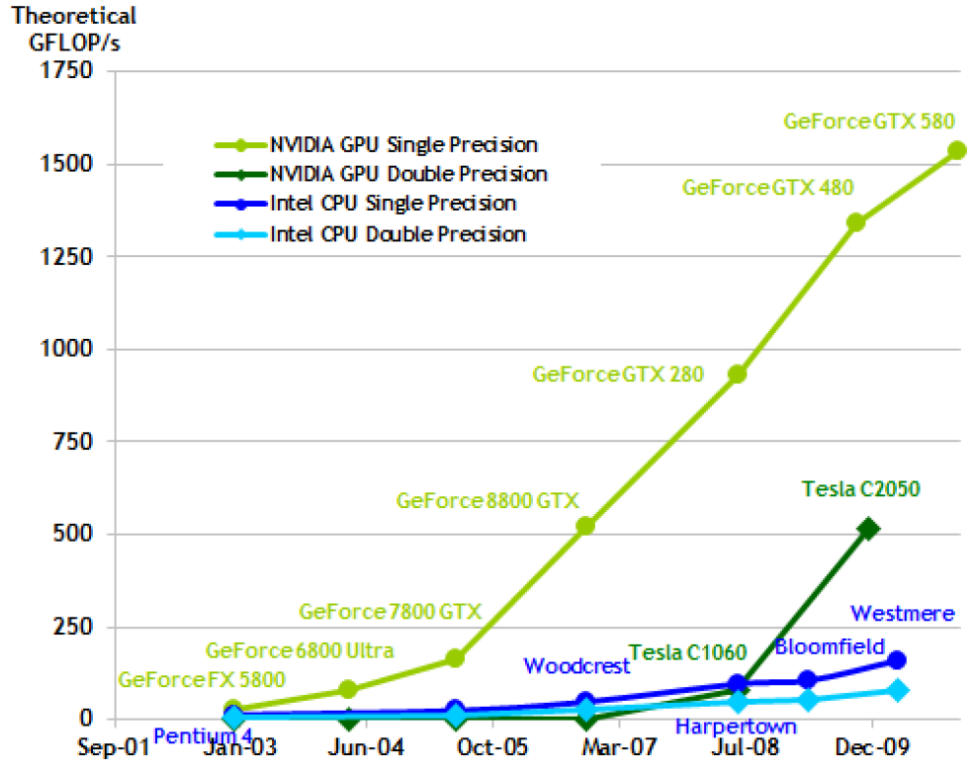


Figura 4.9: Comparação entre CPUs e GPUs em quantidade de operações de ponto flutuante simples por segundo(Nvidia, 2011)

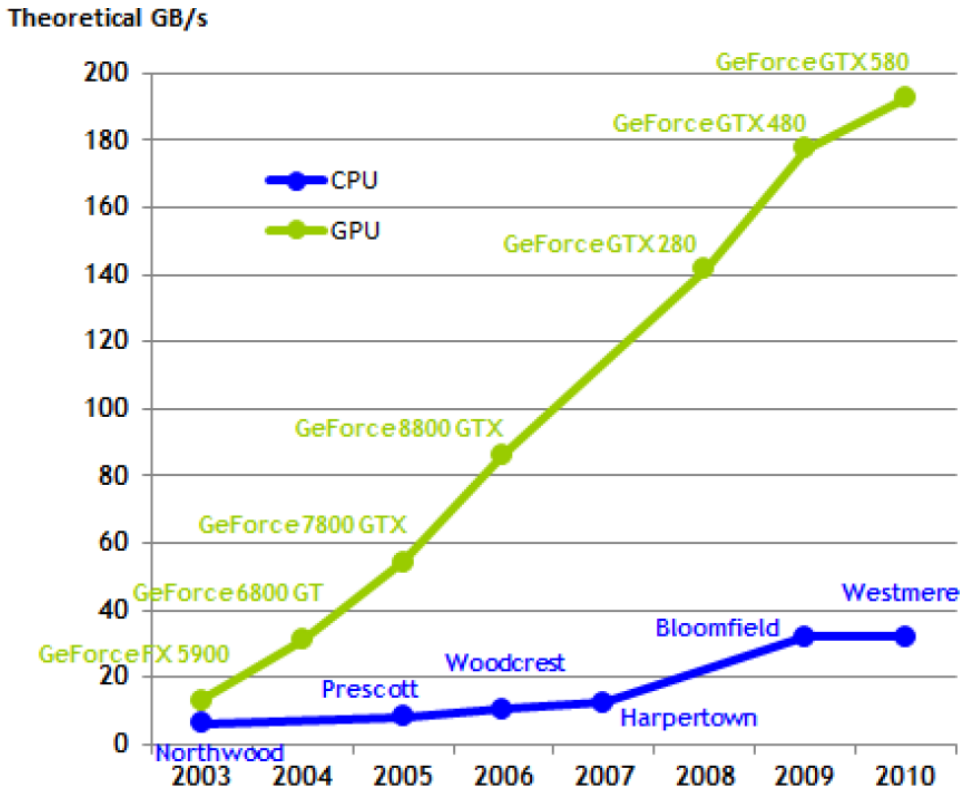


Figura 4.10: Comparação entre CPUs e GPUs em Largura de Banda da Memória (Memory Bandwidth)(Nvidia, 2011)

No desenvolvimento de um software utilizando CUDA é necessário levar em

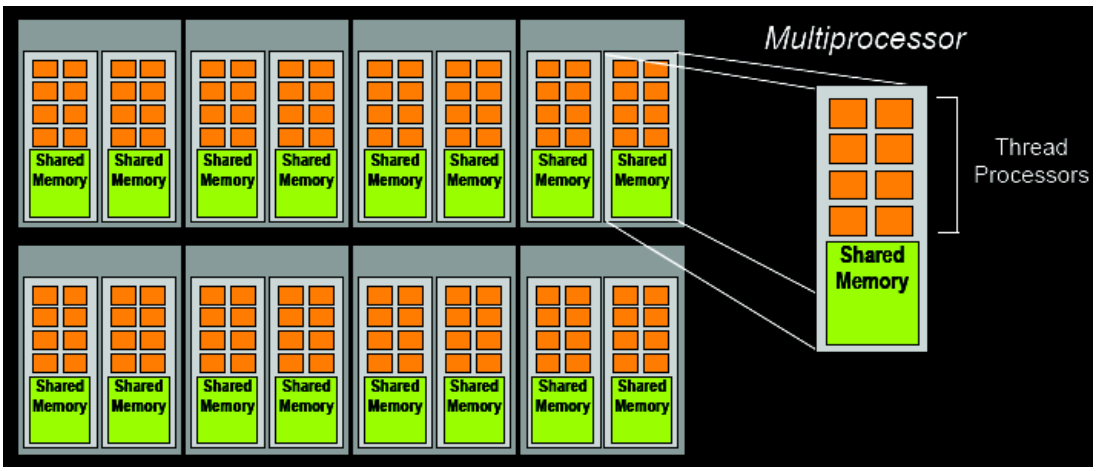


Figura 4.11: Chipset simplificado de uma GPU (Nvidia, 2011)

conta a hierarquia de threads do CUDA, como na Figura 4.12. Quando um kernel é lançado, as threads estão agrupadas em blocos que, por sua vez, estão agrupados em um grid. Em chipsets mais antigos, os grids só poderiam ser unidimensionais ou bidimensionais. Em chipsets modernos, é possivel ter grids tridimensionais. A independência entre os blocos precisa ser levada em conta, dado que para diferentes hardwares, eles poderão ser executados em sequências diferentes, como na Figura 4.13.

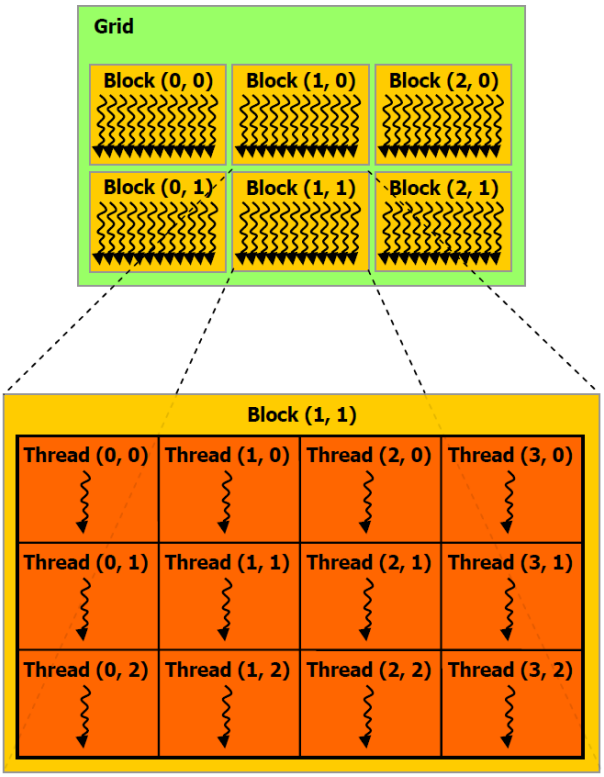


Figura 4.12: Hierarquia de threads (Nvidia, 2011)

Nos chipsets mais antigos, somente um único kernel poderia ser processado por vez, limitando severamente o uso concorrente da GPU para proces-

