Universidade Federal de Juiz de Fora

Faculdade de Engenharia

Programa de Graduação em Engenharia Elétrica com habilitação nas áreas de Robótica e Automação Industrial

Pedro Antônio Alcântara Pires

Navegação de Robôs Móveis através de Campos Potenciais baseados em Problemas de Valor de Contorno

Juiz de Fora

2016

Pedro Antônio Alcântara Pires

Navegação de Robôs Móveis através de Campos Potenciais baseados em Problemas de Valor de Contorno

Trabalho de Conclusão de Curso apresentado ao Programa de Graduação em Engenharia Elétrica com habilitação nas áreas de Robótica e Automação Industrial da Universidade Federal de Juiz de Fora, na área de concentração em Navegação de Robôs Móveis , como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Ana Sophia Cavalcanti Alves Vilas Boas

Coorientador: Exuperry Barros Costa

Juiz de Fora

2016

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Alcântara Pires, Pedro Antônio.
Navegação de Robôs Móveis através de Campos Potenciais baseados em Problemas de Valor de Contorno / Pedro Antônio Alcântara Pires. – 2016.
76 f. : il.

Orientador: Ana Sophia Cavalcanti Alves Vilas Boas
Coorientador: Exuperry Barros Costa
Trabalho de Conclusão de Curso (Graduação) – Universidade Federal de Juiz de Fora, Faculdade de Engenharia. Programa de Graduação em Engenharia Elétrica com habilitação nas áreas de Robótica e Automação Industrial, 2016.

1. Campos Potenciais. 2. Navegação. 3. Robótica Móvel. I. Boas, A. S. C. A. V., orient. II. D.Sc. III. Costa, E. B., coorient. IV. M.sc.

Pedro Antônio Alcântara Pires

Navegação de Robôs Móveis através de Campos Potenciais baseados em Problemas de Valor de Contorno

Trabalho de Conclusão de Curso apresentado ao Programa de Graduação em Engenharia Elétrica com habilitação nas áreas de Robótica e Automação Industrial da Universidade Federal de Juiz de Fora, na área de concentração em Navegação de Robôs Móveis , como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Aprovado em: 22/12/2016

BANCA EXAMINADORA

Professor Dr. Eng. Ana Sophia Cavalcanti Alves Vilas Boas - Orientador
Universidade Federal de Juiz de Fora

Professor Ms. Eng. Exuperry Barros Costa - Coorientador
Universidade Federal de Juiz de Fora

Professor Ms. Eng. Guilherme Márcio Soares
Universidade Federal de Juiz de Fora

AGRADECIMENTOS

Esta talvez seja uma das partes mais difíceis deste trabalho, são tantas as pessoas que fazem parte do nosso dia a dia, do nosso crescimento acadêmico, profissional e principalmente pessoal.

Começo então por agradecer aos meus orientadores, Ana Sophia e Exuperry, assim como a equipe ***RinoBot Team***, por confiarem na minha capacidade, por acreditarem no meu trabalho e terem me dado a melhor experiência como universitário na LARC 2016 (Competição Latino-Americana de Robótica).

Ao amigo de longa data Lucas Rossi, pelos momentos de compreensão e descontração quando os desafios deste trabalho saíam do âmbito acadêmico e passavam para o pessoal.

Aos amigos que fiz durante a graduação na UFJF, em especial, ao Gustavo Hofstatter e Frederick Tavares pelas conquistas e trabalhos realizados, inclusive na Rinobot.

Aos meus irmãos João, Vinícius e Lucas por sempre terem demonstrado confiança, respeito e compreensão nas minhas escolhas, muito da minha personalidade foi moldada por nossa grande amizade.

Ao meu pai Flauridinélio Elias Pires, por me dar a chance de crescer, estudar e trabalhar por mérito próprio, todos os ensinamentos ao longo desses anos, me transformaram na pessoa determinada que sou hoje. Seu apoio emocional e financeiro foram essenciais em todos os momentos do meu ciclo acadêmico.

A minha mãe Adriana Marques de Alcântara, simplesmente por ser a mulher que é, nunca conheci ninguém igual e nem acho que vá. Ao abdicar de seus sonhos, me deu a chance de ter os meus e isto não tem preço. Sempre serei grato a ela por tudo, com toda a certeza minha mãe foi a maior motivação para a conclusão deste curso.

“Há uma sensação de inquietude enquanto
não encontramos soluções, e isso nunca passa.
Achamos que o impossível, sempre é possível.”
(Sebastian Thrun)

RESUMO

Este trabalho, aborda o estudo de sistemas de navegação para robôs móveis utilizando Campos Potenciais baseados em Problemas de Valor de Contorno (PVC), se concentrando em três métodos: Campos Potenciais Harmônicos (CPH), Campos Potenciais Orientados (CPO) e Campos Potenciais Localmente Orientados (CPLO). Tais métodos, dependem de uma etapa de solução de sistemas lineares, na qual se utilizam métodos de relaxação iterativos, decorrentes da aplicação do método de Diferenças Finitas como solucionador das Equações Diferenciais Parciais que geram os PVC. As técnicas de navegação foram implementadas visando suas versões sequenciais, de forma a comprovar suas eficácias mesmo que em um ambiente virtual. Os métodos serão implementados futuramente na equipe de futebol de robôs autônomos da UFJF, ***Rinobot Team***.

Palavras-chave: Campos Potenciais. Navegação. Robótica Móvel.

ABSTRACT

This work is about the study of mobile robot path planning systems using Potential Fields based on Boundary Value Problems (BVP), focusing on three methods: Potential Harmonic Fields (PHF), Oriented Potential Fields (OPF) and Locally Oriented Potential Fields (LOPF). These methods depend on a solution step of linear systems, in which iterative relaxation methods are used, resulting from the application of the Finite Differences method as a solver of the Partial Differential Equations that generate the BVP. The path planning techniques were implemented aiming their sequential versions, in order to prove their efficacies even in a virtual environment. The methods will be implemented in the future in the autonomous robot soccer team of the UFJF, ***Rinobot Team***.

Key-words: Mobile Robotics. Path Planning. Potential Fields.

LISTA DE ILUSTRAÇÕES

Figura 1 – Força de Repulsão na vizinhança de um obstáculo. 24
Figura 2 – Força de Repulsão do objeto O sobre um robô localizado em R. 24
Figura 3 – Campo de Atração Constante 25
Figura 4 – Atração entre o robô R e a meta M. 25
Figura 5 – (a) Potencial Atrativo da Meta, (b) Potencial Repulsivo de dois obstáculos, (c) Soma dos dois campos. 26
Figura 6 – Mínimos Locais gerados pela anulação entres as Forças de Atração e Repulsão. 27
Figura 7 – Exemplo de Campos Potenciais Harmônicos (CPH), o qual não apresenta mínimos locais. 28
Figura 8 – Influência do vetor v no campo potencial. 32
Figura 9 – Comparação entres as trajetórias seguidas para várias taxas de influência ϵ. 32
Figura 10 – Campo instável, gerado por Diferenças Centradas, quando $\epsilon = 4$. 32
Figura 11 – Eficácia do CPH-GS para sistemas estáticos e sem obstáculos. 38
Figura 12 – Eficácia do CPH-GS para sistemas estáticos e sem obstáculos, movimento regressivo. 38
Figura 13 – Eficácia do CPH-GS para sistemas estáticos e com obstáculos. 38
Figura 14 – Eficácia do CPH-GS para sistemas estáticos e com obstáculos, movimento regressivo. 38
Figura 15 – Concorrência de múltiplos agentes em uma mesma malha gerada pelo CPH-GS. 39
Figura 16 – Eficácia do CPH-SOR para sistemas estáticos e sem obstáculos. 39
Figura 17 – Eficácia do CPH-SOR para sistemas estáticos e com obstáculos, movimento regressivo. 40
Figura 18 – Concorrência de múltiplos agentes em uma mesma malha gerada pelo CPH-SOR. 40
Figura 19 – Eficácia do CPO-GS-DC para sistemas estáticos e sem obstáculos, com v orientado em 45 graus. 42
Figura 20 – Eficácia do CPO-GS-DC para sistemas estáticos e sem obstáculos, com v orientado em 90 graus. 42
Figura 21 – Eficácia do CPO-GS-DC para sistemas estáticos e com obstáculos, com v orientado em 90 graus. 42
Figura 22 – Concorrência de múltiplos agentes em uma mesma malha gerada pelo CPO-GS-DC com v orientado em 0 grau. 43
Figura 23 – Estabilidade da malha gerada pelo CPO-GS-DC com $\epsilon = 0.5$. 43
Figura 24 – Estabilidade da malha gerada pelo CPO-GS-DC com $\epsilon = 1.0$. 43
Figura 25 – Estabilidade da malha gerada pelo CPO-GS-DC com $\epsilon = 1.5$. 44
Figura 26 – Estabilidade parcial da malha gerada pelo CPO-GS-DC com $\epsilon = 2.0$. . 44

Figura 27 – Instabilidade da malha gerada pelo CPO-GS-DC com $\epsilon = 3.0$. 44
Figura 28 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 0.5$. 45
Figura 29 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 1.0$. 45
Figura 30 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 1.5$. 45
Figura 31 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 2.0$. 46
Figura 32 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 3.0$. 46
Figura 33 – Eficácia do CPO-SOR-DC para sistemas estáticos e sem obstáculos, com v orientado em 135 graus. 47
Figura 34 – Eficácia do CPO-SOR-DC para sistemas estáticos e com obstáculos, com v orientado em 0 grau. 47
Figura 35 – Concorrência de múltiplos agentes em uma mesma malha gerada pelo CPO-SOR-DC com v orientado em 90 graus. 47
Figura 36 – Estabilidade da malha gerada pelo CPO-SOR-DC com $\epsilon = 0.5$. 48
Figura 37 – Estabilidade da malha gerada pelo CPO-SOR-DC com $\epsilon = 1.0$. 48
Figura 38 – Estabilidade da malha gerada pelo CPO-SOR-DC com $\epsilon = 1.5$. 48
Figura 39 – Estabilidade parcial da malha gerada pelo CPO-SOR-DC com $\epsilon = 2.0$. 48
Figura 40 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 0.5$. 49
Figura 41 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 1.0$. 49
Figura 42 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 1.5$. 49
Figura 43 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 2.0$. 50
Figura 44 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 3.0$. 50
Figura 45 – Comportamento do CPLO-GS-DC. 53
Figura 46 – Comportamento do CPLO-SOR-DC. 53
Figura 47 – Comportamento do CPLO-GS-UP. 53
Figura 48 – Comportamento do CPLO-SOR-UP. 53
Figura 49 – Componentes das velocidades V e ω sobre os eixos cartesianos. 59

LISTA DE TABELAS

Tabela 1 – Comparação entre CPH-GS e CPH-SOR 41
Tabela 2 – Comparação entre CPO-GS-DC e CPO-SOR-DC 51
Tabela 3 – Comparação entre CPO-GS-UP e CPO-SOR-UP 51
Tabela 4 – Comparação entre as possibilidades de implementação do CPLO. . . . 52

LISTA DE ABREVIATURAS E SIGLAS

CP	Campos Potenciais de Khatib
CPH	Campos Potencias Harmônicos
CPO	Campos Potenciais Orientados
CPLO	Campos Potenciais Localmente Orientados
DC	Diferenças Centradas
EDP	Equações Diferenciais Parciais
GS	Método de Relaxação Iterativo de Gauss-Seidel
JR	Método de Relaxação Iterativo de Jacobi-Richardson
PVC	Problema de Valor de Contorno
SOR	Método de Relaxação Iterativo Successive Over-Relaxation
UFJF	Universidade Federal de Juiz de Fora
UP	Diferenças Up Wind

LISTA DE SÍMBOLOS

$\forall$	Para todo
$\in$	Pertence
$\mathbb{R}$	Conjunto dos Números Reais
$\mathbb{R}^2$	Plano Cartesiano de Duas Dimensões
$\|v\|$	Norma do Vetor v
$\|c\|$	Módulo da Constante c
$v\angle a$	Vetor v com ângulo de orientação a graus.

SUMÁRIO

1	**INTRODUÇÃO**	**14**
1.1	OBJETIVO	15
1.2	ORGANIZAÇÃO DO TRABALHO	15
2	**MÉTODOS NUMÉRICOS**	**16**
2.1	FUNÇÃO HARMÔNICA	16
2.2	DIFERENÇAS FINITAS	16
2.2.1	**Diferenças Centradas**	19
2.2.2	**Diferenças Up Wind**	19
2.3	MÉTODOS DE RELAXAÇÃO	20
2.3.1	**Método de Relaxação Iterativo de Jacobi-Richardson (JR)**	20
2.3.2	**Método de Relaxação Iterativo de Gauss-Seidel (GS)**	21
2.3.3	**Método de Relaxação Iterativo Successive Over-Relaxation (SOR)**	21
2.4	CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO	21
3	**CAMPOS POTENCIAIS**	**23**
3.1	CAMPOS POTENCIAIS DE KHATIB (CP)	23
3.1.1	**Força de Repulsão**	24
3.1.2	**Força de Atração**	25
3.1.3	**Força Resultante**	26
3.2	CAMPOS POTENCIAIS HARMÔNICOS (CPH)	27
3.2.1	**Considerações acerca do CPH**	28
3.3	CAMPOS POTENCIAIS ORIENTADOS (CPO)	29
3.3.1	**Solução Numérica Utilizando Diferenças Centradas**	29
3.3.2	**Solução Numérica Utilizando Diferenças Up Wind**	30
3.3.3	**Considerações acerca do CPO**	31
3.4	CAMPOS POTENCIAIS LOCALMENTE ORIENTADOS	33
3.4.1	**Considerações acerca do CPLO**	34
3.5	CONTROLE DE VELOCIDADE DOS AGENTES ROBÓTICOS INSERIDOS NA GRADE	34
3.5.1	**Considerações acerca do Controle de Velocidade**	35
3.6	CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO	36
4	**APLICAÇÃO DOS MÉTODOS DE CAMPOS POTENCIAIS ESTUDADOS EM UM ROBÔ VIRTUAL**	**37**
4.1	PLANEJAMENTO VIA CAMPOS POTENCIAIS HARMÔNICOS	37

4.1.1 **Testes com Gauss-Seidel** . . . 37
4.1.2 **Testes com Successive Over-Relaxaton** . . . 39
4.1.3 **Considerações acerca do Planejamento via CPH** . . . 40
4.2 PLANEJAMENTO VIA CAMPOS POTENCIAIS ORIENTADOS . . . 41
4.2.1 **Testes com Gauss-Seidel** . . . 41
4.2.1.1 *Testes Utilizando Diferenças Centradas* . . . 41
4.2.1.2 *Testes Utilizando Diferenças Up Wind* . . . 44
4.2.2 **Testes com Successive Over-Relaxation** . . . 46
4.2.2.1 *Testes Utilizando Diferenças Centradas* . . . 46
4.2.2.2 *Testes Utilizando Diferenças Up Wind* . . . 49
4.2.3 **Considerações acerca do Planejamento via CPO** . . . 50
4.3 PLANEJAMENTO VIA CAMPOS POTENCIAIS LOCALMENTE ORIENTADOS . . . 52
4.3.1 **Considerações acerca do Planejamento via CPLO** . . . 52
4.4 CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO . . . 54

5 CONCLUSÕES . . . 55
5.1 TRABALHOS FUTUROS . . . 56

REFERÊNCIAS . . . 57

APÊNDICE A – Cinemática de Robôs Móveis . . . 59

APÊNDICE B – Cinemática Implementada em MATLAB® . 60

APÊNDICE C – Classe Environment Implementada em C++ 61

APÊNDICE D – Iterador do CPH Implementado em C++ . 63

APÊNDICE E – Iterador do CPO Implementado em C++ . 65

APÊNDICE F – Iterador do CPLO Implementado em C++ 69

APÊNDICE G – Vizinhança de Manhattam Implementado em C++ . . . 75

1 INTRODUÇÃO

A robótica foi alvo de muitos estudos nos últimos anos e, sem dúvida, continuará sendo ao longo dos próximos anos. Trata-se de uma área que lida com diversos elementos do âmbito da ciência e engenharia. Inicialmente os robôs foram concebidos para operações industriais no período em que a necessidade de produção em larga escala, demandava a substituição do trabalho manual pelo serviço mecanizado. Aos poucos, a robótica foi expandindo seus ambientes de atuação, assim surgiram os agentes robóticos de propósito industrial, móvel (exploração de ambientes), assistivo (próteses e sistemas vinculados a saúde), militar e mais recente o surgimento dos agentes com propósito social (interação homem-máquina), exemplo *Jibo* ([1]).

Todos os propósitos listados anteriormente enfrentam grandes desafios em sua execução e concepção, mais especificamente, os robôs móveis defrontam um grande problema quando estratégias de controle são abordadas, ou seja, não é uma tarefa trivial desenvolver um sistema de navegação *(Path Planning)* para os mesmos. Isto requer o estudo de técnicas inteligentes e que sejam capazes de reagir aos estímulos enviados pelo ambiente.

Em [2], 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 e Mecânica.

