UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FÁBIO DAPPER

# Planejamento de Movimento para Pedestres utilizando Campos Potenciais

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Profª. Dra. Luciana Porcher Nedel
Orientadora

Prof. Dr. Edson Prestes e Silva Júnior
Co-orientador

Porto Alegre, março de 2007

**CIP – CATALOGAÇÃO NA PUBLICAÇÃO**

Dapper, Fábio

Planejamento de Movimento para Pedestres utilizando Campos Potenciais / Fábio Dapper. – Porto Alegre: PPGC da UFRGS, 2007.

73 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2007. Orientadora: Luciana Porcher Nedel; Co-orientador: Edson Prestes e Silva Júnior.

1. Planejamento de Movimento, Animação Comportamental, Simulação de Pedestres, Computação Gráfica. I. Nedel, Luciana Porcher. II. Silva Júnior, Edson Prestes e. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Profª. Valquíria Linck Bassani
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenadora do PPGC: Profª. Luciana Porcher Nedel
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

# AGRADECIMENTOS

Gostaria agradecer a todas as pessoas e instituições que permitiram que este trabalho pudesse ter sido realizado ou de alguma forma me incentivaram a continuar.

Gostaria de agradecer aos meus orientadores, Luciana Nedel e Edson Prestes, pelo empenho e pela referência que foram para mim. Em particular, para a Luciana por insistir na minha permanência no curso, em um momento em que estava decidido a desistir. Ao Edson, pela diponibilidade em conversar e sugerir idéias para o trabalho.

Aos meus pais, pela educação que me deram, pelo incentivo a sempre continuar estudando e pelo suporte financeiro durante muitos anos, sem o qual não teria conseguido concluir a graduação e iniciar o mestrado.

Ao meu orientador na graduação João Comba, por me fazer pensar na possibilidade de um mestrado e de um trabalho de pesquisa avançado, e também pela preocupação quanto a minha desistência do curso. A Carla Freitas, minha primeira professora de computação gráfica, que me inspirou a continuar nesta área e deu um auxílio importante durante a construção do primeiro artigo submetido ao CGI.

Também gostaria de agradecer a todos os colegas do grupo de computação gráfica, por informações valiosas e momentos agradáveis quando pude estar trabalhando no laboratório.

Além destes, agradeço à minha irmã Virgínia e os meus amigos Márcio Araújo, Ranquetat, Luciana (Preta), Wagner, Carla e Sandra pela parceria, diponibilidade em ajudar a resolver algum problema e pelas palavras de apoio.

À minha filha Alícia, que me trouxe sentimentos inimagináveis antes do seu nascimento e sempre é um incentivo a acordar disposto a seguir o trabalho e aproveitar a vida.

À Cláudia, minha mulher, pelo amor que recebo e pelo afeto durante momentos difíceis, pela paciência quando não estive presente e por assumir muito trabalho doméstico, de minha responsabilidade, durante um longo período.

Também gostaria de prestar um agradecimento ao Sport Club Internacional, Campeão do Mundo, por me dar um *presente* que espero desde minha infância e mostrar que é possível conquistar feitos notáveis quando se trabalha arduamente, com humildade e determinação, até o final.

Finalmente, agradeço ao CNPq pelo auxílio financeiro durante parte do curso, e ao Instituto de Informática, pela estrutura oferecida durante a graduação e mestrado.

# SUMÁRIO

# LISTA DE ABREVIATURAS E SIGLAS

DVG Diagrama de Voronoi Generalizado

PRM Probabilistic Roadmaps

PVC Problema de Valor de Contorno

# LISTA DE FIGURAS

# LISTA DE TABELAS

# RESUMO

A simulação de pedestres representados como humanos virtuais em um mundo sintético é de grande interesse em áreas como cinema, arquitetura e jogos. Esta atividade pode ser definida como navegação em ambientes virtuais e envolve principalmente a especificação do ambiente, a definição da posição inicial do agente, assim como sua posição final (ou objetivo). Tendo estes parâmetros definidos, um algoritmo de planejamento de movimento, ou de caminhos, em particular, pode ser usado para encontrar uma trajetória a ser seguida por ele. Entretanto, em um mundo real, se considerar-se várias pessoas, todas na mesma posição inicial e procurando atingir o mesmo objetivo, cada uma seguirá por um caminho diferente. Ou seja, para uma mesma tarefa, a estratégia utilizada por cada pessoa para alcançar seu objetivo vai depender de sua constituição física, personalidade, humor e raciocínio.

Levando em consideração estas questões, este trabalho apresenta um estudo sobre planejamento de movimento para pedestres. Como resultado prático foi desenvolvido um planejador que fornece trajetórias suaves e variadas. As trajetórias são dependentes de características individuais de cada agente, que podem ser alteradas dinâmicamente. O método adotado é baseado na utilização de campos potenciais gerados pela solução numérica de problemas de valores de contorno envolvendo a equação de Laplace (funções harmônicas) e o problema de Sturm-Liouville. Campos potenciais gerados desta forma produzem caminhos suaves e são livres de mínimos locais.

O comportamento de cada agente é determinado pela alteração de seu campo potencial individual, gerado a cada passo da simulação. Dessa forma, é possível alterar dinamicamente o padrão da trajetória e ao mesmo tempo evitar colisões com obstáculos móveis (demais agentes na simulação). Por outro lado, os comportamentos gerados podem tanto ser usados de forma isolada, como combinados em movimentos complexos. Assim, é possível utilizar funções que definem trajetórias ou quantificar um desvio à esquerda ou à direita quando o agente avista um obstáculo a sua frente.

A implementação do método, incluindo técnicas para controlar a velocidade e orientação do agente, e situações de simulação como comportamentos em corredores e regiões abertas, são apresentadas e discutidas.

**Palavras-chave:** Planejamento de Movimento, Animação Comportamental, Simulação de Pedestres, Computação Gráfica.

## Pedestrian Motion Planning using Potential Fields

# ABSTRACT

The simulation of pedestrians represented as virtual humans in a synthetic world is of great interest in areas as cinema, architecture and games. This activity can be defined as navigation in virtual environments and involves mainly the specification of the environment, the definition of the agent's initial position as well as its target position in the world (also called goal). By setting these parameters, a motion planning algorithm, or a path-planning algorithm in particular can be used to find a trajectory to be followed by it. However, in a real world, if we consider several persons (all in the same initial position) trying to reach the same target position, each individual path followed will be different. That is, for the same task, the strategy used by each person to reach their goal will depend on their physical constitution (body type), personality, mood and reasoning.

Taking these questions into consideration, this work presents a study about motion planning for pedestrians. As a practical result, a planner which supplies smooth and varied trajectories was developed. The trajectories are also depending on the individual characteristics of each agent, which can be dynamically changed. The method adopted is based on the use of potential fields generated by numerical solutions of boundary value problems involving the equation of Laplace (harmonic functions) and the problem of Sturm-Liouville. Potential fields generated in this manner produce smooth and local minima free trajectories.