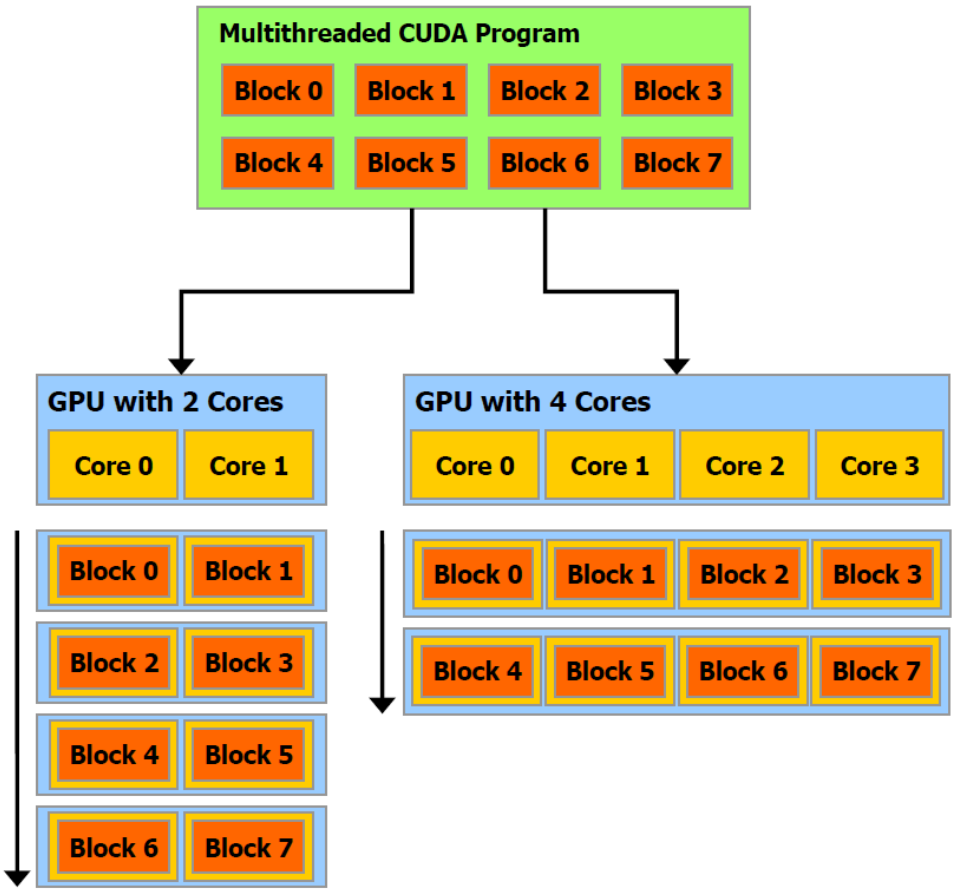


Figura 4.13: Escalabilidade automática (Nvidia, 2011)

samento gráfico e propósito geral em PCs com apenas uma GPU. Nos chipsets mais novos, havendo disponibilidade, os kernels serão executados em paralelo. Isto pode ser observado na Figura 4.14

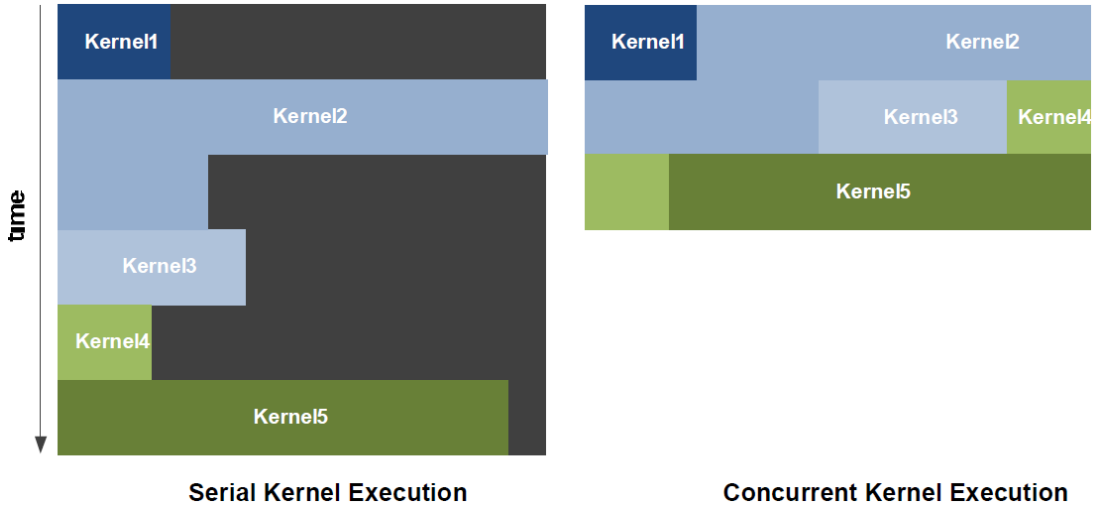


Figura 4.14: Execução sequencial de kernels em chipsets antigos e execução concorrente de kernels em chipsets novos (Nvidia, 2011)

O compilador utilizado no *C for Cuda* é o *NVidia C Compiler*, ou *nvcc*. O

nvcc separa o código que rodará em GPU e passa para *CUDA C Compiler*, ou *cudacc*, e o código que rodará na CPU para o compilador C padrão do sistema operacional (neste caso, o Gnu C Compiler, ou *gcc*).

Dentro da GPU há diversos tipos e acessos a memória. Os principais tipos de memória, a saber, memória global (Figura 4.18), memória compartilhada (Figura 4.17), memória local e registrador (Figura 4.16), são apresentados na Figura 4.15, bem como sua respectiva localização no chip.

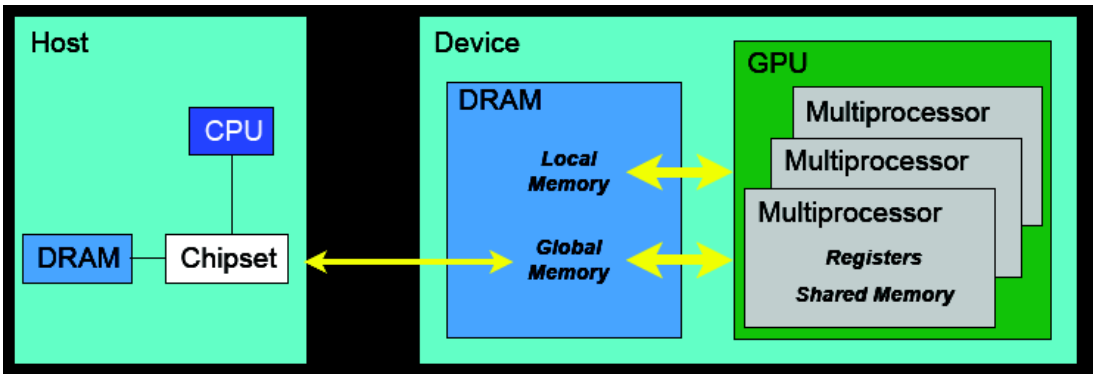


Figura 4.15: Localização física de memórias no dispositivo (Nvidia, 2011)

Figura 4.16: Memória local e registrador (Nvidia, 2011)

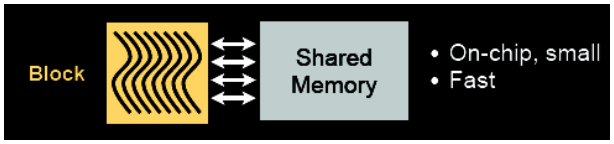


Figura 4.17: Memória compartilhada (Nvidia, 2011)

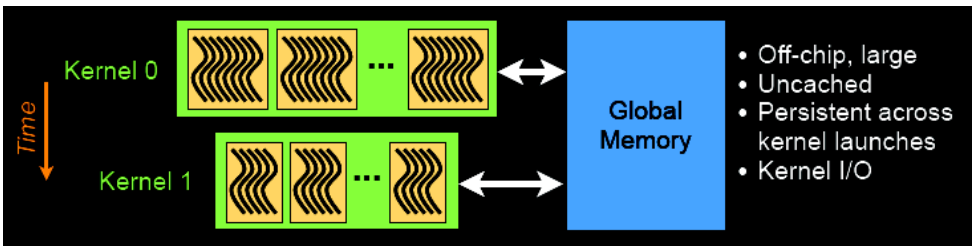


Figura 4.18: Memória global (Nvidia, 2011)

4.2 Implementação em CUDA dos métodos de campos potenciais

Nesta seção, aspectos importantes da implementação em CUDA serão discutidos. Conforme dito anteriormente, a implementação em paralelo de um código pode diferir significativamente da implementação sequecial.

A implementação em CUDA dos métodos de campos potenciais deste projeto foi idealizada de forma a encapsular todas as chamadas ao *C for CUDA*. Isto deve-se a várias incompatibilidades que surgiram entre a API do CUDA e outras APIs, especialmente a Qt, que fornece vários recursos necessários ao desenvolvimento de um sistema inteligente, o qual o módulo de campos potenciais é uma parte.