Partindo então deste desafio (futebol de robôs), pode-se propor várias técnicas e algoritmos de controle dos agentes em campo e testar seus desempenhos e eficiência, uma vez que esse ambiente é dinâmico e imprevisível. Assim, os sistemas projetados devem possuir um alto grau de autonomia, atuando com realimentação e em tempo real.

Existem vários tópicos de pesquisa específicos para estruturar uma equipe competitiva, dentre eles:

- Completa integração entre percepção, ação e cognição num time de múltiplos agentes robóticos;
- Definição de um conjunto de comportamentos reativos robustos para cada agente;
- Percepção em tempo real, robusta e confiável nas técnicas de rastreamento dos agentes e objetos em movimento.

O presente trabalho é motivado então pela oportunidade de investigação e implementação de técnicas que envolvem sistemas de navegação para robôs móveis, em particular, serão estudas técnicas de Campos Potenciais que partem da resolução de Equações Diferenciais Parciais (EDP).

1.1 OBJETIVO

O objetivo deste trabalho, é desenvolver um sistema de planejamento de caminho para a equipe de futebol de robôs autônomos da Universidade Federal de Juiz de Fora (UFJF), ***Rinobot Team***. Com base no estudo dos métodos de Campos Potenciais baseados em Problemas de Valor de Contorno (PVC), a navegação dos agentes robóticos da equipe será aprimorada, permitindo também o refinamento das técnicas deliberativas de estratégia do jogo.

1.2 ORGANIZAÇÃO DO TRABALHO

O trabalho está dividido em cinco capítulos, a começar por este com intuito introdutório.

No Capítulo 2 serão apresentadas as técnicas utilizadas para resolver as EDP decorrentes da teoria de Campos Potenciais baseados em PVC. Dentre elas estão: o Método das Diferenças Finitas, que discretiza e transforma as EDP em sistemas de equações algébricas lineares e métodos de relaxação utilizados para resolução destes sistemas de equações.

No Capítulo 3 serão apresentados os Campos Potenciais baseados em PVC, um comparativo entre tais métodos e o método de Campos Potenciais de Khatib (CP), bem como uma alternativa de Controle da Velocidade do agente robótico com base nestes métodos citados.

No Capítulo 4 serão apresentados resultados experimentais obtidos com a aplicação destes métodos de navegação e considerações sobre os mesmos.

No Capítulo 5 serão apresentadas as conclusões e considerações finais, além de expor algumas sugestões de trabalhos futuros para os problemas que ficaram em aberto.

Por último, nos Anexos, encontram-se os códigos utilizados neste trabalho, tanto os iteradores aplicados em C++ na IDE do VISUAL STUDIO 2013® para os métodos de Campos Potenciais estudados, quanto o código de cinemática aplicado no MATLAB®, além de uma abordagem rápida sobre cinemática de robôs móveis.

2 MÉTODOS NUMÉRICOS

Este Capítulo apresenta o tratamento matemático utilizado para o desenvolvimento deste trabalho, ou seja, os temas aqui apresentados são relacionados a resolução de EDP, em especial os métodos serão aplicados a Equação de Laplace e algumas derivações da mesma.

2.1 FUNÇÃO HARMÔNICA

Segundo [3], estritamente em matemática, Função Harmônica, é qualquer solução não trivial da Equação de Laplace (Equação 2.1), cujas derivadas primeira e segunda sejam contínuas. Encontra-se este tipo de função em vários sub-domínios da própria matemática, tendo imensa e rica utilidade na física, em análise de processos estocásticos e outras vertentes.

$$\sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} = 0 \tag{2.1}$$

Aqui limita-se a análise em duas dimensões da Equação 2.1, uma vez que o ambiente de estudo está confinado ao $\mathbb{R}^2$, assim tem-se:

$$\sum_{i=1}^{2} \frac{\partial^2 f}{\partial x_i^2} = 0 \tag{2.2}$$

De acordo com [4], a Equação de Laplace modela o potencial gravitacional ou eletrostático em pontos do espaço vazio. Encontrar soluções analíticas de um PVC[1], nem sempre é possível, ou mesmo se possível pode não ser uma solução simples, logo é preferível utilizar métodos numéricos para tal.

2.2 DIFERENÇAS FINITAS

Conforme [5], um dos primeiros métodos desenvolvidos para obter soluções numéricas de EDP é o método de Diferenças Finitas, o qual 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. Isto então reduzirá o PVC a um sistema linear de equações algébricas.

Em [6], o método de diferenças finitas é inicialmente desenvolvido para funções de uma única variável. Logo, seja $P(x)$ com $x \in$ ao $\mathbb{R}^2$. Definem-se um espaçamento $\Delta x > 0$ e um inteiro j. Aproxima-se o valor de $P(x)$, quando $x = j \times \Delta x$, pela seguinte expressão:

[1] Mais informações sobre PVC, podem ser encontradas em [3].

$$p_j \approx P(j \times \Delta x) \tag{2.3}$$

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

- Diferenças Atrasadas:
$$\frac{p_j - p_{j-1}}{\Delta x} \tag{2.4}$$
- Diferenças Centradas:
$$\frac{p_{j+1} - p_{j-1}}{2\Delta x} \tag{2.5}$$
- Diferenças Avançadas:
$$\frac{p_{j+1} - p_j}{\Delta x} \tag{2.6}$$

Ainda em [6], as Equações 2.4-2.6, são aproximações corretas e que podem ser provadas a partir da expansão de Taylor, se algumas condições de regularidade apresentadas em [7] forem satisfeitas.

Lembrando da definição matemática do polinômio e também do erro residual de Taylor centrada em um ponto genérico x_0:

$$P_n(x) = \sum_{k=0}^{n} \frac{f^k(x_0)}{k!}(x - x_0)^k \tag{2.7}$$

$$E_n(x) \approx \frac{f^{n+1}(x_0)}{(n+1)!}(x - x_0)^{n+1} \tag{2.8}$$

Então, para obter uma aproximação para a primeira derivada, trabalha-se com um polinômio de grau 1 da expansão de Taylor:

$$P_1(x) = f(x_0) + f'(x_0)(x - x_0) + E_1(x) \tag{2.9}$$

Aplicando uma mudança na Equação 2.9 sugerida por [10], tem-se:

$$P_1(x + \Delta x) = f(x) + f'(x)(\Delta x) + E_1(x) \tag{2.10}$$

$$P_1(x - \Delta x) = f(x) - f'(x)(\Delta x) + E_1(x) \tag{2.11}$$

Desta forma pode-se escrever:

$$\frac{P_1(x + \Delta x) - f(x)}{\Delta x} - \frac{E_1(x)}{\Delta x} = f'(x) \tag{2.12}$$

$$\frac{f(x) - P_1(x - \Delta x)}{\Delta x} + \frac{E_1(x)}{\Delta x} = f'(x) \tag{2.13}$$

Desprezando o erro residual, consegue-se então observar a correlação das novas equações obtidas, com as Equações 2.4 - 2.6, tendo em mente a Equação 2.3 e que $f(x) \equiv P(x)$:

$$\frac{P_1(x + \Delta x) - f(x)}{\Delta x} = f'(x) \tag{2.14}$$

$$\frac{f(x) - P_1(x - \Delta x)}{\Delta x} = f'(x) \tag{2.15}$$

$$\frac{P_1(x + \Delta x) - P_1(x - \Delta x)}{2\Delta x} = f'(x) \tag{2.16}$$

Seguindo o mesmo raciocínio apresentado pela expansão do polinômio de Taylor, pode-se obter também uma aproximação para a segunda derivada, neste caso a forma de aproximação mais comum é a central de segunda ordem:

$$\frac{d^2P(x)}{dx^2} \approx \frac{p_{j+1} - 2p_j + p_{j-1}}{\Delta x^2} \tag{2.17}$$

Então ao referir-se a funções de várias variáveis (duas para este trabalho, conforme dito anteriormente), basta aplicar o método para cada uma das variáveis.

$$\frac{\partial^2 P(x,y)}{\partial x^2} \approx \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} \tag{2.18}$$

$$\frac{\partial^2 P(x,y)}{\partial y^2} \approx \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta y^2} \tag{2.19}$$

Pode-se então reescrever a Equação 2.2 como a seguir:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta y^2} = 0 \tag{2.20}$$

Os valores de Δx e Δy representam o espaçamento utilizado para discretizar o ambiente e transformá-lo em uma grade (*grid*), ao utilizar um sistema discreto com espaçamentos regulares, ou seja, $h = \frac{\Delta x}{\Delta y} = 1$, obtém-se então a solução discreta para a Equação 2.2:

$$p_{i,j} = \frac{1}{4}(p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1}) \tag{2.21}$$

A Equação 2.21 segundo [8] e [9] quando aplicada sobre uma grade, ou seja, sobre cada divisão de tamanho Δx por Δy do ambiente, consiste em resolver um sistema de equações lineares no qual as variáveis p são potenciais das células livres da grade, mais

adiante serão apresentados métodos de relaxação que podem ser utilizados para resolver este sistema numericamente.

É importante destacar que existem vários métodos de discretização para EDP. Anteriormente o presente texto fez referência ao método de diferenças centradas de segunda ordem, como sendo um dos métodos de discretização mais comuns para a segunda derivada. Porém ainda neste texto, será visto um outro tipo de EDP, a qual contém também primeira derivada em sua formulação. Para este tipo de equação, mantém-se a discretização da segunda derivada com as diferenças centradas de segunda ordem e para a primeira derivada utilizam-se três tipos diferentes que terão como base as Equações 2.4-2.6, formalizando então dois grandes métodos de discretização: Diferenças Centradas e Diferenças *Up Wind*.

2.2.1 **Diferenças Centradas**

Considere a seguinte expressão:

$$\frac{\partial^2 P(x,y)}{\partial x^2} + f(x,y)\frac{\partial P(x,y)}{\partial x} = 0 \tag{2.22}$$

Utilizando a Equação 2.5 como base intuitiva e a Equação 2.18 já formalizada, tem-se:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + f_{i,j}\frac{p_{i,j+1} - p_{i,j-1}}{2\Delta x} = 0 \tag{2.23}$$

Após algumas operações algébricas é possível obter a Equação 2.24:

$$p_{i,j} = \frac{2 + f_{i,j}\Delta x}{4}p_{i,j+1} + \frac{2 - f_{i,j}\Delta x}{4}p_{i,j-1} \tag{2.24}$$

2.2.2 **Diferenças Up Wind**

Para este método deve-se considerar as Equações 2.4 e 2.6 como base intuitiva. Quando $f(x,y) > 0$, utiliza-se 2.6, quando $f(x,y) < 0$, utiliza-se 2.4. Desta maneira tem-se:

Para $f(x,y) > 0$:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + f_{i,j}\frac{p_{i,j+1} - p_{i,j}}{\Delta x} = 0 \tag{2.25}$$

Obtendo então a Equação 2.26:

$$p_{i,j} = \frac{p_{i,j+1}(1 + f_{i,j}\Delta x) + p_{i,j-1}}{2 + f_{i,j}\Delta x} \tag{2.26}$$

Para $f(x, y) < 0$:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + f_{i,j}\frac{p_{i,j} - p_{i,j-1}}{\Delta x} = 0 \tag{2.27}$$

Obtendo então a Equação 2.28:

$$p_{i,j} = \frac{p_{i,j-1}(1 - f_{i,j}\Delta x) + p_{i,j+1}}{2 - f_{i,j}\Delta x} \tag{2.28}$$

As subseções 2.2.1 e 2.2.2, terão grande relevância para as deduções do próximo capítulo.

2.3 MÉTODOS DE RELAXAÇÃO

De acordo com [10], para solucionar uma EDP de forma numérica, uma etapa de solução numérica de sistemas lineares é requerida. Tais métodos podem ser divididos em dois grandes grupos conforme [11]:

- **Métodos Diretos:** Métodos que forneceriam a solução exata, não fossem os erros de arredondamento, com um número finito de passos.
- **Métodos Iterativos:** Métodos que permitem obter a solução de um sistema linear dada uma certa precisão.

Ainda em [11] é dito que o arredondamento dos métodos diretos em sistemas de grande porte podem tornar a solução sem consistência (sem significado), ao passo que os erros não se acumulam ao utilizarmos métodos iterativos. Tal característica faz deste tipo de solução numérica a mais apropriada na resolução de EDP.

2.3.1 **Método de Relaxação Iterativo de Jacobi-Richardson (JR)**

Ao utilizar o JR aplicado a Equação 2.21, o resultado é a substituição simultânea do valor de todas as posições livres da grade pela média dos valores de seus vizinhos. Assim a função de atualização de cada célula passa a ser:

$$p_{i,j}^{k+1} = \frac{1}{4}(p_{i+1,j}^{k} + p_{i-1,j}^{k} + p_{i,j+1}^{k} + p_{i,j-1}^{k}) \tag{2.29}$$

Onde k representa o número da iteração. Este método demanda mais iterações se comparado aos demais que serão apresentados a seguir. Porém, possui a vantagem de ser um método naturalmente paralelo e quando utilizado em sistemas *multi-tarefa*, cada ponto

$p_{i,j}$ pode ser calculado por uma unidade de processamento.

2.3.2 Método de Relaxação Iterativo de Gauss-Seidel (GS)

Este método assemelha-se muito ao anterior, diferenciando-se pelo fato de possuir valores de vizinhos já atualizados no cálculo de uma posição, mais especificamente, o vizinho da esquerda ($p_{i,j-1}$) e o vizinho do topo ($p_{i-1,j}$) serão os vizinhos já atualizados. A implementação deste método pode ser feita através da varredura de um *array* bi-dimensional, atualizando-o conforme a Equação 2.21, desta forma tem-se a seguinte função:

$$p_{i,j}^{k+1} = \frac{1}{4}(p_{i+1,j}^{k} + p_{i-1,j}^{k+1} + p_{i,j+1}^{k} + p_{i,j-1}^{k+1}) \tag{2.30}$$

Note que ao atualizar uma posição por vez, a equação contará com metade dos vizinhos já atualizados, o cálculo sequencial de cada valor torna este método o mais natural quando aplicado a um sistema com um núcleo no processador. Obtendo então uma convergência mais rápida se comparado ao JR.

2.3.3 Método de Relaxação Iterativo Successive Over-Relaxation (SOR)

A diferença entre p^k e p^{k+1} tende a diminuir com as sucessivas aproximações, o SOR consiste em aumentar esta pequena diferença de maneira que ele obtenha uma convergência mais rápida em relação ao JR e GS. Este método é uma mudança do GS, como pode ser visto a seguir:

$$p_{SOR}^{k+1} = p_{GS}^{k+1} + \beta(p_{GS}^{k+1} - p^{k}) \tag{2.31}$$

Onde o valor $-1 < \beta < 1$, é um fator de relaxação. Em [12] o valor adotado é de $\beta = 0.8$ para experimentos realizados.

2.4 CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO

Este Capítulo tem suma importância para o texto, todo tratamento matemático para compreensão do trabalho é encontrado aqui. Foi introduzido o conceito de Função Harmônica através da Equação de Laplace (2.1 e 2.2), explicitando que a mesma será utilizada para modelar o potencial eletrostático do método de Campos Potenciais. Introduziu-se também o conceito de Diferenças Finitas, que são os métodos utilizados para transformar uma EDP contínua em discreta, resultando então num sistema de equações

lineares que demandam métodos iterativos de relaxação para sua resolução, métodos estes encontrados na seção 2.3. [2]

[2] Informações maiores sobre os métodos de relaxação, podem ser encontradas nos textos de [10], [11] e [12].

3 CAMPOS POTENCIAIS

O método de Campos Potenciais tem grande importância em sistemas robóticos. Trata-se de um algoritmo capaz de controlar e planejar caminhos. Tal método foi inicialmente proposto por [13] para o controle e proteção contra colisão de robôs manipuladores, e é conhecido por seu termo da língua inglesa *Potential Fields*.

A adaptação para agentes robóticos móveis se deu através de [14], e desde então, tem sido bastante popular em aplicações industriais e acadêmicas. Este algoritmo tem um comportamento bastante simples, baseia-se na ação de partículas com cargas elétricas sob efeito de um campo potencial.

Vários pesquisadores têm apresentado propostas para melhorar a eficiência deste método. Em [15] foi proposto que a velocidade do robô deveria ser reduzida na vizinhança de obstáculos para diminuir o risco de colisão, já em [16] foi proposto a implementação de funções potenciais através da combinação de funções individuais de obstáculos com operadores lógicos.

Estes e vários outros métodos, assumiam um modelo conhecido de mundo, com formas geométricas pré-definidas representando obstáculos ou então o caminho do robô era gerado *off-line*. Porém [17] e [18] foram os primeiros a introduzir a técnica de campos potencias de forma reativa, ou seja, para cada novo passo do robô, uma nova força resultante é calculada para direcionar seu movimento. Esta característica torna o algoritmo muito eficiente para agentes que realizam tarefas em ambientes desconhecidos e/ou com objetos dinâmicos, como é o caso do ambiente de futebol de robôs.

Algumas aplicações da utilização da técnica de campos potenciais aplicada ao planejamento de trajetórias para robôs móveis podem ser encontradas em [19] e [20].

