

**UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
INSTITUTO DE INFORMÁTICA**

LEON EZEQUIEL DEICU

**PLANEJADORES DE CAMINHO BASEADOS EM CAMPOS POTENCIAIS**

Prof. Dr. Dante Barone  
Orientador

Porto Alegre, 2012

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. Carlos Alexandre Netto

Vice-Reitor: Prof. Rui Vicente Oppermann

Pró-Reitora de Graduação: Profa. Valquiria Link Bassani

Diretor do Instituto de Informática: Prof. Flávio Rech Wagner

Coordenador do ECP: Prof. Sérgio Cechin

Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro.

## **AGRADECIMENTOS**

Primeiramente gostaria de agradecer a meus pais, que desde pequeno me incentivaram ao estudo e me deram todas as chances possíveis que eu necessitava para crescer. Eles me apoiaram em todas as minhas decisões e me ajudaram sempre que necessário.

Gostaria de agradecer também a Universidade e aos professores pelo estudo e tempo. Gostaria de agradecer especialmente o professor Dante Barone, pois sem ele este trabalho não teria sido concluído. Ele foi mais do que um professor em um tempo que eu realmente necessitei.

Gostaria de agradecer meus colegas e amigos pelos momentos divertidos e descontraídos que tivemos durante estes anos. Os grupos de estudo, as horas do dia e as noites viradas fazendo trabalhos que pareciam ficar cada vez mais difíceis. Aos encontros na biblioteca para estudar para provas e principalmente pelas saídas nos finais de semana para descontrair.

Gostaria de agradecer finalmente minha namorada pelo suporte, apoio, amizade e companhia. Pelo fato de ela sempre estar lá para me ajudar, pelo fato de ser minha confidente, pelo fato de ter me aguentado todo esse tempo.

# SUMÁRIO

AGRADECIMENTOS .....	3
SUMÁRIO .....	4
LISTA DE ABREVIATURAS E SIGLAS .....	6
LISTA DE FIGURAS .....	7
LISTA DE TABELAS .....	9
RESUMO .....	10
ABSTRACT .....	11
1 INTRODUÇÃO .....	12
2 PLANEJADORES DE CAMINHO .....	15
2.1 Representação E Decomposição De Ambientes .....	16
2.2 Planejadores De Caminho .....	18
2.2.1 Roadmaps .....	19
2.2.2 Cell Decomposition .....	22
2.2.3 Métodos de Planejamento Discreto .....	24
2.2.4 Campos Potenciais .....	29
3 CAMPOS POTENCIAIS HARMONICOS .....	32
3.1 Espaço de configuração e Grids .....	32
3.2 Funções Harmônicas .....	33
3.2.1 Propriedades das funções harmônicas .....	34
3.3 Resolução numérica .....	34
3.3.1 Expansão de Taylor .....	36
3.3.2 Equação de Laplace discreta .....	39
3.3.3 Métodos Iterativos .....	40
3.4 Planejador BVP .....	42
4 IMPLEMENTAÇÃO E RESULTADOS .....	44
4.1 Leitura e interpretação do Mapa .....	44
4.2 Algoritmos .....	49
4.2.1 Harmonic Functions Path Planner Iterado com Gauss-Seidel simples .....	51

4.2.2	Harmonic Functions Path Planner Iterado com Gauss-Seidel simultâneo	52
4.2.3	Harmonic Functions Path Planner Iterado com SOR simples .....	53
4.2.4	Harmonic Functions Path Planner Iterado com SOR simultâneo .....	54
4.2.5	BVP iterado com GS simples .....	55
4.2.6	BVP iterado com GS simultâneo .....	56
4.2.7	BVP iterado com SOR simples.....	57
4.2.8	BVP iterado com SOR simultâneo .....	58
4.2.9	BVP alterado iterado com GS simples .....	60
4.2.10	BVP alterado iterado com GS simultâneo.....	61
4.2.11	BVP alterado iterado com SOR simples .....	62
4.2.12	BVP alterado iterado com SOR simultâneo .....	63
4.3	Resultados .....	64
	Resultados no Mapa 1 .....	66
	Resultados no Mapa 2.....	67
	Resultados no Mapa 3.....	68
4.3.1	Convergência de dados.....	69
5	CONCLUSÃO .....	71
	Bibliografia .....	75

## **LISTA DE ABREVIATURAS E SIGLAS**

IA	Inteligência Artificial
SOR	Successive Over-Relaxation
BVP	Boundary Value Problem
PDE	Partial Differential Equation - Equação Diferencial Parcial
GS	Gaus-Seidel
HFPP	Harmonic Functions Path Planner

## LISTA DE FIGURAS

Figura 1 - Mapa do ambiente.....	17
Figura 2 - Representação do ambiente em uma Occupacy Grid.....	17
Figura 3 - Representação do ambiente em Mapa de Linhas.....	17
Figura 4 - Representação do ambiente em Mapa Topológico.....	17
Figura 5 - Occupacy Grid com escala 1 para 10cm <sup>2</sup> .....	18
Figura 6- Occupacy Grid com escala 1 para 4cm <sup>2</sup> .....	18
Figura 7 - Diagrama de Visibilidade.....	20
Figura 8 - Soluções do Diagrama de Voronoi.....	21
Figura 9 - Soluções escolhida pelo algoritmo.....	21
Figura 10 - Exemplo de Métodos baseados em Amostragem.....	22
Figura 11 - Vertical Decomposition.....	23
Figura 12 - Vertical Decomposition.....	23
Figura 13 - Triangular Cell Decomposition.....	24
Figura 14 - Circular Decomposition.....	24
Figura 15 - Breath First.....	27
Figura 16 - Depth First.....	28
Figura 17 - Campos potenciais.....	30
Figura 18 - Caminho escolhido pelo robô.....	30
Figura 19 - Grid mapeada.....	33
Figura 20 - Grid Estruturada.....	35
Figura 21 - Grid de Escoamento VLS.....	35
Figura 22 - Definição de $\Delta x$ .....	36
Figura 23- Aproximação de 1° ordem, via diferenças progressivas, para $f'$ .....	37
Figura 24- Aproximação de 1° ordem, via diferenças atrasadas, para $f'$ .....	37
Figura 25- Aproximação de 2° ordem, via diferenças centrais, para $f'$ .....	38
Figura 26 - Sistema Linear.....	40
Figura 27 - Software resultado.....	44
Figura 28 - Mapa 1 escala 100x100.....	46
Figura 29 - Mapa 2 escala 250x250.....	46
Figura 30 - Mapa 3 escala 500x500.....	47
Figura 31 - Mapas com Preferências.....	48
Figura 32 - Mapa 1.....	49
Figura 33 - Mapa Puro.....	50
Figura 34 - Mapa Calculado sem Preferencias.....	50
Figura 35 - Mapa Calculado com Preferência.....	51
Figura 36 - Ambiente usando Mapa 1.....	66
Figura 37 - Ambiente usando Mapa 2.....	67
Figura 38 - Ambiente usando Mapa 3.....	68
Figura 39 - Gráfico do erro no Mapa 1.....	69
Figura 40 - Gráfico do erro no Mapa 1 na 20° iteração.....	70

Figura 41 - Gráfico do erro no Mapa 1 na 40ª iteração .....	70
Figura 42 - Ponto da 1ª Convergencia.....	<b>Erro! Indicador não definido.</b>



## LISTA DE TABELAS

Tabela 1 – Mapas em formato de Linha .....	45
Tabela 2 - Arquivos de Preferência .....	48
Tabela 3- Mapa com preferencias.....	50
Tabela 4 - Algoritmo GS simples .....	51
Tabela 5 - Algoritmo GS simultâneo .....	52
Tabela 6 - Algoritmo SOR simples.....	53
Tabela 7 - Algoritmo SOR simultaneo .....	54
Tabela 8 - BVP usando GS simples .....	55
Tabela 9 - BVP com GS simultâneo .....	56
Tabela 10 - BVP com SOR simples.....	58
Tabela 11 - BVP com SOR simultâneo .....	59
Tabela 12 - BVP 2 com GS simples .....	60
Tabela 13 - BVP 2 com GS simultâneo .....	61
Tabela 14 - BVP 2 com SOR simples.....	62
Tabela 15 - BVP 2 com SOR simultâneo .....	63
Tabela 16 - Resultados.....	65
Tabela 17 - Resultados do mapa 1 .....	66
Tabela 18 - Resultados no Mapa 2.....	67
Tabela 19 - Resultados no Mapa 3.....	68

## **RESUMO**

Este trabalho tem por objetivo o estudo e a aplicação de métodos de planejamentos de caminhos baseados em funções harmônicas e melhorias propostas para os mesmos em diversos ambientes. Outro aspecto do estudo são as modificações propostas por Edson Prestes as quais adicionam uma ferramenta para a resolução do “weight problem”, criando áreas de baixa e alta preferência. Também neste estudo verificam-se o tempo de convergência dos métodos e como eles convergem.

## **ABSTRACT**

This article has for objective the study and application of path planning algorithms based in harmonic functions and improvements proposed by Edson Prestes. These improvements add tools to resolve the “weight problem” by adding low preference and high preference areas. Also in the study we verify how and how much time it takes to converge.

# 1 INTRODUÇÃO

Os robôs estão invadindo a sociedade moderna; há robôs no espaço e no fundo dos oceanos explorando lugares aos quais humanos não teriam acesso, há robôs em hospitais ajudando no tratamento de pessoas, há robôs nas fábricas construindo manufaturados de melhor qualidade, há robôs nas fazendas ajudando no plantio e na colheita de produtos agrícolas e há robôs nos lares ajudando com as tarefas do dia-a-dia entre tantas outras aplicações. No futuro espera-se que os robôs tenham um papel mais significativo no cotidiano, desejando-se que eles sejam mais inteligentes e autônomos de modo a realizar um número ainda maior de tarefas. Toma-se por verdade que robôs como Rosie, personagem do desenho animado “Os Jetsons”, capaz de atender toda a rotina de tarefas domésticas, serão parte do nosso dia-a-dia como são os carros e os celulares.

O desejo de construir tais máquinas é comprovado ao vermos diversos livros, filmes, séries e manifestações culturais sobre o assunto. Porém esse desejo é mais antigo do que muitos acreditam, pois no ano de 1200 o engenheiro Al-Jazari construiu alguns dos primeiros robôs humanóides programáveis, um dos quais era capaz de servir água, chá ou qualquer outra bebida através de um complexo sistema hidráulico (1). Al-Jazari também criou robôs capazes de tocar instrumentos musicais como tambores e trombetas (2) novamente utilizando sistemas hidráulicos. A própria palavra “robô” vem de “Robota” utilizada pela primeira vez em uma peça teatral de 1920 na república Checa, chamada “Rossum’s Universal Robots” escrita por Karel Capek e significa trabalhador subordinado, escravo (3). Outro exemplo de literatura amplamente conhecido sobre robótica é o livro de Isaac Asimov de 1940, “I Robot”, onde o autor discute pela primeira vez a interação entre humanos e robôs e as leis e condutas que regem tal interação estabelecendo as conhecidas três leis da robótica<sup>1</sup>. Desde esta época surgiram milhares de outros exemplos de livros, peças e filmes sobre o tema.

Os primeiros robôs industriais foram concebidos nos anos 1960 e se tratavam de robôs manipuladores, também conhecidos como braços robóticos, que replicavam o movimento de braços humanos. Tais robôs eram usados na manufatura de produtos que exigiam precisão e na manipulação de material radioativo. Com o avanço dos circuitos integrados e de técnicas computacionais, nos anos 70-80, os robôs tornaram-se programáveis e capazes de serem controlados por sistemas de computadores. Hoje em dia, podemos ver robôs fora das fábricas e entrando no cotidiano da população. A empresa iRobot possui uma série de robôs para o uso doméstico, o robô comercial Romba promete substituir o aspirador de pó, o robô Scooba promete lavar os pisos, o

---

<sup>1</sup> As leis da robótica criadas por Asimov na obra I Robot são: 1) um robô não pode ferir um ser humano ou, por omissão, permitir que um ser humano sofra alguma mal; 2) um robô deve cumprir as ordens recebidas a não ser que tais ordens contrariem a Primeira Lei; 3) um robô deve proteger a própria existência desde que tal proteção não entre em conflito com a Primeira e/ou a Segunda Lei (I Robot, Isaac Asimov).

robô erro lavar a piscina e o robô Looj promete limpar as calhas (4). Outro uso de robôs atualmente é na segurança do lar, como o robô Roborior da empresa japonesa Sanyo cujo objetivo é detectar invasores ou incêndios na casa. Já na área médica podemos citar o Keepon cujo objetivo é ajudar crianças com autismo. Uma grande quantidade de investimento está sendo direcionada a robôs para o auxílio a cirurgias como o robô “neuroArm” que é usado para realizar complexas cirurgias no cérebro (5), o robô “da Vinci robot” que é utilizado para realizar operações complexas no coração (6), entre outros.

Podemos notar o avanço na robótica e a diversificação de seus objetivos ao observarmos as diferentes definições de robô através dos anos. Vemos claramente a troca de paradigma nas definições, pois os robôs deixam de ser apenas braços manipuladores para se tornarem agentes inteligentes em um ambiente. As definições a seguir foram coletadas de diversas fontes e seguem em ordem cronológica:

1. Qualquer dispositivo que substitua trabalho humano. (Soska, Japan, 1985)
2. Um robô é uma máquina que pode ser programada para uma variedade de operações, da mesma forma que um computador é um circuito elétrico que pode ser programado para realizar uma variedade de operações. (McKerrow, 1986)
3. Um manipulador multifuncional desenhado para mover objetos, materiais e partes através de movimentos programados para realizar uma variedade de tarefas. (Schulussel, 1985 – Robotics Institute of América)
4. Uma máquina multi-funcional controlada por computador cujo principal objetivo é manipular o ambiente externo. (Crabbe, 2002)
5. “É uma máquina capaz de sentir, atuar e reagir no mundo e possivelmente envolvendo algum grau de raciocínio para tal ação, e o faz de maneira autônoma. Por esta definição um termostato seria um robô apesar de não estar ciente de seu objetivo, tal ciência não é necessária”. (Alan Mackworth, diretor do laboratório de inteligência computacional da University of British Columbia Laboratory)

O avanço atual dos robôs se deve em muito ao avanço nas técnicas de inteligência artificial e ao aumento do poder computacional. A IA, Inteligência Artificial, permite ao robô tornar-se capaz de tomar decisões e desta forma tornar-se mais autônomo. Tomar decisões avaliando o ambiente que cerca o robô é um dos maiores desafios a qual a robótica se dedica e envolve muitas áreas. Podemos citar sensores e como avaliá-los, mapeamento de ambiente e a localização do robô no ambiente, e planejamento de ações a serem tomadas. Dentro deste enfoque, podemos citar planejadores de caminho, cujo objetivo é traçar uma trajetória entre a posição atual do robô e uma posição objetivo final sem colidir com os obstáculos presentes no ambiente.

Existem vários métodos para resolver este problema como os algoritmos baseados em vetores, campos potenciais, seguidores de paredes, entre outros. Dentre

eles podem-se destacar os campos potenciais propostos por Khatib (7) em 1985, que apesar de terem sido propostos para aumentar a capacidade de controle em robôs a fim de evitar colisões, criando um campo potencial falso para repelir o robô com uma intensidade proporcional à distância entre ele e o obstáculo, evoluiu de modo a criar diversos outros algoritmos mais refinados para resolver problemas mais complexos como o planejamento de caminhos. Uma destas evoluções é a aplicação de funções harmônicas para traçar a trajetória proposta por Connolly (8) e Akishita (9).

## 2 PLANEJADORES DE CAMINHO

Planejamento é um termo com diferentes conotações para grupos distintos. Para a robótica, algoritmos de planejamento são aqueles que decompõem um comando de alto nível em uma série de comandos em baixo nível. Um exemplo muito utilizado é o “Piano Mover’s Problem”, onde dada a planta de uma casa e um piano com dimensões definidas, é necessário descrever todos os movimentos e rotações necessários para mover o piano de uma sala à outra sem colidir com nenhum objeto ou parede. Sendo assim, podemos resumir planejamento de caminhos como:

*Dado um subconjunto  $U$  em um cenário  $n$ -dimensional e dois sub-conjuntos compactos  $C0$  e  $C1$  pertencentes a  $U$ , onde  $C1$  é derivado de  $C0$  por movimentos contínuos, quais os movimentos necessários para mover-se de  $C0$  para  $C1$  sem sair de  $U$  (10).*

Pode-se citar também a conotação de planejamento pelo ponto de vista da inteligência artificial, a qual vê planejamento como o ato de resolver um problema a partir de uma sequência finita de ações discretas. Por esta conotação a resolução de um cubo mágico é um problema de planejamento, pois se pode executar um conjunto finito de ações cujas consequências são conhecidas com o objetivo de chegar a um estado final conhecido (11).

Podemos classificar os métodos de planejamento em três grandes categorias (12):

1. Métodos Globais: Quando o robô tem conhecimento pleno do ambiente que o cerca. Nestes casos podemos aplicar métodos como Roadmaps, decomposição em células e campos potenciais.
2. Métodos Locais: O robô não possui conhecimento sobre o ambiente ou sua localização, dependendo inteiramente de seus sensores para alcançar seu objetivo. Podemos citar como exemplos algoritmos WallFollow, campos potenciais de Khatib (13) .
3. Métodos Híbridos: é a combinação dos métodos locais e globais, onde o robô tem um conhecimento prévio do ambiente, porém este não é completo e torna-se inútil, muito usado em ambientes externos.

O planejador de caminhos baseado em campos potenciais com funções harmônicas, objetivo deste trabalho, enquadra-se nos métodos globais, pois parte do princípio de que o robô conhece todo o ambiente. Seu algoritmo baseia-se na posição do robô, dos objetos e do objetivo para traçar um caminho viável. Porém para entendermos como funciona o planejador é necessário entender como ele representa o mundo. Na próxima seção 2.1 discutiremos as representações do ambiente e o porquê da representação escolhida. Também se discutirão vantagens e desvantagens das

representações de ambiente. Na seção 2.2 será visto um apanhado geral dos planejadores de caminho tradicionais e algumas comparações entre eles.

## 2.1 Representação E Decomposição De Ambientes

A representação do ambiente tem grande importância na robótica, pois é através dela que os robôs entendem o que está a sua volta. Existem muitos tipos de representações de ambientes e cada uma delas tem suas vantagens e desvantagens, o que torna a escolha da representação correta, extremamente importante. Tal escolha é feita seguindo as três relações citadas abaixo (3):

1. A precisão do mapa deve ser correspondente ao nível de precisão exigido para o robô alcançar o objetivo.
2. A precisão do mapa e os objetos representados devem corresponder à precisão e aos tipos retornados pelos sensores do robô.
3. A complexidade do mapa tem um impacto direto na complexidade computacional à qual será submetido o mapeamento, localização e navegação.

Dentre estes tipos de representações podemos destacar Occupancy Grids, Line Maps e Topological Maps, entre outros. Occupancy Grids são amplamente utilizadas na área de robótica móvel. Tal técnica discretiza o ambiente em uma matriz onde cada célula representa uma área definida do mundo real. Cada célula então é carregada com informações específicas sobre aquela região como, por exemplo, se existe ou não um obstáculo nela. A quantidade de memória necessária para o grid depende diretamente do tamanho do mapa, da escala utilizada e de quanta informação se deseja armazenar por célula. Por outro lado, os Line maps apenas guardam a posição de linhas tornando o armazenamento do mapa extremamente eficaz e a quantidade de memória depende apenas da densidade de objetos e paredes do ambiente. (14)

Os Topological maps, descrevem apenas pontos do mapa e como estes se relacionam entre si, geralmente através de um grafo onde os nodos são locais reais e os vértices representam o caminho a se seguir para alcançar o próximo local. A vantagem desta técnica é o fato de representar o ambiente, posição inicial e objetivo como um grafo (roadmap), permitindo assim usar algoritmos clássicos para busca em grafos como o Breadth First, Depth First, A\*, Dijkstra's, entre outros, para navegar pelo mapa. Este método depende muito do tipo de decomposição escolhida. No exemplo da Figura 4 usou-se a decomposição 2D Vertical. Existem outras técnicas para a criação de roadmaps como Diagramas de Voronoi, decomposição triangular, onde cada vértice se torna um nodo e se ligar a outras arestas quando se for possível, decomposição cilíndrica, entre outras (15). Estas serão apresentadas na próxima sessão.

A Figura 1 representa um ambiente de tamanho de 10 metros. A Figura 2 mostra o mesmo ambiente representado por Occupancy Grid com escala 1 célula para cada 10 cm<sup>2</sup>. A Figura 3 mostra a descrição do ambiente em linhas, Line maps, enquanto a Figura 4 demonstra o ambiente descrito por um grafo associado a ele, mapa topológico.



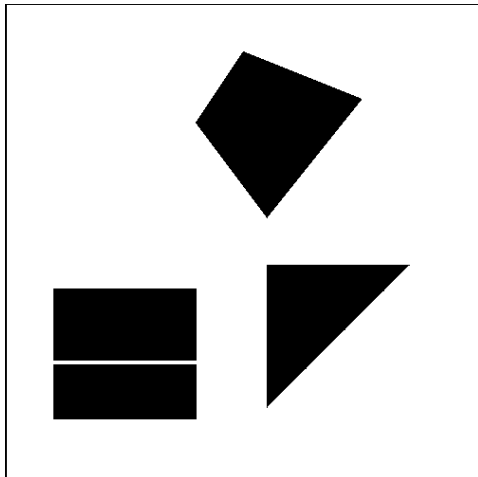


Figura 1 - Mapa do ambiente.