The behavior of each agent is determined by the alteration of its individual potential field which is generated to each step of the simulation. Thus, it is possible to dynamically modify the standard of the trajectory and at the same time to prevent collisions with mobile obstacles (other agents in the simulation). On the other hand, the produced behaviors can be used isolatedly or combined in complex moves. Therefore, it is possible to use a function that defines a trajectory. It is also possible to quantify a detour to the left or to the right when the agent sights an obstacle ahead.

The implementation of the method, including speed control techniques and agent's orientation, and situations like simulation of behaviors in corridors or open regions, will be presented and argued.

**Keywords:** Motion Planning, Behaviour Animation, Pedestrian Simulation, Computer Graphics.

# 1 INTRODUÇÃO

O uso de atores sintéticos, capazes de agir como agentes autônomos, está se tornando cada vez mais comum em aplicações envolvendo ambientes virtuais. Em muitas destas aplicações, um ator sintético deve ter a forma de um humano virtual (TORRES; NEDEL; BORDINI, 2003). Características desejáveis para estes agentes podem incluir a aparência realística na parte gráfica e nos movimentos e a capacidade de ter uma personalidade própria, tomando decisões imprevisíveis.

Em particular, a simulação de pedestres representados como humanos virtuais em um mundo sintético é de grande interesse em áreas como cinema, arquitetura e jogos. Pode-se definir esta atividade como navegação em ambientes virtuais. Esta simulação envolve principalmente a especificação do ambiente, a definição da posição inicial do agente, assim como sua posição final (ou objetivo). Tendo estes parâmetros definidos, um algoritmo de planejamento de movimento, ou de caminhos em particular, pode ser usado para encontrar uma trajetória a ser seguida por ele. Entretanto, em um mundo real, se considerar-se várias pessoas, todas na mesma posição inicial e procurando atingir o mesmo objetivo, cada uma seguirá por um caminho diferente. Ou seja, para uma mesma tarefa, a estratégia utilizada por cada pessoa para alcançar seu objetivo vai depender de sua constituição física, personalidade, humor e raciocínio.

Levando em consideração estas questões, este trabalho apresenta um estudo sobre planejamento de movimento para humanos virtuais. Como resultado prático foi desenvolvido um algoritmo baseado em problemas de valores de contorno que permite encontrar caminhos suaves e variados em ambientes dinâmicos. Estes caminhos dependem das características individuais de cada agente, as quais podem ser alteradas dinamicamente.

## 1.1 Planejamento de movimento

A pesquisa sobre planejamento de movimento teve início nos anos 60 juntamente com o início do desenvolvimento de robôs controlados por computador e intensificou-se nos anos 80. Originalmente, o problema era tratado na robótica, onde o grande objetivo é criar agentes autônomos (robôs[1]) capazes de executar tarefas requeridas pelo usuário sem intervenção humana, ou seja, o usuário especifica o que fazer sem informá-los de como fazer. Hoje em dia, planejamento de movimento possui aplicações desde vídeo games (KAMPHUIS; OVERMARS, 2004), até cirurgias guiadas por computador (TOMBROPOULOS; ADLER; LATOMBE, 1998) e bioinformática (TANG et al., 2004).

[1]Neste texto, o termo robô se refere ao agente autônomo que pode ser tanto real como virtual. O robô pode ser um objeto articulado, um carro ou um foguete. Também pode-se ter um robô simulado em computador ou um humano virtual, entre outros.

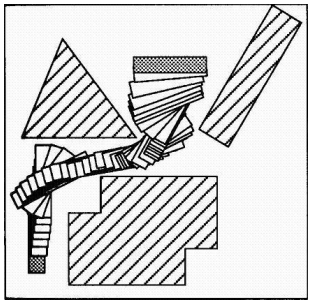


Figura 1.1: Exemplo do problema de Planejamento de Movimento

Em sua forma básica, o problema de planejamento de movimento consiste em encontrar um caminho livre de colisões para movimentar um objeto rígido ou articulado entre uma posição inicial e outra final (Figura 1.1). Nesta situação, este objeto é o único a se mover no ambiente. Desta forma, obtem-se um problema puramente geométrico e que, apesar de parecer simples, possui uma alta complexidade computacional quando são utilizados, por exemplo, robôs com vários graus de liberdade (LATOMBE, 1999).

### 1.1.1 Espaço de Configurações

As dimensões do Espaço de Configurações correspondem aos graus de liberdade do robô. Dessa forma, com o correto mapeamento dos obstáculos para este espaço, o problema se resume ao planejamento de movimento para um ponto. Este processo é interessante porque torna as restrições de movimento do robô mais explícitas e mostra que muitos problemas, aparentemente distintos, podem ser resolvidos da mesma maneira.

Para exemplificar, seja $R$ um robô representado por um polígono simples movendo-se em um ambiente bi-dimensional, ou espaço de trabalho, contendo um conjuto $S = P_1, ..., P_n$ de obstáculos. Uma disposição, ou configuração, do robô pode ser especificada como um vetor de translação. Representa-se o robô transladado por um vetor $(x,y)$ como $R(x,y)$. Por exemplo, se o robô é um polígono com vértices $(1,-1),(1,1),(0,3),(-1,1)$ e $(-1,-1)$, então os vértices de $R(6,4)$ são $(7,3),(7,5),(6,7),(5,5)$ e $(5,3)$, como mostra a Figura 1.2. Com essa notação, um robô pode ser especificado pela lista de vértices de $R(0,0)$. Por definição, $R(0,0)$ é chamado de ponto de referência do robô.

Pode-se especificar uma configuração ou disposição do robô simplesmente indicando as coordenadas do ponto de referência. Assim, $R(x,y)$ especifica que o robô está colocado com seu ponto de referência em $(x,y)$. Já um robô que pode transladar e rotacionar no plano é representado por três parâmetros correspondendo aos seus graus de liberdade. Assim, uma determinada configuração do robô é dada por $R(x,y,\theta)$, onde o espaço de configurações é dado por $R^2 \times [0:360)$. Para não haver colisão, os pontos no espaço de configurações que correspondem à localizações onde o robô intercepta algum dos obstáculos mapeados não devem ser usados. O subconjunto que contém esses pontos é denominado espaço de configurações proibido, ou espaço proibido, denotado por $C_{proibido}(R,S)$. O restante do espaço de configurações, que contém os pontos que correspondem às disposições livres - local onde o robô não intercepta nenhum obstáculo - é chamado de espaço de configurações livre, ou espaço livre, denotado por $C_{livre}(R,S)$.