Neste trabalho serão abordados três métodos de Campos Potenciais, a saber: Campos Potenciais Harmônicos (CPH), Campos Potenciais Orientados (CPO) e Campos Potenciais Localmente Orientados (CPLO) ([8]). Primeiro será introduzida uma técnica mais antiga e mais simples denominada Campos Potenciais de Kathib (CP) ([14]), a qual facilitará a compreensão das outras três técnicas citadas.

3.1 CAMPOS POTENCIAIS DE KHATIB (CP)

Neste método, considera-se a meta como um agente de atração, ou seja, a mesma exerce uma força de atração em direção a sua coordenada cartesiana. Os obstáculos por sua vez, são considerados como agentes de repulsão e tem função análoga a da meta. A direção que o robô deve seguir é dada por um vetor resultante da soma de todos os vetores força envolvidos.

A seguir, será mostrado como [8] modelou as forças de repulsão aos obstáculos e atração à meta, embora deva ser ressaltado que outros modelos de forças vetoriais podem ser criados para representar comportamentos além dos indicados para o ambiente de futebol de robôs.

3.1.1 **Força de Repulsão**

Na Figura 1 observa-se que a força de repulsão decai ao se afastar de um obstáculo, pois a força de repulsão ($\overrightarrow{F_r}$) é um vetor cujo módulo é inversamente proporcional ao quadrado da distância (d) entre o robô (R) e o objeto observado (O) (Figura 2).

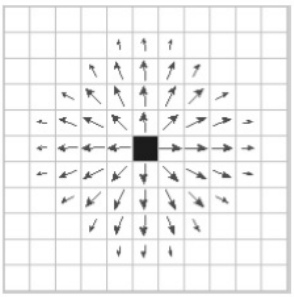

Figura 1 – Força de Repulsão na vizinhança de um obstáculo.

Fonte: Retirado de [8]

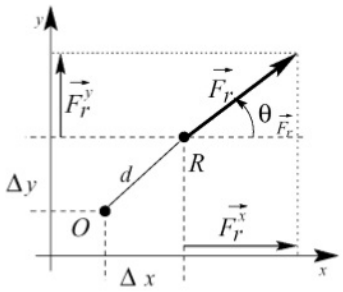

Figura 2 – Força de Repulsão do objeto O sobre um robô localizado em R.

Fonte: Retirado de [8]

O vetor $\overrightarrow{F_r}$ também pode ser representado por suas componentes: módulo e direção, conforme as Equações 3.1 e 3.2:

$$|\overrightarrow{F_r}| = \frac{Q}{d^2} \tag{3.1}$$

$$\theta_{\overrightarrow{F_r}} = \arctan(-\Delta y, -\Delta x) \tag{3.2}$$

Em que Q representa um escalar constante de repulsão 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 na Equação 3.3:

$$\overrightarrow{F_r} = \sum_{i=1}^{n} \overrightarrow{F_{r_i}} \tag{3.3}$$

Em que n representa o número total de obstáculos.

3.1.2 **Força de Atração**

A Figura 3 mostra o campo potencial de atração, o qual considera $|\overrightarrow{F_a}|$ constante. Desta forma o agente é atraído pela meta (M) mesmo estando distante, e não reduz sua velocidade na vizinhança da mesma, o que é essencial num ambiente de futebol de robôs.

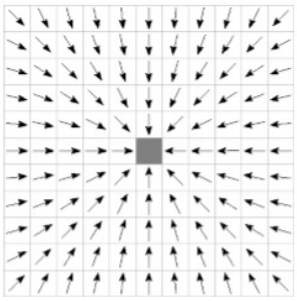

Figura 3 – Campo de Atração Constante

Fonte: Retirado de [8]

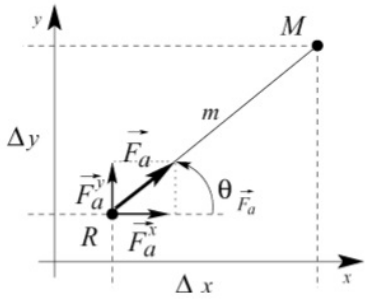

Figura 4 – Atração entre o robô R e a meta M.

Fonte: Retirado de [8]

A Força de Atração é equacionada em módulo e direção conforme as Equações 3.4 e 3.5:

$$|\overrightarrow{F_a}| = C \tag{3.4}$$

$$\theta_{\overrightarrow{F_a}} = \arctan(\Delta y, \Delta x) \tag{3.5}$$

Onde C representa um escalar constante de atração. Note que desta vez, Δx e Δy não

são negativos, uma vez que a orientação do vetor força de atração será na direção da meta (Figura 4).

3.1.3 **Força Resultante**

Ao realizar a soma vetorial de $\overrightarrow{F_a}$ e $\overrightarrow{F_r}$, obtém-se a Força Resultante $\overrightarrow{F}$.

$$\overrightarrow{F} = \overrightarrow{F_a} + \overrightarrow{F_r} \tag{3.6}$$

Para ilustrar a interação entre os vetores de força gerados pelo método CP tem-se a Figura 5:

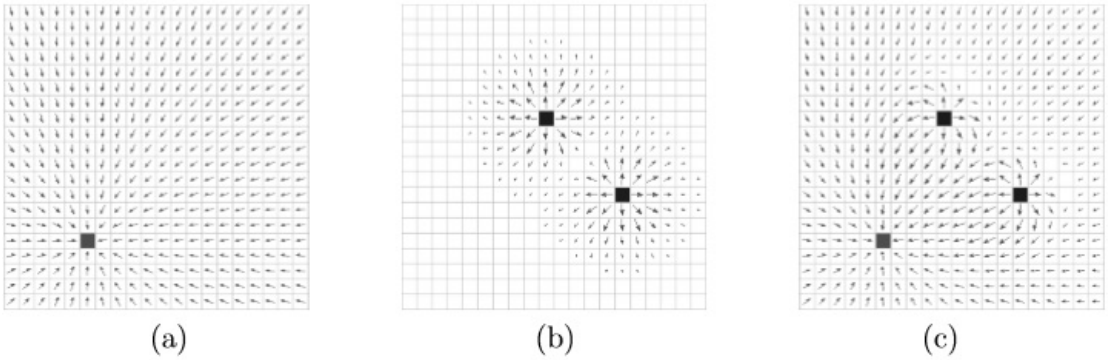

Figura 5 – (a) Potencial Atrativo da Meta, (b) Potencial Repulsivo de dois obstáculos, (c) Soma dos dois campos.

Fonte: Retirado de [8]

Com o exposto até aqui, podem ser feitas observações a cerca de algumas vantagens do CP:

- O custo computacional é baixo se comparado a métodos que utilizam mapas;
- 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;
- É um sistema modular no qual as forças podem ser calculadas em paralelo, o que facilita implementações em hardware.

Porém o CP apresenta também algumas desvantagens:

- O método não realiza exploração de ambientes, uma vez que não guarda informações ou modelos do mesmo.

- A ocorrência de *mínimos locais* é um problema grave em ambientes não dinâmicos. Se o agente estiver em uma região de mínimo local, o mesmo pode ficar sem "saída" e o método nunca irá convergir para a meta (Figura 6).

- Não é trivial determinar Q e C de forma a maximizar a eficiência do CP. Presume-se que o pesquisador se valha de valores empíricos ou que utilize algoritmos de calibragem.

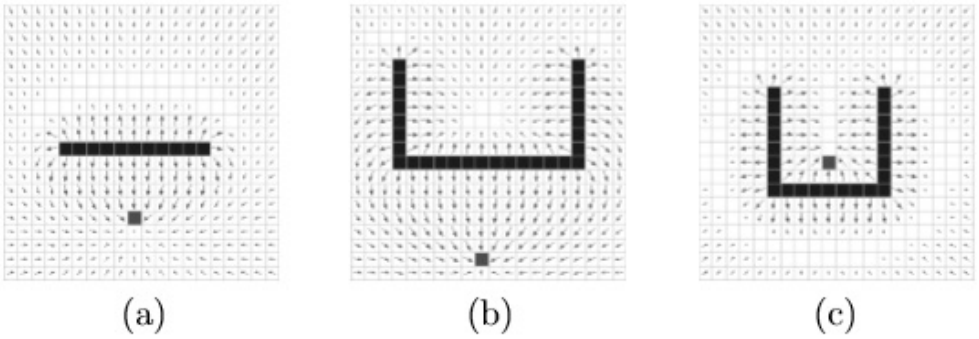

Figura 6 – Mínimos Locais gerados pela anulação entres as Forças de Atração e Repulsão.

Fonte: Retirado de [8]

3.2 CAMPOS POTENCIAIS HARMÔNICOS (CPH)

Apesar das limitações supracitadas, o método CP têm sido bastante utilizado. No trabalho desenvolvido por [12], o problema de mínimos locais foi resolvido com a utilização de Funções Harmônicas para o cálculo do campo potencial de ambientes nos quais as posições das paredes , objetos e metas sejam conhecidas.

Como dito no Capítulo 2, as Funções Harmônicas são soluções para a Equação de Laplace (Equações 2.1 - 2.2), sendo que estas possuem propriedades muito úteis ao desenvolvimento de sistemas de controle robótico como exposto em [8]:

- **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.

Para o problema de planejamento de caminho, o método CPH deve ser aplicado sobre uma representação discreta do ambiente em forma de grade. Desta forma, um PVC é então definido na região de atuação do robô utilizando a condição de Dirichlet, como exposto em [21] e [6], onde as células que representam obstáculos são marcadas com potencial um e as metas com potencial zero.

De acordo com [8], o potencial das células livres[1] deve ser relaxado, ou seja, calculado continuamente através da Equação 2.21. Essa equação então, interpola os valores dos potenciais para todas as células livres entre os obstáculos e a meta.

As soluções da Equação de Laplace não possuem mínimos/máximos locais (exposto em [21]), desta forma, caso exista um caminho, o objetivo é sempre alcançado seguindo-se as linhas de força. Tal fato é constatado na Figura 7, onde os mesmos ambientes utilizados na Figura 6, que receberam o método CP, recebem o tratamento com CPH.

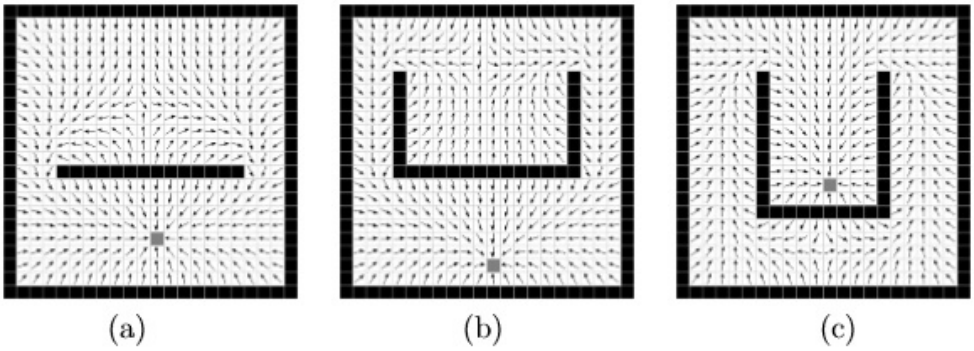

Figura 7 – Exemplo de Campos Potenciais Harmônicos (CPH), o qual não apresenta mínimos locais.

Fonte: Retirado de [8]

As linhas de força são extraídas a partir do gradiente descendente do potencial que direciona o robô para sua meta desviando de obstáculos, conforme mencionado em [12] e [22]. Deste modo, dada a posição de um robô em uma determinada célula (i,j), a direção Θ que deve ser seguida, pode ser calculada como:

$$\Theta_{i,j} = -\arctan(p_{i-1,j} - p_{i+1,j}, p_{i,j-1} - p_{i,j+1}) \tag{3.7}$$

Onde Θ, pertence ao intervalo $[-\pi, \pi]$.

3.2.1 **Considerações acerca do CPH**

Este método têm demonstrado sua eficiência, tanto para o problema de planejamento de caminhos quanto para exploração de ambientes. Assim como os outros métodos que se valem de um modelo do mundo para sua implementação, o custo computacional deste método cresce proporcionalmente com o tamanho do ambiente mapeado. Contudo, o que diferencia este método dos demais métodos de Campos Potenciais é o fato de ser um

[1] Células que não são tratadas como obstáculo ou meta. Seus potenciais estão no intervalo $0 < p_{i,j} < 1$.

gerador de caminhos completo, ou seja, um caminho livre até a meta sempre será gerado caso ele exista.

3.3 CAMPOS POTENCIAIS ORIENTADOS (CPO)

Como visto anteriormente, a solução da Equação de Laplace garante que a geração de Campos Potenciais não irá apresentar mínimos locais, no entanto, este é apenas um caso particular de um conjunto de funções que também possuem esta característica.

Em [23], é introduzida uma outra família de funções escalares geradas por um PVC e que não apresenta mínimos locais, sendo esta família denotada por:

$$\nabla^2 P + F(\nabla P) = 0 \tag{3.8}$$

Desde que $F(0) = 0$ e $F(\nabla P)$ seja uma função contínua.

Ainda em [23], é sugerido uma função linear para o novo termo $F(\nabla p)$:

$$F(\nabla P) = \epsilon \cdot \nabla P \cdot v \tag{3.9}$$

No qual $\epsilon \in \mathbb{R}$ é um escalar e $v \in \mathbb{R}^2$ é um vetor constante ($v = \begin{bmatrix} v_x & v_y \end{bmatrix}$) com $\|v\| = 1$. Substituindo a Equação 3.9 em 3.8, obtém-se:

$$\nabla^2 P + \epsilon \cdot \nabla P \cdot v = 0 \tag{3.10}$$

Observa-se então que as Equações 2.1 e 2.2 são casos especiais da Equação 3.10, quando $\epsilon = 0$. Nas próximas subseções serão apresentados métodos numéricos para discretizar a equação acima, com base nas subseções 2.2.1 e 2.2.2.

3.3.1 **Solução Numérica Utilizando Diferenças Centradas**

Nota-se que $\epsilon \cdot \nabla P \cdot v$ é equivalente ao produto interno entre os vetores ∇P, v e a constante ϵ. Desta forma, utilizando Diferenças Centradas, pode-se obter:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta x^2} + \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta y^2} + \frac{\epsilon}{2}\left[\frac{p_{i+1,j} - p_{i-1,j}}{\Delta y} v_y + \frac{p_{i,j+1} - p_{i,j-1}}{\Delta x} v_x\right] = 0 \tag{3.11}$$

Como foi assumido uma malha regular ($\Delta x = \Delta y = h$) e definindo ainda $\lambda = \frac{\epsilon h}{2}$, como exposto em [9], obtém-se:

$$p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j} + \lambda v_y(p_{i+1,j} - p_{i-1,j}) + \lambda v_x(p_{i,j+1} - p_{i,j-1}) = 0 \quad (3.12)$$

Agora pode-se obter a Equação 3.13, que é a solução discreta para a Equação 3.10.

$$p_{i,j} = \frac{1}{4}[(1 + \lambda v_y)p_{i+1,j} + (1 - \lambda v_y)p_{i-1,j} + (1 + \lambda v_x)p_{i,j+1} + (1 - \lambda v_x)p_{i,j-1}] \quad (3.13)$$

Segundo [9], tal discretização impõe um limite em λ, isto é, quanto maior for ϵ, menor precisa ser o espaçamento da malha. Ainda em [9], é dito que quando $\epsilon < 2$, o novo potencial de uma determinada célula é uma média ponderada dos seus vizinhos. Tal ponderação leva em consideração o vetor v e o seu módulo ϵ.

3.3.2 **Solução Numérica Utilizando Diferenças Up Wind**

De acordo com [9], o uso de Diferenças Up Wind, tem como finalidade obter um sistema de planejamento de caminhos com menos limitações, ou seja, pode-se trabalhar com valores mais elevados de ϵ.

A Equação 3.10, possui duas derivadas de primeira ordem, portanto, ao utilizar Diferenças Up Wind em uma malha regular, podem ser definidos quatro casos:

Para $v_x > 0$ e $v_y > 0$:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2} + \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{h^2} + \epsilon v_x \frac{p_{i,j+1} - p_{i,j}}{h} + \epsilon v_y \frac{p_{i+1,j} - p_{i,j}}{h} = 0 \quad (3.14)$$

Assim, pode-se obter:

$$p_{i,j} = \frac{(1 + h\epsilon v_x)p_{i,j+1} + (1 + h\epsilon v_y)p_{i+1,j} + p_{i,j-1} + p_{i-1,j}}{4 + h\epsilon(v_x + v_y)} \quad (3.15)$$

Para $v_x > 0$ e $v_y < 0$:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2} + \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{h^2} + \epsilon v_x \frac{p_{i,j+1} - p_{i,j}}{h} + \epsilon v_y \frac{p_{i,j} - p_{i-1,j}}{h} = 0 \quad (3.16)$$

Assim, pode-se obter:

$$p_{i,j} = \frac{(1 + h\epsilon v_x)p_{i,j+1} + (1 - h\epsilon v_y)p_{i-1,j} + p_{i,j-1} + p_{i+1,j}}{4 + h\epsilon(v_x - v_y)} \quad (3.17)$$

Para $v_x < 0$ e $v_y > 0$:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2} + \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{h^2} + \epsilon v_x \frac{p_{i,j} - p_{i,j-1}}{h} + \epsilon v_y \frac{p_{i+1,j} - p_{i,j}}{h} = 0 \quad (3.18)$$

Assim, pode-se obter:

$$p_{i,j} = \frac{(1 - h\epsilon v_x)p_{i,j-1} + (1 + h\epsilon v_y)p_{i+1,j} + p_{i,j+1} + p_{i-1,j}}{4 + h\epsilon(-v_x + v_y)} \quad (3.19)$$

Para $v_x < 0$ e $v_y < 0$:

$$\frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2} + \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{h^2} + \epsilon v_x \frac{p_{i,j} - p_{i,j-1}}{h} + \epsilon v_y \frac{p_{i,j} - p_{i-1,j}}{h} = 0 \quad (3.20)$$

Assim, pode-se obter:

$$p_{i,j} = \frac{(1 - h\epsilon v_x)p_{i,j-1} + (1 - h\epsilon v_y)p_{i-1,j} + p_{i,j+1} + p_{i+1,j}}{4 + h\epsilon(-v_x - v_y)} \quad (3.21)$$

Conforme [9], pode ser observado que independente do valor de ϵ, o novo potencial de uma determinada célula, sempre será uma média ponderada entre os seus vizinhos.