Tendo por base a implementação sequencial, a estrutrura do código ficou como na Figura 4.19. Cada componente, tem uma função específica, como segue:

BE de BASE ENVIRONMENT, é responsável por gerenciar as estruturas mais básicas e comuns, como adicionar e remover obstáculos, metas, robôs a serem controlados, robôs adversários (ou, em uma aplicação diferente de futebol de robôs, poderia significar um agente robótico não controlável);

HPF de HARMONIC POTENTIAL FIELD, utilizando os recursos de BE para gerenciar os objetos do ambiente, é responsável por representar o ambiente em uma grade[7], tanto em CPU quanto em GPU. Também contém a implementação do método de Campos Potenciais Harmônicos;

OPF de ORIENTED POTENTIAL FIELD, implementa o método de Campos Potenciais Orientados;

LOPF de LOCALLY ORIENTED POTENTIAL FIELD, implementa o método de Campos Potenciais Localmente Orientados;

Para entender o funcionamento da estrutura, toma-se como exemplo o OPF, adicionando um obstáculo, uma meta e um robô a ser controlado, e utilizando o método de Gauss-Seidel. Ao adicionar cada um destes objetos, as funções do componente OPF chamam funções do componente HPF, que adicionam as informações dos objetos no grid e chamam funções do componente BE, que adicionam as informações na estrutrura básica do ambiente. Em seguida, ao chamar uma função que implementa o método de Gauss-Seidel, esta chama uma função do componente HPF que retorna os endereços de memória dos recursos alocados na GPU e na CPU. Então, de posse destas informações, e de

[7] ou grid

um critério de parada pré-estabelecido, uma iteração consiste em lançar um kernel com implementação do Gauss-Seidel e, logo em seguida, um segundo kernel para o cálculo de estimativa de erro.

Implementado desta forma, encapsulado e compartimentado, espera-se que manutenções futuras no código não sejam onerosas.

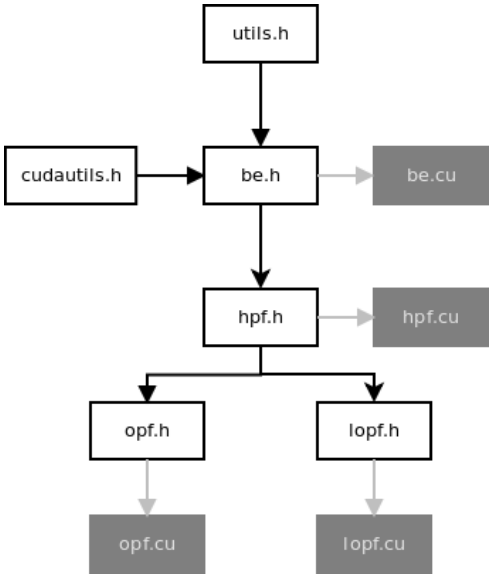


Figura 4.19: Diagrama dos módulos

Na Figura 4.20 é mostrado o fluxo de dados em uma iteração do método de Gauss-Seidel aplicado ao CPO com discretização upwind. Os outros métodos seguem o mesmo fluxo, com ressalvas respectivas particularidades, como no caso do CPH que não envia dados de orientação (v e ϵ), enquanto que o CPLO envia uma informação para cada robô e respectivo raio da área de influência.

A comunicação entre a GPU e a CPU é feita através da PCI Express x16, em que a largura de banda de memória é de $8GB/s$. Essa transferência pode ser vista como o maior gargalo do código, dado que as taxas de transferência entre a CPU ou a GPU e seu respectivo espaço de memória (Figura 4.10) é muito maior.

A fim de evitar problemas de concorrência, foi adotado uma abordagem de calculo de erro baseada no uso de dois kernels. Ao calcular o novo valor de uma determinada célula da grade, o iterador calcula a "distância" entre estes dois valores, tal como o código sequencial em CPU e, então, atribui este valor em uma segunda grade, que tem o objetivo de armazenar os erros (grade de erros na Figura 4.20). Então, um segundo kernel é lançado para efetuar a *redução de soma pararela*[8].

A função iteradora em CPU é responsável por fazer a movimentação dos dados entre os espaços de memória da GPU e da CPU, chamar os kernels e,

[8]parallel sum reduction

devolver o erro daquela iteração. Desta forma, todas as chamadas ao CUDA estão encapsuladas, e uma aplicação que faça uso do sistema implementado neste trabalho torna-se mais simples e clara.

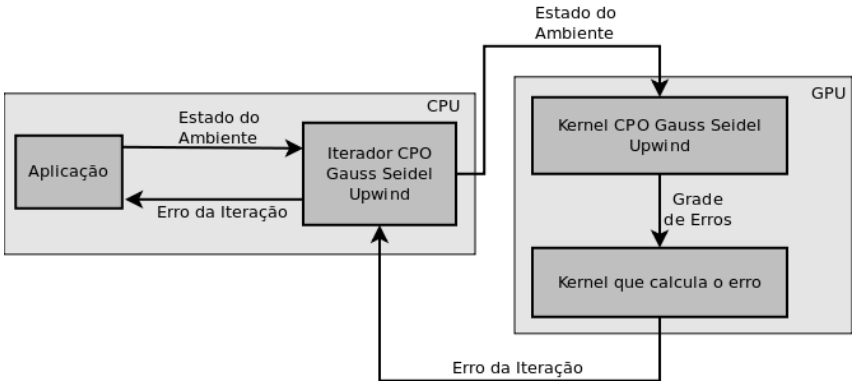


Figura 4.20: Fluxo de uma iteração utilizando CUDA

O Algoritmo 7 descreve o iterador apresentado na Figura 4.20. Nas linhas 1 a 5, são definidos os parâmetros das chamadas ao kernel iterador (Algoritmo 8). Na linha 2, há uma compensação devido ao espalhamento, ou hashing (Algoritmo 9), para evitar que os recursos da GPU sejam superestimados. Nas linhas 6 e 7, o kernel (Algoritmo 8) é lançado para as células pretas e vermelhas, de forma paralela sem concorrência.

A alocação de matrizes de forma linear (linhas ou colunas são enfileiradas), permitiu otimizar e simplificar parte do código, como pode ser visto nas linhas 8 e 9, nas quais são definidos os parâmetros da chamada do kernel redutor. Na linha 10, a chamada ao kernel iterador resulta em $reductionGridSize$ somas parciais, que são reduzidas com uma nova chamada ao kernel, na linha 11. A forma com a qual este kernel foi implementada tem um limite teórico de $reductionGridSize \times reductionBlockSize = 33553920$, ou um grid de 5792×5792, pois $reductionBlockSize$ é fixado em 512, e $reductionGridSize <= 65535$. Com isto, na linha 12, a soma reduzida de todo o grid de erros é trazida para a memória da CPU e utilizada como parâmetro de retorno.

O Algoritmo 8 apresenta o kernel do iterador do CPO, utilizando RedBlack-Gauss-Seidel e diferenças Upwind. Nas linhas 1 e 2, é obtida a posição (j, i) da célula que será atualizada no grid, por esta cudathread. Na linha 3 é executado o procedimento de espalhamento (hashing), que é apresentado no Algoritmo 9. Nas linhas 4 a 6 é feita uma checagem para evitar que cudathreads excedentes realizem cálculos indevidos. Nas linhas seguintes, o algoritmo é igual ao conteúdo do laço de repetição da versão sequencial do Gauss-Seidel utilizando Upwind, com a excessão da forma do cálculo de erro que, aqui não é acumulado, sendo apenas guardado em um grid para posterior operação de redução.

Algoritmo 7 Iterador CUDA CPO Red-Black Gauss Seidel Upwind

requer Um ambiente e para CPO

garante O erro de uma iteração do método de Gauss-Seidel utilizando diferenças Upwind, em GPU, aplicado ao grid contido em e

```
1: procedimento CUDA_CPO_GS_UP(e)
2:     iterationBlockSize ← 32
3:     dimGrid.x ← round( width(e)/(2*iterationBlockSize) )
4:     dimGrid.y ← round( height(e)/iterationBlockSize )
5:     dimBlock.x ← iterationBlockSize
6:     dimBlock.y ← iterationBlockSize
7:     iteratorGS_RB_UP_kernel<<< dimGrid,dimBlock >>>(e, 0)        ▷ Algoritmo 8
8:     iteratorGS_RB_UP_kernel<<< dimGrid,dimBlock >>>(e, 1)        ▷ Algoritmo 8
9:     reductionBlockSize ← 512
10:    reductionGridSize ← round( width(e)*height(e)/ reductionBlockSize )
11:    reduceKernel<<< reductionGridSize,reductionBlockSize >>>(e)
12:    reduceKernel<<< 1,reductionGridSize >>>(e)
13:    cudaMemcpy(error, deviceErrorBuffer(e) )
14:    retorna error
15: fim procedimento
```

4.3 Considerações Finais

Neste capítulo foram apresentados conceitos de computação paralela que serão utilizados para desenvolver este projeto, bem como aspectos da arquitetura CUDA e da implementação em C for CUDA dos métodos de planejamento de caminhos. Pode ser visto que o pseudocódigo do Algoritmo 8 é similar com a sua versão sequencial, no núcleo do Algoritmo 4, mas a forma como são executados difere bastante. Em CPU, há uma ordem de execução bem estabelecida, definida pelos índices dos laços j e i. Em GPU, não há uma ordem bem estabelecida, sendo que grupos células estão sendo executadas paralelamente, a depender do dispositivo utilizado.