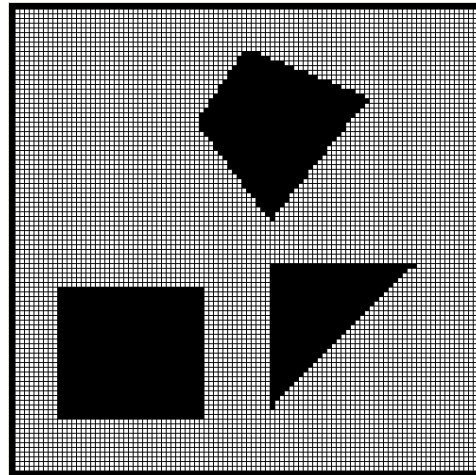


Figura 2 - Representação do ambiente em uma Occupancy Grid.

```

2D-Map
LineMinPos: -10000 -10000
LineMaxPos: 10000 10000
NumLines: 15
LINES
-8000 5000 -2000 5000
-8000 5000 -8000 2000
-2000 2000 -8000 2000
-2000 2000 -2000 5000
-8000 7500 -2000 7500
-8000 5200 -2000 5200
-8000 7500 -8000 5200
-2000 5200 -2000 7500
1000 1000 1000 7000
1000 7000 7000 1000
1000 1000 7000 1000
1000 -1000 5000 -6000
1000 -1000 -2000 -5000
0 -8000 5000 -6000
0 -8000 -2000 -5000
DATA

```

Figura 3 - Representação do ambiente em Mapa de Linhas.

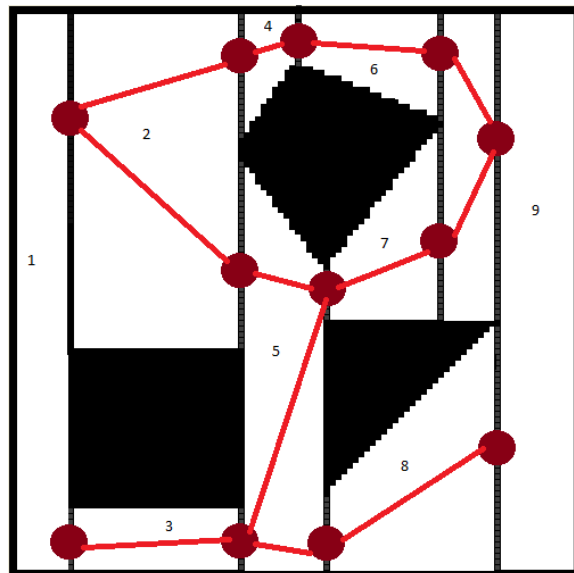
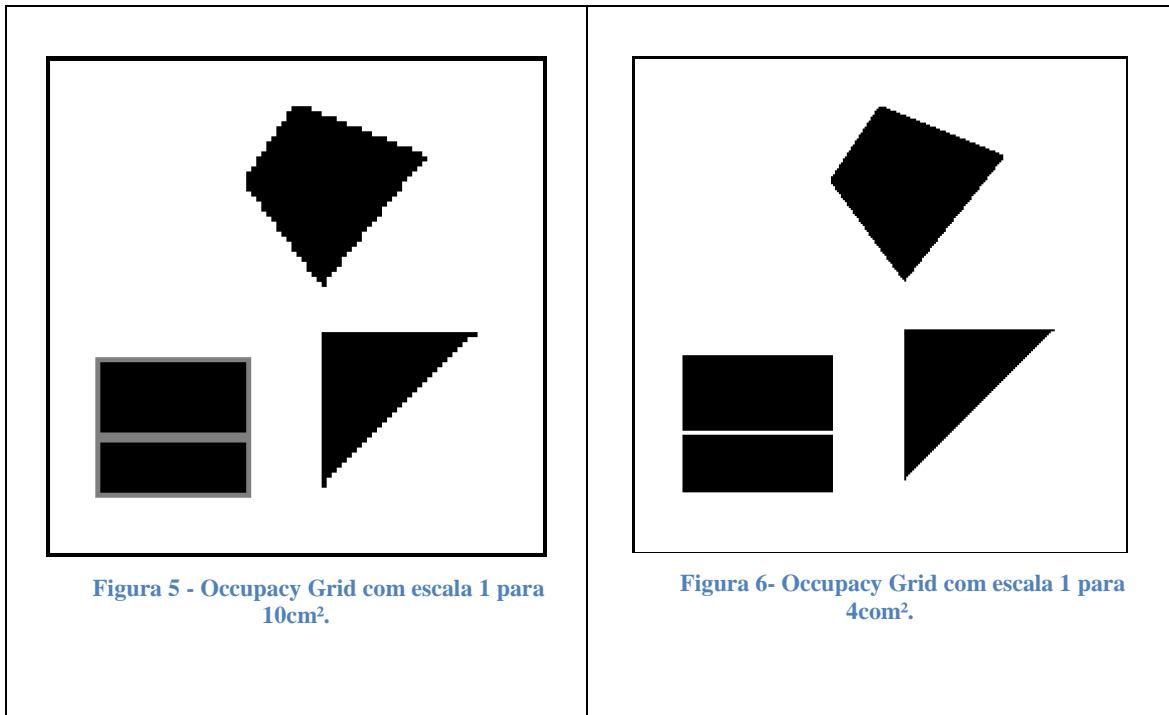


Figura 4 - Representação do ambiente em Mapa Topológico.

Escolheu-se a representação Occupancy Grid para este trabalho, devido às suas características e às peculiaridades dos campos potenciais. Os campos potenciais beneficiam-se muito desta representação, pois cada célula pode facilmente armazenar o valor do campo potencial no ponto. O cálculo do campo potencial também se beneficia muito da estrutura matricial da Grid, pois podemos aplicar diretamente métodos

matemáticos como Gauss-Seidel. Porém, a Grid apresenta suas desvantagens, como o crescimento exponencial de memória e o acréscimo computacional para armazenar e processar mapas maiores.

Tendo isso em vista, um dos parâmetros a se observar neste método é o tamanho da área que cada célula representa, pois células muito grandes descaracterizam objetos a ponto de prejudicar o planejamento de caminhos; por outro lado, células muito pequenas causam matrizes muito grandes, multiplicando em muitas vezes o poder computacional necessário para processar informações úteis sobre elas. Na Figura 5 vemos um exemplo de uma mapa de tamanho 10m X 10m discretizado usando Occupacy Grid utilizando uma escala de 1 célula para cada 10cm<sup>2</sup>. Na Figura 6 temos o mesmo mapa discretizado numa escala de 1 célula para cada 4cm.



As grids foram propostas inicialmente por Marvoravec e Efler (16) com o intuito de serem grades de probabilidade para objetos. Cada célula guarda a probabilidade de que haja ou não um objeto na sua área. Vamos utilizar este conceito na exploração de ambientes. No caso do planejador de caminhos usar funções harmônicas que utilizem a grid para armazenar o potencial de cada célula a fim de calcular o melhor caminho do robô até o objetivo.

## 2.2 Planejadores De Caminho

Os algoritmos de planejamento de caminho têm como objetivo prover ao robô uma rota segura, sem colisões, da posição inicial até a posição objetivo. Surgiram muitos algoritmos para resolver tal problema, dentre estes podemos citar (3):

- 1-Roadmaps: Encontra uma série de rotas no espaço livre;

- 2-Cell Decomposition: Discretiza o espaço livre e separa o espaço ocupado;
- 3-Campos Potencias: Cria uma função matemática caracterizando o espaço.

### 2.2.1 Roadmaps

Os algoritmos de roadmap baseiam-se na construção de mapas de conectividade, como os mapas topológicos da sessão 2.1. Uma vez o mapa construído, o algoritmo seleciona a melhor rota do ponto inicial até o final. Criar o mapa de conectividade (mapa topológico) tem grande importância neste método, pois o mapa deve prover acesso a todas as regiões livres do mapa e ao mesmo tempo conter o menor número possível de caminhos (arestas do grafo) pois os algoritmos que calculam a melhor rota dependem diretamente do número de vértices e arestas que este possui. Como visto na sessão 2.1 existem vários algoritmos para a criação dos roadmap, podendo-se citar decomposição em 2D, diagrama de Voronoi, diagrama de visibilidade entre outros.

#### 2.2.1.1 Diagrama de Visibilidade

O diagrama de Visibilidade é construído a partir dos vértices dos obstáculos, a posição inicial e a posição final. Arestas são traçadas entre os vértices desde que não haja um obstáculo no caminho. Logo após, seleciona-se um algoritmo de planejamento discreto para encontrar um caminho viável entre o ponto inicial e o ponto final. Planejadores discretos serão discutidos rapidamente na seção 2.2.3, visto que este tem um impacto direto no caminho selecionado para o robô. Vemos um exemplo na Figura 7. Na Figura 7(a) A figura demonstra a decomposição em vértices dos objetos e dos pontos iniciais e finais. (b) A figura demonstra a criação das arestas a partir dos vértices. (c) A figura demonstra um possível caminho selecionado por um algoritmo de planejamento discreto.

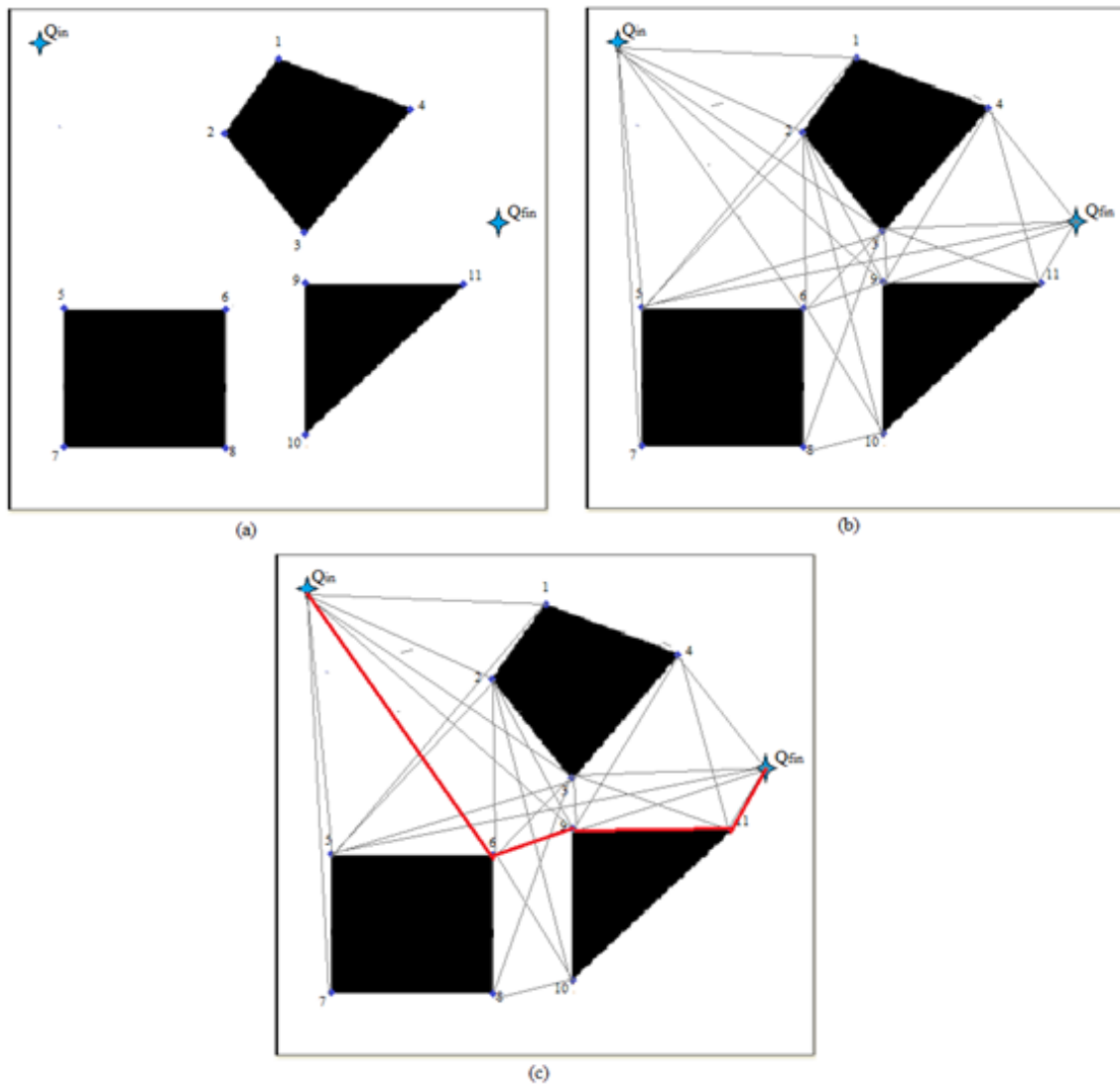


Figura 7 - Diagrama de Visibilidade

Diagramas de visibilidade são facilmente implementados, porém podem provocar alguns problemas. Como se pode observar, o caminho escolhido passa muito próximo aos obstáculos devido às características do diagrama; caso os obstáculos não sejam aumentados de maneira correta, o robô poderia colidir com os mesmos. Outro problema é o aumento exponencial de arestas ao se introduzir um número maior de objetos ou objetos mais complexos. O número de arestas e vértices é linearmente proporcional ao tempo de execução do planejador discreto.

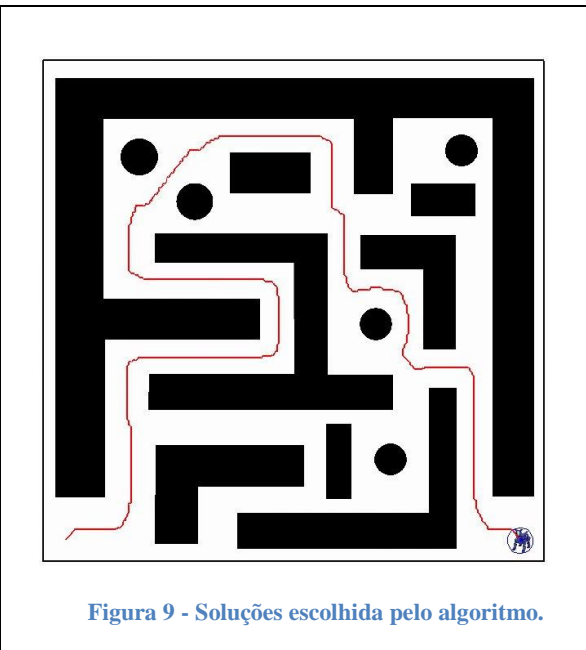
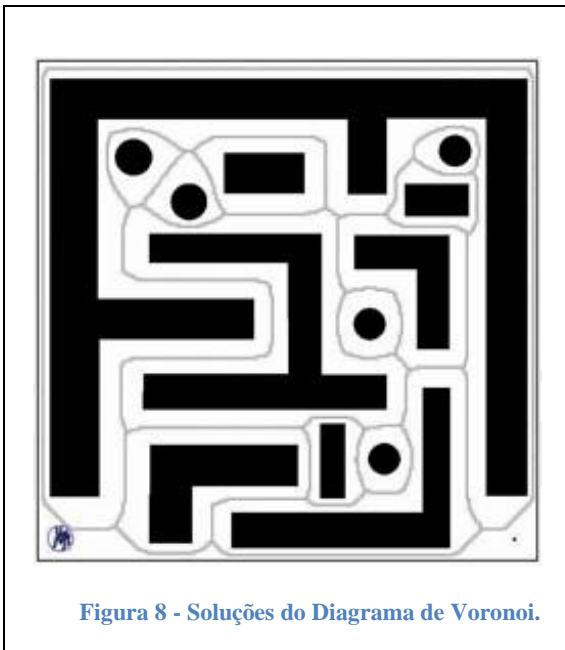
### 2.2.1.2 Diagrama de Voronoi

Para traçarmos um caminho na região bidimensional em que o robô se locomove, cada obstáculo pode ser representado por polígonos côncavos ou convexas. Para encontrar o diagrama generalizado de Voronoi para esta configuração, calcula-se o diagrama através de uma aproximação, convertendo os obstáculos em uma série de pontos. Logo após, é calculado o diagrama para esta coleção de pontos. E, em seguida, os segmentos do diagrama que interceptam algum obstáculo são eliminados. Uma vez calculado o diagrama, é preciso adicionar a posição do robô e o destino ao conjunto de

pontos. Logo após seleciona-se um algoritmo de planejamento discreto para encontrar um caminho viável entre o ponto inicial e o ponto final. A vantagem deste método é que ele tende a maximizar a distância entre os obstáculos e o sistema robótico.

Uma definição formal para o Diagrama de Voronoi é: dado um conjunto  $S$  de  $n$  pontos no plano, deseja-se determinar para cada ponto  $p$  de  $S$  qual é a região  $V(p)$  dos pontos do plano que estão mais próximos de  $p$  do que de qualquer outro ponto em  $S$ . As regiões determinadas por cada ponto formam uma partição do plano chamada de Diagrama de Voronoi [Voronoi, 2004].

Diagramas de Voronoi são completos e graças às suas características, o robô passa longe dos objetos, ao contrário do que acontece no diagrama de visibilidade. Porém, em muitos casos, gerar o diagrama de Voronoi pode não ser trivial e toma muito tempo ao robô.



### 2.2.1.3 Métodos baseados em Amostragem

A ideia dos métodos baseados em amostragem baseia-se em escolher um conjunto de configurações válidas e depois tentar conectá-las criando caminhos e assim um roadmap. É especialmente utilizado em ambientes multidimensionais devido à dificuldade de usar outros métodos. Para aplicar o método, podem-se usar tendências para separar os pontos ou espalhá-los uniformemente no espaço de configurações. Na Figura 10 vê-se um exemplo. Na Figura 10 (a) O primeiro passo dos Métodos baseados em Amostragem escolheu-se pontos aleatórios no espaço de configurações válidas e assim são gerados os vértices do roadmap. (b) Logo se combinam os vértices possíveis. Tais combinações geram as arestas. (c) Como nos outros métodos é escolhido um algoritmo de busca em grafos para encontrar um caminho válido do  $Q_{inicial}$  ao  $Q_{final}$ .

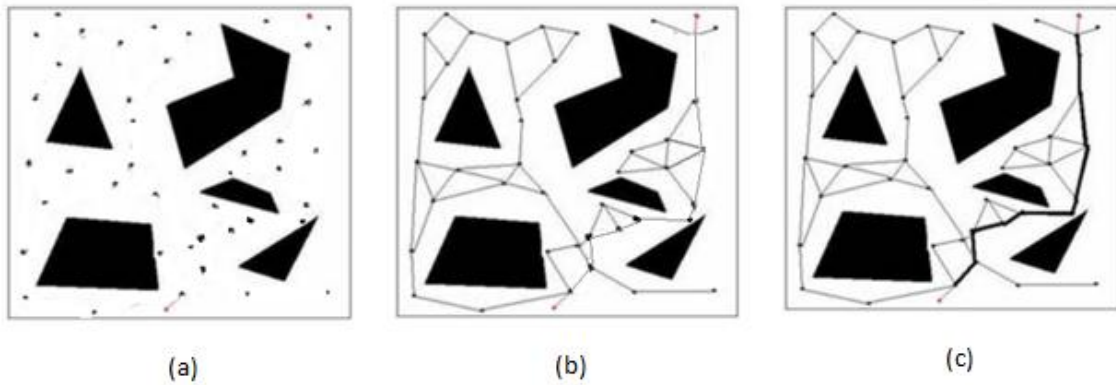


Figura 10 - Exemplo de Métodos baseados em Amostragem

Os métodos baseados em amostragem têm como vantagem a simplicidade, pois muitas vezes é mais complexo gerar configurações válidas através de um algoritmo tradicional do que testar se uma configuração  $Q$  qualquer é válida. Como desvantagens, tais algoritmos possuem uma dificuldade para passar por lugares estreitos e não apresentam caminho ótimo, não importando o algoritmo usado para escolher a rota do ponto inicial ao final.

## 2.2.2 Cell Decomposition

Cell decomposition baseia-se na ideia de dividir o espaço de configurações em regiões simples conectadas. Logo após cria-se um grafo de adjacência e a partir desse grafo determina-se a sequência de células a seguir para chegar do ponto inicial ao final. O caminho seguido pelo robô também tem de ser escolhido. Por exemplo, o robô sempre tem de passar pelo meio da célula.

### 2.2.2.1 Vertical Cell Decomposition

Na decomposição Vertical, também conhecida como decomposição trapezoidal, cada vértice separa o espaço de configuração válido em um segmento vertical como visto na Figura 11 (15). A decomposição vertical é uma técnica válida e simples de ser aplicada, porém apresenta alguns problemas como a escolha de onde os vértices serão feitos. Por exemplo, na Figura 11 os vértices foram criados na metade da linha que separa os subespaços e no meio do subespaço. A linha azul mostra a trajetória ótima e a vermelha a melhor possível tendo em vista a decomposição vertical. Na Figura 11(a) apresenta-se o espaço de configurações. Em 11(b) Cada vértice separa o espaço de configurações em subespaços de configurações. Só o espaço válido de configurações é dividido. Em 11(c) cria-se o Roadmap baseado nos subespaços. Como se observa toda a área de configuração válida está em uma célula e é alcançável. Em 11(d) o algoritmo de busca em grafo encontrou um caminho válido entre  $Q_{inicial}$  e  $Q_{final}$ . Na Figura 12 vemos outra decomposição vertical onde em 12(a) vê-se um ambiente decomposto pela decomposição vertical. Em 12(b) vê-se caminho em vermelho é o melhor caminho escolhido pelo algoritmo de busca em grafo, e o caminho azul é o melhor caminho possível.