3.3.3 **Considerações acerca do CPO**

Devido a influência do vetor v na orientação do campo gradiente, esta abordagem utilizando a Equação 3.10, foi denominada por [8] como Campos Potenciais Orientados (CPO). Na Figura 8, é mostrado o campo potencial gerado utilizando-se a Equação 3.13, 3.15, 3.17, 3.19 ou 3.21, para diferentes vetores de orientação v.

Na Figura 8(a), tem-se um CPH sem influência de vetor, enquanto que nas Figuras 8(b) a (d), tem-se um CPO com $v\angle 0$, $v\angle -45$ e $v\angle -90$ respectivamente.

Por outro lado, na Figura 9, observa-se o comportamento para valores diferentes de ϵ. Segundo [8] e [9], a variável ϵ, pode ser vista como a taxa de influência do vetor v sobre o campo potencial.

Na Figura 9, compara-se os caminhos com $0 \leq \epsilon \leq 1.5$, e é possível observar que, quanto maior o valor de ϵ, mais o caminho se aproxima da direção do vetor v (-90 graus). Contudo, o valor de ϵ não pode crescer indefinidamente, a depender da discretização utilizada, como dito anteriormente. A Figura 10 mostra um campo instável, gerado por Diferenças Centradas, que não garante uma trajetória até a meta ($\epsilon = 4$). Isto acontece pois as condições inciais de contorno, na qual a meta atrai e o obstáculo repele, foram violadas.

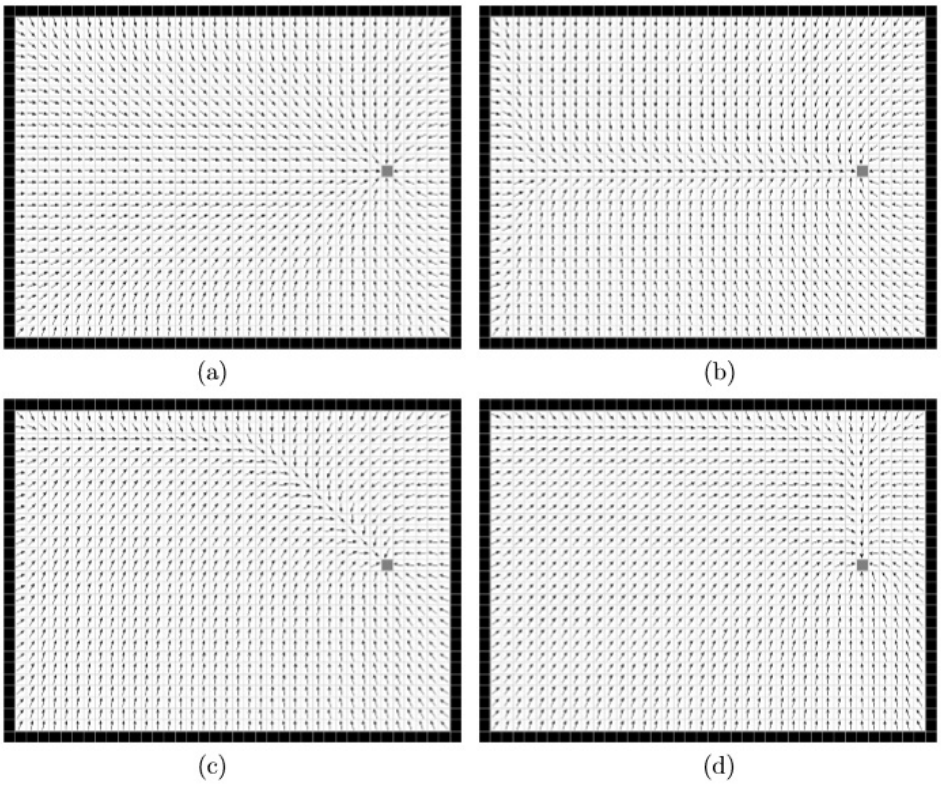

Figura 8 – Influência do vetor v no campo potencial.

Fonte: Retirado de [8]

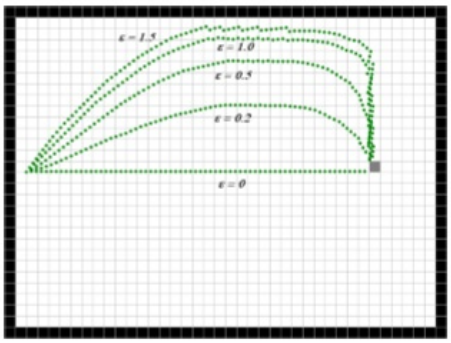

Figura 9 – Comparação entres as trajetórias seguidas para várias taxas de influência ϵ.

Fonte: Retirado de [8]

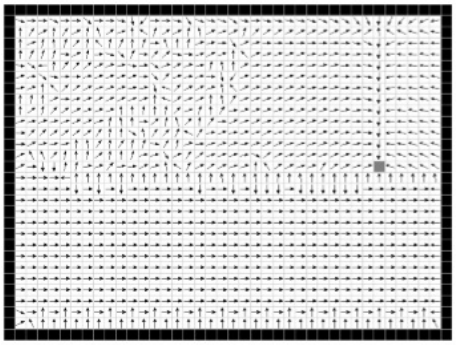

Figura 10 – Campo instável, gerado por Diferenças Centradas, quando $\epsilon = 4$.

Fonte: Retirado de [8]

Assim como o CPH, o CPO demonstra eficiência no planejamento de caminhos e exploração de ambientes, o custo computacional segue a mesma linha de raciocínio do CPH e também trata-se de um método completo. O uso deste método é destinado a situações nas quais seja favorável atingir a meta sob certa orientação (chute a gol).

3.4 CAMPOS POTENCIAIS LOCALMENTE ORIENTADOS

Este método foi proposto por [8], sendo uma alternativa para solução de algumas limitações encontradas durante a formulação de uma estratégia de jogo em seu trabalho. Dentro dessas limitações encontram-se:

- Uma única malha é gerada, logo todos os agentes consultam o mesmo mapa. Desta forma todos seguirão para a mesma meta.
- A alocação de paredes virtuais para delimitação de áreas de atuação dos agentes, podem resultar em direções errôneas, uma vez que algumas regiões do mapa não possuiriam meta definida. Este fato é um resultado natural dos métodos CPH e CPO, pois estes garantem traçar uma trajetória livre até a meta caso ela exista.

O método de Campos Potenciais Localmente Orientados (CPLO), segundo [8], consiste de uma modificação do algoritmo de relaxação do CPO, tornando o CPLO capaz de manusear diferentes comportamentos em um mesmo mapa discreto do ambiente.

A proposta aqui apresentada, utiliza uma única grade na qual muitos robôs possam seguir diferentes vetores de orientação (v) que conduzam à meta por diferentes trajetórias. Este método foi obtido por [8] ao aplicar a Equação 3.13 somente na vizinhança de cada agente. Assume-se como vizinhança, todas as células contidas em uma circunferência de raio fixo com centro na posição do robô.

Como visto na seção anterior, pode-se então aplicar as Equações 3.15, 3.17, 3.19 e 3.21 na vizinhança de cada agente e obter outras abordagens do CPLO como feito em [9].

No trabalho de [8], é mostrado como o método de planejamento através do CPLO atua em um Sistema Baseado em Regras para gerir uma partida do futebol de robôs. Neste trabalho, limita-se a determinação do CPLO apenas como método de navegação com base em PVC.

Como vários robôs estão sendo considerados, supõe-se que cada robô k, que atua sobre a grade, possua um vetor de orientação v_k e um parâmetro ϵ_k. Desta forma, todos os robôs farão consultas a grade para direcionar seus movimentos no campo. Assim o CPLO pode ser descrito da seguinte forma:

$$\nabla^2 P + \epsilon(x, y) \cdot \nabla P \cdot v(x, y) = 0 \tag{3.22}$$

Onde $\epsilon(x, y) = \epsilon_k \in \mathbb{R}$, caso (x, y) esteja na área de influência do robô k, senão $\epsilon(x, y) = 0$. E $v(x, y) = v_k \in \mathbb{R}^2$ com $\|v_k\| = 1$, caso (x, y) esteja na área de influência do robô k, senão $v(x, y) = \vec{0}$.

Pode ser observado pela Equação 3.22, que o CPLO mescla o CPO e o CPH, em uma mesma malha. Desta forma, quando (x, y) não pertencem a área de influência, utiliza-se a Equação 2.21 como solução discreta da Equação 3.22. Quando (x, y) pertencem a área de influência, utiliza-se uma destas Equações (3.13, 3.15, 3.17, 3.19 ou 3.21).

3.4.1 **Considerações acerca do CPLO**

Assim como no CPO, para que a Equação 3.13 quando utilizada pelo CPLO tenha garantia de ser consistente e estável, é necessário que $0 \leq \epsilon < 2$ ([9]). Enquanto que o uso das Equações 3.15, 3.17, 3.19 e 3.21, possibilitam obter um método numérico incondicionalmente convergente.

Segundo [9], embora o CPLO permita controlar vários robôs utilizando uma mesma malha, isto não quer dizer que este seja o melhor método. Considera-se que cada agente tem uma meta específica e única, por isso, o uso de uma grade por agente seria o ideal.

Posto desta forma, conclui-se que para um sistema sequencial, o método CPLO é uma grande alternativa, visto o custo computacional demandado para atualização da grade para cada agente. Enquanto que nos sistemas multitarefa (paralelo), o uso dos métodos CPO e CPH para cada agente, possam representar uma excelente alternativa.

3.5 CONTROLE DE VELOCIDADE DOS AGENTES ROBÓTICOS INSERIDOS NA GRADE

Uma das maiores dificuldades no controle de movimento em um sistema robótico é o controle do hardware, ou seja, ajustar a velocidade de cada motor afim de direcionar o robô para sua meta.

Deseja-se que o agente chegue a seu destino o mais rápido e suave possível, tornando necessário um sistema de controle de baixo nível para ajuste de rotação dos motores e funções geradoras de velocidade para os robôs, oriundas dos sistemas de navegação.

Os sistemas de controle de baixo nível não são escopo deste trabalho, logo, limita-se a análise as funções geradoras de velocidade. Em [8], é feita uma proposta que equilibra velocidade angular e velocidade linear, permitindo então obter o maior rendimento possível dos motores.

A proposta feita por [8], considera que a orientação do robô Φ e a orientação de destino Θ (Equação 3.7), pertençam ao intervalo [-180, 180] graus. Assim tem-se:

$$\alpha = \Theta - \Phi \tag{3.23}$$

Onde $\alpha \in [-180, 180]$ graus.

Porém alguns ajustes em α são necessários, uma vez que existem casos onde seja melhor girar o robô no sentido oposto, ou seja, quando $|\alpha| > 180$ graus. Logo:

$$\alpha = \alpha - 360 \iff \alpha > 180 \tag{3.24}$$

$$\alpha = \alpha + 360 \iff \alpha < -180 \tag{3.25}$$

$$\alpha = \alpha \iff -180 \leq \alpha \leq 180 \tag{3.26}$$

A função que gera a velocidade angular apresentada por [8], é dada como a seguir:

$$\omega(\alpha) = \frac{V_{max} \times \alpha}{180} \tag{3.27}$$

Onde V_{max} é a velocidade linear máxima que o agente robótico pode atingir. Desta forma o robô teria $\omega(\alpha)$ máxima quando $\alpha = 180$ graus e $\omega(\alpha)$ mínima quando $\alpha = 0$ grau.

Por outro lado, a velocidade linear, segue um comportamento inversamente proporcional ao valor da velocidade angular, pois deseja-se que o robô tenha velocidade linear máxima quando $\alpha = 0$ e tenha velocidade linear mínima, quando $L_\alpha \leq \alpha$. A constante L_α foi estabelecida para indicar que o agente deve somente girar sobre o próprio eixo quando $\alpha \geq L_\alpha$.

O limite estabelecido por L_α, ajusta a suavidade do movimento do robô ao mudar de direção. Quanto menor L_α, mais bruscos serão os movimentos. Desta forma, em [8] é proposta a seguinte função que gera a velocidade linear:

$$V(\alpha) = \frac{-V_{max} \times |\alpha|}{L_\alpha} + V_{max} \iff |\alpha| \leq L_\alpha \tag{3.28}$$

Caso contrário, $V(\alpha) = 0$.

3.5.1 Considerações acerca do Controle de Velocidade

Nota-se que a função geradora de velocidade linear proposta por [8], tem como consequência, $V(\alpha) \geq 0 \ \forall \ \alpha$. Com velocidade linear limitada positivamente, o robô se moverá sempre de forma progressiva. No entanto os agentes da ***Rinobot Team***, possuem uma concepção mecânica na qual pode-se aproveitar ambas as direções (progressiva e regressiva), sendo portanto um desperdício desconsiderar o movimento regressivo.

Feita esta observação, é necessário adaptar o trabalho de [8], para oferecer ao agente a possibilidade de se movimentar com velocidade linear negativa, visando metas que estejam por trás do mesmo.

Desta forma, L_α passa a valer noventa graus, ou seja, quando a orientação da meta estiver fora da limitação [-90, 90] graus, significa que a mesma demanda um movimento regressivo do agente robótico. Assim tem-se a seguinte formulação:

Quando $|\alpha| \leq L_\alpha$:

$$V(\alpha) = \frac{-V_{max} \times |\alpha|}{L_\alpha} + V_{max}$$

Quando $|\alpha| > L_\alpha$:

$$\alpha = \alpha + 180 \tag{3.29}$$

$$V(\alpha) = \frac{V_{max} \times |\alpha|}{L_\alpha} - V_{max} \tag{3.30}$$

No primeiro caso, mantém-se o que [8] já havia proposto, no segundo caso, troca-se o sinal da Equação 3.28 obtendo 3.29. Outra mudança importante é acrescer cento e oitenta graus a α, desta forma, garante-se que o sentido de rotação será trocado, fazendo com que $\omega(\alpha)$ acompanhe o movimento regressivo do agente robótico.

Antes de gerar $V(\alpha)$ e $\omega(\alpha)$ é necessário ajustar α conforme as Equações 3.24 - 3.26.

3.6 CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO

Este Capítulo aborda o desenvolvimento teórico necessário para a compreensão dos métodos de Campos Potenciais baseados em Problemas de Valor de Contorno: Campos Potenciais Harmônicos (CPH), Campos Potenciais Orientados (CPO) e Campos Potenciais Localmente Orientados (CPLO). Estes métodos, resolvem o problema de mínimos locais encontrados nos Campos Potenciais de Khatib (seção 3.1), uma vez que são concebidos através de Equações Diferenciais Parciais que não dispõem de mínimos e máximos locais em suas soluções. Todos estes métodos terão suma importância para o desenvolvimento estratégico da ***Rinobot Team***.

4 APLICAÇÃO DOS MÉTODOS DE CAMPOS POTENCIAIS ESTUDADOS EM UM ROBÔ VIRTUAL

O ambiente que será utilizado para aplicação e validação dos métodos de Campos Potenciais mencionados anteriormente, é gerado com auxílio do VISUAL STUDIO 2013® e com o MATLAB®.

O papel do VISUAL STUDIO 2013® é a implementação dos sistemas de navegação, utilizando a linguagem de programação C++, sendo então possível desenvolver várias grades de teste para os métodos de Campos Potenciais.

Os papéis do MATLAB® são o desenvolvimento do robô virtual, bem como a plotagem do ambiente para aferição da funcionalidade dos sistemas de navegação.

Para todos os testes realizados, utiliza-se uma grade com espaçamentos regulares de 5×5 centímetros. O ambiente mede 100×100 centímetros e o robô virtual tem as mesmas dimensões das células. Foram aplicados os métodos de relaxação Gauss-Seidel e Successive Over-Relaxation para os testes, devido sua facilidade de implementação em algoritmos sequenciais. Outros parâmetros, como os limites de velocidade, se encontram nos códigos dos apêndices.