Um caminho para o robô no espaço de trabalho é mapeado para uma curva no espaço de configurações $C(R)$ e vice-versa. Um caminho livre de colisões é mapeado para uma curva no espaço livre $C_{livre}(R,S)$. A Figura 1.3 ilustra esse conceito para um robô que translada no plano. A Figura 1.3(a) corresponde ao espaço de trabalho, onde

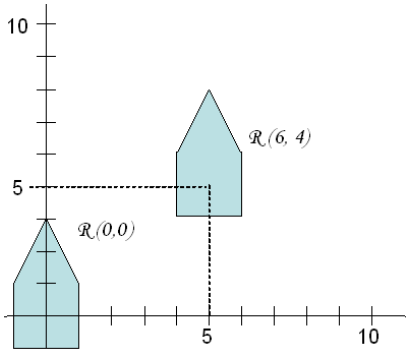
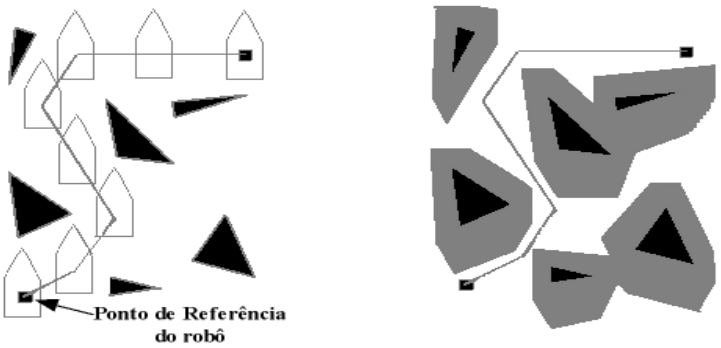



Figura 1.2: Robô poligonal representado por um ponto de referência

é mostrado um caminho livre de colisões para o robô entre a posição inicial e a final, sendo que os obstáculos correspondem aos polígonos pretos. O desenho da direita corresponde ao espaço de configurações, com as áreas cinzas representando $C_{proibido}$ (ver Figura fig:espacoConf(b)). Os obstáculos (em preto, na figura) são mostrados nesse espaço para melhor compreensão, porém, eles não possuem qualquer significado. No caso de um robô que executa translações, $C_{proibido}$ pode ser gerado através do aumento dos obstáculos utilizando um procedimento chamado soma de Minkowski (BERG et al., 2000).




(a) (b)

Figura 1.3: Robô poligonal transladando no plano: (a) Espaço de Trabalho e (b) Espaço de Configurações. Fonte: (BERG et al., 2000)

Características como a geometria do robô e restrições de movimento influenciam na maneira como são definidas as dimensões e na dificuldade em se gerar $C_{proibido}$ dentro do espaço de configurações.

Durante os anos 80, ferramentas clássicas da matemática foram utilizadas para estudar propriedades algébricas, geométricas e topológicas do espaço de configurações. Propriedades físicas como força e fricção foram mapeadas de forma elegante para esta representação. Isto criou um *framework* para o estudo do problema de planejamento de movimento. Muitos destes resultados matemáticos podem ser encontrados no trabalho de Latombe (LATOMBE, 1991).

Através desse formalismo matemático, alguns autores passaram a diferenciar dois termos muitas vezes utilizados indistintamente: Planejamento de Movimento e de Caminhos.

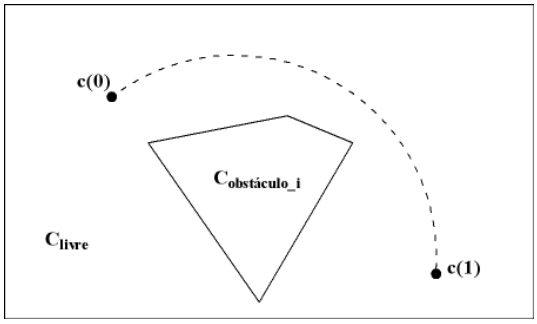



Figura 1.4: Curva em $C_{livre}$ formando um caminho

**Planejamento de Caminhos:** Um caminho é uma curva contínua no espaço de configurações. É representado por uma função contínua que mapeia alguns parâmetros, normalmente utilizados no intervalo [0,1] para uma curva em $C_{livre}$. Este intervalo é arbitrário, sendo que qualquer tipo de parametrização pode ser usada. A solução para um problema de planejamento de caminhos é uma função contínua $c \in C^0$ tal que:[2]

$$c : [0,1] \rightarrow C \; onde \; c(0) = p_{inicio}, \; c(1) = p_{objetivo} \; e \; c(s) \; \in \; C_{livre} \; \forall s \in [0,1]$$

A Figura 1.4 mostra uma curva em $C_{livre}$ unindo $c(0)$ e $c(1)$, dessa forma, definindo um caminho livre de colisões para o robô.

**Planejamento de Movimento:** também chamado de planejamento de trajetória, consiste na parametrização do caminho pelo tempo $t$. Assim, velocidade e aceleração podem ser calculadas através da primeira e segunda derivadas de $c$ em relação ao tempo. Portanto, $c(t)$ define uma *trajetória* no espaço de configurações.

### 1.1.2 Classificação dos problemas de planejamento de movimento

À medida que avançam as pesquisas nas diversas áreas onde planejamento de movimento é utilizado, várias extensões ao problema básico, descrito anteriormente, têm sido estudadas . Estas extensões podem ser classificadas de acordo com as tarefas que são executadas, as propriedades do robô que resolve a tarefa e do algoritmo utilizado (CHOSET et al., 2005).

#### *1.1.2.1 Tarefa executada pelo robô*

A principal forma de classificar um planejador de movimentos é pela tarefa que ele resolve. Entre as tarefas que podem ser citadas estão a navegação, a cobertura, a localização e o mapeamento. Navegação é o problema de encontrar um caminho livre de colisões entre uma configuração do robô e outra. No problema de cobertura, um robô deve passar um sensor ou ferramenta sobre todos os pontos de um determinado espaço. Na localização, ele deve interpretar um mapa e dados de um sensor para determinar sua configuração (ou localização no mapa). Mapeamento consiste em explorar um ambiente desconhecido com a utilização de um sensor para construir uma representação que possa ser útil para navegação, cobertura ou localização. Pode-se combinar tarefas como localização e mapeamento em um processo de exploração de um ambiente desconhecido.

Pode-se pensar em diversas extensões interessantes como a navegação entre obstáculos móveis, o controle de múltiplos robôs e a possibilidade de agarrar e manipular objetos.

[2] O conjunto de funções contínuas é denotado por $C^0$. Se a derivada de uma função contínua $c$ também é contínua, então $c$ é dita diferenciável e pertence ao conjunto denotado por $C^1$

Também pode-se pensar na escolha de caminhos e a definição de trajetórias segundo critérios individuais do agente. Todos estes problemas, em muitos estudos, são tratados em separado, e em muitas aplicações aparecem em conjunto.