Programação em CUDA, e em geral, programação em GPGPU, ainda é algo recente e está em constante evolução. Ainda não há muitos recursos avançados, como os disponíveis em CPU. Entre eles, por exemplo, o cacheamento de memória precisa ser feito pelo desenvolvedor de forma explícita. Mesmo o uso de recursividade, ensinado como ferramenta básica em cursos introdutórios de programação, em GPU ainda não está completamente disponível.

Algoritmo 8 CUDA CPO Red-Black Gauss Seidel Upwind Kernel

requer Um ambiente e para CPO
requer Uma flag $hashFlag$ para identificar o tipo de célula (vermelha ou preta)
garante Uma iteração do método de Gauss-Seidel utilizando diferenças Upwind, em GPU, aplicado ao grid contido em e

1: **procedimento** ITERATORGS_RB_UP_KERNEL(e, $hashFlag$)
2: $i \leftarrow blockIdx.y * blockDim.y + threadIdx.y$
3: $j \leftarrow blockIdx.x * blockDim.x + threadIdx.x$
4: $(j, i) \leftarrow rbHash(j, i, hashFlag)$ ▷ Algoritmo 9
5: **se** $i >= height(e)$ **ou** $j >= width(e)$ **então**
6: **retorna**
7: **fim se**
8: **se** $FreeCell =$ getOccupancy(e, j, i) **então**
9: $oldPotential \leftarrow$ getPotential(e, j, i)
10: $(top, bottom, left, right) \leftarrow$ getNeighborhood(e, j, i) ▷ Algoritmo 2
11: $ro.y \leftarrow 1+$ getEpsilon(e) * abs(sin(getVector(e)))
12: $ro.x \leftarrow 1+$ getEpsilon(e) * abs(cos(getVector(e)))
13: $den = 2 + ro.x + ro.y$
14: **se** sin(getVector(e)) > 0 **então**
15: **se** cos(getVector(e)) > 0 **então**
16: $newPotential \leftarrow (ro.x * right + left + ro.y * top + bottom)/den$
17: **senão**
18: $newPotential \leftarrow (right + ro.x * left + ro.y * top + bottom)/den$
19: **fim se**
20: **senão**
21: **se** cos(getVector(e)) > 0 **então**
22: $newPotential \leftarrow (ro.x * right + left + top + ro.y * bottom)/den$
23: **senão**
24: $newPotential \leftarrow (right + ro.x * left + top + ro.y * bottom)/den$
25: **fim se**
26: **fim se**
27: setErrorGrid(e, j, i, $(newPotential - oldPotential)^2$)
28: setPotential(e, j, i, $newPotential$)
29: **senão**
30: setErrorGrid(e, j, i, 0)
31: **fim se**
32: **retorna**
33: **fim procedimento**

Algoritmo 9 Red-Black Hashing

requer Uma flag $hashFlag$ para identificar o tipo de célula (vermelha ou preta)
requer Uma posição (j, i)
garante Uma posiçao (j, i) espalhada de acordo com $hashFlag$

```
1: procedimento rbHash(hashFlag, j, i)
2:     se hashFlag = 1 então
3:         se (i mod 2) = 1 então
4:             j ← (j * 2) + 1
5:         senão
6:             j ← j * 2
7:         fim se
8:     senão
9:         se (i mod 2) = 1 então
10:            j ← j * 2
11:        senão
12:            j ← (j * 2) + 1
13:        fim se
14:    fim se
15:    retorna (j, i)
16: fim procedimento
```

CAPÍTULO

5

Aplicações de Campos Potenciais em 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; Augusto Silva 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 comite 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

[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

Laboratories (sediado em Seattle, EUA), e o time *SOTY*, do KAIST, foram campeão e vice-campeão mundial, respectivamente. Inicialmente, a FIRA focava 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 extendida 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.

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

5.1 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-se em dois aspectos principais: tamanho do campo e quantidade de robôs por time. A categoria *IEEE Very Small Size* utiliza 3 robôs e um campo de $150cm \times 130cm$, enquanto que a MiroSot tem duas categorias, uma com 5 robôs, na qual é utilizada um campo com $220cm \times 180cm$ e outra categoria com 11 robôs, na qual é utilizada um campo com $400cm \times 280cm$.

O objetivo de um time nesta categoria é obter o maior saldo de gols durante uma partida (que tem dois tempos de 5 minutos e um intervalo de 10 minutos entre estes).

5.2 USPDroids

A equipe *USPDroids* é composta por alunos de graduação e pós-graduação (em sua maioria, bolsistas) do campus de São Carlos da USP. Iniciou suas atividades em 2005, encerrando em 2010 e, obteve os seguintes títulos:

- TERCEIRO LUGAR na categoria *IEEE Very Small Size* na *Latin American Robots Competition*, em Salvador/BA - 2008;
- VICE-CAMPEÃO na categoria *IEEE Very Small Size* na *Competição Brasileira de Robótica*, em Brasília/DF - 2009.
- VICE-CAMPEÃO na categoria *IEEE Very Small Size* na *Latin American Robots Competition*, em São Bernardo do Campo/SP - 2010.
- TERCEIRO LUGAR na categoria *RoboCup 2D Simulated Soccer League* na *Latin American Robots Competition*, em São Bernardo do Campo/SP - 2010.

Foi desenvolvido um time de futebol de robôs para participar de competições *IEEE Very Small Size*. Abaixo, segue alguns detalhes do sistema integrado utilizado no futebol de robôs. Como pode ser visto em (Silva et al., 2009), o time (Figura 5.2) é composto por quatro módulos (Figura 5.1):

- **Visão** Módulo responsável pela localização dos robôs e classificação entre robôs *aliados* e robôs *inimigos*;
- **Estratégia** Módulo responsável pelas decisões (autônomas), do sistema.
- **Eletrônica** Módulo responsável pelo tratamento das informações recebidas via rádio, e controle dos motores.

- **Mecânica** Módulo que constitui a parte física do robô (carcaça, rodas, engrenagens e etc).

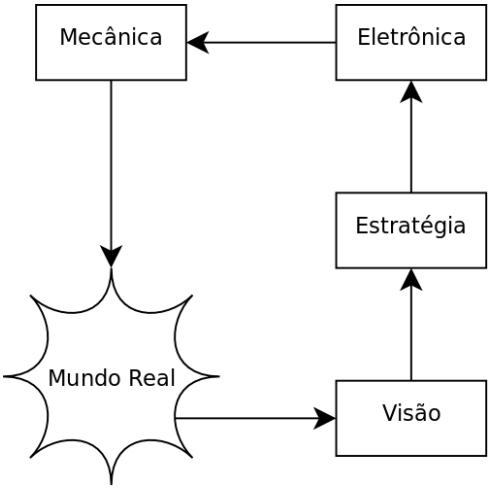


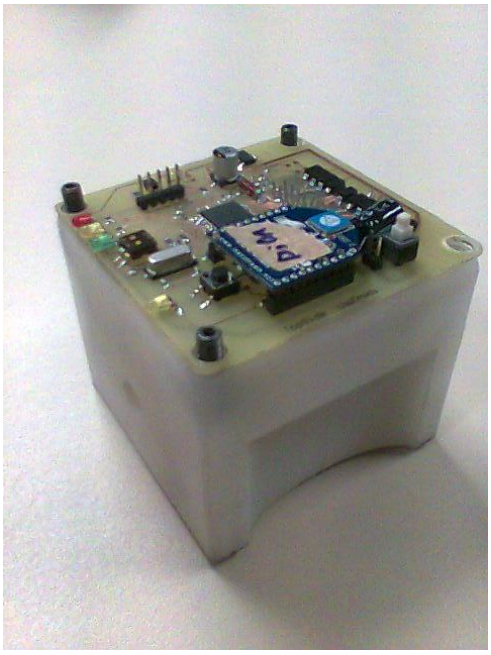
Figura 5.1: Visão geral dos módulos e suas relações de fluxo de informação

Figura 5.2: Campo oficial das competições IEEE Very Small Size e time USP-Droids/2009

5.3 Warthog Robotics

Em 2011, as duas equipes de desenvolvimento de robôs capazes de disputar partidas de futebol robótico (USPDroids e GEAR) uniram-se, criando uma nova equipe, chamada *Warthog Robotics*. Este ano, na Competição Brasileira da Robótica, em São João Del Rei/MG, a equipe já obteu os seguintes títulos:

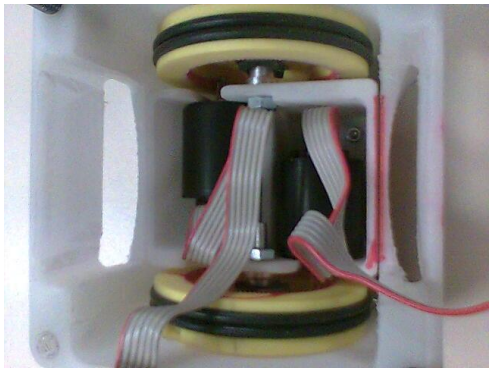
(a) Robô sem a etiqueta que define time



(b) Robô sem a tampa protetora

(c) Robô sem a placa eletrônica

(d) Robô sem a bateria



(e) Robô sem o suporte da bateria

Figura 5.3: Fotos das partes internas do robô do time USPDroids/2009

- TERCEIRO LUGAR na categoria *Robocup Small Size League*;
- VICE-CAMPEÃO na categoria *RoboCup 2D Simulated Soccer League*
- CAMPEÃO na categoria *RoboCup 3D Simulated Soccer League*
- CAMPEÃO na categoria *IEEE Very Small Size*

5.4 Simulador USPDroidsSS

De acordo com (Doi et al., 2010), 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 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.