4.1 PLANEJAMENTO VIA CAMPOS POTENCIAIS HARMÔNICOS

No primeiro teste, verificou-se a eficácia do método CPH em situações as quais os únicos obstáculos são as paredes externas ao ambiente. Enquanto que no segundo teste, verificou-se a capacidade de desvio de obstáculos do CPH. Por fim, verificou-se o comportamento de múltiplos agentes no ambiente. (Figuras 11 a 18).

4.1.1 **Testes com Gauss-Seidel**

Nas Figuras 11 e 12, pode-se notar que para situações onde não existem obstáculos e quanto menor for α (Equação 3.23), mais suave é o movimento realizado pelo agente robótico. Verificou-se também a funcionalidade do movimento regressivo proposto na subseção 3.5.1.

Os agentes são representados pelos quadrados amarelos, a meta é representada pelo quadrado vermelho, os obstáculos são representados pelos quadrados cinzas, o caminho seguido pelo agente é sinalizado pela linha azul, a orientação do robô pela linha ciano e as direções que o robô deve seguir, são sinalizadas pelas setas pretas.

Nas Figuras 13 e 14, nota-se que o método CPH-GS[1] é capaz de lidar com obstáculos. Como dito anteriormente neste trabalho, o CPH é um sistema completo, existindo um

[1] As siglas apresentadas neste capítulo fazem referência ao método de navegação, ao sistema de relaxamento e a discretização utilizada (quando necessário), respectivamente.

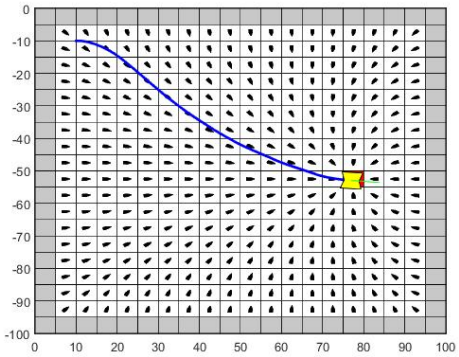

Figura 11 – Eficácia do CPH-GS para sistemas estáticos e sem obstáculos.

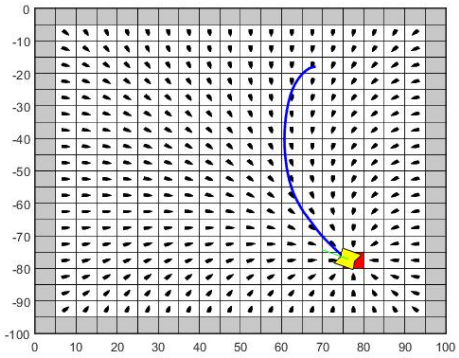

Figura 12 – Eficácia do CPH-GS para sistemas estáticos e sem obstáculos, movimento regressivo.

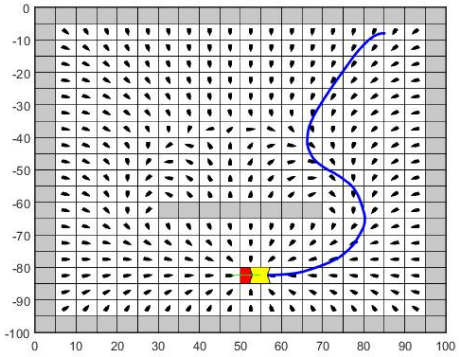

Figura 13 – Eficácia do CPH-GS para sistemas estáticos e com obstáculos.

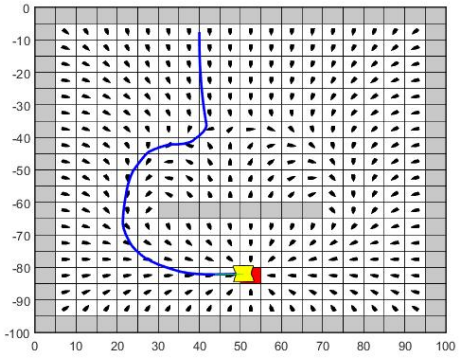

Figura 14 – Eficácia do CPH-GS para sistemas estáticos e com obstáculos, movimento regressivo.

caminho livre até a meta, este sempre será encontrado.

Por fim, na Figura 15, observa-se que na ocorrência de múltiplos agentes na malha, um evento de competição pela meta é criado como consequência. O que se torna uma grande desvantagem no uso deste método em sistemas sequenciais com vários agentes.

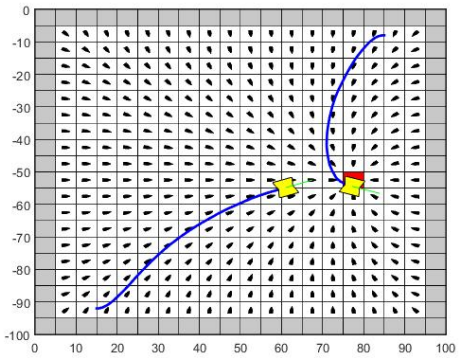

Figura 15 – Concorrência de múltiplos agentes em uma mesma malha gerada pelo CPH-GS.

4.1.2 **Testes com Successive Over-Relaxaton**

Os mesmos testes da subseção anterior foram aplicados nesta subseção. Assim, as observações feitas no primeiro teste com GS, são mantidas para este teste com SOR (foi adotado $\beta = 0.8$ conforme [12], Equação 2.31), o que pode ser observado na Figura 16.

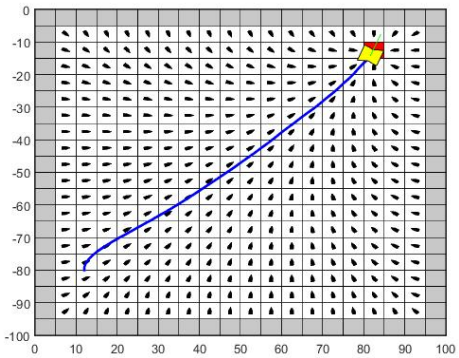

Figura 16 – Eficácia do CPH-SOR para sistemas estáticos e sem obstáculos.

O mesmo ocorre no segundo teste, as observações feitas com base no método GS, são mantidas para o teste com SOR. Conforme mostra a Figura 17.

Assim como nos demais testes, mantém-se as afirmações feitas na subseção anterior, conforme Figura 18.

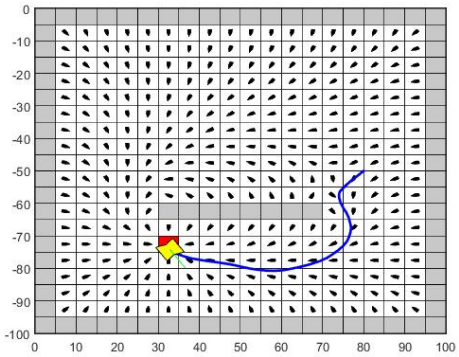

Figura 17 – Eficácia do CPH-SOR para sistemas estáticos e com obstáculos, movimento regressivo.

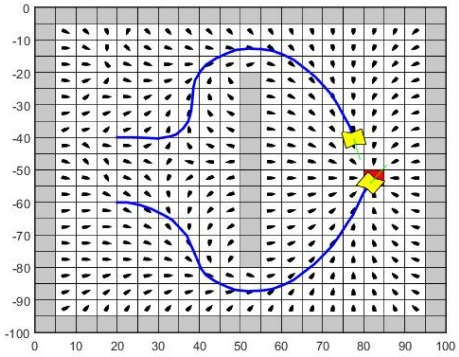

Figura 18 – Concorrência de múltiplos agentes em uma mesma malha gerada pelo CPH-SOR.

4.1.3 **Considerações acerca do Planejamento via CPH**

Para ambos os métodos de relaxação (GS e SOR), nos testes 1 e 2, observaram-se resultados que comprovam a funcionalidade do planejamento de caminhos via CPH, ou seja, em ambos os casos, o robô virtual atinge a meta.

O que diferencia o GS do SOR, é o número médio de iterações que cada método precisou para que a malha fosse criada, embora a relaxação contínua do ambiente permita uma maior suavidade nas transições de direcionamento de uma célula a outra. Para estes testes, os quais consideram ambientes estáticos para geração da grade, adotou-se um critério de convergência estimado pelo erro apresentado na seguinte fórmula[2]:

$$erro = \sum_{j=1}^{n}\sum_{i=1}^{m}(p_{i,j}^{k} - p_{i,j}^{k-1})^2 \tag{4.1}$$

Desta forma, quando $erro \leq 10^{-6}$, pode-se dizer que houve uma convergência e a malha gerada é satisfatória. A Tabela 1[3] compara então os dois métodos.

[2] Critério utilizado por [9].

[3] O valor incial do potencial das células livre foi de 0.5, para todos os testes.

Teste	N. Iterações GS	N. Iterações SOR
1	175	41
2	117	44

Tabela 1 – Comparação entre CPH-GS e CPH-SOR

Logo, como o método SOR converge de forma mais rápida, para ambientes dinâmicos, onde é demandado relaxação contínua do ambiente, este método apresentará vantagens.

Para o terceiro teste, verificou-se que para múltiplos agentes inseridos numa mesma malha, a situação de competição pela meta seria consequência direta, uma vez que o CPH direciona toda a malha para o objetivo.

Posto desta forma, ocorreriam problemas relacionados a estratégia de jogo em uma equipe de futebol de robôs. Em [8] é proposto o CPLO para resolução deste problema. Em [9] é proposto a geração de grades individuais para cada agente. Neste trabalho foi dito que a alternativa de [9] poderia gerar um alto custo computacional, o que demandaria uma implementação do CPH em arquiteturas paralelas.

4.2 PLANEJAMENTO VIA CAMPOS POTENCIAIS ORIENTADOS

Os testes da seção anterior foram também aplicados ao CPO, o intuito foi comprovar as semelhanças entre os métodos. Porém, no teste mais importante desta seção, verificou-se o quanto ϵ (Equações 3.13, 3.15, 3.17, 3.19 e 3.21) tem influência na trajetória e na estabilidade da grade gerada pelo CPO.

4.2.1 **Testes com Gauss-Seidel**

Esta subseção se divide em duas outras partes, uma vez que no trabalho de [9], foram propostas duas formas de discretização da Equação 3.10, conforme visto anteriormente.

4.2.1.1 *Testes Utilizando Diferenças Centradas*

Para os três primeiros testes, $\epsilon = 1.5$, com este valor, tem-se uma grade estável.

Para as situações em que não existem obstáculos internos, somente as paredes externas ao ambiente, tem-se as Figuras 19 e 20.

Para as situações em que existem obstáculos internos juntamente com paredes externas, tem-se a Figura 21.

Para as situações em que existem múltiplos agentes na malha, tem-se a Figura 22.

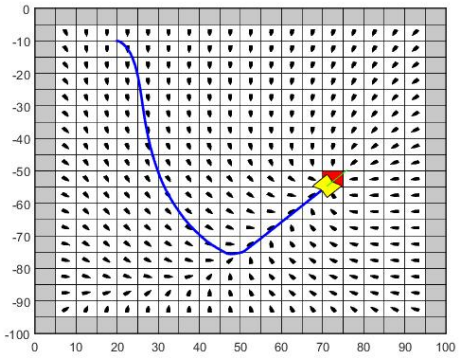

Figura 19 – Eficácia do CPO-GS-DC para sistemas estáticos e sem obstáculos, com v orientado em 45 graus.

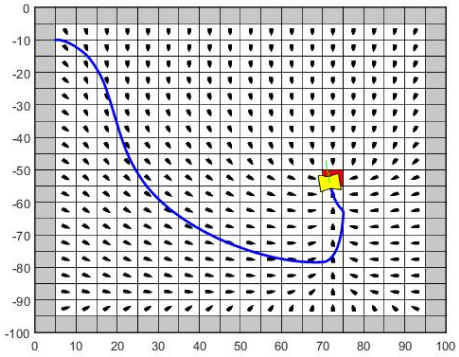

Figura 20 – Eficácia do CPO-GS-DC para sistemas estáticos e sem obstáculos, com v orientado em 90 graus.

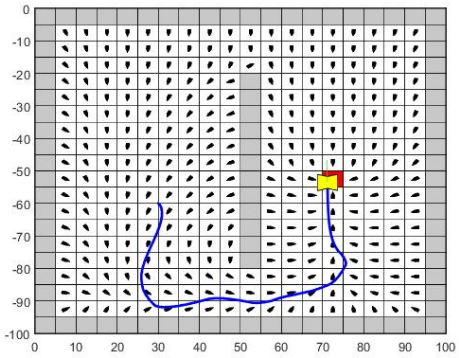

Figura 21 – Eficácia do CPO-GS-DC para sistemas estáticos e com obstáculos, com v orientado em 90 graus.

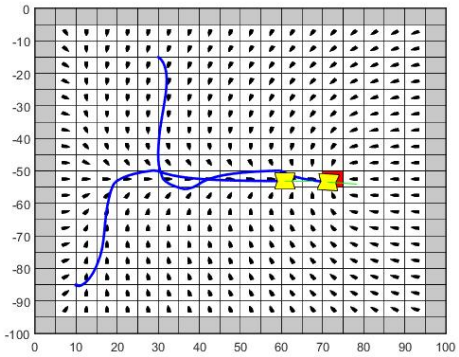

Figura 22 – Concorrência de múltiplos agentes em uma mesma malha gerada pelo CPO-GS-DC com v orientado em 0 grau.

Para o próximo teste, utilizou-se $\epsilon = (0.5, 1.0, 1.5, 2.0, 3.0)$ e $v\angle -90$. Conforme Figuras 23-27.

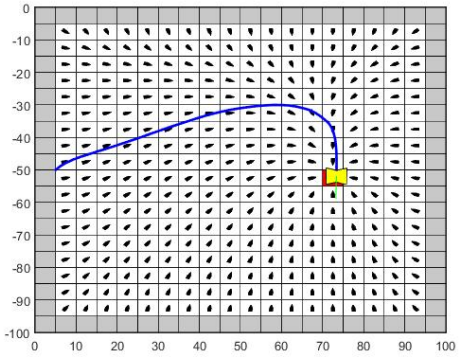

Figura 23 – Estabilidade da malha gerada pelo CPO-GS-DC com $\epsilon = 0.5$.

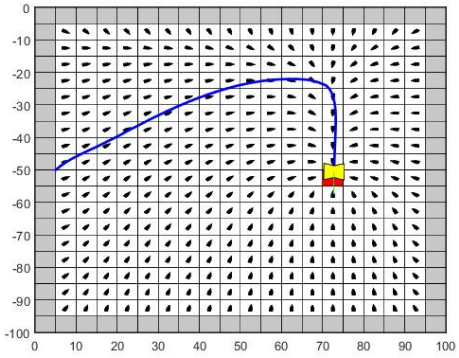

Figura 24 – Estabilidade da malha gerada pelo CPO-GS-DC com $\epsilon = 1.0$.

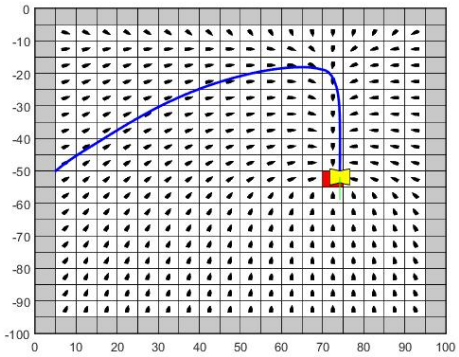

Figura 25 – Estabilidade da malha gerada pelo CPO-GS-DC com $\epsilon = 1.5$.

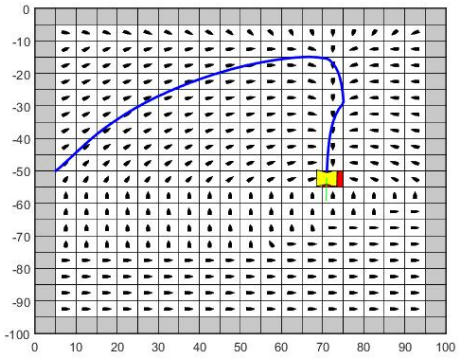

Figura 26 – Estabilidade parcial da malha gerada pelo CPO-GS-DC com $\epsilon = 2.0$.

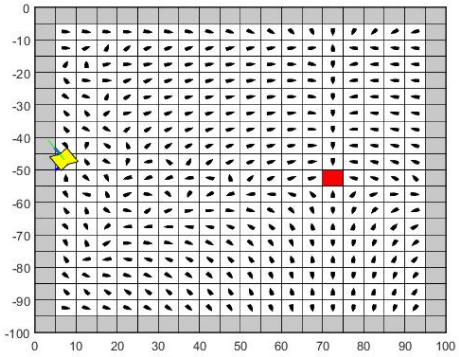

Figura 27 – Instabilidade da malha gerada pelo CPO-GS-DC com $\epsilon = 3.0$.

Quanto maior for ϵ, mais a trajetória tende a convergir para a orientação do vetor v (Figuras 23-25). Porém, quando $\epsilon \geq 2$, o sistema começa a apresentar instabilidade (Figuras 26 e 27). Fatos estes que vão de encontro ao exposto no capítulo anterior.