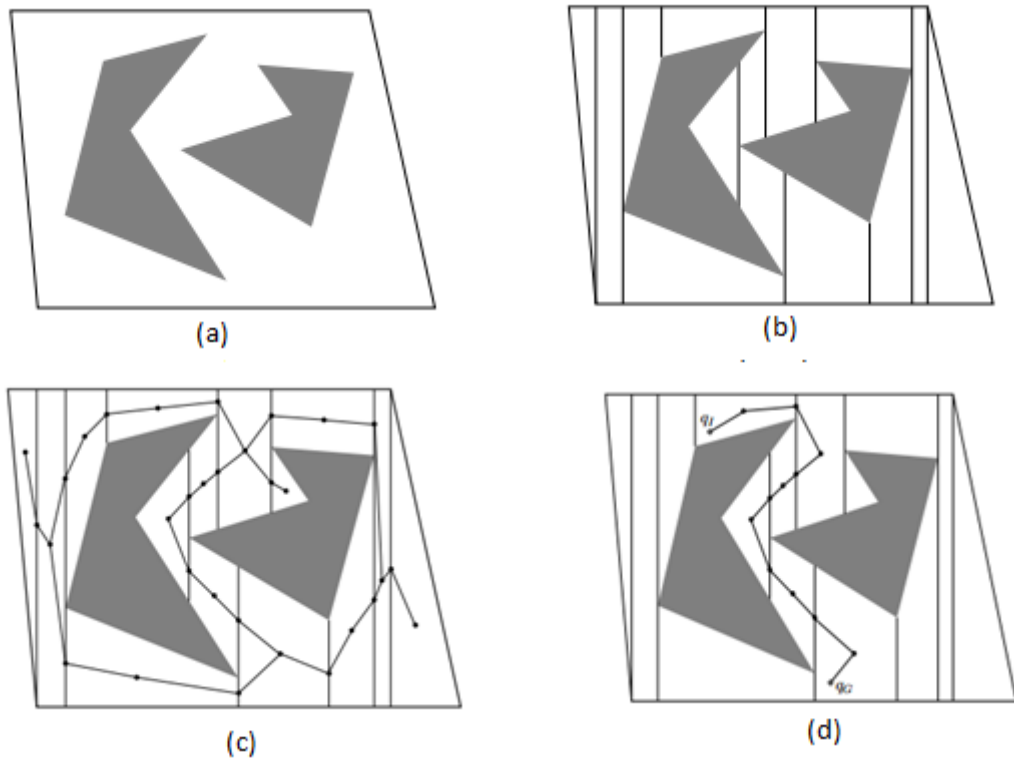


Figura 11 - Vertical Decomposition

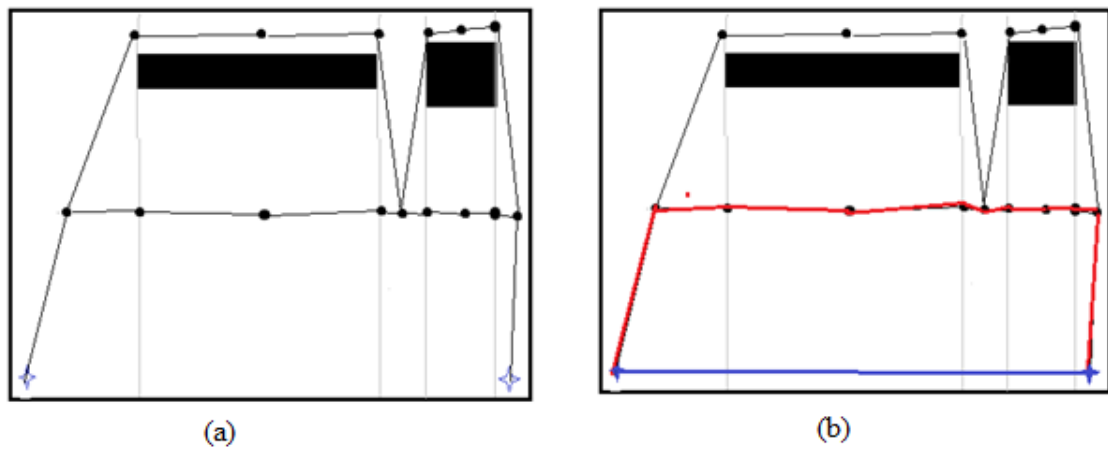


Figura 12 - Vertical Decomposition

### 2.2.2.2 *Triangular Cell Decomposition*

Semelhante ao diagrama de visibilidade vista em 2.2.1.1, na decomposição triangular cada vértice de obstáculo liga-se a outra de modo a dividir o espaço de





Os algoritmos de planejamento discretos podem ser vistos como algoritmos de busca em grafos. Para tal, é importante que tais técnicas sejam sistemáticas, de modo que se existe pelo menos um caminho válido o método o encontrará. O método também deve ser capaz de notificar caso não exista solução (15). No bloco 1, vê-se um algoritmo genérico para buscas em grafos FORWARD SEARCH \*.

FORWARD SEARCH	
1	Q.Inserir( $X_i$ , NULL) & marcar ( $X_i$ , Visitado)
2	Enquanto Q ã vazia faça
3	$x \leftarrow$ Q.Primeiro()
4	se $x \in X_f$
5	retornar SUCESSO
6	paratodo $u \in$ filho ( $x$ )
7	$x' \leftarrow$ u
8	se $x'$ ã visitado
9	marcar( $x'$ , Visitado)
10	Q.Inserir( $x'$ , x)
11	senão
12	resolver $x'$ duplicado
13	retornar FALHA
Onde:	
	Q: lista de estados
	$x, x', u$ : estado
	$X_i$ : estado inicial
	$X_f$ : estado(s) final

Algoritmo:

- 1) O algoritmo inicia salvando o estado inicial em uma lista vazia de estados.
- 2) Logo, para cada entrada de Q:
  - 2.1) Salva-se o primeiro nodo da lista na variável  $x$  com o procedimento Primeiro(). A implementação de tal procedimento varia dependendo do método a se escolher.

2.2) Caso o estado  $x$  seja o estado final, o algoritmo termina em Sucesso. Caso contrário para cada filho de  $x$  não visitado ainda é adicionado à lista  $Q$  com o  $Q.Inserir(<nodo>, <pai do nodo>)$ . A implementação de tal procedimento varia dependendo do método a se escolher.

2.3) Caso o estado filho já tenha sido visitado, o algoritmo deve resolver como agir, pois tal ação tem ligação direta com as características do método a se usar.

3) Caso  $Q$  esteja vazia, ou seja, todos os estados foram alcançados e nenhum dos estados era um estado final, o algoritmo retorna falha.

4) O resultado do algoritmo é uma lista de nodos que leva do estado final até o inicial baseando-se nos pais dos nodos.

A diferença entre os vários métodos de planejamento discreto é a forma como  $Q$  é ordenada. Caso  $Q$  seja uma FIFO, ou seja, uma fila onde o primeiro elemento a ser colocado vai ser o primeiro elemento a ser retirado, temos o algoritmo “Breadth first”. Nas sessões a seguir, é dada uma breve explicação sobre alguns destes métodos.

#### **2.2.3.1 Breadth first**

Como explicado anteriormente o Breadth first utiliza  $Q$  como uma fila de modo que o nível  $k+1$  só será visitado depois que todos os nodos do nível  $k$  forem visitados, garantindo que a primeira solução encontrada será a com menos passos. O tempo de execução asymptotic do Breadth first é  $O(|V|+|E|)$  onde  $V$  é o número de vértices e  $E$  é o número de arestas no grafo ( assumindo que as outras tarefas como testar se o nodo já foi visitado leve sempre o mesmo tempo). Vê-se um exemplo do Breadth first na Figura 15:

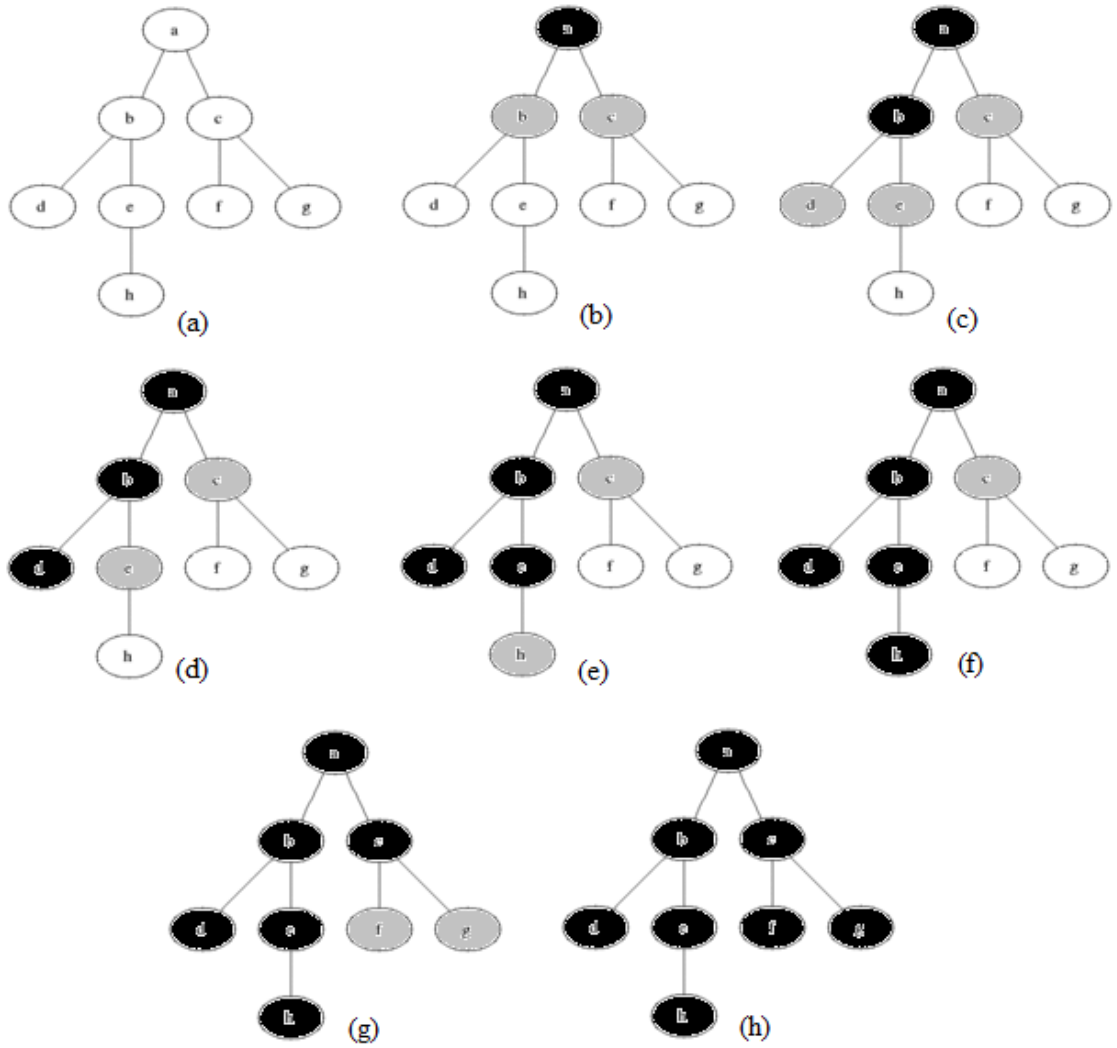


Figura 15 - Breath First

### 2.2.3.2 Depth First

O algoritmo Depth first utiliza Q como uma Pilha de modo que o último nodo a ser adicionado em Q será o próximo a ser visitado. Isto garante um algoritmo que mergulha no grafo de maneira mais agressiva, encontrando uma solução não ótima, porém em um tempo menor. O tempo de execução asymptotic do Depth first é  $O(|V|+|E|)$  onde V é o número de vértices e E é o número de arestas no grafo (assumindo que as outras tarefas como testar se o nodo já foi visitado leve sempre o mesmo tempo). Vê-se um exemplo do Depth first na Figura 16:

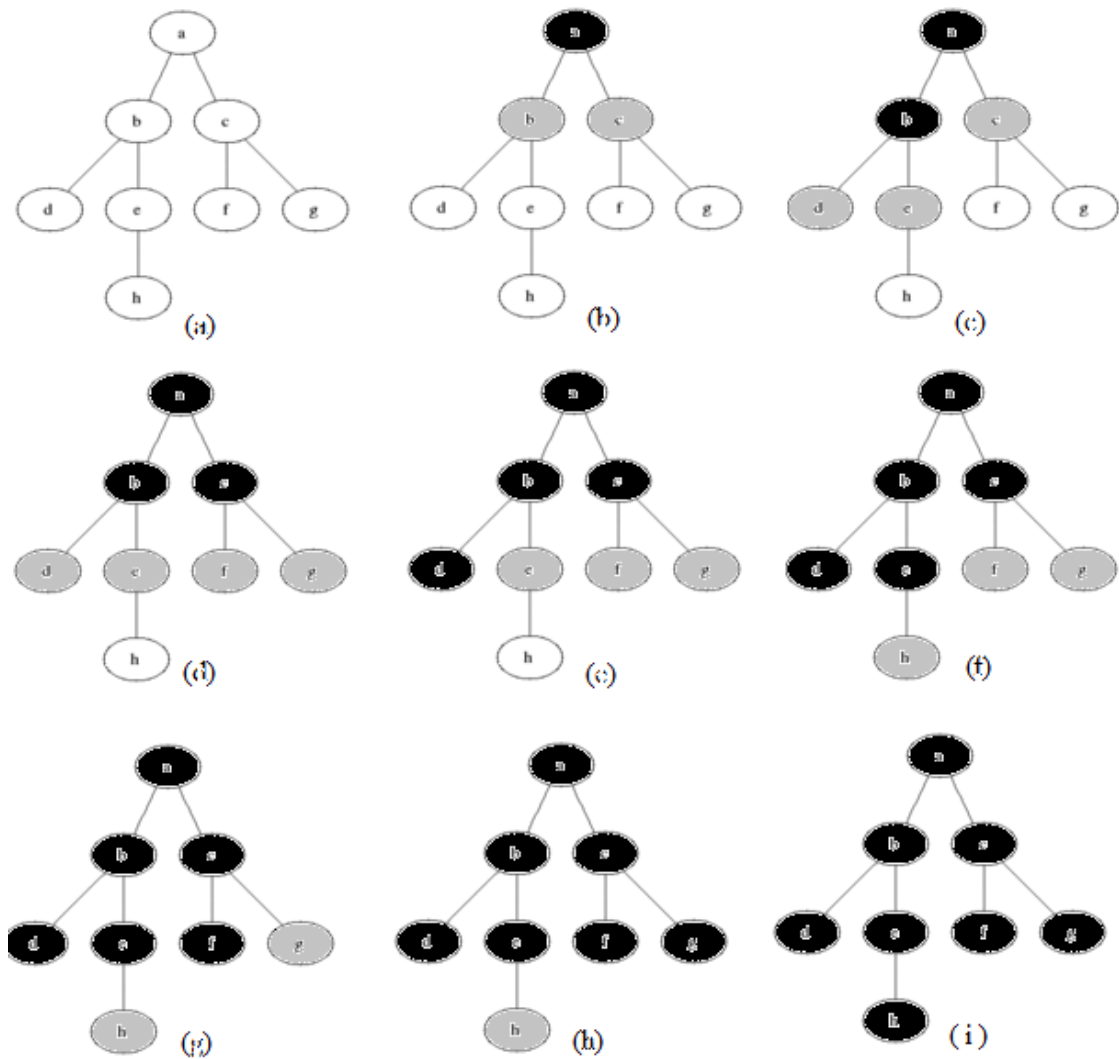


Figura 16 - Depth First

### 2.2.3.3 Dijkstra's

O algoritmo de Dijkstra difere dos anteriores, pois assume que as arestas tem pesos diferentes, dado que tal peso pode vir da distancia entre dois estados, dificuldade de se mover de um estado a outro entre outros. Tal modificação reflete no algoritmo mostrado no bloco 1 fazendo com que o custo, chamado de  $C$ , seja calculado e a lista  $Q$  reordenada de forma a refletir sempre o menor custo. No passo 12, caso o nodo já haja sido visitado o custo  $C$  para chegar a ele pode mudar, caso o novo custo seja menor do que o custo antigo. O tempo de execução do algoritmo de Dijkstra é  $O(|V| \lg |V| + |E|)$ , onde  $V$  é o numero de nodos e  $E$  representa o número de arestas do grafo.

### 2.2.3.4 A-Star

O algoritmo A-star também conhecido como  $A^*$  pode ser considerado uma evolução do algoritmo de Dijkstra, pois tenta reduzir o número de estados visitados a fim de chegar ao estado final incorporando uma estimativa heurística ao custo que leva

o determinado estado a chegar ao estado final. De forma que reordenamos Q levando em conta o custo C mais a estimativa G. O maior problema da técnica é encontrar uma estimativa adequada para G. Caso G seja 0 para todos os casos o algoritmo A\* se torna igual ao algoritmo de Dijkstra, e seu tempo de execução é  $O(|V| \lg |V| + |E|)$ , onde V é o número de nodos e E representa o número de arestas do grafo.

Existem outros métodos de planejamento discreto como o Best First o qual ordena Q de acordo com uma estimativa total do melhor custo ou o Iterative deepening, entre outros. Também podemos aplicar estes algoritmos usando uma abordagem BACKWARD SEARCH a qual inicia pelo estado final e tenta chegar ao inicial. Para todos estes casos, o tempo de execução depende diretamente do número de nodos e arestas, de modo que a escolha correta do método para criar os grafos impacta diretamente no tempo de processamento do robô. Outro fator que podemos levar em conta é o fato de que ao se ter mais informações sobre o ambiente, algoritmos mais refinados como o A\* e o Dijkstra se tornam mais eficientes e tendem a encontrar uma solução melhor.

#### 2.2.4 Campos Potenciais

Os planejadores de caminhos baseados em campos potenciais criam um campo de vetores no espaço, ou gradiente, pelos quais o robô navega. Tal método trata o robô como um ponto que sofre influencia do campo potencial artificial e o move de acordo com o gradiente, tal como uma bola se moveria ladeira abaixo caso o campo indicasse altitude. O objetivo, ou estado final é o ponto mais baixo e atua como um vale enquanto os obstáculos, ou estados não válidos, são os pontos mais altos e atuam como picos. Tal planejador move o robô de maneira suave e ao mesmo tempo evita que o robô colida com qualquer objeto. O método indica o que o robô deve fazer em qualquer posição e não apenas se localizado no estado inicial.

A idéia base do algoritmo é o fato de que o objetivo atrai o robô enquanto os obstáculos o repelem. Caso novos obstáculos sejam encontrados, seus campos de repulsão são adicionados e um novo campo surge (11).

Se assumirmos um campo potencial diferenciável  $U(q)$  2D, pode-se encontrar a força  $F(q)$  agindo na posição  $q = (x,y)$  pela formula:

$$F(q) = -\nabla U(q)$$

Onde  $\nabla U(q)$  denota o gradiente vetorial do campo

$$\nabla U = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}$$

O potencial final atuando sobre o robô na posição q,  $U(q)$  é igual a soma do  $U(q)$  atrativo no ponto q ( $U_{att}(q)$ ) mais a soma do  $U(q)$  repulsivos no ponto q ( $U_{rep}(q)$ ).

$$U(q) = U_{att}(q) + U_{rep}(q)$$

As funções  $U_{att}$  e  $U_{rep}$  podem diferir dependendo da aplicação. Por exemplo, deseja-se uma repulsiva muito grande quando o robô se aproximar do obstáculo, porém deseja-se que tal força não o influencie depois que ele se afaste. Podemos então expressar  $U_{rep}(q)$  como:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}k \left( \frac{1}{p(q)} - \left( \frac{1}{p_0} \right) \right)^2 & \text{se } p(q) < p_0 \\ 0 & \text{se } p(q) \geq p_0 \end{cases}$$

De modo que  $k$  é uma constante,  $p(q)$  indica a distancia do robô ao objeto e  $p_0$  é a distância a partir da qual o robô começa a ser repellido pelo objeto. Quanto maior o  $p_0$  maior será o efeito do obstáculo sobre o robô. Podemos ver o efeito do método na Figura 17 e na Figura 18:

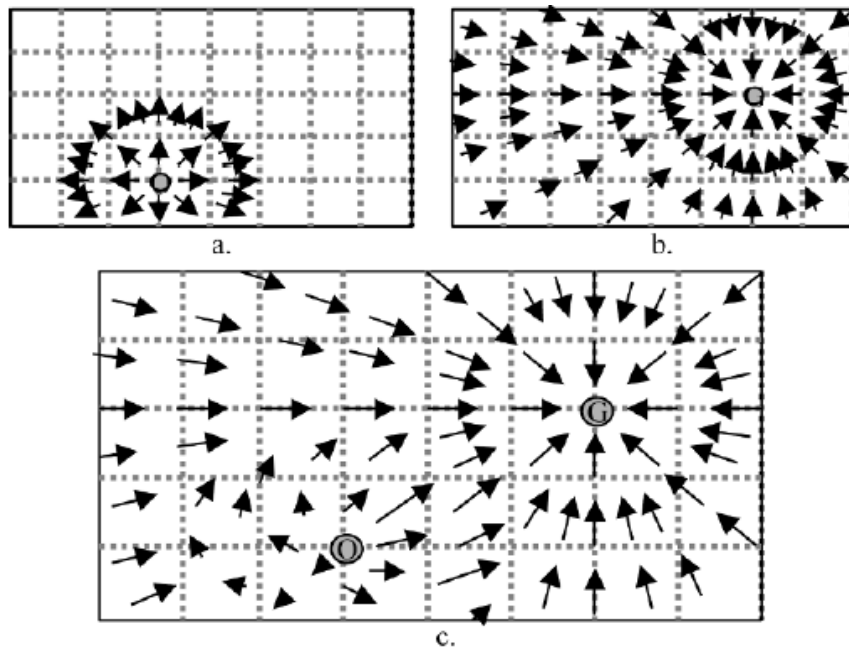


Figura 17 - Campos potenciais

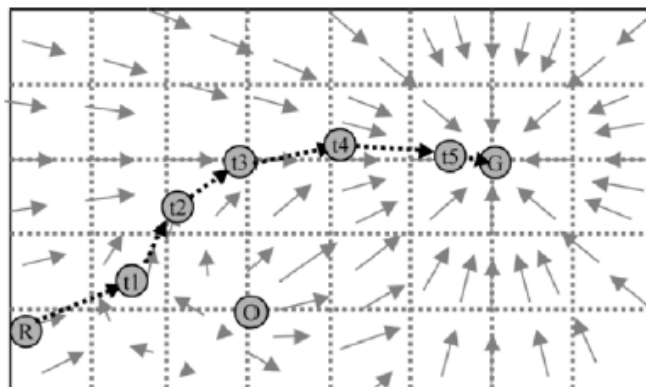


Figura 18 - Caminho escolhido pelo robô

Tal método é extremamente eficaz e simples de aplicar, porém está sujeito a mínimos locais o que o torna ineficiente. Para se eliminar tais mínimos locais é

necessário o uso de pesadas ferramentas matemáticas ou a adição de novos objetos atrativos e repulsivos (16).