Alguns exemplos de problemas serão descritos a seguir:

- Exploração de ambientes: em muitos casos, o problema de planejamento deve ser modificado para permitir com que um robô visite um ambiente completamente desconhecido a fim de localizar um determinado objeto ou simplesmente fazer o levantamento da estrutura do ambiente (PRESTES et al., 2002). Neste caso pode-se tratar em conjunto um problema de aprendizado autônomo do ambiente;

- Manipulação de objetos: em alguns casos o robô deve ser capaz de manipular objetos presentes no ambiente. Em animações é interessante que um ator virtual mova objetos como uma pessoa real, por exemplo, que seja capaz de guardar objetos em uma prateleira ou colocar um par de óculos (KOGA et al., 1994). Outro exemplo de aplicação é um robô que encontra um caminho em uma sala empurrando ou puxando alguns obstáculos específicos (STILMAN; KUFFNER, 2005);

- Caminhos individuais baseados em comportamentos: Neste caso, procuram-se caminhos que não sejam estereotipados como nos métodos tradicionais e sua principal utilização está na simulação de humanos virtuais. O objetivo é construir caminhos que sejam diferentes para cada agente e que representem um comportamento específico. Este é um dos problemas que motivou o desenvolvimento do trabalho apresentado neste texto, como comentado na Seção 1.3.

#### *1.1.2.2 Propriedades do robô*

A Seção 1.1.1 apresenta o que foi denominado de espaço de configurações do robô, para o qual características como geometria do robô, restrições de movimento e graus de liberdade podem ser mapeados. As propriedades do robô como graus de liberdade influenciam diretamente no tempo de processamento necessário, pois definem o número de dimensões do espaço de configurações. As restrições de movimento (ou cinemática) também possuem influência sobre o número de dimensões. Estas restrições podem ser divididas entre restrições holonômicas e não-holonômicas (LATOMBE, 1991).

Restrições holonômicas são aquelas que podem ser expressas como equações relacionando os parâmetros do espaço de configurações. Estas equações podem ser usadas para eliminar algum parâmetro e reduzir a dimensão deste espaço. Um exemplo típico ocorre em um braço articulado. Outros tipos de restrições, ditas não-holonômicas, representam as impostas na movimentação de um objeto tipo carro, que pode andar para frente, para trás e fazer curvas (FRAICHARD; AHUACTZIN, 2001). A relação entre estes parâmetros gera funções não integráveis que não permitem reduzir a dimensão do Espaço de Configurações, como no caso anterior.

Outra dificuldade é o mapeamento dos obstáculos para o espaço de configurações, como comentado anteriormente. A geometria do robô exerce influência direta sobre este mapeamento.

#### *1.1.2.3 Propriedades do algoritmo*

Uma vez definida a tarefa e as propriedades do robô, pode-se escolher o(s) algoritmo(s) adequado(s) que satisfaçam, por exemplo, restrições de tempo, memória, qualidade de solução, consumo de energia, entre outros.

Outra forma de classificar as propriedades dos algoritmos é pela completude, ou seja, o algoritmo sempre fornece um caminho válido caso ele exista ou a informação sobre a inexistência de tal caminho, caso contrário. Como muitos algoritmos são intratáveis computacionalmente, existem outras formas (ditas fracas) de completude.

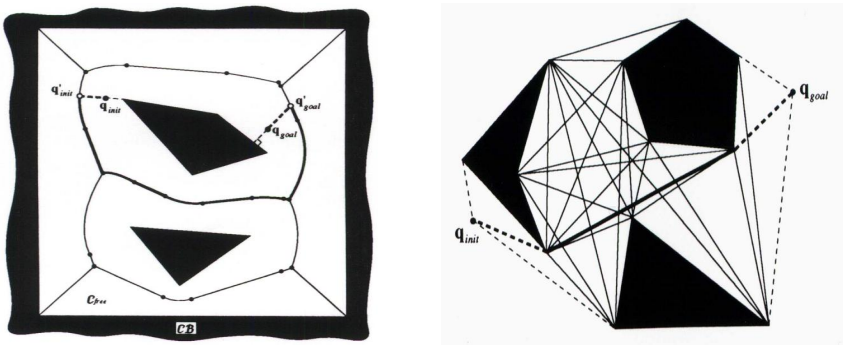
Assim, um algoritmo pode ser probabilisticamente completo quando a probabilidade dele retornar uma solução, se existir alguma, converge para o valor 1 quando o tempo tende ao infinito. Também pode ser completo para uma determinada resolução de discretização, ou seja, se uma solução pode ser encontrada para um determinado nível de resolução, o algoritmo deve retorná-la (*resolution completeness*).

## 1.2 Abordagens para planejamento de movimento

Existe uma enorme quantidade de métodos para a resolução do problema básico. Nenhum deles é totalmente genérico e alguns exigem condições bem específicas como, por exemplo, que o ambiente seja bi-dimensional e os objetos poligonais. Mas de uma forma geral, os métodos clássicos são baseados em três principais abordagens: mapas de caminhos, decomposição celular e campos potenciais.

### 1.2.1 Mapas de Caminhos

Os métodos que utilizam mapas de caminhos representam a estrutura do espaço livre através de um conjunto de segmentos interligados formando um grafo. Após construídos estes caminhos (segmentos), pode-se procurar um caminho através de algoritmos de busca comumente usados para realizar pesquisas em grafos. Métodos clássicos incluem a utilização de diagramas de Voronoi, mostrado na Figura 1.5(a), e grafos de visibilidade, exibido na Figura 1.5(b).




(a) (b)

Figura 1.5: Mapas de Caminhos: Diagramas de Voronoi (a) e Grafos de Visibilidade (b). Fonte (LATOMBE, 1991)

Um método recente utiliza hardware gráfico para construir diagramas de Voronoi generalizados (HOFF et al., 1999). Este método evoluiu para um planejador de movimentos que utiliza a noção de mapas de caminhos probabilísticos e diagramas de Voronoi generalizados (FOSKEY et al., 2001).

### 1.2.2 Decomposição Celular

A decomposição celular consiste em dividir o espaço livre em regiões mais simples, chamadas células, de forma que um caminho entre duas configurações possa ser gerado