4.2.1.2 *Testes Utilizando Diferenças Up Wind*

Como as Diferenças Up Wind são apenas outra maneira de discretização do ambiente, não se fez necessário a aplicação dos três primeiros testes básicos, uma vez que

os resultados seriam os mesmos obtidos nos testes utilizando Diferenças Centradas.

Limita-se aqui então o teste com $\epsilon = (0.5, 1.0, 1.5, 2.0, 3.0)$ e $v\angle -90$. Conforme Figuras 28-32.

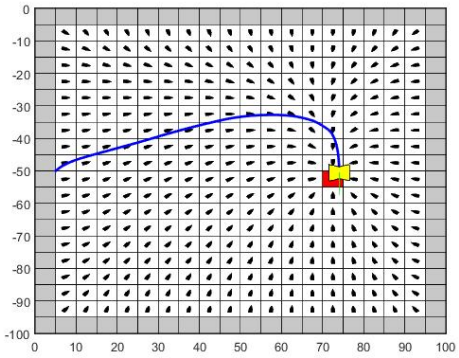

Figura 28 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 0.5$.

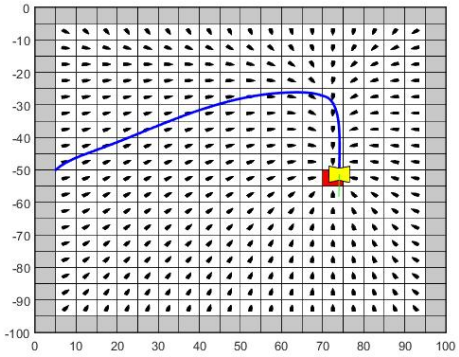

Figura 29 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 1.0$.

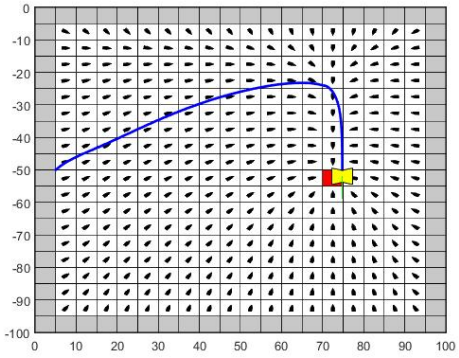

Figura 30 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 1.5$.

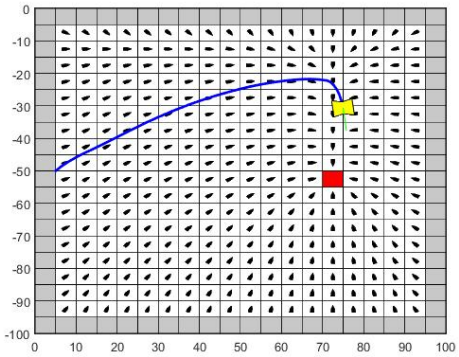

Figura 31 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 2.0$.

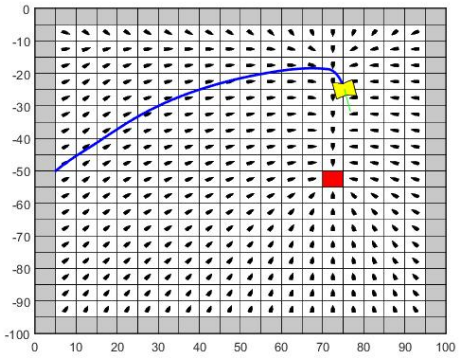

Figura 32 – Estabilidade da malha gerada pelo CPO-GS-UP com $\epsilon = 3.0$.

Quanto maior for ϵ, mais a trajetória tende a convergir para a orientação do vetor v (Figuras 28-32), ou seja, o CPO utilizando Diferenças Up Wind é incondicionalmente estável. Fato este que vai de encontro ao exposto no capítulo anterior.

4.2.2 **Testes com Successive Over-Relaxation**

Esta subseção também divide-se em duas outras partes, pelo mesmo motivo apresentado na subseção anterior.

4.2.2.1 *Testes Utilizando Diferenças Centradas*

Para os três primeiros testes foi utilizado $\epsilon = 1.5$, devido a estabilidade da malha gerada com este valor. Logo, as Figuras 33-35 ilustram os três testes respectivamente.

Como esperado, os resultados dos primeiros testes são iguais aos da subseção anterior. O que também é observado para o teste final com $\epsilon = (0.5, 1.0, 1.5, 2.0)$ e $v\angle 90$, conforme as Figuras 36-39. Para $\epsilon > 2$, a malha além de se tornar instável, não consegue retornar valores numéricos para sua plotagem.

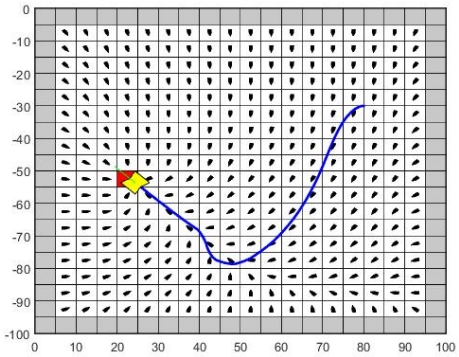

Figura 33 – Eficácia do CPO-SOR-DC para sistemas estáticos e sem obstáculos, com v orientado em 135 graus.

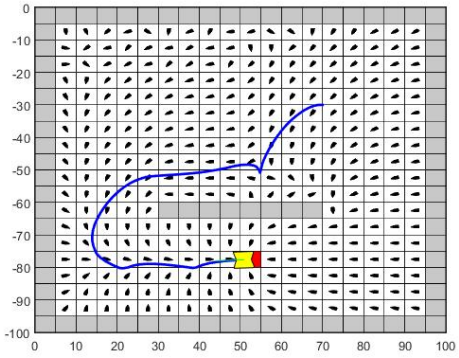

Figura 34 – Eficácia do CPO-SOR-DC para sistemas estáticos e com obstáculos, com v orientado em 0 grau.

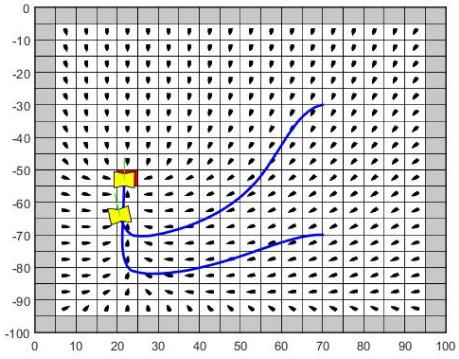

Figura 35 – Concorrência de múltiplos agentes em uma mesma malha gerada pelo CPO-SOR-DC com v orientado em 90 graus.

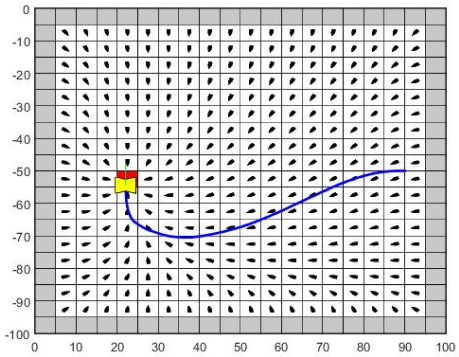

Figura 36 – Estabilidade da malha gerada pelo CPO-SOR-DC com $\epsilon = 0.5$.

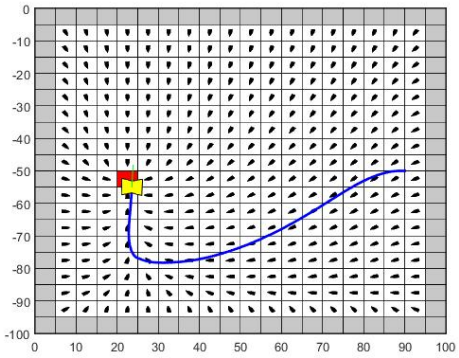

Figura 37 – Estabilidade da malha gerada pelo CPO-SOR-DC com $\epsilon = 1.0$.

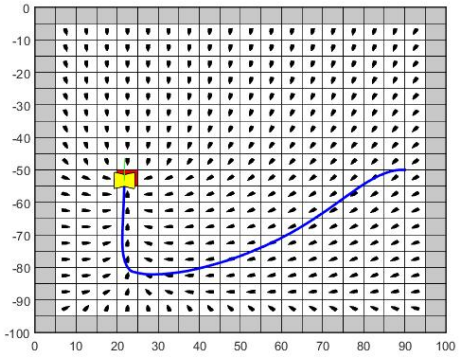

Figura 38 – Estabilidade da malha gerada pelo CPO-SOR-DC com $\epsilon = 1.5$.

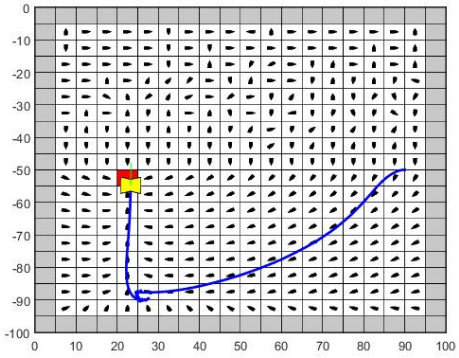

Figura 39 – Estabilidade parcial da malha gerada pelo CPO-SOR-DC com $\epsilon = 2.0$.

4.2.2.2 *Testes Utilizando Diferenças Up Wind*

Como dito anteriormente neste capítulo, aqui não se faz necessário a aplicação dos testes básicos, limitando-se então ao teste com $\epsilon = (0.5, 1.0, 1.5, 2.0, 3.0)$ e $v\angle 90$. Nota-se então pelas Figuras 40-44 que os resultados aqui obtidos se equivalem aos obtidos na subseção anterior.

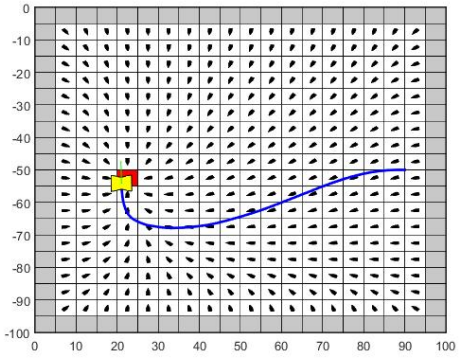

Figura 40 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 0.5$.

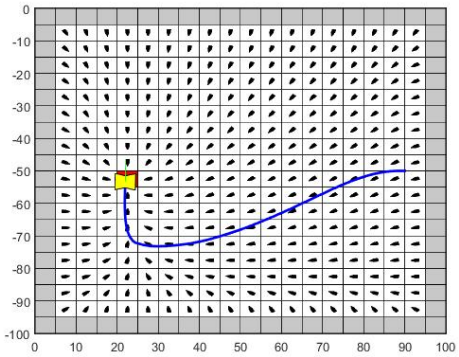

Figura 41 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 1.0$.

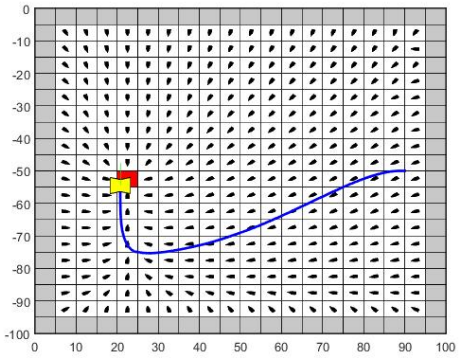

Figura 42 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 1.5$.

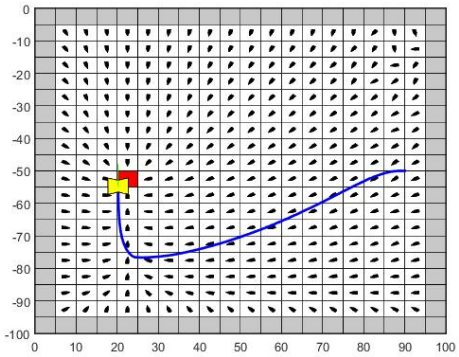

Figura 43 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 2.0$.

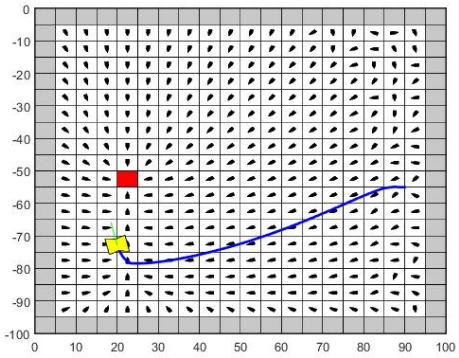

Figura 44 – Estabilidade da malha gerada pelo CPO-SOR-UP com $\epsilon = 3.0$.

4.2.3 **Considerações acerca do Planejamento via CPO**

Como pôde ser visto, para os testes básicos, o CPO apresenta o mesmo resultado que o CPH, ou seja, o planejamento de caminhos é funcional. Assim como o CPH, o CPO também apresenta concorrência pelo objetivo com múltiplos agentes inseridos na malha em um sistema com arquitetura sequencial. A comparação do número de iterações médias entre o CPO-GS e o CPO-SOR, segue a lógica da Tabela 1 mostrada anteriormente.

O teste com variação do valor de ϵ, apresentou resultados compatíveis com o apresentado em [9]. Para os testes utilizando Diferenças Centradas e Up Wind (GS e SOR) foi observado que quanto maior o valor de ϵ, mais a trajetória tende a convergir para a orientação do vetor v. Porém utilizando DC em sistemas com $\epsilon \geq 2$, ocorrem instabilidades na geração da malha, especialmente nos testes em que o método de relaxação utilizado foi o SOR, não sendo possível determinar uma grade com ϵ elevado.

A Tabela 2 a seguir, compara o número médio de iterações demandadas pelo CPO-GS-DC e CPO-SOR-DC para todos os valores de ϵ testados. Seguindo o mesmo critério de convergência da Equação 4.1.

Por mais que o CPO-GS-DC melhore sua velocidade de convergência conforme ϵ

ϵ	N. Iterações GS-DC	N. Iterações SOR-DC
0.5	104	44
1.0	58	67
1.5	36	99
2.0	23	319
3.0	18	–

Tabela 2 – Comparação entre CPO-GS-DC e CPO-SOR-DC

cresce, não pode-se esquecer de que $\epsilon \geq 2$ gera instabilidades na grade. Logo para um sistema dinâmico a melhor solução a nível de velocidade na atualização da malha, com o uso do GPO-GS-DC é utilizar $\epsilon = 1.5$. Enquanto que para o GPO-SOR-DC a melhor solução com relação a velocidade de atualização da malha, é utilizar $\epsilon = 0.5$.

Por apresentarem números de iterações muito próximos, a escolha dentre uma dessas soluções ótimas, ficaria condicionada pelo ambiente, visto que uma impõe ao agente um grande raio de curvatura até a meta e a outra um raio pequeno de curvatura até a meta.

Utilizando UP, os testes mostraram que a malha se torna incondicionalmente convergente, para todos os valores de ϵ testados a grade continuou estável. Pode-se observar em alguns casos, que para valores de $\epsilon \geq 2$ (Figuras 31, 32 e 44) a convergência da trajetória para a orientação do vetor v, pode se tornar mais abrupta, ou seja, não se tem uma curva suave do agente robótico. Para solucionar este problema, pode-se desprezar o movimento regressivo do robô, desta forma estas transições seriam amenizadas. Aumentar o nível do hardware do agente, ou seja, permitir que o mesmo ainda fosse controlável sob grandes requisições de velocidade angular é outra alternativa. Porém estas correções acarretam em desvantagens, na primeira, a desvantagem é não aproveitar completamente a estrutura do robô e perder mobilidade para o caso do futebol de robôs. Na segunda, a desvantagem é que o aumento do nível de hardware, pode acarretar custos elevados ao projeto, tornando-o inviável sob falta de recursos.

A Tabela 3 a seguir, compara o número médio de iterações demandadas pelo CPO-GS-UP e CPO-SOR-UP para todos os valores de ϵ testados. Seguindo o mesmo critério de convergência da Equação 4.1.

ϵ	N. Iterações GS-UP	N. Iterações SOR-UP
0.5	113	42
1.0	76	51
1.5	57	57
2.0	46	64
3.0	33	77

Tabela 3 – Comparação entre CPO-GS-UP e CPO-SOR-UP

Pela tabela nota-se que para o caso da utilização do CPO-GS-UP, a solução ótima é o uso de $\epsilon = 1.5$, por questões de velocidade na atualização da malha e suavidade no movimento do robô. Enquanto que o uso do CPO-SOR-UP, a solução ótima é o uso de $\epsilon = 0.5$. A escolha entre estas duas soluções, fica condicionada pelo ambiente, por razões análogas as escolhas entre o CPO-GS-DC e o CPO-SOR-DC.

Nota-se então que para $\epsilon = 0.5$, o CPO-SOR-DC e o CPO-SOR-UP são equivalentes, ficando a critério do programador a escolha. Enquanto que para $\epsilon = 1.5$, o CPO-GS-DC apresenta ligeira vantagem na velocidade de atualização da grade quando comparado com o CPO-GS-UP.

4.3 PLANEJAMENTO VIA CAMPOS POTENCIAIS LOCALMENTE ORIENTADOS

Nesta seção, os testes serão diferentes, como dito anteriormente, o CPLO atua igual ao CPO para as células na vizinhança do agente robótico e igual ao CPH para células distantes.