Os algoritmos de planejamento de caminho usando campos potenciais se beneficiam muito da representação em grid, pois a grid é capaz de representar facilmente o valor de  $U(q)$  em cada uma de suas células. Tal método também é capaz de se atualizar rapidamente ao encontrar um obstáculo desconhecido ou um novo objetivo.

O objetivo deste trabalho é o estudo e a aplicação de campos potenciais harmônicos, pois estes são uma subcategoria de campos potenciais, os quais não apresentam mínimo local. Tal método será aplicado e depois modificado de modo a adicionar campos de atração e repulsão criando áreas de baixa e alta preferencia. Também será usado para explorar um ambiente e obter tais dados através de sensores a fim de encontrar um caminho que o leve um objetivo. O estudo dos campos potenciais harmônicos é visto a seguir, no capítulo 3.

### 3 CAMPOS POTENCIAIS HARMONICOS

Campos potenciais harmônicos funcionam da mesma forma que os campos potenciais tradicionais criando um campo de vetores que age sobre o robô levando-o suavemente até o objetivo seguindo o gradiente do campo, porém usam uma função baseada na equação de Laplace para gerar tal campo. A resposta de uma equação laplaciana é sempre uma função harmônica, o que dá nome à técnica. Tal método foi proposto por Connolly e Grupen (8) para corrigir a debilidade dos campos potenciais propostos por Khatib (13) de apresentar mínimos locais. A principal característica das funções harmônicas, no ponto de vista dos planejadores de caminho, é o fato de garantir que não haverá mínimos locais como ve-se nas sessões abaixo. Outras vantagens deste método são as propriedades obtidas ao se resolver o algoritmo computacionalmente. Por exemplo, o método resolve o mapa de maneira completa, robustez na presença de objetos desconhecidos e erros, habilidade de exibir diferentes comportamentos e métodos ágeis e rápidos computacionais (tanto por hardware analógico quanto por software) (8).

#### 3.1 Espaço de configuração e Grids

A representação do espaço de configuração nos campos potenciais harmônicos é geralmente a grid, pois ela permite em uma simples matriz armazenar todas as informações necessárias de maneira simples e direta garantindo um acesso rápido às informações. A grid pode ser vista como um plano cartesiano onde cada célula é o menor espaço representado possível por exemplo 5cm x 5cm. O tamanho do ambiente e da grid influenciam diretamente o tamanho que cada célula representa.

Na Figura 19 demonstra uma grid mapeada de um espaço de configurações. As células em branco representam um estado de configuração válido ou um espaço livre no ambiente, as células pretas demonstram espaços ocupados por objetos enquanto as azuis indicam o objetivo ou estado final. Caso seja necessário pode-se adicionar mais objetos ou objetivos.



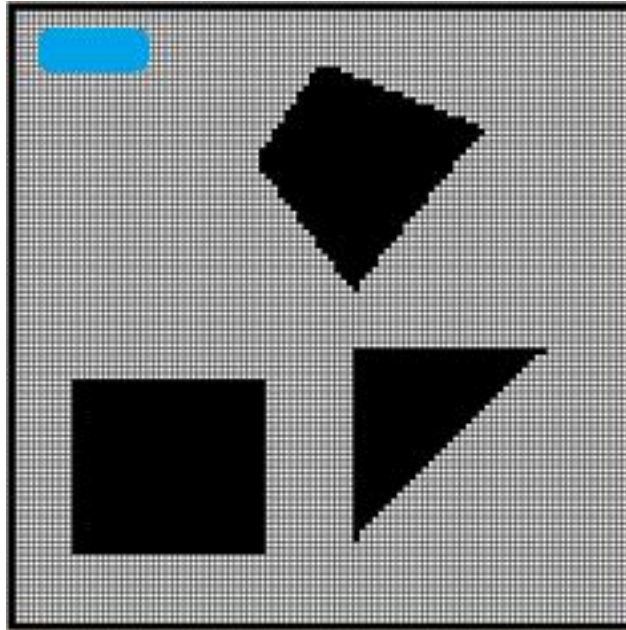


Figura 19 - Grid mapeada

### 3.2 Funções Harmônicas

Funções harmônicas são usadas com várias finalidades diferentes. Na física são empregadas para descreverem campos conservativos, como por exemplo o campo elétrico gerado por partículas carregadas, a velocidade de um líquido não viscoso ou as equações de condução de calor. Tendo em vista tais efeitos, quando utilizamos equações harmônicas para descrever o campo potencial pelo qual o robô se moverá, pode-se aplicar todas as propriedades destes campos conhecidos e imaginar o robô como um ser que busca o ponto mais frio em uma superfície quente ou uma partícula carregada se movendo para o potencial elétrico mais baixo em um campo elétrico. Pode-se extrair a equação de Laplace, cuja resposta será uma função harmônica, a partir das leis do campo elétrico propostas por Maxwell:

$$\nabla \times E = 0 \rightarrow \text{onde } E = \text{campo elétrico numa posição } (x,y)$$

$$\nabla = \left(\frac{\partial}{\partial x}\right)i + \left(\frac{\partial}{\partial y}\right)j$$

$$E = -\nabla\phi \rightarrow \text{sendo } \phi \text{ fluxo do campo elétrico}$$

$$E \cdot E = 0$$

Substituindo obtemos:

$$\nabla^2\phi = 0 \rightarrow \text{Equação de Laplace}$$

### 3.2.1 Propriedades das funções harmônicas

Como dito acima as características dos campos gerados pelas funções harmônicas são o principal motivo pelo qual se utilizam funções harmônicas para descrever o campo potencia (16). As características mas importantes são:

1) Superposição: Qualquer combinação linear de 2 funções harmônicas resultará em uma outra função harmônica e uma solução para a equação de Laplace.

2) Propriedade do valor médio (mean-value) : o valor do potencial em um ponto  $(x_1, y_1)$  centrado em um círculo qualquer é igual a media do potencial integrando a circunferência do círculo, não importando o raio do mesmo,

3) The Maximum Potential Property : O valor máximo de um campo harmônico não constante ocorre nos limites do espaço, onde o potencial tende a infinito.

4) The Minimum Potential Property : O valor mínimo de um campo harmônico não constante ocorre nos limites do espaço, onde o potencial tende a infinito. Esta característica garante que apenas exista um mínimo e não vários mínimos locais.

### 3.3 Resolução numérica

A solução para um equação diferencial em uma região  $R$ , como a equação de Laplace, geralmente implica na obtenção de uma equação analítica que obtenha os valores da variável dependente para todo o  $R$ . Computacionalmente só podemos lidar com uma região  $R$  continua obtendo equação analítica e resolvendo-a para o ponto desejado na região  $R$ . Devido ao fato de muitos dos problemas envolvendo equação diferenciais não possuírem soluções analíticas conhecidas aplicam-se técnicas numéricas. Ao aplicar técnicas numéricas de resolução, trata-se o espaço de forma discreta e obtém-se a solução para cada ponto na região  $R$ .

Inicia-se a resolução numérica pela desratização da região  $R$ , ou seja dividimos a região  $R$  em um conjunto de pontos e somente nesses pontos a solução será encontrada. A esse conjunto dá-se o nome de malha (Grid em inglês). A distribuição e densidade de pontos é essencial para o método como para o seu tempo de processamento computacional. Após isso se descreve os pontos em função dos seus pontos adjacentes, resultando em um sistema de equações algébricas, geralmente linear. Tais expressões recebem o nome de “aproximações por diferenças finitas”. Logo após acrescentam-se as variáveis de contorno, modificando a equação apropriadamente nos pontos de fronteira. Logo após as equações algébricas lineares são resolvidas para cada ponto obtendo assim a solução do problema.

A maneira mais comum e simples de discretizar uma região  $R$  contínua é espalhando os pontos uniformemente separados por um espaço constante  $Dx$  e  $Dy$ . Vemos um exemplo disso na Figura 20 de uma região  $R$  sendo discretizada de maneira uniforme com espaçamento  $Dx$  e  $Dy$ . Dependendo de experimento uma malha uniforme pode não representar bem o ambiente, por exemplo, a malha usada em dinâmica de fluidos para calcular o escoamento sobre VSL é muito mais densa perto da sua dobra, pois ali concentram-se os valores mais importantes, veja na Figura 21. Malhas que apresentam uma regularidade na distribuição dos pontos são conhecidas como estruturadas enquanto as não-estruturadas não apresentam regularidade. Podemos considerar a grid representada na Figura 20 como estruturada enquanto a representada na Figura 21 é não estruturada.

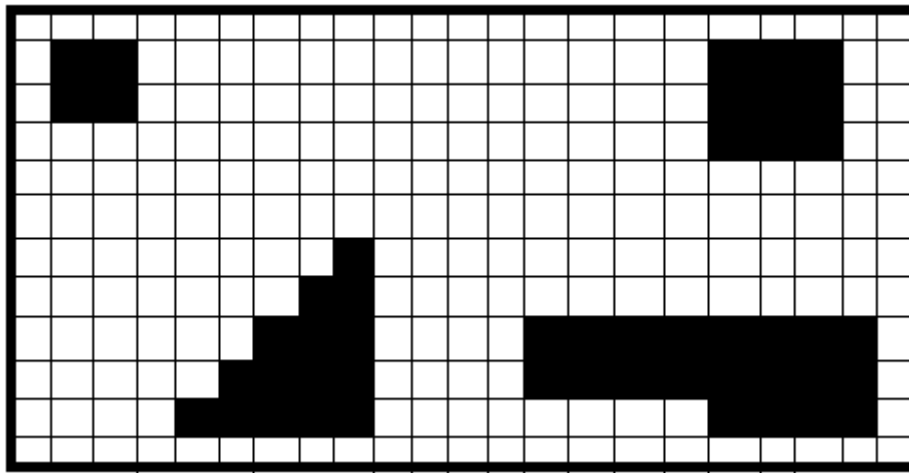


Figura 20 - Grid Estruturada

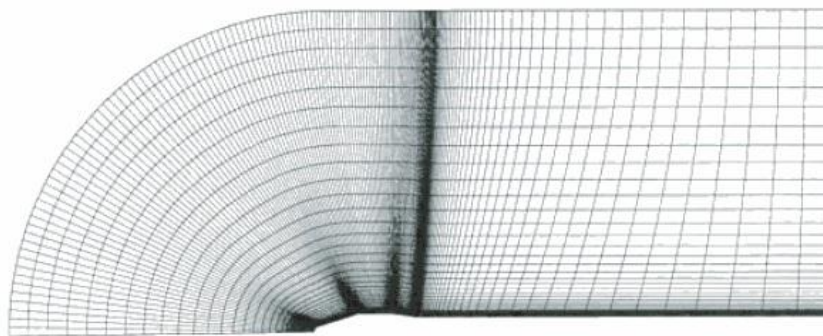


Figura 21 - Grid de Escoamento VLS

A obtenção equações algébricas, ou aproximações por diferenças finitas, podem ser obtidas através de dois métodos, expansões de Taylor ou interpolação polinomial. As expansões de Taylor são mais comumente usados para a representação de malhas estruturadas enquanto a interpolação polinomial para as não-estruturadas (17). Podemos considerar as aproximações por diferenças finitas como o inverso do processo de determinação do limite, utilizado para definir a derivada de uma função  $f$ :

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

### 3.3.1 Expansão de Taylor

Para discretizar uma função o teorema da Taylor nos permite escrever uma função  $f$  continua no intervalo  $[a,b]$  e que possua derivadas definidas de ordem  $N$  como:

$$f(x) = f(x_0) + (\nabla x) \frac{df}{dx} \Big|_{x_0} + \frac{(\nabla x)^2}{2!} \frac{d^2 f}{dx^2} \Big|_{x_0} + (\nabla x)^3 \frac{d^3 f}{dx^3} \Big|_{x_0} + \dots + R_n$$

Em que  $\Delta x = x - x_0$  e  $R_n$  é o resto, definido como:

$$R_n = \frac{(\nabla x)^n}{n!} \frac{d^n f}{dx^n} \Big|_{x_0}$$

A Figura 22 mostra alguns pontos da uma malha unidimensional separados uniformemente por  $x_i - x_{i-1} = \Delta x$  :

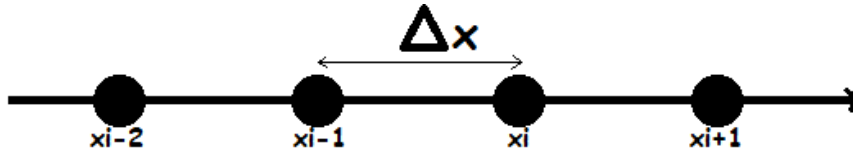


Figura 22 - Definição de  $\Delta x$

Logo, o teorema nos diz, que para determinar a primeira derivada de uma função  $f(x)$  no ponto  $x_i = i\Delta x$  denotada por  $\frac{\delta f}{\delta x} \Big|_i$  expandimos a função  $f(x_i + \Delta x)$  e isolamos o termo  $\frac{\delta f}{\delta x} \Big|_i$  obtendo assim:

$$\frac{\delta f}{\delta x} \Big|_i = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} + \left[ -\frac{(\nabla x)}{2!} \frac{d^2 f}{dx^2} \Big|_i - (\nabla x)^2 \frac{d^3 f}{dx^3} \Big|_i - \dots \right]$$

Logo podemos descrever  $\frac{\delta f}{\delta x} \Big|_i$  como:  $\frac{f(x_i + \Delta x) - f(x_i)}{\Delta x}$  mais os termos  $R_n$ , também conhecidos como erro local de truncamento (ELT). Tal número aparece devido ao número finito de termos usados na serie de Taylor. O ELT fornece uma diferença entre o valor exato da derivada e o valor aproximado numérico. Logo podemos escrever, como uma equação de diferenças progressivas de primeira ordem para a derivada de  $f$ , representando o ELT por  $O(\Delta x)$ :

$$\left. \frac{\delta f}{\delta x} \right|_i = \frac{f_{i+1} - f_i}{\Delta x} + O(\Delta x)$$

Dizemos que a equação acima é de primeira ordem pois  $O(\Delta x)$  aparece elevado a primeira potência e é dita de diferenças progressivas pois utiliza um ponto no futuro para calcular a derivada. Podemos observar o efeito na Figura 23:

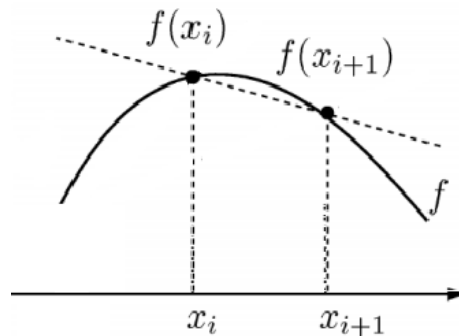


Figura 23- Aproximação de 1º ordem, via diferenças progressivas, para f

Uma segunda aproximação poderia ser feita através da expansão de  $f(x_i - \Delta x)$ , onde obtemos:

$$\left. \frac{\delta f}{\delta x} \right|_i = \frac{f_i - f_{i-1}}{\Delta x} + O(\Delta x)$$

Dizemos que a equação acima é de primeira ordem pois  $O(\Delta x)$  aparece elevado a primeira potência e é dita de diferenças atrasadas pois utiliza um ponto no passado para calcular a derivada. Podemos observar o efeito na Figura 24:

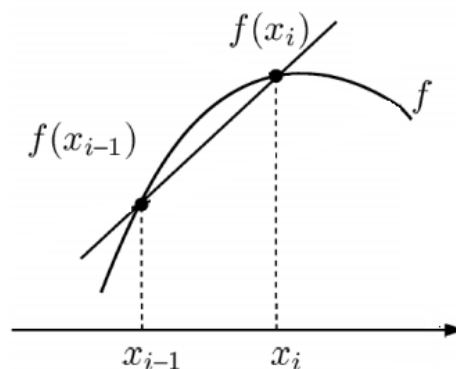


Figura 24-Aproximação de 1º ordem, via diferenças atrasadas, para f

Podemos obter uma aproximação de  $O(\Delta x)^2$  apenas manipulando as equações acima, fazendo com que as derivadas segundas se anulem:

$$f(x_i + \Delta x) - f(x_i - \Delta x) = 2(\Delta x) \left. \frac{\delta f}{\delta x} \right|_i + O(\Delta x)^3$$

Ou

$$\left. \frac{\delta f}{\delta x} \right|_i = \frac{f_{x+1} - f_{x-1}}{2\Delta x} + o(\Delta x)^2$$

Dizemos que a equação acima é de segunda ordem pois  $O(\Delta x)$  aparece elevado a segunda potência e é dita de diferenças centrais pois utiliza um ponto no passado e o futuro para calcular a derivada. Podemos observar o efeito na Figura 25:

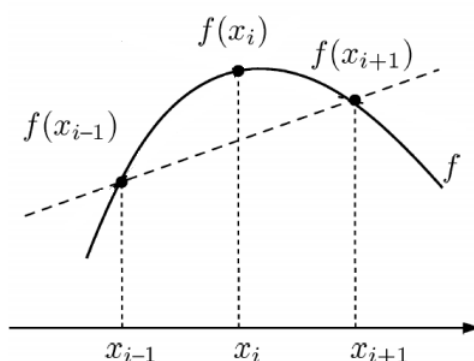


Figura 25-Aproximação de 2º ordem, via diferenças centrais, para f

Expressões para derivadas de ordem superior a 1 podem ser construídas de forma parecida, como vemos na equação abaixo:

$$f(x_i + \Delta x) - f(x_i - \Delta x) = 2f(x_i) + (\Delta x)^2 \left. \frac{\delta^2 f}{\delta x^2} \right|_i + O(\Delta x)^4$$

Rearranjamos os termos temos:

$$\left. \frac{\delta^2 f}{\delta x^2} \right|_i = \frac{f_{x+1} - 2f_i + f_{x-1}}{(\Delta x)^2} + O(\Delta x)^2$$

Tendo em mão as aproximações por diferenças finitas e adicionando as condições de contorno pode-se calcular a solução para cada ponto. Para tal á dois métodos ao qual se pode recorrer, o método analítico ou o iterativo. Visto que o método analítico é penoso computacionalmente devido ao elevado grau de complexidade envolvido em sua resolução, usa-se o método iterativo descritos na sessão a seguir.