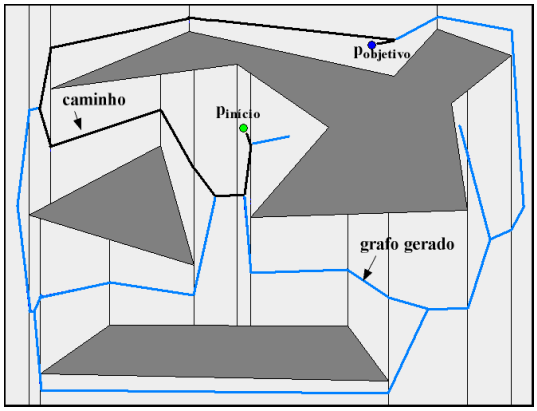



Figura 1.6: Planejamento de Caminhos por Decomposição Celular

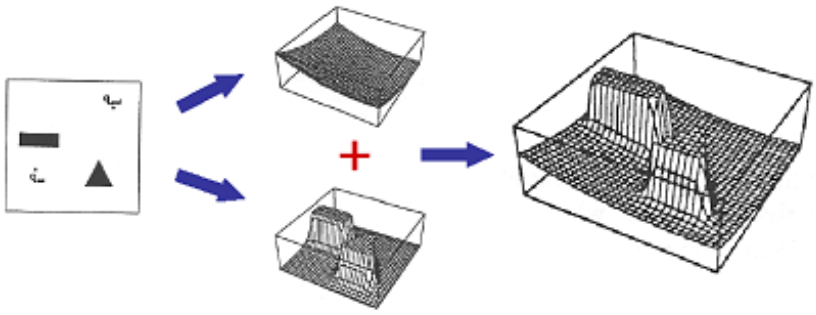


Figura 1.7: Planejamento de Movimento através de Campos Potenciais

facilmente em uma célula. Após esta etapa, cada célula é numerada e cria-se um grafo ligando as células adjacentes. Este método pode ser dividido em decomposição exata e parcial. Entre os métodos desenvolvidos, pode-se citar decomposição trapezoidal e decomposição por quadtrees. A Figura 1.6 mostra um exemplo de planejamento de caminhos através de decomposição trapezoidal. O grafo é formado pelas linhas que unem as células do espaço livre, sendo que as linhas escuras definem um caminho entre a posição inicial e a final.

### 1.2.3 Campos Potenciais

A abordagem chamada de campos potenciais sugere uma metáfora onde o robô é tratado como uma partícula no espaço de configurações, sob a influência de um campo potencial artificial produzido pelo objetivo e pelos obstáculos. Tipicamente, a configuração objetivo gera um potencial atrativo que puxa o robô em sua direção, enquanto os obstáculos produzem um potencial repulsivo, empurrando-o para longe deles. Nesta abordagem, o espaço é discretizado em uma matriz como mostra a Figura 1.7. Muitos métodos que se enquadram nesta abordagem costumam ser muito eficientes, porém costumam sofrer com o problema de mínimos locais. Para isso devem ser usadas técnicas para se escapar destas regiões, ou funções que não possuam mínimos locais.

Um típico exemplo de mínimo local ocorre quando existem objetos simétricos e côncavos como mostra a Figura 1.8. Desta forma o robô é atraído pelo menor potencial, e fica preso em uma região onde a força de atração é igual a força de repulsão dos obstáculos.

A melhor forma de se obter campos potenciais sem mínimos locais é através da solução de equações diferenciais apropriadas como a equação de Laplace. Esta equação foi

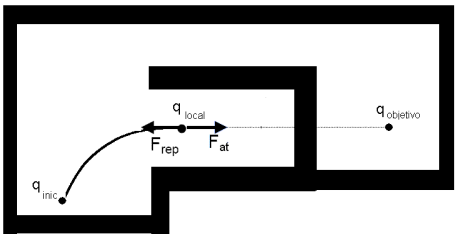



Figura 1.8: Exemplo de Mínimo Local

utilizada como base para o desenvolvimento deste trabalho e será apresentada e discutida convenientemente nos Capítulos 3 e 4.

### 1.2.4 Métodos probabilísticos

Métodos probabilísticos possibilitaram trabalhar com espaços de configurações de dimensões elevadas, o que se torna impraticável com os métodos clássicos. Existem algoritmos randomizados para as três abordagens mencionadas anteriormente.

O primeiro que obteve sucesso foi o planejador chamado *Randomized Path Planner* (RPP) baseado em campos potenciais. Este planejador alterna movimentos fornecidos pelo gradiente descendente e outros movimentos aleatórios para tentar escapar de mínimos locais (BARRAQUAND; LATOMBE, 1991). Depois dele, duas abordagens baseadas em mapas de caminhos obtiveram grande sucesso e serviram de base para vários planejadores recentes: o PRM - *Probabilistic Roadmaps* (KAVRAKI et al., 1996) e o RRT - *Rapidly-exploring Random Tree* (LAVALLE, 1998).

Recentemente, Lingelbach (LINGELBACH, 2004) propôs uma técnica denominada *Probabilistic Cell Decomposition* (PCD) baseada em decomposição celular. Segundo o autor, seu trabalho está sendo melhorado, porém obteve sucesso na solução de problemas como encontrar caminhos em labirintos para robôs rígidos e articulados e no controle de um braço articulado com 6 graus de liberdade.

## 1.3 Objetivos do trabalho

Este trabalho consiste em investigar métodos de planejamento de movimento que possam ser aplicados ao problema de geração de trajetórias para humanos virtuais. O critério principal foi investigar a possibilidade de se construir caminhos baseados em características individuais dos agentes, de forma que pareçam menos estereotipados como são normalmente os caminhos produzidos pelos planejadores existentes. Para tanto, foi construido um protótipo baseado em campos potenciais gerados a partir da solução numérica de um Problema de Valor de Contorno (PVC) envolvendo a equação de Laplace.

Dentre os requisitos considerados para a escolha do método usado neste trabalho estão: a geração de trajetórias em tempo real, a presença de múltiplos agentes, que os caminhos sejam suaves, variados e dependentes de características individuais de cada agente, além de parecerem naturais, isto é, semelhantes a de um ser humano real. Tais requisitos impõe uma maior dificuldade à construção de um planejador e necessitaram a pesquisa e a definição de uma arquitetura adequada, considerando ainda o custo elevado da solução de um PVC.

## 1.4 Organização do texto

O restante do texto está organizado como segue. No Capítulo 2 são apresentados trabalhos relacionados ao planejamento de movimento na área da computação gráfica com o objetivo principal de auxiliar na animação de humanos virtuais.

O Capítulo 3 descreve o método básico de planejamento atráves de funções harmônicas e no Capítulo 4 são apresentadas extensões a este método que permitem a criação de comportamentos individuais para cada agente.

No Capítulo 5 é descrita a implementação do planejador de caminhos gerado, sendo feitas algumas considerações sobre o motivo da arquitetura desenvolvida. Estas considerações são complementadas no capítulo seguinte que apresenta os resultados obtidos em vários experimentos realizados.

Por fim são feitas considerações finais e apresentados trabalhos futuros.

# 2 TRABALHOS RELACIONADOS

Lengyel e seus colegas (LENGYEL et al., 1990) publicaram um dos primeiros trabalhos onde o problema de planejamento de movimento foi visto como um problema de computação gráfica. O trabalho resolvia o clássico problema da movimentação do piano através da utilização do hardware gráfico para rasterizar obstáculos e gerar o espaço de configurações. Desde então, a maior parte das pesquisas se concentrou na tentativa de gerar atores sintéticos capazes de navegar em ambientes virtuais de forma autônoma e realizar tarefas como pegar e manipular objetos. Na área da robótica já existem soluções eficientes para o problema da navegação e existem diversos robôs que manipulam objetos em linhas de produção. Um diferencial na computação gráfica é que os personagens devem ser realísticos, isto é, devem se parecer com humanos reais, tanto na parte gráfica, quanto na forma como são executados os movimentos.