Nas Figuras 45-48, podem ser observados os distintos comportamentos do CPLO utilizando as relaxações GS e SOR, assim como as discretizações DC e UP. Os círculos representam a região de vizinhança considerada. Logo observa-se que para as células presentes no interior dos mesmos, houve uma distorção do CPH causada pelo CPO que atua na vizinhança.

Nos testes com relaxação GS, foi utilizado $\epsilon = 1.5$ e para os testes com relaxação SOR, foi utilizado $\epsilon = 0.5$. Todos os testes tinham $v\angle -90$ como ângulo de orientação do robô superior e $v\angle 90$ como ângulo de orientação do robô inferior.

4.3.1 **Considerações acerca do Planejamento via CPLO**

Nota-se que dentro da área de influência do agente robótico, o campo potencial distorce o comportamento padrão do CPH, a malha visa orientar o robô para a trajetória determinada pelo vetor v. Desta forma, o caminho independerá da meta, por isso o CPLO evita a concorrência de múltiplos agentes. Para que haja convergência até o objetivo, o robô deve seguir instruções que o alinhem com a meta.

A Tabela 4, seguindo o critério de convergência da Equação 4.1, compara o número de iterações médias demandadas pelos métodos CPLO-GS-DC, CPLO-SOR-DC, CPLO-GS-UP e CPLO-SOR-UP.

	CPLO-GS-DC	CPLO-SOR-DC	CPLO-GS-UP	CPLO-SOR-UP
N. Iterações	153	43	158	44

Tabela 4 – Comparação entre as possibilidades de implementação do CPLO.

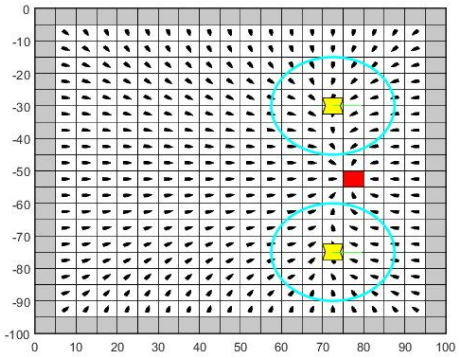

Figura 45 – Comportamento do CPLO-GS-DC.

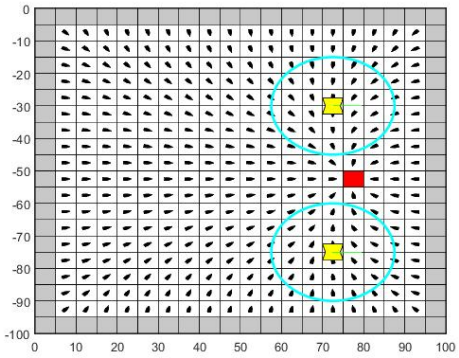

Figura 46 – Comportamento do CPLO-SOR-DC.

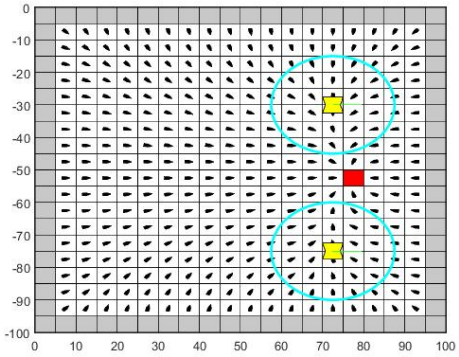

Figura 47 – Comportamento do CPLO-GS-UP.

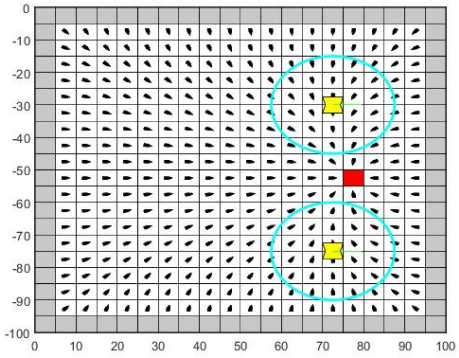

Figura 48 – Comportamento do CPLO-SOR-UP.

Os resultados acima condizem com os mostrados nas Tabelas 1-3, ou seja, são mesclados os resultados para o CPH e o CPO nos resultados do CPLO.

Para sistemas dinâmicos, o uso dos métodos CPLO-SOR-DC e CPLO-SOR-UP são os mais indicados, visto sua velocidade de convergência.

4.4 CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO

As aplicações realizadas neste capítulo, visaram demonstrar a eficiência dos Sistemas de Navegação Baseados em Problemas de Valor de Contorno (Campos Potenciais Harmônicos, Campos Potenciais Orientados e Campos Potenciais Localmente Orientados).

Devido a dificuldade de realizar plotagens gráficas com a linguagem C++, não foi possível demonstrar testes com ambientes dinâmicos, o que de certa forma, prejudicou os testes realizados com Campos Potenciais Localmente Orientados, uma vez que a representação do movimento em ambiente dinâmico poderia ilustrar de forma mais sucinta o comportamento do mesmo.

Os códigos que aparecem nos apêndices deste trabalho, focam em algoritmos não triviais para a concepção computacional dos Sistemas de Navegação. Algumas funções implementadas como o caso dos pares *set* e *get* e conversão entre ambiente contínuo para discreto ou discreto para contínuo, são de formulação pessoal do desenvolvedor, não sendo portanto apresentadas.

A classe *Environment*, foi desenvolvida para que a estrutura do código em C++ seguisse o paradigma de Programação Orientada a Objetos, otimizando o código em termos de escrita, uma vez que diminui o número de comandos executados pelo programador além de aumentar a robustez do programa.

5 CONCLUSÕES

Este trabalho foi motivado pelo estudo de técnicas de navegação de robôs móveis baseadas em PVC, que pudessem ser implementadas pela equipe de futebol de robôs ***Rinobot Team*** da UFJF. Para tal, foram analisados os métodos: *Campos Potenciais Harmônicos* (CPH), *Campos Potenciais Orientados* (CPO) e *Campos Potenciais Localmente Orientados* (CPLO).

Utilizando o método de *Diferenças Finitas*, mais especificamente *Diferenças Centradas* (DC) e *Diferenças Up Wind* (UP), foi possível discretizar as Equações Diferenciais Parciais (EDP) transformando-as então em sistemas de equações lineares. Além disso, algoritmos de relaxação de sistemas lineares como *Jacobi-Richardson* (JR), *Gauss-Seidel* (GS) e *Successive Over-Relaxation* (SOR) foram analisados e dentre eles, os que tinham facilidade de implementação em versão sequencial foram utilizados nos testes realizados para validação dos sistemas de planejamento de caminhos estudados.

Uma proposta de controle de velocidade dos agentes robóticos, com base nas direções obtidas através do gradiente descente do potencial das células da grade, foi analisada e incrementada de tal forma que o robô pudesse ter também movimento regressivo, permitindo então que o robô pudesse otimizar o uso de seus recursos mecânicos e melhorar sua mobilidade.

Os testes mostraram que dentre os métodos de relaxação de Campos Potenciais abordados neste trabalho, o SOR, na maioria dos testes, mostrou-se mais eficiente (mais rápido), conforme o que foi dito no Capítulo 2. Desta forma, para ambientes dinâmicos, como o futebol de robôs, é recomendado relaxação contínua do ambiente pelo SOR com $\epsilon = 0.5$.

Com os testes, ainda foi possível observar que o UP resolve o problema de instabilidade gerado pelo DC quando $\epsilon \geq 2$, confirmando as afirmações feitas no Capítulo 3. Porém para o UP, valores altos de ϵ, podem acarretar em problemas cujas correções demandam um maior custo no projeto ou perda de mobilidade, fatos estes que não são desejáveis.

Por fim, os testes mostraram também que os sistemas CPH e CPO são eficientes quanto ao planejamento de caminhos, porém demandam uma implementação em arquiteturas paralelas quando o objetivo é controlar múltiplos agentes. Para o caso do CPLO, foi mostrado que o mesmo influencia apenas a vizinhança de cada agente, assim, seria possível determinar comportamentos diferentes para cada robô dentro de uma mesma malha, já que o agente seguirá o caminho que o vetor v lhe impor.

5.1 TRABALHOS FUTUROS

Com um sistema de navegação bem estruturado e eficiente, abrem-se brechas para alguns estudos futuros:

- **Comportamentos de jogo:** Definir estratégias deliberativas de ataque, defesa, goleiro e etc.
- **Sistemas baseados em regras para posicionamento dos robôs:** Através de análises do ambiente com técnicas de inteligência artificial, definir o melhor posicionamento para cada agente na malha.
- **Arquitetura de Controle:** Implementar uma arquitetura que seja capaz de gerir os dados enviados pelo ambiente e as ações de estratégia tomadas sobre estes dados.

Todos estes estudos serão direcionados a Rinobot, o objetivo é tornar a UFJF uma referência no cenário nacional e latino-americano nas categorias de futebol de robôs autônomos diferenciais.

REFERÊNCIAS

[1] *Jibo Social Robot*™. Disponível em <https://www.jibo.com>. Acesso em 22 de novembro de 2016.

[2] MACWORTH, A. K. On seeing robots. Em Basu, A. e World, X. L., editors, *Computer Vision: Systems, Theory, and Applications*, pages 1-13. World Scientific Press, Singapore, 1993.

[3] EVANS, L. *Partial Differential Equations*, 2 ed. American Mathematical Society [S.I], 2010.

[4] AMES, W. F. *Numerical Methods for Partial Differential Equations*. Thomas Nelson and Sons Ltd., 1969.

[5] LARSSON, S. and THOMEÉ, V. *Partial Differential Equations with Numerical Methods*. Springer, 2005.

[6] SMITH, G. D. *Numerical Solution of partial differential equations: finite difference methods.* Oxford University, 1992.

[7] GUIDORIZZI, H. L. *Um Curso de Cálculo - Volume 1*. LTC, 5th Edition, 2001.

[8] FARIA, G. *Uma Arquitetura de Controle Inteligente para Múltiplos Robôs*. PhD thesis, ICMC-USP São Carlos, 2006.

[9] DA SILVA, M. O. *Campos Potenciais Modificados Aplicados ao Controle de Múltiplos Robôs*. Master's thesis, ICMC-USP São Carlos, 2011.

[10] BURDEN, R. L. and FAIRES, J. D. *Análise Numérica* . Editora Cengage Learning, 8th edition, 2008.

[11] FRANCO, N. B. *Cálculo Numérico*. Pearson, 2008.

[12] CONNOLLY, C. I. and GRUPEN, R. A. *On the application of harmonic functions to robotics*. Journal of Robotic Systems , 10, 913-946, 1993.

[13] ANDREWS, J. R., and HOGAN, N.*Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator.* In D. E. Hardt and W. J. Book. In AMSE Winter Meeting 1983 , pp. 243-251 , Boston, MA.

[14] KHATIB, O. *Real-time obstacle avoidance for manipulators and mobile robots.* The International Journal of Robotics Research 5 (1): 90–98, 1986.

[15] KROGH, B. H. *A generelazed potential field approach to obstacle avoidance.* In International Robotics Research Conference. Bethlehem, Pennsylvania, 1984.

[16] NEWMAN, W. S. and HOGAN, N. *High speed robot control and obstacle avoidance.* In Proceedings of the 1987 IEEE International (pp. 14-24). Raleigh, North Carolina.

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

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

[19] GOMES, M. R. S. and CAMPOS, M. F. M. *Desvio de obstáculos para micro-robôs móveis utilizando campo potencial.* In Anais do V Simpósio Brasileiro de Automação Inteligente (SBAI), Canela - RS, 2001.

[20] ROMERO, R. A. F, TEIZEN, L. and FARIA, G. *Controle de trajetórias utilizando Campos Potenciais aplicado ao futebol de robôs.* In Anais do II Encontro de Robótica Inteligente, Salvador, 2004.

[21] GILBARG, D. and TRUDINGER, N. S. *Elliptic Partial Differential Equations of Second Order.*Springer-Verlag, 2nd edition, 1983.

[22] CONNOLLY, C. I., BURNS, J. B. and WEISS, R. *Path Planning using laplaces equation.* Em IEEE International Conference on Robotics and Automation, 1990.

[23] PRESTES, E. *Navegação Explanatória Baseada em Problemas de Valores de Contorno.* PhD thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2003.

[24] CORK, P. *Robotics, Vision and Control: Fundamental Algorithms in Matlab.* Springer, 2011.

[25] SIEGWART, R. and NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots.* The MIT Press, Massachusetts, 2004.

APÊNDICE A – Cinemática de Robôs Móveis

As velocidades V e ω (Equações 3.27, 3.28 e 3.30), podem ser divididas em componentes sobre os eixos cartesianos, uma vez que o modelo cinemático no *frame* (conjunto de coordenadas espaciais utilizadas como referência) do robô e seu respectivo ângulo de orientação Φ estejam definidos.

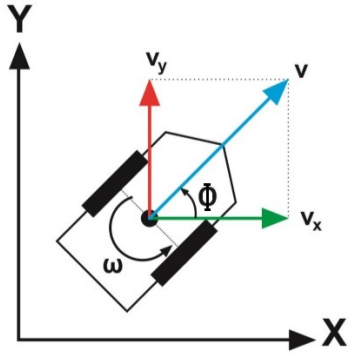

Figura 49 – Componentes das velocidades V e ω sobre os eixos cartesianos.

Assim pode-se definir:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Phi} \end{bmatrix} = \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix}$$

Onde:

$$V_x = V \cdot \cos(\Phi)$$
$$V_y = V \cdot \sin(\Phi)$$

Então, reescrevendo as equações anteriores em modo matricial tem-se:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Phi} \end{bmatrix} = \begin{bmatrix} \cos(\Phi) & 0 \\ \sin(\Phi) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V \\ \omega \end{bmatrix}$$

Obtém-se, portanto, o **modelo cinemático não-linear de velocidades** para o agente robótico, relativo ao referencial global. Os parâmetros de controle do robô (entradas) são V e ω. A partir deste modelo, a posição do agente robótico pode ser estimada ao transformar o resultado matricial acima em infinitesimais de posição, ou seja, basta multiplicar o resultado por uma taxa Δt de amostragem temporal. Mais informações podem ser obtidas em [24] e [25].

APÊNDICE B – Cinemática Implementada em MATLAB®

O algoritmo a seguir, implementado em MATLAB®, é responsável por gerar a cinemática do robô virtual utilizado nos testes do Capítulo 4.

```
function [rPos] = MOVER(rPos,goal,theta)
% Saida da planta:
S = @(th,v,w)([cosd(th),0;sind(th),0;0,1]*[v;w])*0.03;
% Caracteristicas do sistema:
dx = 5; % em cm.
dy = 5; % em cm.
vmax = 80; % em cm/s.
limiarTheta = 90; % em graus.
%Verifica em qual posicao do grid o robo se encontra:
[ir,jr] = convertCToG(rPos,dx,dy,20,20);
%Determina a diferenca entre o angulo de orientacao pedido
%no grid e o angulo de orientacao no qual de encontra o robo:
alpha = theta(ir,jr)-rPos(3); % em graus.
%Aplica a correcao em alpha:
alpha = ajustaAngulo(alpha);
%Verifica se o robo deve realizar movimento progressivo
%ou regressivo:
if abs(alpha) <= limiarTheta
        w = 20*vmax*alpha/180; %em centi-rad/s
        v = -vmax*abs(alpha)/limiarTheta + vmax; %em cm/s.
else
        alpha = ajustaAngulo(alpha+180); % em graus.
        w = 20*vmax*alpha/180;  %em centi-rad/s
        v = vmax*abs(alpha)/limiarTheta - vmax; %em cm/s.
end
%Calculo da Saida - Deslocamento no Ambiente:
Saida = S(rPos(3),v,w);
%Atualizacao da Posicao do Robo:
rPos = rPos + Saida;
end
```

A função *convertCtoG()*, transforma uma posição cartesiana em uma posição na grade. O eixo j da malha, cresce no mesmo sentido do eixo das abscissas, e o eixo i da malha, cresce no sentido oposto do eixo das ordenadas. A função *ajustaAngulo()*, atua conforme as Equações 3.24-3.26.