### 3.3.2 Equação de Laplace discreta

A equação de Laplace é classificada como equação elíptica e está relacionada com problemas de equilíbrio e geralmente não dependem de tempo. A discretização de uma equação elíptica usando expansões de Taylor, vista na sessão anterior, mostra que podemos discretizar a equação de Laplace da seguinte forma:

$$\nabla^2 f = 0 \quad \rightarrow \quad \text{Equação de Laplace}$$

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

Logo, quando usamos a equação de Laplace em para descrever a derivada em um ponto  $i, j$  podemos escrever  $\nabla^2 f(x, y)$  como:

$$\left. \frac{\delta^2 f}{\delta x^2} \right|_{i,j} + \left. \frac{\delta^2 f}{\delta y^2} \right|_{i,j} \approx \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(\Delta x)^2} + \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{(\Delta y)^2} = 0$$

Ao multiplicar a equação acima por  $(\Delta x)^2$  obtemos uma forma simplificada:

$$f_{i+1,j} + f_{i-1,j} - 2(1 + \beta^2)f_{i,j} + \beta^2(f_{i,j+1} + f_{i,j-1}) = 0 \quad \rightarrow \quad \text{onde } \beta = \frac{\Delta x}{\Delta y}$$

$$f_{i,j} = \frac{(f_{i+1,j} + f_{i-1,j} + \beta^2 f_{i,j+1} + \beta^2 f_{i,j-1})}{2(1 + \beta^2)}$$

Visto que ela não depende do tempo, o cálculo da equação de Laplace se resume a resolver um sistema linear, e como a matriz dos coeficientes do sistema não é, em geral triangular, a solução é obtida por métodos iterativos. Vemos um exemplo do sistema linear na **Erro! Fonte de referência não encontrada.**:

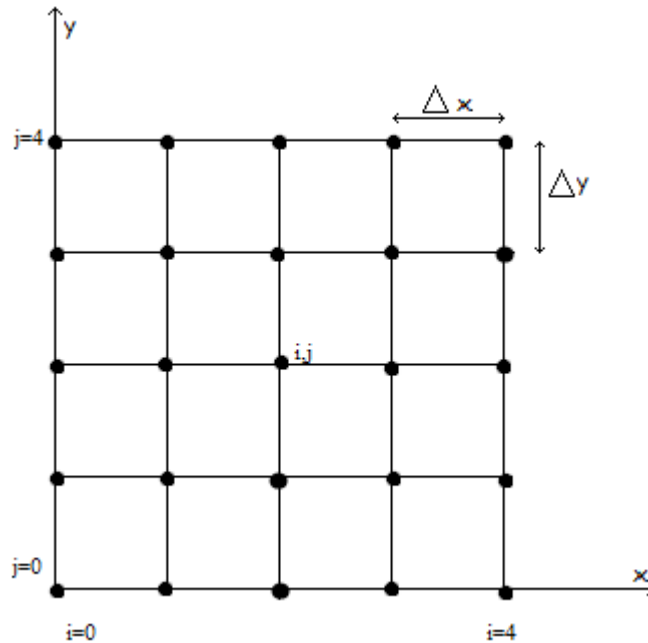


Figura 26 - Sistema Linear

$$f(x) = \begin{cases} f_{2,1} + f_{0,1} - 2(1 + \beta^2)f_{1,1} + \beta^2(f_{1,2} + f_{1,0}) & \text{em (1,1)} \\ f_{3,1} + f_{1,1} - 2(1 + \beta^2)f_{2,1} + \beta^2(f_{2,2} + f_{2,0}) & \text{em (2,1)} \\ f_{4,1} + f_{2,1} - 2(1 + \beta^2)f_{3,1} + \beta^2(f_{3,2} + f_{3,0}) & \text{em (3,1)} \\ f_{2,2} + f_{0,2} - 2(1 + \beta^2)f_{1,2} + \beta^2(f_{1,3} + f_{1,1}) & \text{em (1,2)} \\ f_{3,2} + f_{1,2} - 2(1 + \beta^2)f_{2,2} + \beta^2(f_{2,3} + f_{2,1}) & \text{em (2,2)} \\ f_{4,2} + f_{2,2} - 2(1 + \beta^2)f_{3,2} + \beta^2(f_{3,3} + f_{3,1}) & \text{em (3,2)} \\ f_{2,3} + f_{0,3} - 2(1 + \beta^2)f_{1,3} + \beta^2(f_{1,4} + f_{1,2}) & \text{em (1,3)} \\ f_{3,3} + f_{1,3} - 2(1 + \beta^2)f_{2,3} + \beta^2(f_{2,4} + f_{2,2}) & \text{em (2,3)} \\ f_{4,3} + f_{2,3} - 2(1 + \beta^2)f_{3,3} + \beta^2(f_{3,4} + f_{3,2}) & \text{em (3,3)} \end{cases} \quad \text{onde } \beta = \frac{\Delta x}{\Delta y}$$

Para resolver tal sistema linear, podem-se utilizar dois tipos de métodos diferentes, os métodos diretos ou os métodos iterativos. A solução por métodos diretos não é recomendada, pois apresenta o problema “fill in<sup>2</sup>”. Os métodos iterativos não sofrem tal problema visto que requerem somente o resultado da multiplicação da matriz coeficiente por um vetor (18). Na próxima sessão veremos um estudo dos métodos iterativos.

### 3.3.3 Métodos Iterativos

Métodos iterativos resultam da aplicação repetida de um algoritmo, em geral simples, que a partir de uma aproximação conhecida constrói uma nova

<sup>2</sup> Ao se aplicar um método direto, como o método de Gauss, ao se eliminar os elementos da parte triangular inferior da matriz, muitos elementos que eram nulos na matriz original tornam-se não-nulos ao longo do processo.



aproximação mais próxima da solução exata. Portanto, eles fornecem a solução exata somente como limite de uma sequência, mesmo quando os erros de arredondamento não são levados em conta (18). Uma característica muito positiva é que os métodos iterativos são auto-corrigíveis, isto é, sua convergência é independente da aproximação inicial, e sua estrutura permite a introdução de parâmetros de controle, tais como sub e sobre-relaxação.

Um método é dito iterativo quando fornece uma sequência de aproximações, cada um das quais obtida das anteriores pela repetição do mesmo processo. Dentre os métodos iterativos podemos citar o método Jacobi, Gauss-Seidel ou o SOR, Successive Over-Relaxation.

### 3.3.3.1 Método Jacobi

O método de relaxamento baseado em iterações de Jacobi resume-se simplesmente em substituir o valor das células livres pela média de seus vizinhos simultaneamente. Tal método geralmente resulta em um número maior de iterações, porém em casos onde utiliza-se uma arquitetura SIMD pode-se refinar o algoritmo de forma a obter um resultado mais eficaz. Pode-se resumir o método aplicando a equação abaixo a todas as células livres:

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

### 3.3.3.2 Método Gauss-Seidel

O método Gauss-Seidel é parecido com o método Jacobi, porém utiliza termos da próxima iteração já calculados no cálculo da célula. Como vantagens tem-se a economia de memória, pois só se faz necessário armazenar uma matriz de informações. Pode-se resumir o método aplicando a equação abaixo a todas as células livres:

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

### 3.3.3.3 Método SOR

O método o SOR é conhecido por ser o mais rápido, ou seja, ele converge com menos iterações, pois utiliza um fator de aceleração  $w$ . Em contrapartida o erro aumenta inicialmente antes de convergir.

$$f_{i,j}^{k+1} = f_{i,j}^k + \frac{w}{4} (f_{i+1,j}^{k+1} + f_{i-1,j}^k + f_{i,j+1}^{k+1} + f_{i,j-1}^k - 4f_{i,j}^k)$$

Para  $0 < w < 1$ , o esquema é conhecido como sub-relaxação. Para  $1 < w < 2$ , sobre-relaxação.

### 3.4 Planejador BVP

Os primeiros a propor algoritmos de planejamento de caminhos baseados em funções harmônicas, conhecido também como “Harmonic Functions Path Planner” foram Connolly e Grupen em (8). Em seu método, uma equação diferencial parcial (PDE) gera um campo potencial. O gradiente descendente deste campo indica as rotas que levam de qualquer ponto do campo ao objetivo. O centro desse método é uma equação de Laplace, a qual é calculada para todas as células livres do espaço. O resultado de tal método são caminhos suaves e livres de mínimos locais.

Com o passar do tempo surgiram alguns algoritmos para melhorar o controle sobre as rotas de navegação, entre eles o algoritmo BVP visto em (19). O centro desse algoritmo baseia-se na adição de um novo termo a equação de Laplace, porém mantendo as características básicas das funções harmônicas. Tais termos adicionam tendência ao campo criando regiões de maior e menor atração, com isso fazendo com que o robô tenda a passar por áreas de maior atração e evitar as de menor atração. A diferença entre áreas de menor atração e obstáculos é que o robô pode passar por áreas de menor atração, porém não pode passar por obstáculos. Este problema é conhecido como “weight problem” e foi adereçado por J. S. B. Mitchell e C. H. Papadimitriou em (20). O método BVP baseia-se na equação abaixo:

$$\begin{aligned} \nabla^2 p(r) - ev \cdot \nabla p(r) &= 0 & \begin{aligned} v &\in R^2 \\ |v| &= 1 \\ r &\in R^2 \end{aligned} \end{aligned}$$

Para calcular associamos cada célula do grid com uma posição e iniciamos as células correspondentes a espaços abertos com um potencial baixo(0) e os obstáculos com potencial alto(1). A seguir o algoritmo proposto por Prestes em (19).

---

**Algorithm 1 Basic Algorithm - Usando Gauss-Sindel**

---

```

1: for all cell  $c$  that contains obstacle do
2:   set its potential value to high potential      ▷  $p(c) = 1$ 
3: end for
4: for all cell  $c$  that contains goal do
5:   set its potential value to low potential      ▷  $p(c) = 0$ 
6: end for
7: for all cell  $c$  do
8:   initialize  $\epsilon(c) \leftarrow preference(c)$ ,  $v_x(c) \leftarrow 0$ ,  $v_y(c) \leftarrow 0$ .
9: end for
10: while TRUE do
11:   for all cell  $c$  that represents an environment free-space do
12:      $h(c) \leftarrow \frac{1}{4}(p(c_n) + p(c_s) + p(c_w) + p(c_l))$ 
13:      $d(c) \leftarrow (p(c_l) - p(c_w))v_x(c) + (p(c_n) - p(c_s))v_y(c)$ 
14:      $p(c) \leftarrow h(c) + \frac{\epsilon(c)}{8}d(c)$ 
15:   end for
16:   if potential converged and vector field is not saved then
17:     for all cell  $c$  do
18:        $v(c) \leftarrow$  normalized gradient at cell  $c$ 
19:     end for
20:   end if
21: end while

```

---

Na próxima sessão veremos a implementação deste e de outros métodos baseados em BVP implementados pelo autor em linguagem C++ assim como os resultados obtidos.

## 4 IMPLEMENTAÇÃO E RESULTADOS

O objetivo deste trabalho foi a implementação e testes de algoritmos baseados no método BVP. Tais algoritmos são baseados na proposta de Canolly e Grupen em (8) e as modificações propostas por Prestes vistas em (19). Para tal objetivo foi criado um software em C++, com as capacidades de ler um mapa e uma lista de preferências e a partir dessas gerar os resultados necessários. Na Figura 27 podemos ver o aspecto deste software. O software interage com o simulador MOBILESIM da empresa Mobile Robots, empresa a qual fabrica vários robôs para a pesquisa e ensino como o Pioneer. Tal simulador tem a vantagem de responder em condições similares ao robô real, ou seja, os comandos dados ao simulador e as respostas do mesmo serão iguais as do robô real. O mesmo software usado para se comunicar com o simulador é usado para se comunicar com o robô.

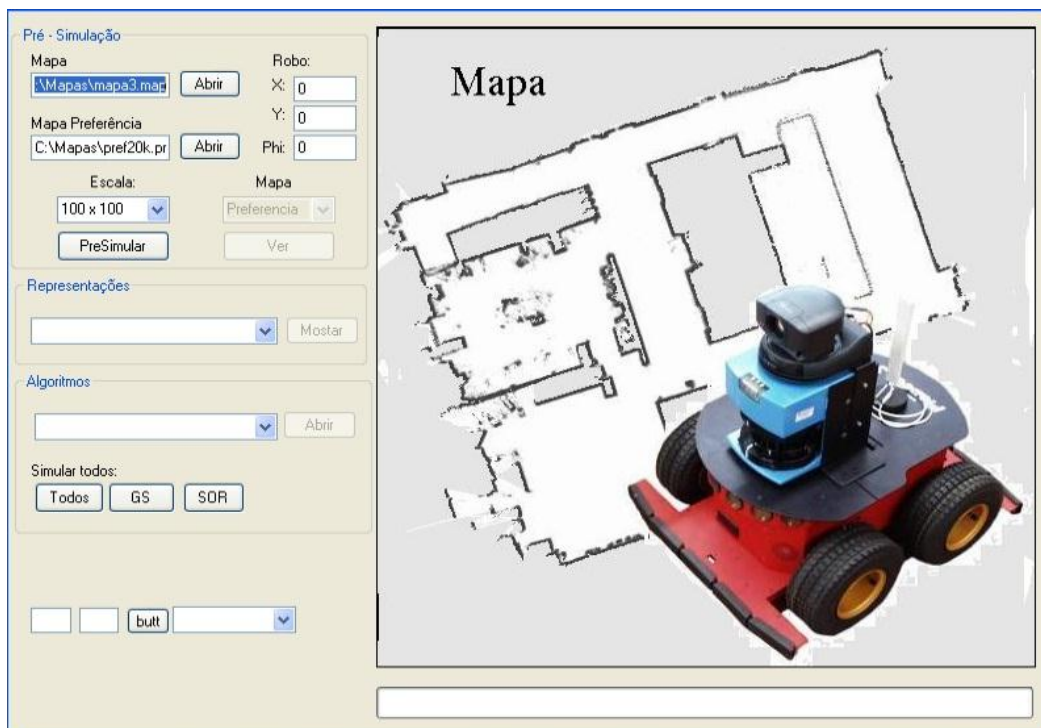


Figura 27 - Software resultado

### 4.1 Leitura e interpretação do Mapa

Um dos aspectos mais importantes do algoritmo é o ambiente no qual ele é executado e como o robô o vê. O mapa de entrada do software é o mesmo que o mapa do simulador, pois dessa forma apenas é necessário criar um ambiente que será o mesmo para os dois programas. Tal mapa é descrito como um mapa de linhas, pode-se ver um exemplo na Tabela 1.

Para criar o arquivo seguem-se 6 passos simples:

- 1 – O mapa começa com a tag 2D, pois todos os mapas são bidimensionais.
- 2 – Descreve-se a menor posição e logo a maior no formato (x, y) Para o simulador isso representa milímetros.
- 3 – Descreve-se o número de linhas existentes no mapa.
- 4 – Escreve-se a tag LINES indicando o início da descrição das linhas.
- 5 – Descreve-se uma linha por linha descrevendo o ponto inicial e o final de cada linha no formato  $P_i(x_i, y_i) P_f(x_f, y_f)$
- 6 – O Arquivo termina com a tag DATA.

**Tabela 1 – Mapas em formato de Linha**

Mapa 1	Mapa 2	Mapa 3
2D-Map	2D-Map	2D-Map
LineMinPos: -20000 -20000	LineMinPos: -25000 -25000	LineMinPos: -10000 -10000
LineMaxPos: 20000 20000	LineMaxPos: 25000 25000	LineMaxPos: 10000 10000
NumLines: 36	NumLines: 10	NumLines: 15
LINES	LINES	LINES
-20000 -20000 -20000 20000	-10000 -10000 +10000 -10000	-8000 5000 -2000 5000
-20000 -20000 20000 -20000	+10000 -10000 +10000 +10000	-8000 5000 -8000 2000
20000 20000 -20000 20000	+10000 +10000 -10000 +10000	-2000 2000 -8000 2000
20000 20000 20000 -20000	-10000 +10000 -10000 +5000	-2000 2000 -2000 5000
-18000 18000 -18000 -10000	-10000 -10000 -10000 -5000	-8000 7500 -2000 7500
-18000 18000 -16000 18000	-20000 -20000 +20000 -20000	-8000 5200 -2000 5200
-16000 18000 -16000 -8000	-20000 -20000 -20000 +20000	-8000 7500 -8000 5200
-16000 -8000 -12000 -8000	+20000 +20000 -20000 +20000	-2000 5200 -2000 7500
-12000 -8000 -12000 -10000	+20000 +20000 +20000 +5000	1000 1000 1000 7000
-12000 -10000 -18000 -10000	+20000 -20000 +20000 -5000	1000 7000 7000 1000
-10000 7500 -10000 -17500	DATA	1000 1000 7000 1000
-10000 -17500 -2000 -17500		1000 -1000 5000 -6000
-2000 -17500 -2000 -5500		1000 -1000 -2000 -5000
-2000 -5500 -8000 -5500		0 -8000 5000 -6000
-8000 -5500 -8000 -2000		0 -8000 -2000 -5000
-8000 -2000 -2000 -2000		DATA
-2000 -2000 -2000 0		
-2000 0 -8000 0		
-8000 0 -8000 5500		
-8000 5500 -2000 5500		

-2000 5500 -2000 7500		
-2000 7500 -10000 7500		
2000 5000 8000 5000		
8000 5000 8000 -5000		
8000 -5000 2000 -5000		
2000 -5000 2000 5000		
10000 5000 14000 5000		
14000 5000 17000 -5000		
17000 -5000 17000 18000		
17000 18000 18000 18000		
18000 18000 18000 -10000		
18000 -10000 16000 -10000		
16000 -10000 12000 2000		
12000 2000 12000 -18000		
12000 -18000 10000 -18000		
10000 -18000 10000 5000		
DATA		

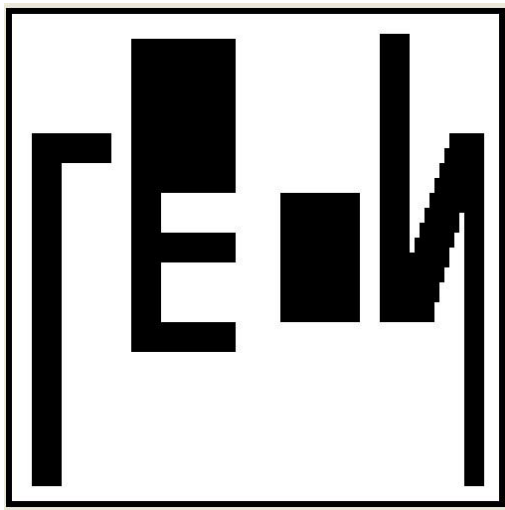


Figura 28 - Mapa 1 escala 100x100

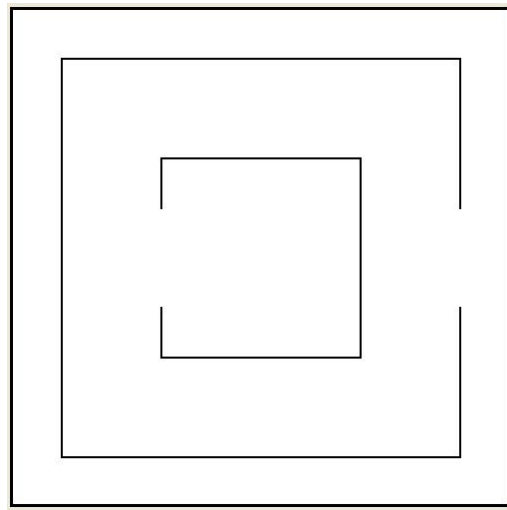


Figura 29 - Mapa 2 escala 250x250

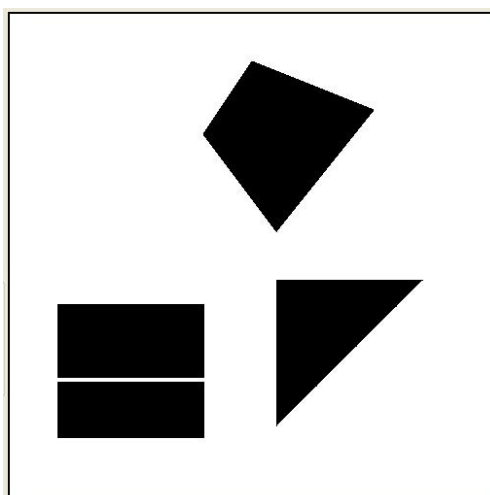


Figura 30 - Mapa 3 escala 500x500

Outro ponto importante é a escala usada, pois o software divide o espaço do mapa em um número determinado de células, e dependendo do número desses pode mudar. Tal efeito pode ser observado na comparação da Figura 1 com a Figura 30. O método proposto por Prestes visto em (19) também requer a criação de um outro arquivo de entrada o qual descreve áreas de alta ou baixa preferência, para tal o software importa um segundo arquivo descrevendo tais áreas. Vemos os arquivos de mapa de preferência na Tabela 2.

Para criar o arquivo seguem-se 6 passos simples:

1 – O mapa começa com a tag 2D, pois todos os mapas e as áreas são bidimensionais.

2 – Descreve-se a menor posição e logo a maior no formato (x, y) Para o simulador isso representa milímetros.

3 – Descreve-se o número de áreas existentes com o objetivo que se deseja adicionar no mapa.

4 – Escreve-se a tag GOAL indicando o início da descrição das áreas objetivo.

5 – Descreve-se uma área objetivo por linha descrevendo o ponto superior a esquerda e o ponto inferior a direita de cada área no formato  $G_i(x_i, y_i) G_f(x_f, y_f)$

3 – Descreve-se o número de áreas existentes com preferencias que se deseja adicionar no mapa.

6 – Escreve-se a tag POINTS indicando o início da descrição das áreas de preferencia.

7 – Descreve-se uma área de preferencia por linha descrevendo o ponto superior a esquerda e o ponto inferior a direita de cada área seguido do valor de  $\varepsilon$  no formato  $A_i(x_i, y_i) A_f(x_f, y_f) \varepsilon$

8 – O Arquivo termina com a tag END.

Tabela 2 - Arquivos de Preferência

Arquivo de Preferencia 1	Arquivo de Preferencia 2
<pre> 2D-Map LineMinPos: -20000 -20000 LineMaxPos: 20000 20000 NumGoal: 1 GOAL -19000 -19000 -19500 -19500 NumPoint: 5 POINTS 5000 -15000 4000 -16000 0.5 7500 7500 10000 10000 0.5 -5000 -5000 -4000 -4000 -0.5 -15500 +17500 -10000 +10000 -0.5 -18000 -18500 18000 -19000 0.5 END                     </pre>	<pre> 2D-Map LineMinPos: -20000 -20000 LineMaxPos: 20000 20000 NumGoal: 1 GOAL -3000 -3000 -1000 -1000 NumPoint: 5 POINTS 5000 -15000 4000 -16000 0.5 7500 7500 10000 10000 0.5 -5000 -5000 -4000 -4000 -0.5 -15500 +17500 -10000 +10000 -0.5 -18000 -18500 18000 -19000 0.5 END                     </pre>

Tais mapas de preferencias podem ser usados em qualquer mapa, vemos os resultados da adiç o dos mapas de preferencia acima aos mapas constando nas Figura 28 e Figura 29.