5.5 Considerações Finais

Neste capítulo foi descrito a plataforma robótica de onde dados serão extraídos para a realização de testes.

Como dito anteriormente, o futebol de robôs engloba todas as dificuldades da robótica móvel autônoma e, por ser uma tarefa padronizada, pode ser vista como um benchmark, facilitando posteriores comparações de métodos.

CAPÍTULO

6

Experimentos

Neste capítulo serão apresentadas as avaliações dos métodos apresentados anteriormente. Para os algoritmos sequenciais foi utilizado um computador com as configurações apresentadas na Tabela 6.1 e para os algoritmos paralelos, foi utilizada a GPU descrita na Tabela 6.2,

A seguir, serão apresentados a média do número de iterações e tempo (em milisegundos), bem como intervalos de confiança com nível de significância $\alpha = 0.05$ para cada técnica de campos potenciais baseada em problemas de valor de contorno descritas no Capítulo 3. Para cada técnica, também foi calculado o speedup em relação ao Gauss-Seidel sequencial, que é método mais comum e é considerado eficiente na literatura encontrada.

Não foram executados testes do método de Jacobi e Weighted-Jacobi sequenciais, pois estes são mais lentos que o Gauss-Seidel ou o SOR, em suas versões sequenciais Franco (2008); de Oliveira Fortuna (2000).

Exceto quando estiver indicado o contrário, os testes utilizaram uma grade de 85×65 células com tolerância de erro menor do que 10^{-6}.

Duas bases de dados foram utilizadas. A primeira delas, composta de dados aleatórios, serviu para obter uma estimativa de w_{otimo} do método WJR (Weighted Jacobi-Richardson) e SOR (Successive Over-Relaxation). Ao utilizar uma base composta de dados aleatórios e uniformemente distribuídos, acredita-se diminuir possíveis viés, explicados adiante.

A segunda base é composta de cinco minutos de estados sucessivos de uma partida de futebol de robôs realizada no simulador USPDroidsSS entre a estratégia utilizada na CBR/2010 e uma estratégia aleatória. Muitas das vezes a diferença entre um estado num tempo t e o estado $t+1$ é tão pequena, que o módulo de captura de dados do sistema da estratégia não é capaz de

notá-las.

Também, devido ao uso de uma estratégia em particular, nem todos os possíveis valores de ϵ e v são explorados, dado que a modificação nestas variáveis é o que acaba por constituir grande parte de um sistema inteligente que utilize estas técnicas.

Ainda, pode ser ressaltado a diferença na relaxação da primeira iteração com as demais (Silva et al., 2010b). Como as modificações do ambiente entre ciclos sucessivos é pequena, e o sistema linear resultante das EDPs que regem os métodos aqui apresentados, a solução de ambos ciclos é bem similar, senão igual. Então, muitas vezes o relaxador irá convergir em apenas uma única iteração. Em (Silva et al., 2010b), foi mostrado que mais de 80% das execuções do CPLO convergem em apenas uma iteração.

Tabela 6.1: Especificações do Computador

CPU	i7-860 (Intel)
Instrução	x86-64
Quantidade de Núcleos	4 (8 em HyperTheading)
Frequência dos Núcleos	2.8GHz
Tipo de Memória RAM	DDR 3
Frequência da Memória RAM	1333MHz
Tamanho da Memória RAM	6GB
Máxima Largura de Banda	21GB/s
Tamanho do Cache L1	256KB
Tamanho do Cache L2	1MB
Tamanho do Cache L3	8MB
Sistema Operacional	Ubuntu 10.04 (Lucid Lynx)
Kernel	2.6.32

Tabela 6.2: Especificações da GPU

GPU	GTX480 (NVidia)
Instrução	CUDA
Quantidade de SPs	480
Frequência dos SPs	700MHz
Quantidade de SMPs	15
Quantidade de SPs por SMPs	32
Tipo de Memória RAM	GDDR5
Frequência da Memória RAM	1848MHz
Tamanho da Memória RAM	1.5GB
Máxima Largura de Banda	177.4GB/s

6.1 Estimação de w_{otimo} dos métodos SOR e WJR

Como dito anteriormente, antes de se executar os experimentos, buscou-se por uma estimativa do w ótimo dos métodos. Nas Figuras 6.1 - 6.3 são apresentados os valores encontrados para o CPH, CPO e CPLO, respectivamente.

Para o CPH, na Figura 6.1, foram obtidos os seguintes valores:

- **SOR Sequencial:** 1.9;
- **SOR CUDA:** 1.9;
- **WJR CUDA:** 1.8;

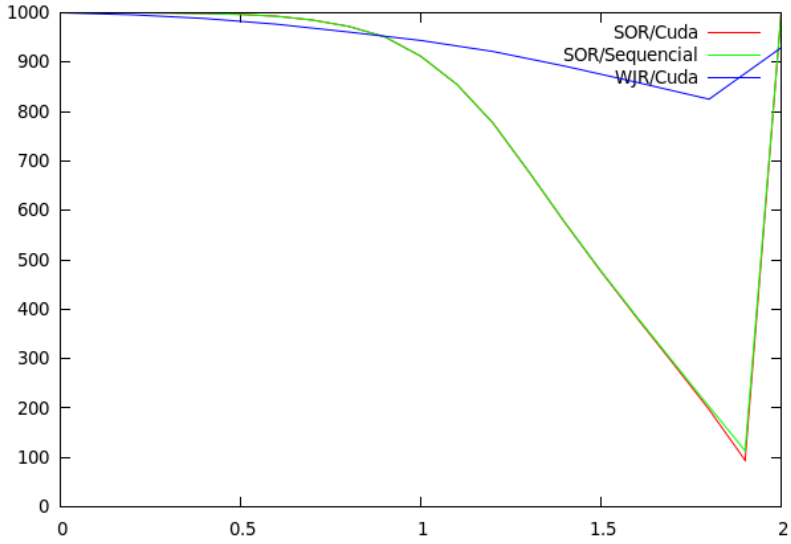


Figura 6.1: CPH - w_{otimo}

Para o CPO, na Figura 6.2, foram obtidos os seguintes valores:

- Diferenças Centradas:
  - **SOR Sequencial:** 1.9;
  - **SOR CUDA:** 1.9;
  - **WJR CUDA:** 1.8;
- Diferenças Upwind:
  - **SOR Sequencial:** 1.5;
  - **SOR CUDA:** 1.5;
  - **WJR CUDA:** 1.8;

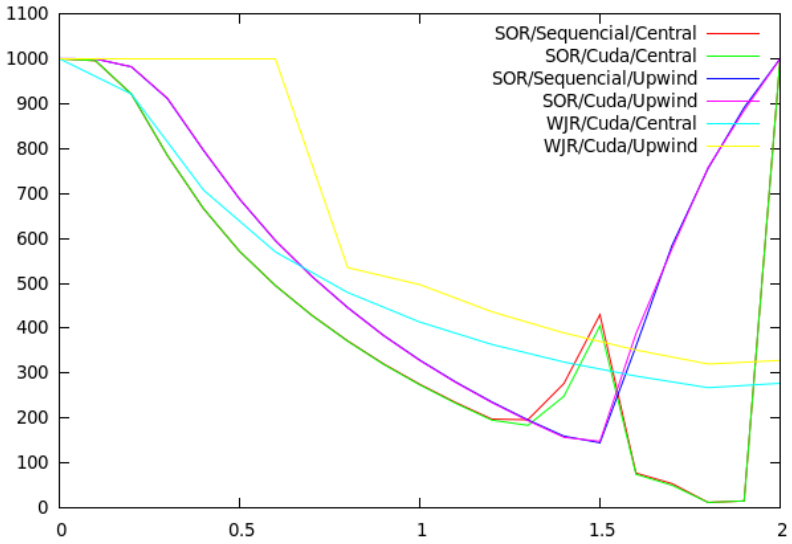


Figura 6.2: CPO - w_{otimo}

E, para o CPLO, na Figura 6.3, foram obtidos os seguintes valores:

- Diferenças Centradas:
  - **SOR Sequencial:** 1.8;
  - **SOR CUDA:** 1.8;
  - **WJR CUDA:** 1.8;
- Diferenças Upwind:
  - **SOR Sequencial:** 1.7;
  - **SOR CUDA:** 1.9;
  - **WJR CUDA:** 1.8;

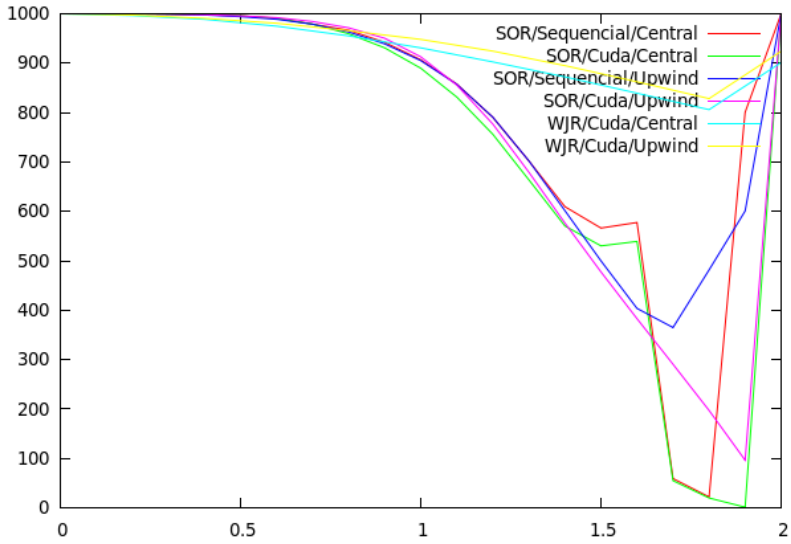


Figura 6.3: CPLO - w_{otimo}

6.2 Avaliação do Epsilon

Na Figura 6.4, é mostrado a convergência do CPO com Gauss-Seidel, paralelo em CUDA e sequencial, com diferenças upwind e centradas, para vários valores de ϵ.