O objetivo deste capítulo é fazer um relato sobre a pesquisa de planejamento de movimento em animação por computador, as principais aplicações e problemas relacionados, incluindo, como principal ponto de interesse, a simulação de pedestres em ambientes virtuais.

## 2.1 Abordagens para a animação de humanos virtuais

No contexto deste capítulo, é importante mencionar as duas técnicas desenvolvidas para auxiliar o processo de animação de humanos virtuais: baseadas em modelo e guiadas por dados. De acordo com Yamane, as abordagens baseadas em modelo usam simulação, busca e otimização para gerar movimentos para o personagem restringindo o espaço de movimentos possíveis através de modelos cinemáticos, dinâmicos e bio-mecânicos (YAMANE; KUFFNER; HODGINS, 2004). Este tipo de abordagem fornece uma representação de movimentos flexível e compacta, mas que pode ser difícil de construir e controlar, ou pode falhar na geração de movimentos com aparência natural se os modelos não restringem suficientemente o movimento. Abordagens baseadas em dados utilizam movimentos capturados de um ator humano para criar animações com movimentos sutis e detalhados. O problema desta abordagem está na dificuldade em adaptar os movimentos capturados previamente a novas situações, devido a existência, em diversos casos, de restrições de interação com o ambiente ou com outros personagens.

## 2.2 Navegação em ambientes virtuais

Para tratar o problema da navegação, existem diversas abordagens, que variam de acordo com o objetivo da aplicação. Pode-se estar interessado em simular um comporta-

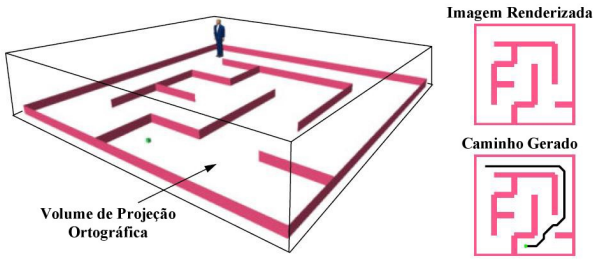



Figura 2.1: Planejamento em duas Fases (KUFFNER, 1998)

mento coletivo através de multidões (*crowd animation*), ou definir comportamentos individuais para cada pedestre, com a preocupação de dar um maior realismo a cena tanto na geração da trajetória a ser seguida, quanto na movimentação do resto do corpo. Por exemplo, quando uma pessoa passa perto de obstáculos, ela precisa mover os braços, ou girar o tórax, para desviar. Quando uma pessoa percorre um caminho, existem certas características que são fáceis de se identificar, por exemplo, uma pessoa não caminha seguindo um caminho reto em direção ao seu objetivo e quando ela encontra obstáculos dinâmicos ou fixos, procura desviar deles com certa antecedência, quando possível.

Para gerar resultados mais realísticos e permitir aplicações em tempo real, muitos autores apresentaram soluções basedas no planejamento de movimento em camadas. Em geral, a primeira etapa se preocupa em definir um caminho e as demais, se preocupam em solucionar questões de animação para tornar o movimento mais realístico ou para desviar de obstáculos dinâmicos. Para a segunda etapa, muitos trabalhos utilizam bibliotecas de movimento geradas previamente, normalmente através de captura de movimento para animações mais realísticas.

Kuffner (KUFFNER, 1998) propõe uma separação em duas fases: uma fase de planejamento de caminhos (*path planning*) e outra em que o ator percorre este caminho (*path following*). Em seu trabalho, o cenário 3D é projetado em um plano e o humanóide é tratado como um cilindro. Desta forma, obtém-se um espaço de trabalho 2D, ou seja, o problema é reduzido ao planejamento de movimento para um disco (Figura 2.1). A rotação, isto é, a direção em que a face do humanóide deve estar voltada é controlada ao se gerar as animações através da biblioteca de movimentos.

A técnica apresentada acima separa o problema em dois subproblemas: um problema de planejamento global e um problema de planejamento local. Metoyer e colaboradores (METOYER; HODGINS, 2004) apresentam uma ferramenta de auxílio à animação por computador que utiliza uma abordagem reativa para a animação de múltiplos personagens em uma cena dinâmica (Figura 2.2). Neste método, os personagens possuem um percurso pré-definido e são utilizados campos de força locais (abordagem reativa) para conduzir os personagens de maneira suave, sem colidir com os demais e parecendo ter um comportamento mais natural. Neste caso, o planejamento global não é executado, sendo definido manualmente pelo usuário (ou poderia ser utilizado um método qualquer de planejamento global). Da mesma forma que o anterior, este trabalho considera os humanóides como discos em um cenário plano e utiliza bibliotecas de movimento pré-definidas.

## 2.3 Técnicas recentes de navegação

O surgimento de algoritmos randomizados, em especial o PRM e o RRT permitiu que as aplicações utilizassem espaços de configurações maiores e mais complexos, gerando

Figura 2.2: Seguindo caminhos de forma reativa (METOYER; HODGINS, 2004)

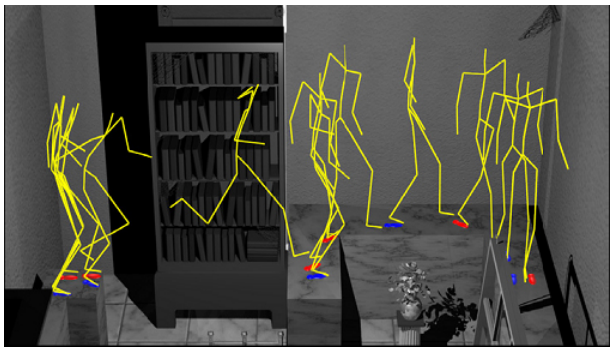


Figura 2.3: Locomoção de Bípedes através de PRM (CHOI; LEE; SHIN, 2003)

caminhos de forma mais rápida e eficiente. Dessa forma, a preocupação passou a ser como tornar as animações cada vez mais realísticas.

Choi et al. (CHOI; LEE; SHIN, 2003) utilizaram a idéia de bibliotecas feitas a partir de captura de movimentos juntamente com o método de PRM para gerar planejamento de movimento para um bípede baseado no cálculo de suas pegadas no cenário. Conforme a posição de uma pegada em relação à próxima, são geradas animações como caminhar, correr, pular, ou andar de lado em uma passagem estreita. Esta técnica é utilizada para ambientes estáticos, uma vez que o mapa de caminhos é gerado em uma fase de pré-processamento. Apenas a busca por um caminho e a geração dos movimentos são feitas em tempo real. No entanto, esta técnica consiste em uma abordagem interessante para muitas aplicações devido ao grande realismo atingido (Figura 2.3).