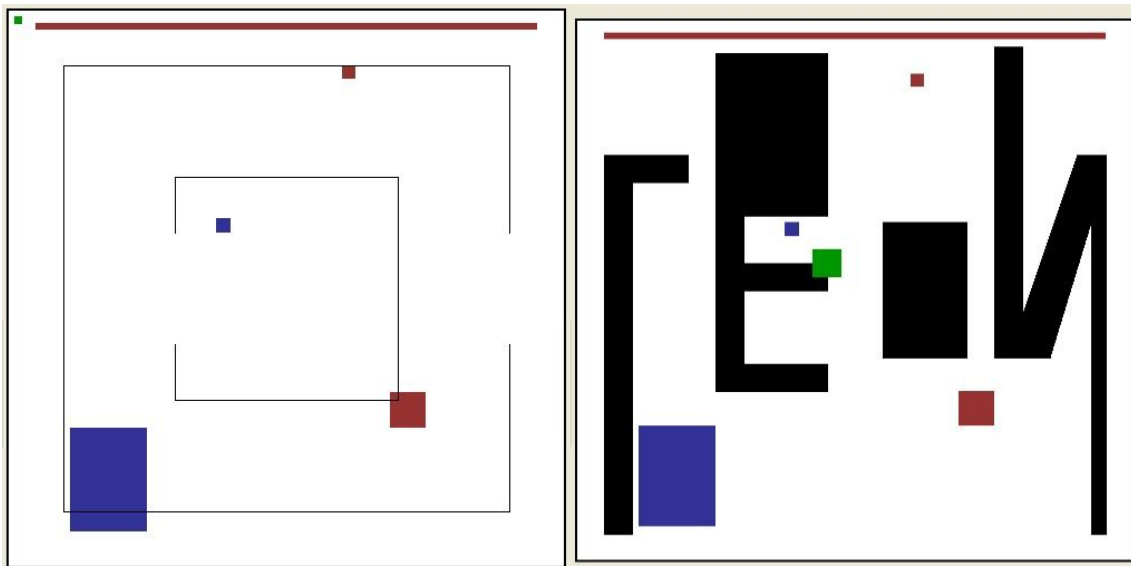


Figura 31 - Mapas com Prefer ncias

O ultimo item a se observar   a posiç o inicial do rob , pois como o mapa   apenas descrito como um conjunto de linhas, n o se sabe diretamente quais as posiç es inalcanç veis. Vemos na Figura 32 o que acontece no mapa quando o rob    iniciado no meio da letra E no mapa da Figura 28.



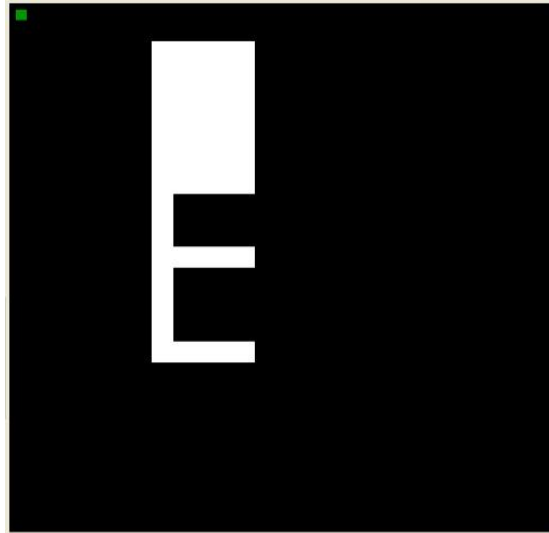


Figura 32 - Mapa 1

## 4.2 Algoritmos

O software trabalha em duas partes. Inicialmente ele carrega o mapa e as preferências em uma matriz e logo após ele passa o algoritmo escolhido e o armazena em outra. Os métodos escolhidos para a simulação são:

- 1 – Método Iterativo Gauss-Seidel;
- 2 – Método de Relaxamento SOR.

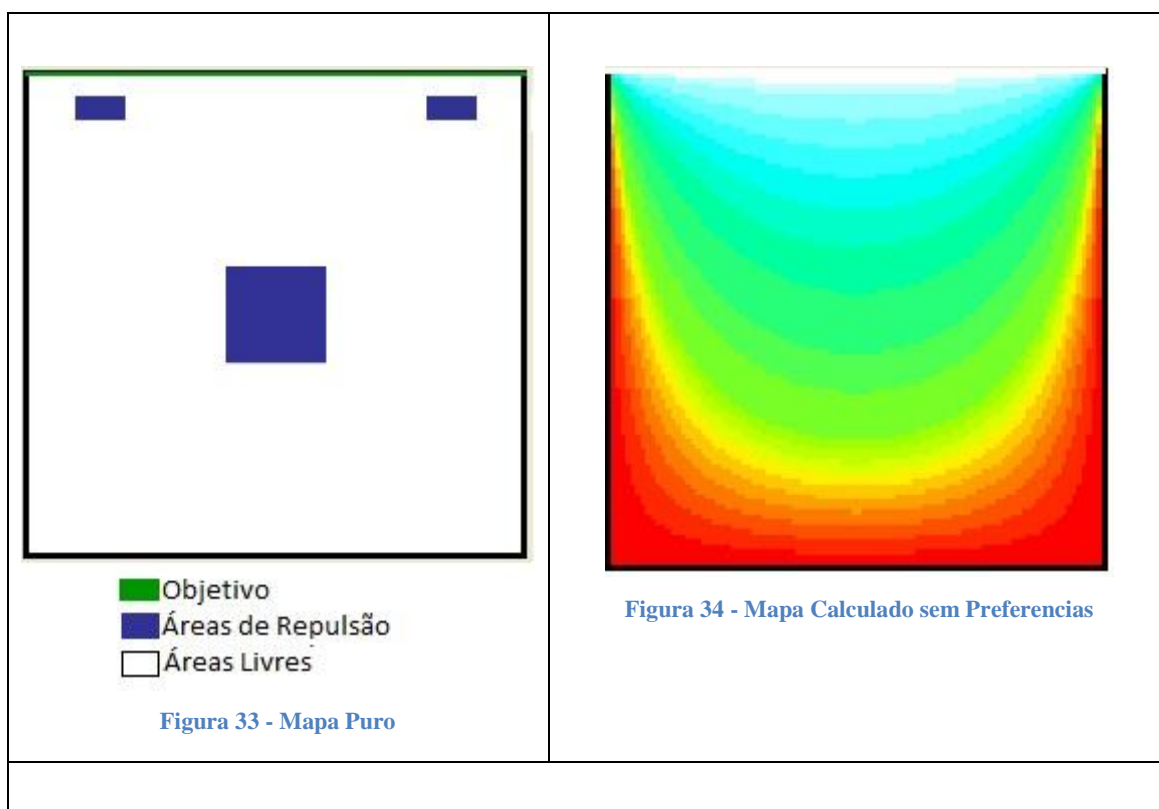
Cada método será executado de três maneiras diferentes. A primeira resume-se ao método clássico proposto por Cannoly e Grupen em (8) sem o uso de áreas de preferências. Os outros dois métodos baseiam-se na proposta de Prestes em (19) e em melhorias propostas posteriores. Além disso cada um desses algoritmos será iterado de duas maneiras diferentes, uma sendo iterado sempre de cima abaixo da esquerda para a direita e outra com ele sendo iterado pelos quatro lados a cada iteração. Isso significa que haverá doze algoritmos:

- 1º: Harmonic Functions Path Planner Iterado com Gauss-Seidel simples;
- 2º: Harmonic Functions Path Planner Iterado com Gauss-Seidel simultâneo;
- 3º: Harmonic Functions Path Planner Iterado com SOR simples;
- 4º: Harmonic Functions Path Planner Iterado com SOR simultâneo;
- 5º: BVP Iterado com Gauss-Seidel simples;
- 6º: BVP Iterado com Gauss-Seidel simultâneo;
- 7º: BVP Iterado com SOR simples;
- 8º: BVP Iterado com SOR simultâneo;

- 9º: BVP melhorado Iterado com Gauss-Seidel simples;
- 10º: BVP melhorado Iterado com Gauss-Seidel simultâneo;
- 11º: BVP melhorado Iterado com SOR simples;
- 12º: BVP melhorado Iterado com SOR simultâneo.

Os resultados da aplicação destes algoritmos são praticamente os mesmos, porém o número de iterações, o decrescimento do erro e os estados intermediários entre o início e o final do cálculo variam bastante. Neste trabalho serão apresentados os resultados da simulação dos algoritmos e algumas dessas características. Na Figura 35 mostra-se o resultado final do algoritmo, vale ressaltar que este é praticamente igual independente do método utilizado, desde que leve-se em conta o uso ou não uso de áreas de atração e repulsão. Podemos ver a atuação de campos de preferências na Tabela 3.

Tabela 3- Mapa com preferencias



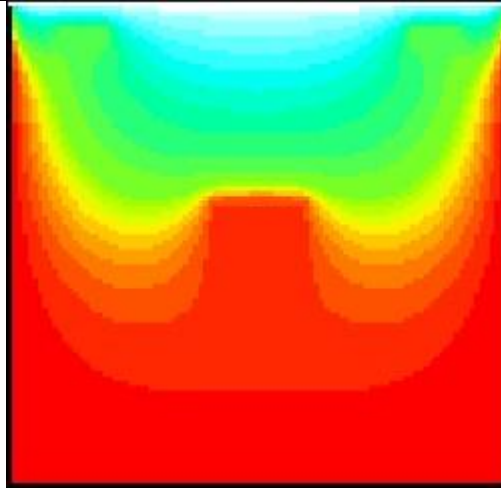


Figura 35 - Mapa Calculado com Preferência

Nas próximas sessões descreveremos os algoritmos individuais e como foram implementados.

#### 4.2.1 Harmonic Functions Path Planner Iterado com Gauss-Seidel simples

Este é o algoritmo mais simples e mais usado. Na Tabela 4 vemos o método implementado para o primeiro algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno. Vemos o exemplo de loop eterno no algoritmo no final da sessão 3.4, tal algoritmo roda em um robô enquanto ele se move e descobre novos objetos com os quais ele recalcula o novo mapa. A convergência, vista no algoritmo na linha 11, resume-se a diferença entre o somatório de erros na rodada anterior e na ultima rodada calculada. Caso tal diferença seja menor que um valor fixo é dito que o algoritmo convergiu.

Tabela 4 - Algoritmo GS simples

Algoritmo	
1:	<b>for all</b> cell c that contains obstacle <b>do</b>
2:	set its potential value to high potential      → $p(c) = 1$
3:	<b>end for</b>
4:	<b>for all</b> cell c that contains goal <b>do</b>
5:	set its potential value to low potential      → $p(c) = 0$
6:	<b>end for</b>
7:	<b>for all</b> cell c <b>do</b>
8:	initialize $\varepsilon(c) \leftarrow \text{preference}(c)$ ,
9:	initialize $v(c)$

```

10: end for

11: while not converged
12:   for all cell c that represents an environment free-space do           left, down3
13:      $p(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$ 
14:   end for
15: end while

16: END

```

#### 4.2.2 Harmonic Functions Path Planner Iterado com Gauss-Seidel simultâneo

Este algoritmo usa o Gauss-Seidel, porém muda a iteração, vista no passo 12, em uma tentativa de diminuir o número de iterações necessárias, fazendo com que o algoritmo convirja mais rápido. Na Tabela 5 vemos o algoritmo implementado para o segundo algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno.

Tabela 5 - Algoritmo GS simultâneo

Algoritmo	
1: <b>for all</b> cell c that contains obstacle <b>do</b>	
2:   set its potential value to high potential	→ p(c) = 1
3: <b>end for</b>	
4: <b>for all</b> cell c that contains goal <b>do</b>	
5:   set its potential value to low potential	→ p(c) = 0
6: <b>end for</b>	
7: <b>for all</b> cell c <b>do</b>	
8:   initialize $\varepsilon(c) \leftarrow \text{preference}(c)$ ,	
9:   initialize v(c)	
10: <b>end for</b>	

<sup>3</sup> Left, down significa que a cada rodada o valor c inicia com (1,1) passando para (1,2), (1,3) até (1,N) logo após passa pra (2,1) e então (2,2) até (N,N).

```

11: while not converged
12:   for all cell c that represents an environment free-space do           alter4
13:      $p(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$ 
14:   end for
15: end while

16: END

```

### 4.2.3 Harmonic Functions Path Planner Iterado com SOR simples

Este é o algoritmo simples e mais usado quando não é necessária a atuação do robô até o modelo ter convergido devido ao fato de usar o método SOR. É igual ao algoritmo apresentado na sessão anterior, porém muda o método de resolução da equação. Na Tabela 6 vemos o algoritmo implementado para o terceiro algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno. Calculamos  $w$  como proposto por Prestes em (21).

$$w = \frac{4}{2 + \sqrt{4 - c^2}} \rightarrow c = \cos \frac{\pi}{m} + \cos \frac{\pi}{n} \quad \text{onde } m \text{ e } n \text{ representam o número de linhas e colunas do mapa}$$

Tabela 6 - Algoritmo SOR simples

Algoritmo	
1:	<b>for all</b> cell c that contains obstacle <b>do</b>
2:	set its potential value to high potential → $p(c) = 1$
3:	<b>end for</b>
4:	<b>for all</b> cell c that contains goal <b>do</b>
5:	set its potential value to low potential → $p(c) = 0$
6:	<b>end for</b>

<sup>4</sup> Alter: significa que na primeira rodada o valor c inicia com (1,1) passando para (1,2), (1,3) até (1,N) logo após passa pra (2,1) e então (2,2) até (N,N). Na segunda rodada o valor c inicia com (1,N) passando para (1,N-1), (1,N-2) até (1,1) logo após passa pra (2,N) e então (2,N-1) até (N,1). Na terceira rodada o valor c inicia com (N,1) passando para (N,2), (N,3) até (N,N) logo após passa pra (N-1,1) e então (N-2,2) até (N,N) e assim por diante.

```

7: for all cell c do
8:   initialize  $\varepsilon(c) \leftarrow \text{preference}(c)$ ,
9:   initialize  $v(c)$ 
10: end for

11: while not converged
12:   for all cell c that represents an environment free-space do           left,down
13:      $p(c) \leftarrow p(c) - w \left( \frac{P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c)}{4} \right)$ 
14:   end for
15: end while

16: END

```

#### 4.2.4 Harmonic Functions Path Planner Iterado com SOR simultaneo

Este é o algoritmo usa o SOR, porém muda a iteração, no passo 12, em uma tentativa de diminuir o número de iterações necessárias, fazendo com que o algoritmo convirja mais rápido. Na Tabela 7 vemos o algoritmo implementado para o quarto algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno. Calculamos  $w$  como proposto por Prestes em (21).

$$w = \frac{4}{2 + \sqrt{4 - c^2}} \rightarrow c = \cos \frac{\pi}{m} + \cos \frac{\pi}{n} \quad \text{onde } m \text{ e } n \text{ representam o número de linhas e colunas do mapa}$$

Tabela 7 - Algoritmo SOR simultaneo

Algoritmo	
1: <b>for all</b> cell c that contains obstacle <b>do</b>	
2:   set its potential value to high potential	→ $p(c) = 1$
3: <b>end for</b>	
4: <b>for all</b> cell c that contains goal <b>do</b>	
5:   set its potential value to low potential	→ $p(c) = 0$
6: <b>end for</b>	

```

7: for all cell c do
8:   initialize  $\varepsilon(c) \leftarrow \text{preference}(c)$ ,
9:   initialize  $v(c)$ 
10: end for

11: while not converged
12:   for all cell c that represents an environment free-space do           alter
13:      $p(c) \leftarrow p(c) - w\left(\frac{P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c)}{4}\right)$ 
14:   end for
15: end while

16: END

```

#### 4.2.5 BVP iterado com GS simples

Algoritmo simples baseado na proposta de Prestes vista em (19). Na Tabela 8 vemos o algoritmo implementado para o quinto algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno.

Tabela 8 - BVP usando GS simples

Algoritmo
<pre> 1: <b>for all</b> cell c that contains obstacle <b>do</b> 2:   set its potential value to high potential   →   <math>p(c) = 1</math> 3: <b>end for</b>  4: <b>for all</b> cell c that contains goal <b>do</b> 5:   set its potential value to low potential   →   <math>p(c) = 0</math> 6: <b>end for</b>  7: <b>for all</b> cell c <b>do</b> 8:   initialize <math>\varepsilon(c) \leftarrow \text{preference}(c)</math>, 9:   initialize <math>v(c)</math> 10: <b>end for</b> </pre>

```

11: while not converged
12:   for all cell c that represents an environment free-space do           left, down
13:      $p(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$ 
14:   end for
15: end while

16: while not converged
17:   for all cell c that represents an environment free-space do           left, down
18:      $h(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$ 
19:      $d(c) \leftarrow (P_{cl} - P_{cw})v_x(c) + (P_{cn} - P_{cs})v_y(c)$ 
20:      $p(c) \leftarrow h(c) + \frac{\varepsilon(c)}{8}d(c)$ 
21:   end for
22: end while

23: END

```

#### 4.2.6 BVP iterado com GS simultâneo

Algoritmo simples baseado na proposta de Prestes vista em (19), porém muda a iteração, no passo 12, em uma tentativa de diminuir o número de iterações necessárias, fazendo com que o algoritmo convirja mais rápido. Na Tabela 8 vemos o algoritmo implementado para o sexto algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno.

Tabela 9 - BVP com GS simultâneo

Algoritmo	
1:	<b>for all</b> cell c that contains obstacle <b>do</b>
2:	set its potential value to high potential → $p(c) = 1$
3:	<b>end for</b>
4:	<b>for all</b> cell c that contains goal <b>do</b>
5:	set its potential value to low potential → $p(c) = 0$



```

6: end for

7: for all cell c do
8:   initialize  $\varepsilon(c) \leftarrow \text{preference}(c)$ ,
9:   initialize  $v(c)$ 
10: end for

11: while not converged
12:   for all cell c that represents an environment free-space do           alter
13:      $p(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$ 
14:   end for
15: end while

16: while not converged
17:   for all cell c that represents an environment free-space do           alter
18:      $h(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$ 
19:      $d(c) \leftarrow (P_{cl}(c) - P_{cw}(c))v_x(c) + (P_{cn}(c) - P_{cs}(c))v_y(c)$ 
20:      $p(c) \leftarrow h(c) + \frac{\varepsilon(c)}{8}d(c)$ 
21:   end for
22: end while

23: END

```

#### 4.2.7 BVP iterado com SOR simples

Algoritmo simples baseado na proposta de Prestes vista em (19) utilizando um método de iteração baseado em SOR. Na Tabela 8 vemos o algoritmo implementado para o sétimo algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno. Calculamos  $w$  como proposto por Prestes em (21).

$$w = \frac{4}{2 + \sqrt{4 - c^2}} \rightarrow c = \cos \frac{\pi}{m} + \cos \frac{\pi}{n} \quad \text{onde } m \text{ e } n \text{ representam o número de linhas e colunas do mapa}$$

Tabela 10 - BVP com SOR simples

Algoritmo	
1:	<b>for all</b> cell c that contains obstacle <b>do</b>
2:	set its potential value to high potential      → $p(c) = 1$
3:	<b>end for</b>
4:	<b>for all</b> cell c that contains goal <b>do</b>
5:	set its potential value to low potential      → $p(c) = 0$
6:	<b>end for</b>
7:	<b>for all</b> cell c <b>do</b>
8:	initialize $\varepsilon(c) \leftarrow \text{preference}(c)$ ,
9:	initialize $v(c)$
10:	<b>end for</b>
11:	<b>while</b> not converged
12:	<b>for all</b> cell c that represents an environment free-space <b>do</b> left, down
13:	$p(c) \leftarrow p(c) - w \left( \frac{P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c)}{4} \right)$
14:	<b>end for</b>
15:	<b>end while</b>
16:	<b>while</b> not converged
17:	<b>for all</b> cell c that represents an environment free-space <b>do</b> left, down
18:	$h(c) \leftarrow \frac{w}{4} (P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c))$
19:	$d(c) \leftarrow (P_{cl}(c) - P_{cw}(c))v_x(c) + (P_{cn}(c) - P_{cs}(c))v_y(c)$
20:	$p(c) \leftarrow h(c) + \frac{\varepsilon(c)}{8} d(c)$
21:	<b>end for</b>
22:	<b>end while</b>
23:	<b>END</b>

#### 4.2.8 BVP iterado com SOR simultâneo

Algoritmo simples baseado na proposta de Prestes vista em (19) utilizando um método de iteração baseado em SOR (19), porém muda a iteração, no passo 12 e 17, em uma tentativa de diminuir o número de iterações necessárias, fazendo com que o algoritmo convirja mais rápido. Na Tabela 11Tabela 8 vemos o algoritmo implementado para o oitavo algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno. Calculamos  $w$  como proposto por Prestes em (21).