Pode ser notado que todos os métodos convergem $\epsilon < 2$, o que ilustra as provas de convergência feitas durante este trabalho. Porém, não há garantias de convergência, tampouco de divergência, para quando $\epsilon \geq 2$ usando diferenças centradas. Neste experimento, para a particular configuração do ambiente, apenas para $\epsilon > 3.4$ que passou a haver divergência no caso das diferenças centradas, enquanto que ao utilizar diferenças upwind convergiu normalmente.

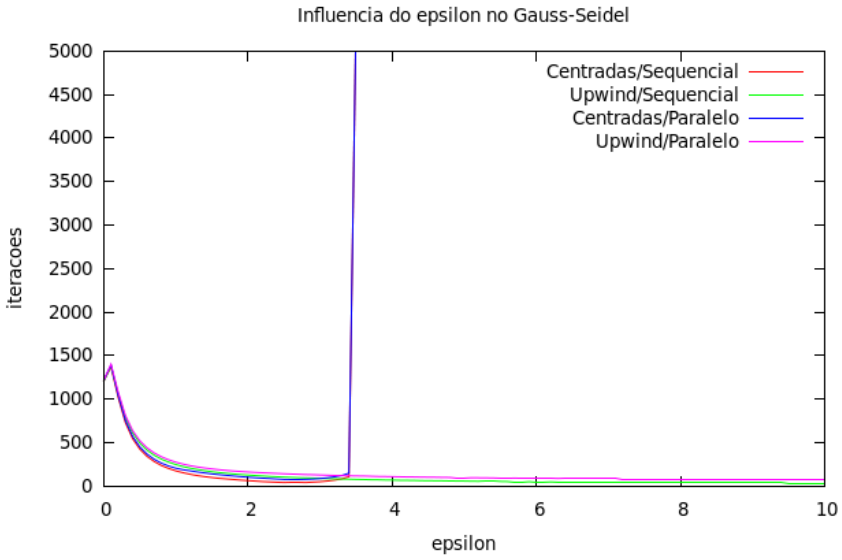


Figura 6.4: CPO / Gauss Seidel - ϵ

6.3 Avaliação do CPH

Na Tabela 6.3, são apresentados os tempos de execução em milisegundos dos métodos para solução do BVP associado ao CPH dos experimentos realizados. Tomando o tempo do Gauss-Seidel sequencial como base ($5.38ms$), foram calculados os speedups (ganhos ou perdas de desempenho), como na Tabela 6.4.

Tabela 6.3: CPH - Tempo de Execução

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	0.60	$[0.58, 0.63]$
WJR	-	-	0.39	$[0.38, 0.41]$
GS	05.38	$[05.15, 05.62]$	0.62	$[0.60, 0.64]$
SOR	05.20	$[05.05, 05.36]$	0.49	$[0.49, 0.51]$

Tabela 6.4: CPH - Speedup

	Sequencial/CPU	Paralelo/GPU
JR	-	8.97
WJR	-	13.80
GS	1	8.68
SOR	1.04	10.98

Na Tabela 6.5, são apresentadas as quantidades de iterações dos mesmos experimentos.

Tabela 6.5: CPH - Quantidade de Iterações

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	32.57	$[30.84, 34.30]$
WJR	-	-	24.16	$[23.07, 25.26]$
GS	19.40	$[18.52, 20.21]$	19.44	$[18.59, 20.29]$
SOR	17.65	$[17.12, 18.17]$	15.50	$[15.04, 15.96]$

Para o CPH, o método mais rápido, se for considerado o tempo de execução, é o WJR em CUDA, (com média de $0.39ms$ por execução) enquanto que for considerado a quantidade de iterações, o método mais rápido é o SOR em CUDA (com média de 15.5 por execução).

Enquanto o WJR/CUDA fez, em média, 62 iterações por milisegundo, o SOR/CUDA fez 32.

6.4 Avaliação do CPO

Nas Tabelas 6.6 e 6.9, são apresentados os tempos de execução em milisegundos dos métodos para solução do BVP associado ao CPO, utilizando diferenças centradas e upwind, respectivamente, dos experimentos realizados. Tomando o tempo do Gauss-Seidel sequencial, em diferenças centradas, como base ($8.04ms$), foram calculados os speedups, como na Tabelas 6.7 e 6.10. Nas Tabelas 6.8 e 6.11, são apresentadas as quantidades de iterações dos mesmos experimentos.

Pode ser observado que, para as diferenças centradas, como no CPH, o WJR/CUDA é o mais rápido se for considerado tempo de execução, ou quantidade de iterações por milisegundo, enquanto que se for considerado apenas a quantidade de iterações, o SOR/CUDA é o mais rápido. Enquanto que para as diferenças upwind, o método mais rápido é o SOR/CUDA, seja considerando quantidade de iterações, quantidade de iterações por milisegundos, ou milesegundos.

Tabela 6.6: CPO - Tempo de Execução - Diferenças centradas

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	0.55	$[0.47, 0.63]$
WJR	-	-	0.48	$[0.40, 0.55]$
GS	08.04	$[07.70, 08.38]$	0.69	$[0.59, 0.79]$
SOR	08.20	$[07.29, 09.10]$	0.55	$[0.50, 0.60]$

Tabela 6.7: CPO - Speedup - Diferenças centradas

	Sequencial/CPU	Paralelo/GPU
JR	-	14.62
WJR	-	16.75
GS	1	11.65
SOR	0.98	14.62

Tabela 6.8: CPO - Quantidade de Iterações - Diferenças centradas

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	34.35	$[28.70, 39.99]$
WJR	-	-	29.51	$[24.33, 34.70]$
GS	17.17	$[16.46, 17.90]$	10.96	$[09.84, 12.88]$
SOR	16.41	$[14.61, 18.22]$	13.62	$[11.87, 15.37]$

Tabela 6.9: CPO - Tempo de Execução - Diferenças upwind

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	0.57	$[0.50, 0.67]$
WJR	-	-	0.52	$[0.43, 0.60]$
GS	11.01	$[09.27, 12.75]$	0.75	$[0.64, 0.86]$
SOR	07.91	$[06.87, 08.94]$	0.48	$[0.43, 0.54]$

Tabela 6.10: CPO - Speedup - Diferenças upwind

	Sequencial/CPU	Paralelo/GPU
JR	-	14.11
WJR	-	15.46
GS	0.73	10.72
SOR	1.02	16.75

Tabela 6.11: CPO - Quantidade de Iterações - Diferenças upwind

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	34.96	$[29.15, 40.76]$
WJR	-	-	29.93	$[24.60, 35.26]$
GS	23.54	$[19.84, 27.24]$	22.08	$[18.62, 25.54]$
SOR	16.34	$[14.22, 18.46]$	13.62	$[11.87, 15.37]$

6.5 Avaliação do CPLO

Nas Tabelas 6.12 e 6.15, são apresentados os tempos de execução em milisegundos dos métodos para solução do BVP associado ao CPO, utilizando diferenças centradas e upwind, respectivamente, dos experimentos realizados. Tomando o tempo do Gauss-Seidel sequencial, em diferenças centradas, como base ($29.18ms$), foram calculados os speedups, como na Tabelas 6.13 e 6.16. Nas Tabelas 6.14 e 6.17, são apresentadas as quantidades de iterações dos mesmos experimentos.

Pode ser observado que, diferentemente do CPO, o WJR/CUDA, para diferenças upwind é o mais rápido se for considerado tempo de execução, ou quantidade de iterações por milisegundo, enquanto que o SOR/CUDA é mais rápido se for considerado apenas a quantidade de iterações. Já para as diferenças centradas, o SOR/CUDA é o mais rápido nos três aspectos levantados.