Apesar do PRM ter sido usado em muitas aplicações e de ser eficiente para um espaço de configurações com muitos graus de liberdade, ele pode ser lento quando o robô (ou humanóide) precisa atravessar uma passagem estreita para atingir o objetivo. Foskey e colaboradores (FOSKEY et al., 2000) sugerem uma abordagem híbrida para robôs rígidos e articulados que se movimentam em um espaço de trabalho 3D através de rotações e translações. Nesta técnica é feita uma análise geométrica global, baseada num diagrama de Voronoi generalizado (DVG) para gerar um caminho aproximado no espaço de trabalho. Este caminho gera alguns segmentos inválidos que são removidos e então utiliza-se PRM para gerar novos segmentos válidos. Portanto, neste método o uso de PRM é feito em alguns lugares e guiados pelo DVG.

A técnica acima foi utilizada por Salomon e colaboradores (SALOMON et al., 2003) para gerar uma navegação interativa em um ambiente 3D complexo. Durante a execução, o usuário indica um destino para o humanóide e o algoritmo calcula um caminho livre de colisões respeitando restrições de movimento do personagem. Esta aplicação permite também que o personagem se mova por vários andares subindo escadas (Figura 2.4).

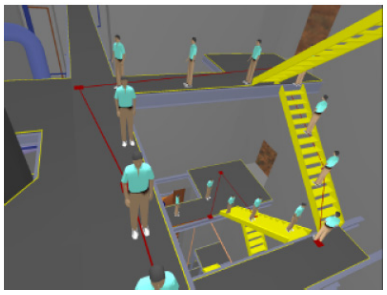


Figura 2.4: Navegação em um ambiente complexo [Salomon et al. 2003]

Como entrada, o algoritmo requer apenas a geometria da cena, a orientação do avatar[1] e parâmetros relacionados ao tamanho do modelo e do avatar. Esta aplicação demonstra a capacidade de se fazer planejamento de movimento em tempo real em ambientes complexos, porém, mesmo utilizando algumas técnicas de suavização de caminhos, a trajetória gerada, em muitos casos, não parece muito natural para um ser humano.

Para gerar trajetórias mais suaves, Pettré et al. (PETTRE; SIMEON; LAUMOND, 2002) utilizam curvas de Bézier através de uma plataforma genérica para planejamento de movimento denominada Move3d (SIMEON; LAUMOND; LAMIRAUX, 2001). O trabalho desenvolvido pelo grupo também utiliza uma biblioteca de movimentos juntamente com o auxílio de interpolação entre os movimentos armazenados. O movimento do humanóide acontece em um cenário plano.

Em 2003, o mesmo grupo publicou um trabalho relatando um aperfeiçoamento desta técnica através de uma aplicação para animação de humanos virtuais contendo dois estágios (PETTRE; LAUMOND; SIMEON, 2003). Como em outras técnicas citadas, existe um planejador de caminhos como etapa inicial. Neste caso um planejador inicial determina uma trajetória livre de colisões para um cilindro representando as pernas do humanóide. O humanóide é representado por um esqueleto humano contendo 57 graus de liberdade no total. As pernas são consideradas partes ativas do humanóide às quais outras partes como braços e espinha estão vinculadas e são consideradas reativas, isto é, possuem regras conforme o deslocamento das pernas. Um controlador de movimentos age em uma segunda fase, detectando quando os braços do humanóide colidem com algum obstáculo e realizando um *warping*[2] no movimento natural dos braços e espinha. A Figura 2.5 mostra o processo de *warping* através de duas possibilidades para evitar a colisão do braço com um obstáculo: na Figura 2.5(a), o braço é movido para junto do corpo enquanto que na Figura 2.5(b), o braço é levantado sobre o obstáculo.

Em 2004, Chestnutt e Kuffner (CHESTNUTT; KUFFNER, 2004) exploraram uma característica importante de bípedes, que é caminhar sobre superfícies irregulares. Um robô tipo humanóide pode se locomover passando por cima de pequenos obstáculos, porém os planejadores usados nas aplicações anteriores não são capazes de lidar com esta possibilidade de uma forma satisfatória. A técnica apresentada utiliza planejadores em três níveis para gerar uma trajetória com as pegadas a serem seguidas pelo robô. O nível mais alto fornece um caminho até o objetivo que é utilizado como guia para os níveis mais baixos. O nível intermediário seleciona um subobjetivo dentro do caminho fornecido, adquire informações do ambiente através de sensores e envia todos os dados para o nível

[1] Um avatar é uma representação virtual do usuário

[2] Warping é qualquer distorção de uma imagem produzida por transformações geométricas genéricas. Para tanto, define-se uma função que mapeia pontos de uma imagem original para uma nova(deformada)

(a) (b)

Figura 2.5: *Warping* para evitar colisão [Pettré et al. 2003]

Figura 2.6: Manipulação de múltiplos braços [Koga et al. 1994]

mais baixo. Este último determina a seqüência de passos para passar pelos obstáculos ou sobre o terreno irregular utilizando os dados anteriores como uma heurística e calculando o caminho partindo do objetivo até a posição do robô.

## 2.4 Manipulação de objetos

Uma variação do problema de planejamento de movimento é a sua utilização na manipulação de objetos por um robô. Pode-se chamar este problema de planejamento de manipulação. A principal diferença entre os dois problemas é que na manipulação deve-se considerar a possibilidade de o robô modificar a forma como ele segura o objeto. Para isso, em algumas tarefas é necessário que o robô solte o objeto e o segure de uma nova maneira. Isto pode incluir a cooperação entre dois robôs e o controle dos braços de um humanóide, onde o objeto pode ser girado e tranferido de um braço a outro.

Em 1994, Koga e colaboradores (KOGA et al., 1994) mostraram um algoritmo para gerar a animação dos braços de um personagem em tarefas de manipulação de objetos. Esta técnica combina planejamento de movimento e algoritmos de cinemática inversa para definir o movimento do braço humano. Entre os exemplos estão a manipulação de um par de óculos e a colocação de um tabuleiro em determinada posição sobre uma mesa (Figura 2.6).

Kuffner e Latombe (KUFFNER; LATOMBE, 2000) melhoraram esta técnica, principalmente em termos de velocidade de processamento utilizando o algoritmo de planejamento de movimento randomizado RRT. Como resultado, era possível realizar o planejamento de movimento para um único braço, em tempo real sem utilizar bibliotecas de movimento prontas.