Tabela 11 - BVP com SOR simultâneo

Algoritmo	
1:	<b>for all</b> cell $c$ that contains obstacle <b>do</b>
2:	set its potential value to high potential $\rightarrow p(c) = 1$
3:	<b>end for</b>
4:	<b>for all</b> cell $c$ that contains goal <b>do</b>
5:	set its potential value to low potential $\rightarrow p(c) = 0$
6:	<b>end for</b>
7:	<b>for all</b> cell $c$ <b>do</b>
8:	initialize $\varepsilon(c) \leftarrow \text{preference}(c)$ ,
9:	initialize $v(c)$
10:	<b>end for</b>
11:	<b>while</b> not converged
12:	<b>for all</b> cell $c$ that represents an environment free-space <b>do</b> alter
13:	$p(c) \leftarrow p(c) - w \left( \frac{P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c)}{4} \right)$
14:	<b>end for</b>
15:	<b>end while</b>
16:	<b>while</b> not converged
17:	<b>for all</b> cell $c$ that represents an environment free-space <b>do</b> alter
18:	$h(c) \leftarrow \frac{w}{4} (P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c))$
19:	$d(c) \leftarrow (P_{cl}(c) - P_{cw}(c))v_x(c) + (P_{cn}(c) - P_{cs}(c))v_y(c)$
20:	$p(c) \leftarrow h(c) + \frac{\varepsilon(c)}{8} d(c)$
21:	<b>end for</b>

22: **end while**

23: **END**

#### 4.2.9 BVP alterado iterado com GS simples

Algoritmo simples baseado na proposta de Prestes vista em (19), porém sem a utilização do vetor  $v$  como entrada. Na Tabela 12 vemos o algoritmo implementado para o nono algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno.

Tabela 12 - BVP 2 com GS simples

Algoritmo	
1: <b>for all</b> cell $c$ that contains obstacle <b>do</b>	
2:     set its potential value to high potential	→ $p(c) = 1$
3: <b>end for</b>	
4: <b>for all</b> cell $c$ that contains goal <b>do</b>	
5:     set its potential value to low potential	→ $p(c) = 0$
6: <b>end for</b>	
7: <b>for all</b> cell $c$ <b>do</b>	
8:     initialize $\varepsilon(c) \leftarrow \text{preference}(c)$ ,	
9: <b>end for</b>	
10: <b>while</b> not converged	
11: <b>for all</b> cell $c$ that represents an environment free-space <b>do</b>	left, down
12: $p(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$	
13: <b>end for</b>	
14: <b>end while</b>	
15: <b>while</b> not converged	
16: <b>for all</b> cell $c$ that represents an environment free-space <b>do</b>	left, down
17: $h(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$	

```

18:       $d(c) \leftarrow \frac{|(P_{cl} - P_{cw})|}{2} + \frac{|(P_{cn} - P_{cs})|}{2}$ 
19:       $p(c) \leftarrow h(c) + \frac{\varepsilon(c)}{4} d(c)$ 
20:  end for
21: end while

22: END

```

#### 4.2.10 BVP alterado iterado com GS simultâneo

Algoritmo simples baseado na proposta de Prestes vista em (19), porém sem a utilização do vetor  $v$  como entrada Seidel, porém muda a iteração, no passo 12 e 17, em uma tentativa de diminuir o número de iterações necessárias, fazendo com que o algoritmo convirja mais rápido. Na Tabela 13 vemos o algoritmo implementado para o décimo algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno.

Tabela 13 - BVP 2 com GS simultâneo

Algoritmo	
1:	<b>for all</b> cell $c$ that contains obstacle <b>do</b>
2:	set its potential value to high potential $\rightarrow p(c) = 1$
3:	<b>end for</b>
4:	<b>for all</b> cell $c$ that contains goal <b>do</b>
5:	set its potential value to low potential $\rightarrow p(c) = 0$
6:	<b>end for</b>
7:	<b>for all</b> cell $c$ <b>do</b>
8:	initialize $\varepsilon(c) \leftarrow \text{preference}(c)$ ,
9:	initialize $v(c)$
10:	<b>end for</b>
11:	<b>while</b> not converged
12:	<b>for all</b> cell $c$ that represents an environment free-space <b>do</b> alter

```

13:       $p(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$ 
14:  end for
15: end while

16: while not converged
17:  for all cell c that represents an environment free-space do      alter
18:       $h(c) \leftarrow \frac{1}{4}(P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c))$ 
19:       $d(c) \leftarrow \frac{|(P_{cl}-P_{cw})|}{2} + \frac{|(P_{cn}-P_{cs})|}{2}$ 
20:       $p(c) \leftarrow h(c) + \frac{\varepsilon(c)}{4}d(c)$ 
21:  end for
22: end while

23: END

```

#### 4.2.11 BVP alterado iterado com SOR simples

Algoritmo simples baseado na proposta de Prestes vista em (19) utilizado o iterador SOR. Na Tabela 13 vemos o algoritmo implementado para o décimo primeiro algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno.

Tabela 14 - BVP 2 com SOR simples

Algoritmo	
1:	<b>for all</b> cell c that contains obstacle <b>do</b>
2:	set its potential value to high potential → $p(c) = 1$
3:	<b>end for</b>
4:	<b>for all</b> cell c that contains goal <b>do</b>
5:	set its potential value to low potential → $p(c) = 0$
6:	<b>end for</b>
7:	<b>for all</b> cell c <b>do</b>
8:	initialize $\varepsilon(c) \leftarrow \text{preference}(c)$ ,

```

9:   initialize v(c)
10: end for

11: while not converged
12:   for all cell c that represents an environment free-space do           left, down
13:      $p(c) \leftarrow p(c) - w \left( \frac{P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c)}{4} \right)$ 
14:   end for
15: end while

16: while not converged
17:   for all cell c that represents an environment free-space do           left, down
18:      $h(c) \leftarrow \frac{w}{4} (P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c))$ 
19:      $d(c) \leftarrow \frac{|(P_{cl}(c) - P_{cw}(c))|}{2} + \frac{|(P_{cn}(c) - P_{cs}(c))|}{2}$ 
20:      $p(c) \leftarrow h(c) + \frac{\varepsilon(c)}{4} d(c)$ 
21:   end for
22: end while

23: END

```

#### 4.2.12 BVP alterado iterado com SOR simultâneo

Algoritmo simples baseado na proposta de Prestes vista em (19) utilizado o iterador SOR, porém muda a iteração, no passo 12 e 17, em uma tentativa de diminuir o número de iterações necessárias, fazendo com que o algoritmo convirja mais rápido. Na Tabela 13 vemos o algoritmo implementado para o décimo primeiro algoritmo. Visto que este programa apenas calcula os mapas e não roda em tempo real não é necessário um loop eterno.

Tabela 15 - BVP 2 com SOR simultâneo

Algoritmo
<pre> 1: for all cell c that contains obstacle do 2:   set its potential value to high potential   →   p(c) = 1 3: end for </pre>

```

4: for all cell c that contains goal do
5:   set its potential value to low potential      →   p(c) = 0
6: end for

7: for all cell c do
8:   initialize  $\varepsilon(c) \leftarrow \text{preference}(c)$ ,
9:   initialize v(c)
10: end for

11: while not converged
12:   for all cell c that represents an environment free-space do           alter
13:      $p(c) \leftarrow p(c) - w \left( \frac{P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c)}{4} \right)$ 
14:   end for
15: end while

16: while not converged
17:   for all cell c that represents an environment free-space do           alter
18:      $h(c) \leftarrow \frac{w}{4} (P_{cn}(c) + P_{cs}(c) + P_{cw}(c) + P_{cl}(c) - 4p(c))$ 
19:      $d(c) \leftarrow \frac{|(P_{cl}(c) - P_{cw}(c))|}{2} + \frac{|(P_{cn}(c) - P_{cs}(c))|}{2}$ 
20:      $p(c) \leftarrow h(c) + \frac{\varepsilon(c)}{4} d(c)$ 
21:   end for
22: end while

23: END

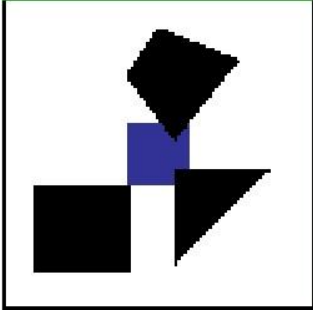
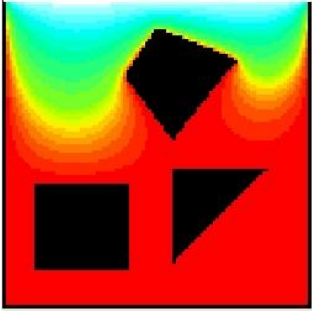
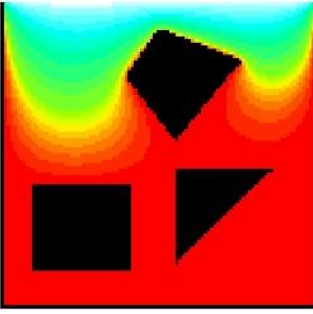
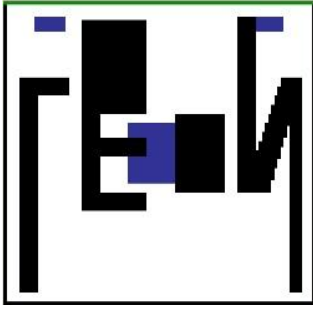
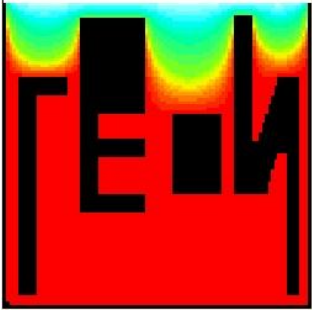
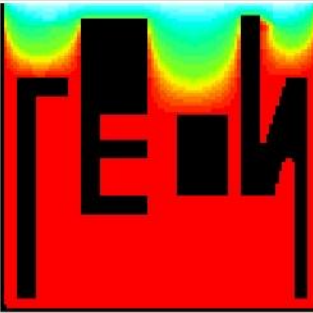
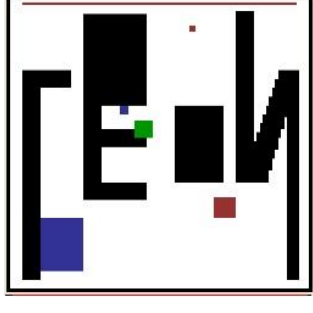
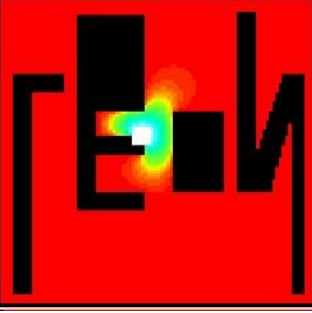
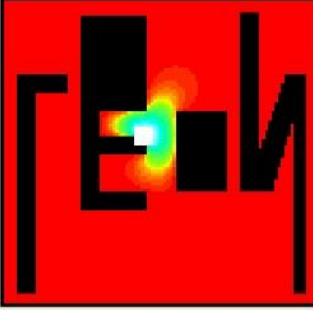
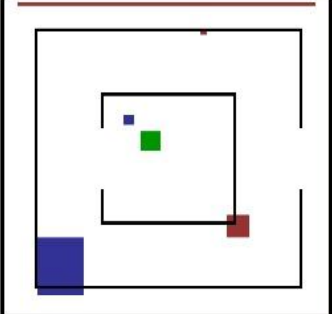
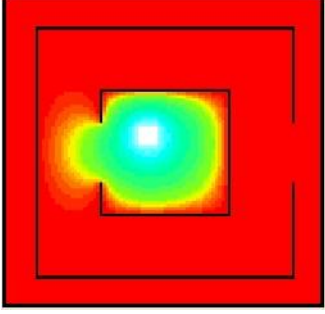
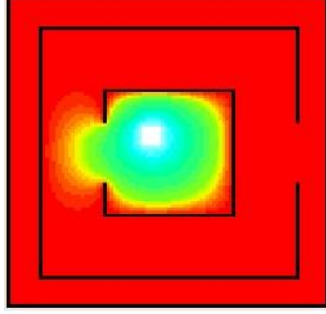
```

### 4.3 Resultados

Como dito anteriormente o resultado final é sempre muito parecido independente do método, por isso na Tabela 16 será mostrado o mapa com as preferencias e o resultado final apenas. O resultado representa as linhas do campo potencial, um robô ao ser deixado em qualquer área livre tentará sempre ir em direção ao objetivo branco descendo tais linhas potencias em busca do valor zero.



Tabela 16 - Resultados

Mapa Puro	Sem Preferências	Com Preferências
		
		
		
		

--	--	--

Um dos principais objetivos destes algoritmos é a rápida convergência, ou seja, a tentativa de reduzir o número de iterações ao máximo para acelerar o funcionamento do robô. O SOR tende a ter menos iterações do que o GS, como vistos na próxima sessão.

### 4.3.1 Resultados no Mapa 1

Os resultados a seguir são baseados no cálculo usando o ambiente da Figura 36. Podemos observar os resultados na Tabela 17.

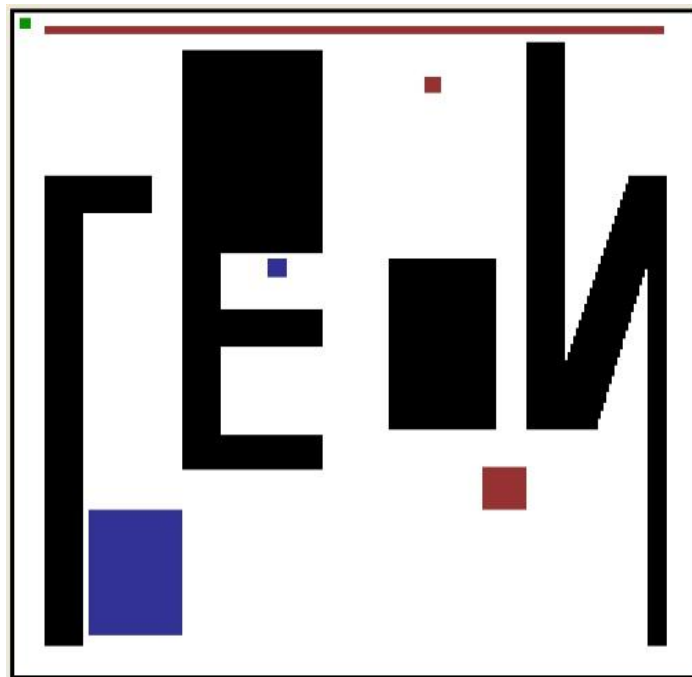


Figura 36 - Ambiente usado Mapa 1

Tabela 17 - Resultados do mapa 1

Algoritmo x Escala	Número de iterações antes da preferencia		Número de iterações depois da preferencia	
	100x100	250x250	100x100	250x250
HFPP GS simples	1408	8924	x <sup>5</sup>	x
HFPP GS simultâneo	1411	8929	x	x
HFPP SOR simples	124	355	x	x
HFPP SOR simultâneo	194	512	x	x

<sup>5</sup> O método é baseado no HFPP, o qual não calcula preferencias.

BVP GS simples	1408	8924	1495	9821
BVP GS simultâneo	1411	8929	1502	9838
BVP SOR simples	124	355	181	569
BVP SOR simultâneo	194	512	237	783
BVP 2 GS simples	1408	8924	1542	10373
BVP 2 GS simultâneo	1411	8929	1551	10393
BVP 2 SOR simples	124	355	270	679
BVP 2 SOR simultâneo	194	512	310	686

### 4.3.2 Resultados no Mapa 2

Os resultados a seguir são baseados no cálculo usando o ambiente da Figura 37. Podemos observar os resultados na Tabela 18 - Resultados no Mapa 2.

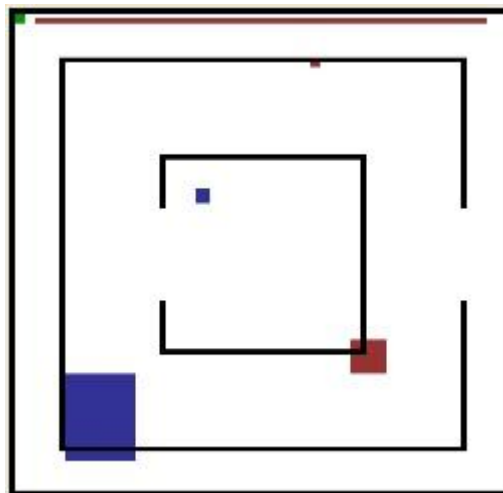


Figura 37 - Ambiente usando Mapa 2

Tabela 18 - Resultados no Mapa 2

Algoritmo x Escala	Número de iterações antes da preferencia		Número de iterações depois da preferencia	
	100x100	250x250	100x100	250x250
HFPP GS simples	884	5572	x	x
HFPP GS simultâneo	887	5575	x	x
HFPP SOR simples	166	434	x	x
HFPP SOR simultâneo	189	477	x	x
BVP GS simples	884	5572	919	5840

BVP GS simultâneo	887	5575	923	5846
BVP SOR simples	166	434	238	669
BVP SOR simultâneo	189	477	242	676
BVP 2 GS simples	884	5572	1100	7352
BVP 2 GS simultâneo	887	5575	1106	7363
BVP 2 SOR simples	166	434	266	876
BVP 2 SOR simultâneo	189	477	277	886

### 4.3.3 Resultados no Mapa 3

Os resultados a seguir são baseados no cálculo usando o ambiente da figura. Podemos observar os resultados na tabela.

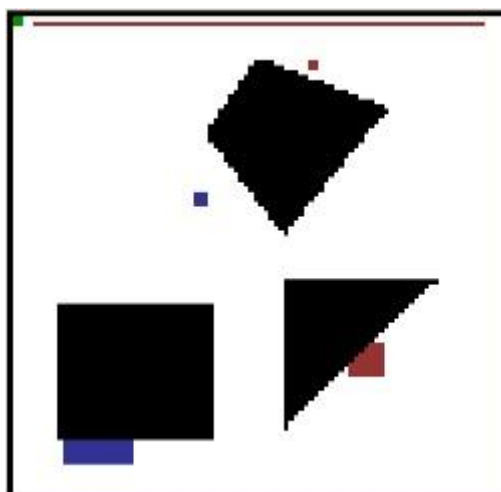


Figura 38 - Ambiente usando Mapa 3

Tabela 19 - Resultados no Mapa 3

Algoritmo x Escala	Número de iterações antes da preferencia		Número de iterações antes da preferencia	
	100x100	250x250	100x100	250x250
HFPP GS simples	2287	14369	x	x
HFPP GS simultâneo	2292	14360	x	x
HFPP SOR simples	189	471	x	x
HFPP SOR simultâneo	270	750	x	x
BVP GS simples	2287	14369	2689	18816

BVP GS simultâneo	2292	14360	2707	18866
BVP SOR simples	189	471	279	731
BVP SOR simultâneo	270	750	366	1565
BVP 2 GS simples	2287	14369	2914	22134
BVP 2 GS simultâneo	2292	14360	2939	22156
BVP 2 SOR simples	189	471	288	841
BVP 2 SOR simultâneo	270	750	378	1312

#### 4.4 Convergência de dados

Como visto na sessão anterior, o número de iterações varia drasticamente de um método para o outro, o que ocorre devido ao fato do algoritmo SOR possuir um fator de aceleração. A tentativa de modificar a iteração não foi bem sucedida e acabou aumentando as iterações necessárias para a convergência do método. Tal modificação porém é útil ao rodar o algoritmo em tempo real, pois a atualização do mapa acontece de modo mais rápida a todos os lados do mapa. Na Figura 39 - Gráfico do erro no Mapa 1 vemos como o erro se comporta para o Ambiente com o Mapa 1 em uma escala de 100x100 células.

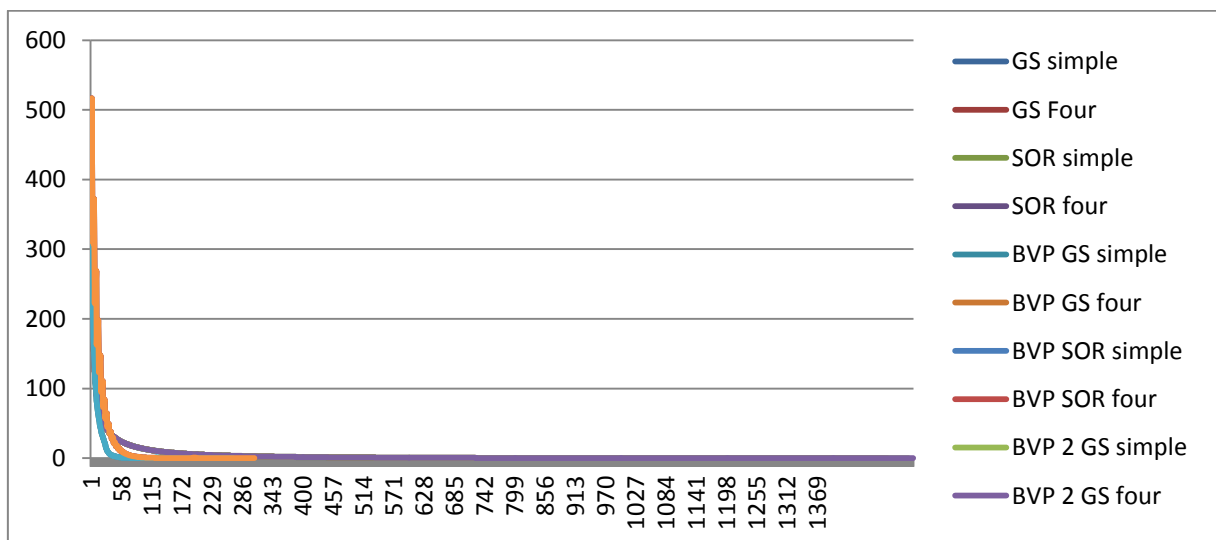


Figura 39 - Gráfico do erro no Mapa 1

Como se pode perceber os algoritmos envolvendo SOR termina mais rápido e convergem mais rápido. Na Figura 40 - Gráfico do erro no Mapa 1 na 20ª iteração vemos o mesmo gráfico, porém a partir da 21ª iteração. Na figura vemos o mesmo gráfico, porém a partir da 41ª iteração. Na Figura 41 vemos os pontos aonde dois métodos convergiram pela primeira vez e as preferências começaram a ser calculadas.