Tabela 6.12: CPLO - Tempo de Execução - Diferenças Centradas

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	0.88	[0.73, 1.02]
WJR	-	-	0.75	[0.63, 0.86]
GS	29.18	[22.89, 35.48]	1.16	[0.99, 1.34]
SOR	21.30	[18.88, 23.72]	0.70	[0.64, 0.77]

Tabela 6.13: CPLO - Speedup - Diferenças Centradas

	Sequencial/CPU	Paralelo/GPU
JR	-	33.16
WJR	-	38.91
GS	1	25.16
SOR	1.37	41.69

Tabela 6.14: CPLO - Quantidade de Iterações - Diferenças Centradas

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	49.13	[40.50, 57.76]
WJR	-	-	40.78	[33.85, 47.70]
GS	20.43	[22.48, 34.38]	32.46	[27.31, 37.60]
SOR	20.52	[18.20, 22.83]	18.32	[16.82, 20.84,]

Tabela 6.15: CPLO - Tempo de Execução - Diferenças upwind

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	0.83	[0.67, 0.98]
WJR	-	-	0.68	[0.57, 0.80]
GS	28.41	[22.41, 34.39]	1.07	[0.90, 1.25]
SOR	16.92	[14.50, 19.35]	0.85	[0.76, 0.93]

Tabela 6.16: CPLO - Speedup - Diferenças Upwind

	Sequencial/CPU	Paralelo/GPU
JR	-	35.17
WJR	-	42.91
GS	1.03	27.27
SOR	1.73	34.33

6.6 Considerações Finais

Conforme resultados apresentados nas Seções 2- 4, pode ser visto que o uso de GPU diminui sensivelmente (alguns códigos ficaram $40\times$ mais rápidos) o tempo de execução dos métodos, ainda que haja a necessidade de tranferir a

Tabela 6.17: CPLO - Quantidade de Iterações - Diferenças upwind

	Sequencial/CPU		Paralelo/GPU	
	Média	I. C.	Média	I. C.
JR	-	-	49.13	[40.50, 47.76]
WJR	-	-	40.78	[33.85, 47.70]
GS	28.44	[22.54, 34.37]	27.42	[22.72, 37.12]
SOR	17.16	[14.72, 19.61]	21.30	[19.07, 23.52]

grade do método numérico da CPU para a GPU no início do algoritmo e da GPU para a CPU no final do mesmo, além do cálculo de erro que não é eficiente, embora seja eficaz (isto é, resolve a tarefa, mas não da melhor forma possível; em futuras versões do código, pretende-se resolver este problema).

A quantidade de iterações permanece praticamente a mesma, quando se compara a versão em CUDA com a versão sequencial, em grande parte devido a abordagem de sincronia na hora de obter o erro de aproximação de uma determinada iteração. Isto indica que a execução dos métodos é mais rápida pela paralelização massiva fornecida pela arquitetura da GPU e da arquitetura CUDA. Os maiores Speedups foram obtidos no CPLO (que tem uso mais intenso dos recursos computacionais) e os menores no CPH (que tem uso menos intenso dos recursos computacionais), o que confirma as orientações fornecidas pela Nvidia no uso do CUDA.

Não há um método que seja melhor do que todos os outros, em todas as execuções, mas pode ser dito que tanto o SOR/CUDA quanto o WJR/CUDA são mais rápidos e eficientes do que os outros, mas indistinguíveis entre si.

Apenas considerado o tempo de execução, em uma implementação sequencial, utilizando diferenças centradas e Gauss-Seidel, para até 3 execuções distintas, i.e. para 3 robôs diferentes, o CPO é mais rápido que o CPLO, pois isto totalizaria $24.12ms$, que é menos do que os $29.18ms$ de uma execução do CPLO. Mas, conforme já foi tratado, o uso de um CPO para cada agente tem mais vantagens do ponto de visto de eficiência da técnica de planejamento de caminhos (não necessariamente da eficiência do iterador), pois permite controlar a trajetória do agente de forma mais precisa, além de evitar indesejáveis competições pela meta.

Com um tempo de execução de $1ms$, em teoria, equivale a dizer que o algoritmo pode rodar a mais de $1MHz$, isto é, mais de 1000 ciclos por segundo, o que, para fins práticos, é mais do que suficiente para ser considerado uma execução em tempo real. Alguns algoritmos aqui apresentados, como o SOR/CUDA para CPO com diferenças upwind, teve média de execuções de $0.48ms$, isto é, média de 2083 ciclos por segundo.

CAPÍTULO

7

Conclusões

Este trabalho foi motivado pelo aprimoramento das técnicas de métodos de planejamentos de caminhos baseadas em problemas de valor de contorno, visando sua utilização em ambientes dinâmicos, como o do futebol de robôs. Para isto, foram procuradas formas de acelerar a execução de tais métodos e, a escolha mais promissora mostrou-se correta, o método de iteração para solucionar as EDPs dos BVPs que formam métodos.

Ao fazer a revisão bibliográfica, notou-se que, em outras áreas que utilizam soluções numéricas de EDPs, uma nova abordagem para obter estas soluções estava sendo utilizada. O uso de GPUs para processar grandes volumes de dados mostrou-se eficaz em várias dessas áreas. Partindo disto, decidiu-se por implementar os métodos de planejamento de caminhos baseados em EDPs em GPU, além da sua versão sequencial.

Porém, o uso de GPUs além de não ser trivial, costuma forçar o programador a más práticas de programação, como o uso de estrutura de arrays em vez de array de estruturas. Com o objetivo de disponibilizar este trabalho para que outros pesquisadores usufruam dele sem que necessitem passar pelas mesmas etapas de desenvolvimento, optou-se pelo desenvolvimento de uma biblioteca de funções em alto-nível, em C, que em breve será disponibilizada sob licensa GPL.

Como mostrado no Capítulo 6, a utilização da GPU aumenta consideravelmente a velocidade dos métodos de planejamento de caminhos.

Também, conforme mostrado no Capítulo 3, o uso de discretizações upwind resolve o problema do ϵ do CPO e do CPLO ter um limitante. Além disto, o uso de diferenças upwind permite a elaboração de novos comportamentos do sistema inteligente que utilizar estes métodos.

As principais contribuições deste trabalho são:

- Prova de existência de solução numérica incondicionalmente convergente para o CPO e o CPLO, ao utilizar diferenças upwind;

- O uso de GPUs em robótica é viável, e permite que alguns sistemas críticos, como o de planejamento de caminhos, funcione tão rápido quanto necessário. Atualmente, poucos algoritmos irão funcionar a mais de $1000Hz$, isto é, o sistema de planejamento de caminhos deixa de ser um gargalo para qualquer sistema, da forma como foi apresentada neste trabalho;

- Em arquiteturas paralelas do tipo manycore, como as GPUs, os métodos WJR e o SOR são os mais promissores. No presente trabalho, foram os métodos mais rápidos, em todos os testes;

- O CPO pode ser utilizado para criar estratégias de controle de múltiplos robôs mais complexas do que o CPLO, dado que permite estabelecer múltiplas metas e permitir tratar melhor a cooperação entre robôs;

- A escolha de um bom chute inicial para os métodos de relaxação afeta sensivelmente o desempenho destes.

Pretende-se, numa próxima etapa, explorar outras possibilidades da GPU, como o uso de coprocessadores matemáticos dentro dos SMPs, acesso coalescido à memória global e cacheamento em memória compartilhada.

A utilização de C for CUDA fica restrita aos chipsets desenvolvidos pela NVidia. Com isto, pretende-se utilizar outras linguagens de programação, como OpenHMPP ou OpenCL, que permitem utilizar não só os produtos da NVidia, como o de outras fabricantes, como a AMD.

Quanto a teoria, embora tenha sido provado que se existir soluções analíticas para as EDPs do CPO e CPLO, existirão soluções numéricas, não encontrou-se na literatura provas da existência de soluções analíticas para as tais. Isto dá a possibilidade de um maior aprofundamento na teoria por trás dos métodos de campos potenciais baseados em problemas de valor de contorno.

Como trabalhos futuros, pretende-se:

- *Investigar o processo de surgimento dos platôs*

  Os platôs são regiões em que o gradiente é nulo. Isto resulta no sistema de planejamento de caminhos não gerar uma rota, ainda que ela exista. Em algumas execuções dos algoritmos, houve o surgimento de platôs, sem padrão identificável, até então.

- *Utilizar relaxadores alternativos aos relaxadores clássicos*

  Nos algoritmos Gauss-Seidel e SOR paralelos, existe a possibilidade de definir uma *coloração*, que evita colisões de acesso a memória compartilhada e pode até acelerar o desempenho do algoritmo. Neste trabalho apenas a coloração Reb-Black foi adotada, por ser amplamente aceita. Outras colorações poderiam ser avaliadas.

  Utilizar os métodos de sub-espaços de Krylov, que são amplamente utilizados na área de mecânica dos fluídos, para resolver BVPs similares aos das técnicas de campos potenciais apresentadas neste trabalho.

- *Fundamentar teóricamente os métodos utilizados*

  Embora tenha sido provado a existência de soluções numéricas, condicionais quando usadas diferenças centradas e incondicionais quando usadas diferenças upwind, uma das etapas da prova é assumir a existência de uma solução analítica (ainda que não trivial de ser encontrada) dos PVCs. E, é possível que alguns casos hoje tratados como tendo solução, não a possuam. (Vide exemplo de Zaremba, para o CPH (Figueiredo, 2005)).

- *Planejar Trajetórias*

  Utilizar informação de como meta, objetos e agentes estão se movimentando no ambiente, bem como características do robô a ser controlado, para gerar um caminho até a meta móvel, que não cruze a rota dos outros obtáculos.