APÊNDICE C – Classe Environment Implementada em C++

A partir desta classe, se criam as funções e variáveis que são responsáveis pelas aplicações de verificação de eficácia do Capítulo 4.

```
1  #ifndef ENVIRONMENT_H
2  #define ENVIRONMENT_H
3
4  class Environment
5  {
6  public:
7          Environment(int, int); //Construtor da classe
8
9          //iteradores dos Campos Potenciais
10         double iteradorGSCPH();
11         double iteradorGSCPO(int);
12         double iteradorGSCPLO(int);
13
14         //Algoritmo para as condicoes de contorno
15         double getNeighborhood(int, int, int);
16
17         //Dupla set e get para setar e coletar o valor de
               potencial das celulas do grid.
18         void setPotential(int, int, double);
19         double getPotential(int, int);
20
21         //Dupla set e get para setar e coletar o valor das
               direcoes das celulas do grid.
22         void setDirection();
23         double getDirection(int, int);
24
25         //Dupla set e get para setar e coletar uma posicao
               cartesiana dos robos
26         void setPosition(double, double, double = 0.0, int = 0);
27         double getPosition(int, int = 0);
28
29         //Funcoes para transformar coordenadas cartesianas em
               grid e vice-versa.
30         int convertCToG(double[], int);
31         double convertGToC(int, int, int);
32
```

```
33          //Funcao para verificar se uma celula esta livre ou
               ocupada.
34          bool getOccupancy(int, int);
35
36          //Dupla set e get para setar e coletar o numero de robos
               inseridos na malha.
37          void setNumberRobots(int = 1);
38          int getNumberRobots();
39
40          //Dupla set e get para setar e coletar o angulo escolhido
                para orientacao de chegada do robo na meta.
41          void setAngOrientation(double, int = 0);
42          double getAngOrientation(int = 0);
43
44          //Funcoes para inicializar o grid. (utilizar uma delas)
45          void initGrid(double[]);
46          void initGrid2(double[]);
47          void initGrid3(double[]);
48
49          //funcao para printar o grid e gerar o txt carregado no
               matlab
50          void printGrid(int = 0);
51
52  protected:
53          double pGrid[20][20]; //matriz de potenciais
54          double tGrid[20][20]; //matriz de direcoes
55          const double dX; //tamanho da discretizacao em x
56          const double dY; //tamanho da discretizacao em y
57          double rPos[6][3]; //salva as coordenadas de posicao de
               cada robo
58          int number; //salva o numero de robos
59          double vecAng[6]; //salva os angulos de orientacao do
               grid de cada robo
60  };
61  #endif
```

APÊNDICE D – Iterador do CPH Implementado em C++

O algoritmo a seguir, implementado em C++, é responsável por gerar o iterador que calcula os potenciais das células segundo os métodos CPH-GS e CPH-SOR.

```
1  double Environment::iteradorGSCPH()
2  {
3  int j = 0,i;
4  double oldPotential, newPotential, error = 0;
5  double top,bottom,left,right; //vizinhos da celula i,j
6
7  while (j<20)
8  {
9          i = 0;
10         while (i<20)
11         {
12                 //Verifica se a celula esta livre
13                 if (getOccupancy(i, j))
14                 {    //Salva o potencial antigo
15                         oldPotential = getPotential(i, j);
16                         //Atualiza as condicoes de contorno
17                         top = getNeighborhood(i, j, 0);
18                         bottom = getNeighborhood(i, j, 1);
19                         left = getNeighborhood(i, j, 2);
20                         right = getNeighborhood(i, j, 3);
21                         //Equacao CPH-GS
22                         newPotential = (top+bottom+left+right)/4;
23                         //Equacao CPH-SOR
24                         //newPotential = newPotential+0.8*(
                              newPotential-oldPotential);
25                         //calculo do erro segundo Equacao 4.1
26                         error += pow((newPotential-oldPotential)
                              ,2);
27                         //Atualiza o potencial
28                         setPotential(i, j, newPotential);
29                 }
30                 i += 1;
31         }
32         j += 1;
33 }
34 return error;
35 }
```

A classe *Environment*, detém todos os iteradores, desta forma, na rotina *main()*, deve ser criado um objeto *env* proveniente da classe *Environment*, para utilizar o iterador. Para que o iterador mude de CPH-GS para CPH-SOR, basta descomentar a linha referente à transformação.

APÊNDICE E – Iterador do CPO Implementado em C++

O algoritmo a seguir, implementado em C++, é responsável por gerar o iterador que calcula os potenciais das células segundo os métodos CPO-GS-DC, CPO-SOR-DC, CPO-GS-UP e CPO-SOR-UP.

```
1  double Environment::iteradorGSCPO(int aux)
2  {
3  int j = 0, i;
4  double oldPotential, newPotential, error = 0;
5  double top, bottom, left, right; //vizinhos da celula i,j
6  double vec[2], e, h, lambda; //parametros do CPO
7  double ro[2],den; //parametros CPO-UP
8  //Vetor de orientacao do CPO
9  vec[0] = cos(getAngOrientation() / 180 * 3.1415);
10 vec[1] = sin(getAngOrientation() / 180 * 3.1415);
11 //Taxas de influencia do CPO
12 h = dX / dY;
13 e = 1.5;
14 lambda = e * h / 2;
15 //denominador do CPO-UP
16 ro[0] = 1 + e * abs(vec[0]);
17 ro[1] = 1 + e * abs(vec[1]);
18 den = 2 + ro[0] + ro[1];
19 //Verifica se sera DC ou UP
20 if (aux == 0)
21 {
22         while (j<20)
23         {
24                 i = 0;
25                 while (i<20)
26                 {
27                     //Verifica se a celula esta livre
28                         if (getOccupancy(i, j))
29                         {
30                                 //Salva o potencial antigo
31                                 oldPotential = getPotential(i, j)
                                       ;
32                                 //Atualiza as condicoes de
                                       contorno
33                                 top = getNeighborhood(i, j, 0);
```

```
34                                      bottom = getNeighborhood(i, j, 1)
                                           ;
35                                      left = getNeighborhood(i, j, 2);
36                                      right = getNeighborhood(i, j, 3);
37                                      //Equacao CPO-GS-DC
38                                      newPotential = ((1+lambda*vec[0])
                                           *right+(1-lambda*vec[0])*left
                                           +(1+lambda*vec[1])*top+(1-
                                           lambda*vec[1])*bottom)/4;
39                                      //Equacao CPO-SOR-DC
40                                      //newPotential = newPotential
                                           +0.8*(newPotential-oldPotential
                                           );
41                                      //calculo do erro segundo Equacao
                                            4.1
42                                      error += pow((newPotential-
                                           oldPotential),2);
43                                      //Atualiza o potencial
44                                      setPotential(i, j, newPotential);
45                              }
46                              i += 1;
47                      }
48                      j += 1;
49              }
50      }
51      else
52      {
53              while (j<20)
54              {
55                      i = 0;
56                      while (i<20)
57                      {
58                              //Verifica se a celula esta livre
59                              if (getOccupancy(i, j))
60                              {
61                                      //Salva o potencial antigo
62                                      oldPotential = getPotential(i, j)
                                           ;
63                                      //Atualiza as condicoes de
                                           contorno
64                                      top = getNeighborhood(i, j, 0);
```

```
65                  bottom = getNeighborhood(i, j, 1)
                        ;
66                  left = getNeighborhood(i, j, 2);
67                  right = getNeighborhood(i, j, 3);
68                  //Verifica as condicoes do UP -
                        Equacoes CPO-GS-UP
69                  if (vec[1] > 0)
70                  {
71                          if (vec[0] > 0)
72                                  newPotential = (
                                        ro[0]*right+
                                        left+ro[1]*top+
                                        bottom)/den;
73                          else
74                                  newPotential = (
                                        right+ro[0]*
                                        left+ro[1]*top+
                                        bottom)/den;
75                  }
76                  else
77                  {
78                          if (vec[0] > 0)
79                                  newPotential = (
                                        ro[0]*right+
                                        left+top+ro[1]*
                                        bottom)/den;
80                          else
81                                  newPotential = (
                                        right+ro[0]*
                                        left+top+ro[1]*
                                        bottom)/den;
82                  }
83                  //Equacao CPO-SOR-UP
84                  //newPotential = newPotential
                        +0.8*(newPotential-oldPotential
                        );
85                  //calculo do erro segundo Equacao
                         4.1
86                  error += pow((newPotential-
                        oldPotential),2);
87                  //Atualiza o potencial
88                  setPotential(i, j, newPotential);
```

```
89                              }
90                              i += 1;
91                      }
92                      j += 1;
93          }
94  }
95  return error;
96  }
```

Para que o iterador atue com as relaxações SOR, basta descomentar as linhas referentes às transformações. Para que o método atue com discretizações DC, *aux* deve ser igual a zero, caso contrário o iterador atua com discretizações UP.

APÊNDICE F – Iterador do CPLO Implementado em C++

O algoritmo a seguir, implementado em C++, é responsável por gerar o iterador que calcula os potenciais das células segundo os métodos CPLO-GS-DC, CPLO-SOR-DC, CPLO-GS-UP e CPLO-SOR-UP.

```
1  double Environment::iteradorGSCPLO(int aux)
2  {
3  int j = 0, i;
4  double oldPotential, newPotential, error = 0;
5  double top, bottom, left, right;//vizinhos da celula i,j
6  double vec[2], e, h, lambda;//parametros do CPLO
7  double ro[2], den;//parametros CPLO-UP
8  //Parametros exclusivos do CPLO
9  double k, kLessDist, lessDist, dist;
10 double posRobot[2];
11 int iRobot, jRobot;
12 ////Taxas de influencia do CPLO
13 h = dX / dY;
14 e = 0.5;
15 lambda = e * h / 2;
16 //Verifica se sera DC ou UP
17 if (aux == 0)
18 {
19         while (j<20)
20         {
21                 i = 0;
22                 while (i<20)
23                 {
24                         //Verifica se a celula esta livre
25                         if (getOccupancy(i, j))
26                         {
27                                 //Salva o potencial antigo
28                                 oldPotential = getPotential(i, j)
                                       ;
29                                 //Atualiza as condicoes de
                                       contorno
30                                 top = getNeighborhood(i, j, 0);
31                                 bottom = getNeighborhood(i, j, 1)
                                       ;
32                                 left = getNeighborhood(i, j, 2);
33                                 right = getNeighborhood(i, j, 3);
```

```
34                  //encontra o robo k, dentr todos
                       do grid, com a menor distancia
                       entre ele e a celula i,j
35                  k = 0;
36                  lessDist = 20000;
37                  kLessDist = -1;
38                  while (k < getNumberRobots())
39                  {
40                          posRobot[0] = getPosition
                               (k,0);
41                          posRobot[1] = getPosition
                               (k,1);
42                          iRobot = convertCToG(
                               posRobot, 0);
43                          jRobot = convertCToG(
                               posRobot, 1);
44                          dist = pow(pow((jRobot-j)
                               ,2)+pow((iRobot-i),2)
                               ,0.5);
45                          if (dist < lessDist)
46                          {
47                                  lessDist = dist;
48                                  kLessDist = k;
49                          }
50                          k += 1;
51                  }
52                  //Verifica se o robo k pertence a
                        vizinhanca de i,j
53                  if (lessDist <= 3)
54                  {
55                          //Vetor de orientacao do
                               CPLO
56                          vec[0] = cos(
                               getAngOrientation(
                               kLessDist)/180*3.1415);
57                          vec[1] = sin(
                               getAngOrientation(
                               kLessDist)/180*3.1415);
58                          //Equacao CPLO-GS-DC
59                          newPotential = ((1 +
                               lambda*vec[0])*right
                               +(1-lambda*vec[0])*left
```

```
                                                +(1+lambda*vec[1])*top
                                                +(1-lambda*vec[1])*
                                                bottom)/4;
60                                      }
61                                      else
62                                      {
63                                              //Equacao CPH-GS
64                                              newPotential = (right+
                                                left+top+bottom)/4;
65                                      }
66                                      //Equacao CPLO-SOR-DC
67                                      //newPotential = newPotential
                                        +0.8*(newPotential-oldPotential
                                        );
68                                      //calculo do erro segundo Equacao
                                         4.1
69                                      error += pow((newPotential-
                                        oldPotential),2);
70                                      //Atualiza potencial
71                                      setPotential(i, j, newPotential);
72                              }
73                              i += 1;
74                      }
75                      j += 1;
76              }
77      }
78      else
79      {
80              while (j<20)
81              {
82                      i = 0;
83                      while (i<20)
84                      {
85                              //Verifica se a celula esta livre
86                              if (getOccupancy(i, j))
87                              {
88                                      //Salva o potencial antigo
89                                      oldPotential = getPotential(i, j)
                                        ;
90                                      //Atualiza as condicoes de
                                        contorno
91                                      top = getNeighborhood(i, j, 0);
```

```
92                  bottom = getNeighborhood(i, j, 1)
                        ;
93                  left = getNeighborhood(i, j, 2);
94                  right = getNeighborhood(i, j, 3);
95              //encontra o robo k, dentr todos do
                    grid, com a menor distancia entre
                    ele e a celula i,j
96                  k = 0;
97                  lessDist = 20000;
98                  kLessDist = -1;
99                  while (k < getNumberRobots())
100                 {
101                         posRobot[0] = getPosition
                                (k,0);
102                         posRobot[1] = getPosition
                                (k,1);
103                         iRobot = convertCToG(
                                posRobot, 0);
104                         jRobot = convertCToG(
                                posRobot, 1);
105                         dist = pow(pow((jRobot-j)
                                ,2)+pow((iRobot-i),2)
                                ,0.5);
106                         if (dist < lessDist)
107                         {
108                                 lessDist = dist;
109                                 kLessDist = k;
110                         }
111                         k += 1;
112                 }
113                 //Verifica se o robo k pertence a
                        vizinhanca de i,j
114                 if (lessDist <= 3)
115                 {
116                         //Vetor de orientacao do
                                CPLO
117                         vec[0] = cos(
                                getAngOrientation(
                                kLessDist)/180*3.1415);
118                         vec[1] = sin(
                                getAngOrientation(
                                kLessDist)/180*3.1415);
```

```
119                     //denominador do CPLO-UP
120                     ro[0] = 1+e*abs(vec[0]);
121                     ro[1] = 1+e*abs(vec[1]);
122                     den = 2+ro[0]+ro[1];
123                     //Verifica as condicoes
                           do UP - Equacoes CPLO-
                           GS-UP
124                     if (vec[1] > 0)
125                     {
126                             if (vec[0] > 0)
127                                     newPotential
                                           = (ro
                                           [0]*
                                           right+
                                           left+ro
                                           [1]*top
                                           +bottom
                                           )/den;
128                             else
129                                     newPotential
                                           = (
                                           right+
                                           ro[0]*
                                           left+ro
                                           [1]*top
                                           +bottom
                                           )/den;
130                     }
131                     else
132                     {
133                             if (vec[0] > 0)
134                                     newPotential
                                           = (ro
                                           [0]*
                                           right+
                                           left+
                                           top+ro
                                           [1]*
                                           bottom)
                                           /den;
135                             else
```

```
136                                                                 newPotential
                                                                        = (
                                                                    right+
                                                                    ro[0]*
                                                                    left+
                                                                    top+ro
                                                                    [1]*
                                                                    bottom)
                                                                    /den;
137                                                         }
138                                                 }
139                                                 else
140                                                 {
141                                                         //Equacao CPH-GS
142                                                         newPotential = (right+
                                                               left+top+bottom)/4;
143                                                 }
144                                                 //Equacao CPLO-SOR-UP
145                                                 //newPotential = newPotential
                                                       +0.8*(newPotential-oldPotential
                                                       );
146                                                 //calculo do erro segundo Equacao
                                                        4.1
147                                                 error += pow((newPotential-
                                                       oldPotential),2);
148                                                 //Atualiza potencial
149                                                 setPotential(i, j, newPotential);
150                                 }
151                                 i += 1;
152                         }
153                         j += 1;
154                 }
155 }
156 return error;
157 }
```

O tratamento é análogo ao iterador do CPO. Assim como no algoritmo implementado em MATLAB®, a função *convertCToG()*, transforma uma posição cartesiana em uma posição na grade.

APÊNDICE G – Vizinhança de Manhattam Implementado em C++

O algoritmo a seguir, implementado em C++, é responsável por calcular as condições de contorno dos PVC's estudados, levando-se em consideração as condições de Dirichlet ([21] e [6]).

```
1  double Environment::getNeighborhood(int i, int j, int aux)
2  {
3  double neighbor[4]; //vetor dos vizinhos top, bottom, left e
      right
4
5  if (i == 0)//verifica se a celula esta na primeira linha do grid
6  {
7          neighbor[0] = 1;//neste caso, top recebe maxP
8          neighbor[1] = getPotential(i + 1, j);//bottom recebe o
              valor normal
9  }
10 else if (i + 1 == 20)//verifica se a celula esta na ultima linha
       do grid
11 {
12         neighbor[0] = getPotential(i - 1, j);//top recebe o
              valor normal
13         neighbor[1] = 1; //neste caso, bottom recebe maxP
14 }
15 else
16 {
17         neighbor[0] = getPotential(i - 1, j);//top recebe o
              valor normal
18         neighbor[1] = getPotential(i + 1, j);//bottom recebe o
              valor normal
19 }
20 if (j == 0)//verifica se a celula esta na primeira coluna do
      grid
21 {
22         neighbor[2] = 1;//neste caso, left recebe maxP
23         neighbor[3] = getPotential(i, j + 1);//right recebe o
              valor normal
24 }
25 else if (j + 1 == 20)//verifica se a celula esta na ultima
      coluna do grid
26 {
```

```
27          neighbor[2] = getPotential(i, j - 1);//left recebe o
               valor normal
28          neighbor[3] = 1;//neste caso, right recebe maxP
29  }
30  else
31  {
32          neighbor[2] = getPotential(i, j - 1);//left recebe o
               valor normal
33          neighbor[3] = getPotential(i, j + 1);//right recebe o
               valor normal
34  }
35  return neighbor[aux];//retorna o valor do vizinho escolhido
36  }
```