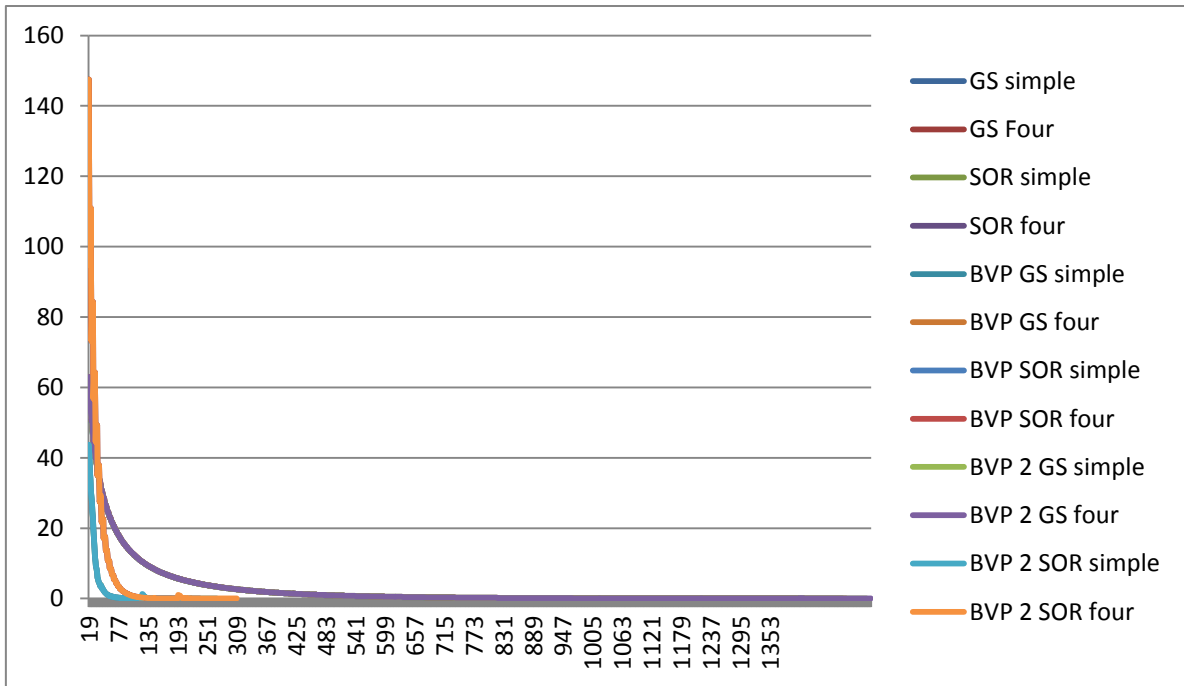


Figura 40 - Gráfico do erro no Mapa 1 na 20ª iteração

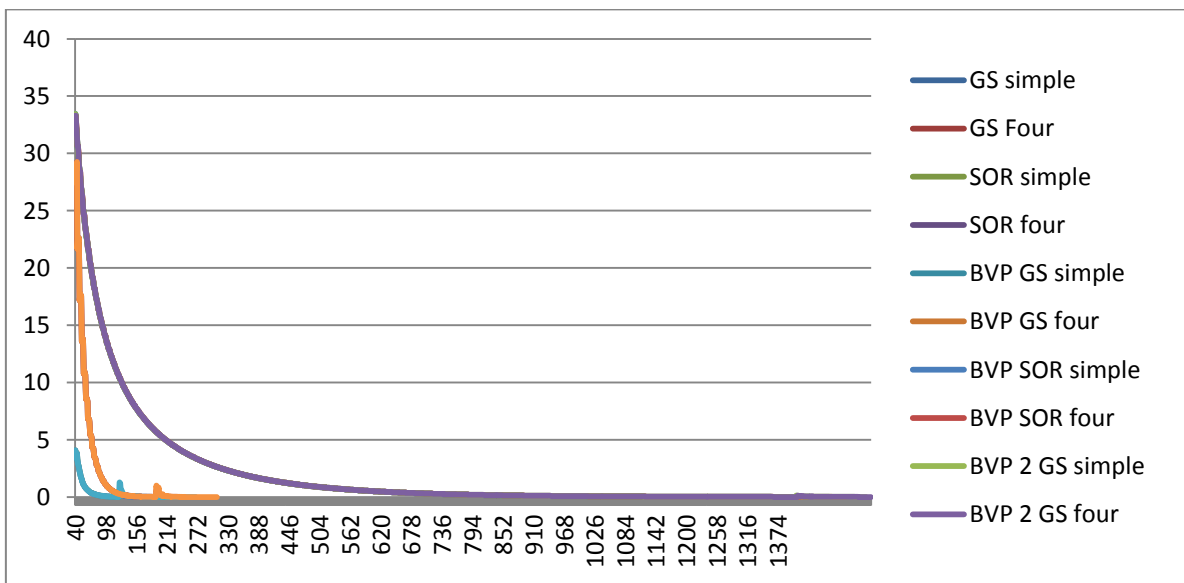


Figura 41 - Gráfico do erro no Mapa 1 na 40ª iteração

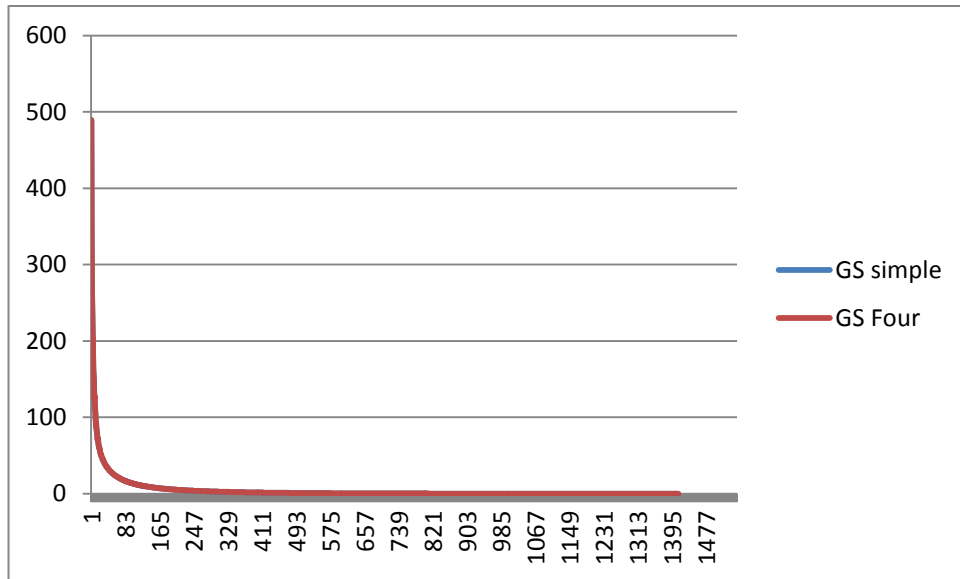


Figura 42: Decrescimento do erro nos algoritmos GS Simples e GS Four.

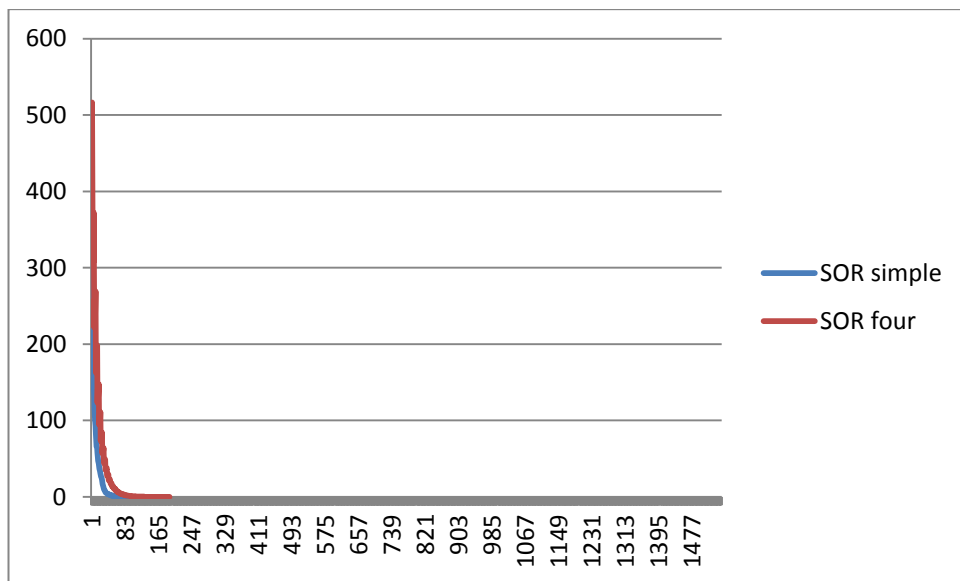


Figura 43: Decrescimento do erro nos algoritmos SOR Simples e SOR Four.

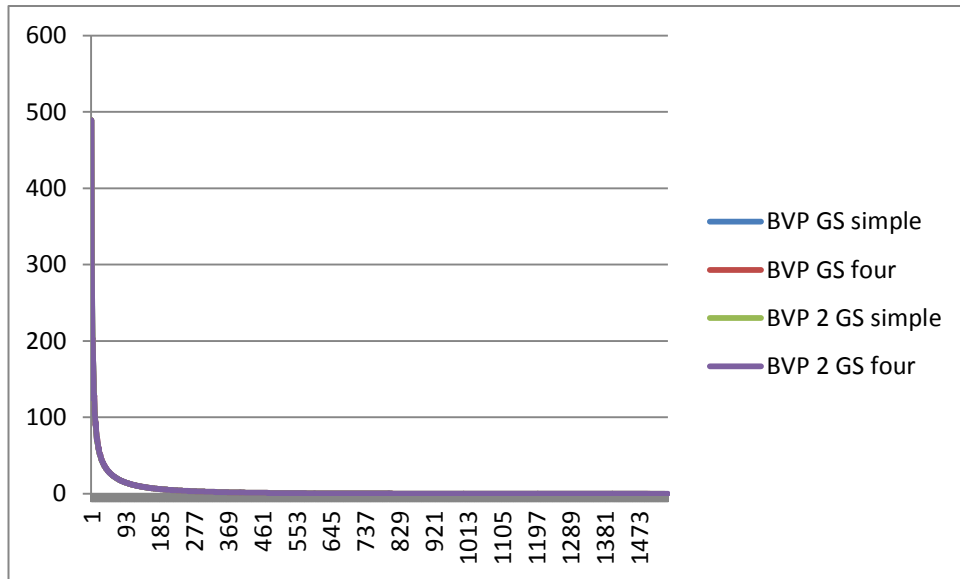


Figure 44 - Decrescimento do erro nos algoritmos BVP GS`

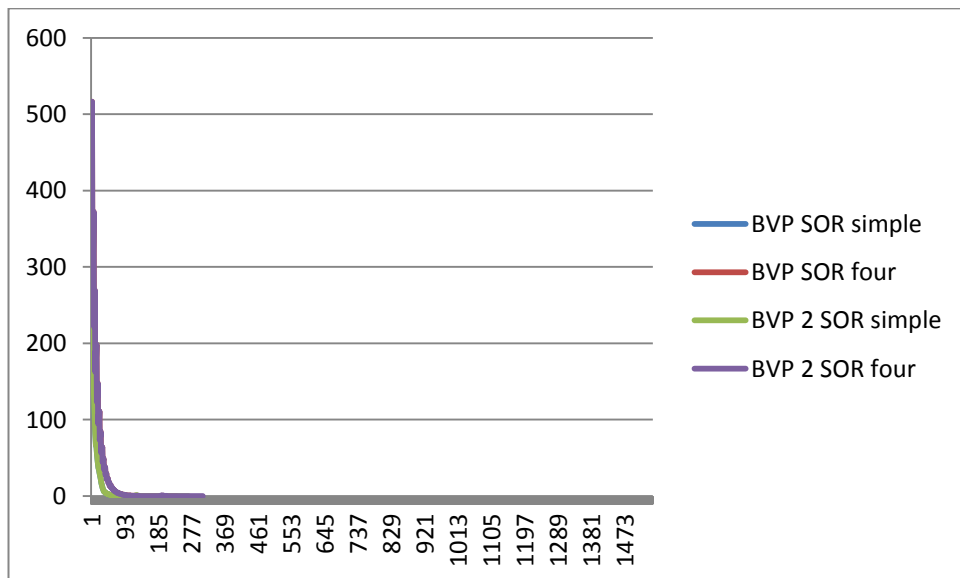


Figure 45 - Decrescimentos do erro nos algoritmos BVP SOR



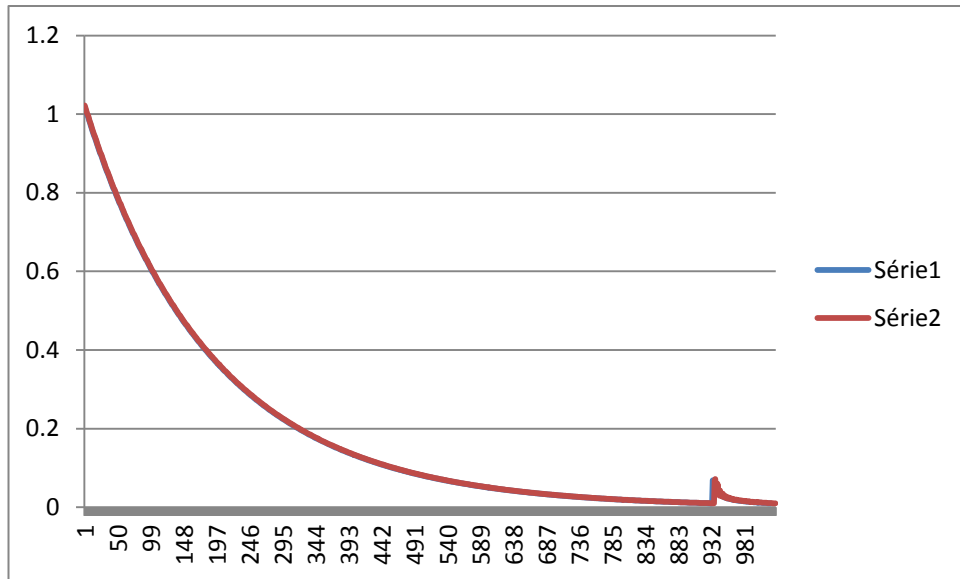


Figure 46 - Adição das áreas de preferencia ao Algoritmo GS

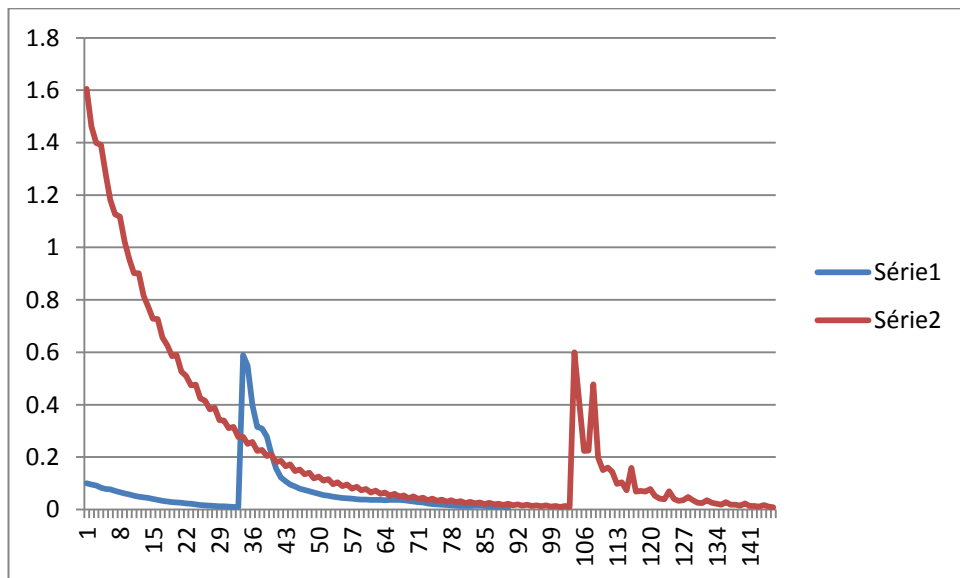


Figure 47 - Adição das áreas de preferencia ao Algoritmo GS

## 5 CONCLUSÃO

Neste trabalho se estudaram os métodos para criar algoritmos de planejamento de caminho com ferramentas para resolver o “weight problem”. Para tal foi criado um software com o qual realizou-se uma série de simulações

No capítulo 2 descreveu-se a importância da representação do ambiente de forma correta, assim como as possíveis escolhas a se tomar na hora de representar o ambiente. Foi também exposto a escolha grid como método para mapear o ambiente devido as características ao representar equações de diferenças. Neste capítulo também foram traçadas as principais características de outros planejadores de caminho e uma comparação entre eles. Também percebemos porque usamos os mapas de linha para representar o ambiente nos arquivos mapa e preferência.

No capítulo 3 descreveu-se a fundamentação teórica dos métodos que geram as funções harmônicas e como discretizar as equações diferenciais como a equação de Laplace. Tal conhecimento foi usado para a criação de um algoritmo no capítulo seguinte.

No capítulo 4 descreveram-se os algoritmos usados no software e seus resultados. Através desses resultados, percebe-se que o método SOR quando usado no contexto descrito é consideravelmente melhor de que os algoritmos que usavam Gauss-Seidel. Também se verifica que ao iterar de maneira simultânea o número de iterações aumentou, o que neste contexto é ruim. É perceptível que o algoritmo nunca atinge zero, ou seja o erro entre iterações é próximo de zero mas nunca zero, e que o erro decresce de maneira exponencial para todos os algoritmos.

Percebe-se também que ao adicionar termos novos ao ambiente pré-calculado são necessárias muito menos iterações do que ao se reiniciar com tal termo novo. Isso se verifica ao ver os saltos no erro ao adicionarmos as áreas de preferência ao cálculo.

Para concluir, pode-se dizer que os métodos baseados em BVP são uma ferramenta extremamente poderosa na robótica, principalmente pelo fato que: a) curvas serem suaves; b) a atualização e adição de novos termos tem um custo pequeno; c) com pequenas modificações podem-se tornar ferramentas de exploração também. Outros algoritmos clássicos para solucionar este problema baseiam-se em grafos, os quais não calculam o ambiente como um todo e ao se adicionar elementos novos a própria representação do ambiente muda. Para estudos futuros pode-se citar a criação de algoritmos de exploração, adição de uma dimensão a mais ou uso de robôs complexos.

## BIBLIOGRAFIA

1. *Ancient discoveries - Machines of the East*. 2007.
2. *Mechanical Engineering in the Medieval Near East (May 1991)*. **Hill, Donald R.** 1991, Scientific American.
3. **Roland Siegwart, Illah Reza Nourbakhsh.** *Introduction to Autonomous Mobile Robots*. s.l. : MIT Press, 2004. 321.
4. *iRobot*. [Online] 2012. <http://www.irobot.com/en/us/robots/home/looj.aspx>.
5. *NeuroArm*. [Online] Health Research Innovation Centre, 2012. [www.neuroarm.com](http://www.neuroarm.com).
6. *da Vinci Surgical System*. [Online] Intuitive Surgical, 2012. <http://www.intuitivesurgical.com/>.
7. **Khatib, Oussama.** The Potencial Field Approach and Operacional Space Formulation in Robot Control. 1985.
8. **Christopher I. Connolly and Roderic A. Grupen.** On the Applications of Harmonic Functions to Robotics. *Journal of Robotic Systems*. 1993.
9. **S. Akishita, S. Kawamura, and K. Hayashi.** Laplace potential for moving obstacle avoidance and approach of a mobile robot. *Japan-USA Symposium on Flexible Automation, A Pacific Rim Conferenc*. 1990.
10. **Weisstein, Eric W.** Piano Mover's Problem. *MathWorld*. [Online] 2012. <http://mathworld.wolfram.com/PianoMoversProblem.html>.
11. **Murphy, Robin.** *Introduction to Ai Robotics*. s.l. : MIT Press, 2000. 466.
12. **Junior, Edson Prestes e Silva.** Navegação exploratória baseada em Problemas de Valores de Contorno. Porto Alegre : s.n., 2003.
13. *The Potencial Field Approach and Operacional Space Formulation in Robot Control*. **Khatib, Oussama.** 1985.
14. **Bruno Siciliano, Oussama Khatib.** *Springer Handbook of Robotics*. s.l. : Springer, 2008.
15. **LaValle, Steven M.** *Planning Algorithms*. s.l. : Cambridge University Press, 2006.
16. **Fahimi, Farbod.** *Autonomous Robots: Modeling, Path Planning, and Control*. s.l. : Springer, 2010.
17. **Fortuna, Armando de Oliveira.** *Técnicas Computacionais ara Dinâmica de Fluidos*. s.l. : Eduusp, 2000.

18. **Cheung, Peter Batista.** *ALGORITMOS DIRETOS E MÉTODOS ITERATIVOS.*  
[Notas de Aula] Aula 7.

19. **Edson Prestes, Marco Idiart.** *Sculpting Potential Fields in the BVP Path Planner.* 2009.

20. **J. S. B. Mitchell, C. H. Papadimitriou.** *The weighted region problem.* Third Annual Symposium on Computational Geometry : s.n., 1987.

21. **Renata Silveira, Edson Prestes and Luciana Nedel.** *Fast Path Planning using Multi-Resolution Boundary Value Problems.* 2010.

22: Ribacki, Arthur Vicente. Comparação experimental de métodos de exploração de ambientes desconhecidos usando robôs móveis autônomos.\_ 2011.