- *Investigar o uso de outras condições de contorno (Newmann, Robin...) e EDPs (Parabólicas e Hiperbólicas) para gerar novas técnicas de planejamento de caminhos.*

- *Melhor aproveitamento dos recursos da GPU*

  Utilizar o cachê L1/L2, os co-processadores matemáticos e etc. Reduzir a transferência de memória pela PCIExpress, bem como evitar acessos conflitantes e não-coesos na memória interna da GPU.

- *Portabilidade*

  Utilizar uma linguagem de programação que permita o uso de diversos tipos de GPUs (OpenCL).

  Disponibilizar os códigos desenvolvidos, bem como material de referência, como open-sources na internet

APÊNDICE

A

Definições e Teoremas

Neste Apêndice, são apresentados alguns conceitos matemáticos necessários para a prova da convergência (condicional no caso das diferenças centradas e incondicional no caso das diferenças upwind) dos métodos baseados em PVC.

Definição 1. *É dito que um grafo é* fortemente conectado se e somente se para qualquer par de nós P_i, P_j existir um caminho

$$\overrightarrow{P_iP_{i_1}}, \overrightarrow{P_{i_1}P_{i_2}}, \ldots, \overrightarrow{P_{i_m}P_j}$$

conectando P_i e P_j.(Cormen et al., 2001; Ortega, 1972)

Definição 2. *Uma matriz $A \in \mathcal{L}(\Re^n)$ (onde $\mathcal{L}(\Re^n)$ é o espaço dos operadores lineares de $\Re^n$ a $\Re^n$) é diagonalmente dominante se*

$$|a_{ii}| \geq \sum_{\substack{j=0 \\ j\neq i}}^{n} |a_{ij}|, \qquad i = 1, \ldots, n \tag{A.1}$$

e estritamente diagonalmente dominante *se as inequações são estritas na Equação A.1 para todo i. Uma matriz é* diagonalmente dominante e irredutível *se é irredutível, diagonalmente dominante e, para pelo menos i na Equação A.1 a inequação é estrita. (Ortega, 1972)*

Definição 3. *Com uma matriz $A_{n\times n}$, pode ser associado um grafo como a seguir. Sejam n pontos distintos $P_1, \ldots, P_n$, que serão chamandos vértice e, para cada elemento não nulo a_{ij} de A constrói-se um aresta de P_i a P_j. Como resultado, há um digrafo (grafo dirigido) associado com a matriz A.(Ortega, 1972)*

Teorema 1. *Uma matriz $A \in \mathcal{L}(\Re^n)$ é irredutível se e somente se o grafo associado a ela é fortemente conectado.(Ortega, 1972)*

Teorema 2. *Assumindo que $b \in \Re^n$ é arbitrário e que $A \in \mathcal{L}(\Re^n)$ ou é estritamente diagonalmente dominante ou é diagonalmente dominante e irredutível, então, o método de Jacobi-Richardson e o método de Gauss-Seidel convergem para $A^{-1}b$ para qualquer x_0 dado.(Ortega, 1972)*

Bibliografia

Ames, W. F. (1969). *Numerical Methods for Partial Differential Equations*. Thomas Nelson and Sons Ltd.

Arkin, R. C. (1989). Motor schema-based mobile robot navigator. Em *The International Journal of Robotics Research*, pages 92–112.

Augusto Silva Pereira, G., Borges Torres, F., 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.

Barney, B. (2011). Introduction to parallel computing. Disponível em https://computing.llnl.gov/tutorials/parallel_comp/.

Borenstein, J. e Koren, Y. (1989). Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187.

Borenstein, J. e Koren, Y. (1991). The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288.

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

Burden, R. L., Faires, J. D., e Reynolds, A. C. (1978). *Numerical Analysis*. Prindle, Weber and Schimidt.

Connolly, C. I., Burns, J. B., e Weiss, R. (1990). Path planning using laplaces equation. Em *IEEE International Conference on Robotics and Automation*.

Connolly, C. I. e Grupen, R. A. (1993). On the application of harmonic functions to robotics. *Journal of Robotic Systems*, 10:931–946.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2001). *Introduction to Algorithms*. McGraw-Hill.

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

Courant, R. e Hilbert, D. (1962). *Methods of mathematical physics*, volume 2. New York: John Wiley and Sons.

da Silva, M. O., Romero, R. A. F., e de Paula Caurin, G. A. (2008). Analysis of the convergence of locally oriented potential fields. Em Reis, L. P., Correia, L., Lau, N., e Bianchi, R., editors, *IROBOT 2008 - 3rd International Workshop on Intelligent Robotics*, Lisbon University Institute - ISCTE, Lisbon, Portugal. ISBN 972886208-3.

de Oliveira Fortuna, A. (2000). *Técnicas Computacionais para Dinâmica dos Fluídos*. EDUSP.

Doi, D. T., da Silva, M. O., Montanari, R., da Silva, W. C., e Romero, R. A. F. (2010). Uspdroidsss: Ambiente de simulação para futebol de robôs da categoria ieee very small size - versão 2010.a. Technical Report 357, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, São Carlos / SP. ISSN 0103-2569.

Faria, G. (2006). *Uma Arquitetura de controle inteligente para múltiplos robôs*. PhD thesis, ICMC-USP São Carlos.

Faria, G. e Romero, R. A. F. (2000). Incorporating fuzzy logic to reinforcement learning. Em *Proceedings of the 9th IEEE International Conference on Fuzzy Systems*, volume 1, pages 847–851.

Figueiredo, D. G. (2005). *Análise de Fourier e Equações Diferenciais Parciais*. Projeto Euclides / IMPA.

FIRA (2008). *www.fira.net*. Federation of International Robot-Soccer Association. visitado em Fevereiro/2008.

Flynn, M. (1972). Some Computer Organizations and Their Effectiveness. *IEEE Trans. Comput.*, C-21:948+.

Franco, N. B. (2008). *Cálculo numérico*. Pearson.

Gilbarg, D. e Trudinger, N. S. (1983). *Elliptic Partial Differential Equations of Second Order*. Springer-Verlag, 2nd edition.

Gomes, M. R. S. e Campos, M. F. M. (2001). Desvio de obstáculos para microrobôs móveis utilizando campo potencial. Em *Anais do V Simpósio Brasileiro de Automação Inteligente (SBAI)*.

Guidorizzi, H. L. (2001). *Um Curso de Cálculo - Volume 1*. LTC, 5th edition.

Khatib, O. (1980). *Command Dynamic dans l'Espace Opérationnel des Robots Manipulateurs en Présence d'Osbtacles*. PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse. em Francês.

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.

Krause, E. F. (1986). *Taxicab Geometry : An Adventure in Non-Euclidean Geometry*. Dover.

Krogh, B. H. (1984). A generalizes potential field approacho to obstacle avoidance. Em *International Robotics Research Conference Bethlehem*.

Krogh, B. H. e Thorpe, C. E. (1986). Integrated paht planning and dynamic steering control. Em *IEEE International Conference on Robotics and Automation*, pages 1664–1669.

Larsson, S. e Thomée, V. (2005). *Partial Differential Equations with Numerical Methods*. Springer.

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.

Masoud, A. A. (2006). A discrete harmonic potential field for optimun point-to-point routing on a weighted graph. Em *IEEE International Conference on Intelligent Robots and Systems*, pages 1779–1784.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8).

Newman, W. S. e Hogan, N. (1987). High speed robot control and obstacle avoidance. Em *Proceedings of the 1987 IEEE International*, pages 14–24, Raleigh, North Carolina.

Nvidia (2011). Nvidia cuda c programming guide. 5/6/2011.

Ortega, J. M. (1972). *Numerical Analysis - A Second Course*. Academic Press.

Ostrovsky, I. (2011). How gpu came to be used for general computation. http://igoro.com/archive/how-gpu-came-to-be-used-for-general-computation/.

Pacheco, R. N. e Costa, A. H. R. (2002). Navegação de robôs móveis utilizando o método de campos potenciais. Em Sakude, M. T. S. e de A. Castro Cesar, C., editors, *Workshop de Computação WORKCOMP'2002*, pages 125–130. SBC.

Post, D. (2011). The future of computing performance. *Computing in Science Engineering*, 13(4):4 –5.

Prestes, E. (2003). *Navegação Exploratória Baseada em Problemas de Valores de Contorno*. PhD thesis, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS.

Shi, C., Zhang, M., e Peng, J. (2007). Harmonic potential field method for autonomous ship navigation. Em *7th International Conference on ITS Telecommunications*.

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.

Silva, M. O., Romero, R. A. F., e Oliveira, S. P. (2010a). Improving the stability of algorithms for path planning based on boundary value problems. Resumo Extendido no XXXIII CNMAC - Congresso Nacional de Matemática Aplicada e Computacional.

Silva, M. O., Silva, W. C., e Romero, R. A. F. (2010b). Performance analysis of path planning techniques based on potential fields. Em *2010 Latin American Robotics Symposium and Intelligente Robotics Meeting*. IEEE.

Smith, G. D. (1992). *Numerical Solution of partial differential equations: finite difference methods*. Oxford University.

Strauss, W. A. (1992). *Partial Differential Equations: an introduction*. John Wiley and Sons.

Sutter, H. (2005). The free lunch is over- a fundamental turn toward concurrency in software. Disponível em http://www.gotw.ca/publications/concurrency-ddj.htm.

Tenenbaum, A. M., Yedidyah, L., e Augenstein, M. J. (1995). *Estruturas de dados usando C*. Makron Books.

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.