Já em 2004, Yamane e colaboradores (YAMANE; KUFFNER; HODGINS, 2004) propuseram adicionar às técnicas anteriores o uso de captura de movimentos permitindo uma animação com aparência mais natural. Esta aplicação é indicada para auxiliar na geração de animações por computador como a mostrada na Figura 2.7. Segundo os autores, o trabalho que levaria no mínimo uma hora feito por um artista experiente, pode ser reduzido

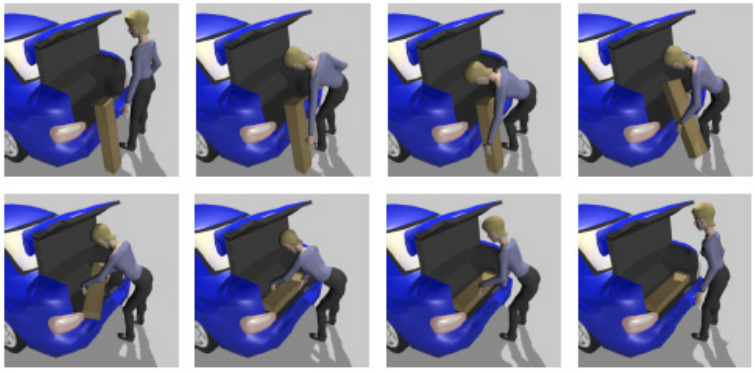


Figura 2.7: Geração de movimento para manipulação de objetos [Yamane et al. 2004]

a menos de 10 minutos com o auxílio de interfaces de pacotes de computação gráfica 3D. A principal vantagem do método é que a biblioteca não precisa corresponder exatamente às tarefas que estão sendo realizadas, pois como a aplicação é baseada em modelos, o planejador de movimentos e o processo de cinemática inversa permitem a complementação da animação. Este método também acontece em duas etapas, uma de planejamento, outra de pós-processamento onde são feitas operações como a suavização do caminho gerado.

Uma limitação desta aplicação é que o sistema falha se o objeto a ser manipulado não está ao alcance do personagem, ou este não consegue ficar equilibrado. Uma solução para este último problema seria que o personagem estendesse o braço para se equilibrar, isto não é feito se esta configuração específica, isto é, a pose com o personagem estendendo o braço, não estiver no banco de dados. Desta forma, é necessário que as tarefas sejam representadas de alguma forma no banco de dados para que o sistema funcione corretamente.

## 2.5 Simulação realística de pedestres

A simulação realística de pedestres é uma área importante e desafiadora dentro da computação gráfica. Muitos dos problemas referentes a criação de atores sintéticos 3D foram resolvidos, mas a dificuldade agora está em se criar comportamentos que sejam verossímeis. As seções anteriores mostraram vários trabalhos referentes à animação que utilizam técnicas de planejamento de movimento em conjunto com bibliotecas de movimentos pré-definidos. Quando se utilizam bibliotecas de movimento, consegue-se um realismo maior, porém fica-se preso a movimentos pré-definidos e de difícil automatização.Um planejador de movimento possui muita dificuldade em criar caminhos que pareçam naturais para um humano de forma automática.

Um fator muito importante para que as animações, em particular, as trajetórias geradas, pareçam mais reais é considerar características individuais dos agentes, como comentado anteriormente.

Rymill e Dodgson (RYMILL; DODGSON, 2005) apresentaram um trabalho que faz uso de teorias da psicologia para a definição de comportamentos para humanos navegando em um ambiente virtual, neste caso, simulação de multidões. O trabalho se concentra no processo de evitar colisões entre os agentes. A idéia é que cada agente possui um caminho pré-definido e quando são detectadas possíveis colisões com outros agentes é iniciado um processo de desvio e posterior retorno ao caminhos normal. A Figura 2.8 mostra 3 tipos de colisões que são detectadas pelo agente.

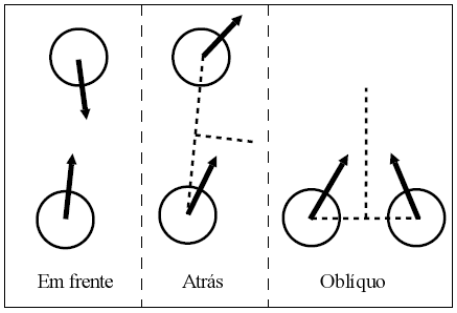



Figura 2.8: Predição de colisões (RYMILL; DODGSON, 2005)

Os estudos psicológicos demonstram que o comportamento de uma pessoa varia conforme a densidade de pedestres que encontra. Quando esta densidade é pequena, as pessoas costumam antecipar o movimento de desvio, enquanto que se for grande, o desvio ocorre, bem mais próximo da possível colisão.

Existem outros comportamentos que podem ser observados como girar os ombros quando se passa muito próximo de outra pessoa e um quase imperceptível passo para o lado.

Um outro trabalho que se preocupa com a definição de comportamentos para pedestres foi apresentado por Shao e Terzopoulos (SHAO; TERZOPOULOS, 2005) e se difere das técnicas tradicionais de animação de multidões. A preocupação é apresentar o modelo para um ser humano individual, através da utilização de várias camadas, integrando componentes motor, perceptivo, comportamental e cognitivo. O objetivo é criar um modelo completo que possa fornecer um elevado grau de realismo considerando diversos aspectos de um ser humano.

Neste trabalho são definidos comportamentos em vários níveis, desde um nível reativo para desviar dos obstáculos até tarefas de alto nível como parar e observar o movimento dos outros pedestres, conversar com amigos ou entrar em uma fila para comprar algum produto.

## 2.6 Considerações sobre animação de humanos virtuais

Este capítulo apresentou diversos trabalhos referentes à animação de humanos virtuais e à utilização de métodos de planejamento de movimento para a automatização deste processo, sejam eles globais ou simplesmente técnicas reativas.

Muitos trabalhos apresentam resultados com grande realismo em ambientes estáticos através da utilização de bibliotecas de movimento pré-definidas e com interferência direta de um usuário. Outros apresentam métodos de planejamento completos para a navegação, mas que não possuem um grau de realismo elevado na geração de trajetórias.

Para resolver os problemas de forma mais efetiva, pode-se criar sistemas em camadas, onde cada camada se preocupa com determinado nível de detalhe. Trabalhos recentes sugerem a construção de um modelo de ser humano completo através da utilização de várias camadas, desde a definição de movimentos a partir das articulações até um raciocínio cognitivo. Em uma camada intermediária pode-se inserir um planejador de trajetórias que defina caminhos individuais e realísticos para os humanóides baseados em características individuais. Estudo psicológicos podem ser um fonte interessante para definir comportamentos realísticos.

# 3 PLANEJAMENTO DE CAMINHOS UTILIZANDO FUNÇÕES HARMÔNICAS

A navegação de um ator sintético, entre uma posição inicial e outra final é normalmente dividida em pelo menos duas fases: uma fase de planejamento, onde um caminho é determinado e uma segunda fase, onde o caminho é percorrido pelo agente móvel. A primeira fase se preocupa com conceitos como eficiência, tratamento de riscos, computabilidade, etc. A segunda fase se preocupa em adaptar o movimento para situações não previstas na primeira devido a mudanças no ambiente, ou tratar problemas como suavização dos caminhos gerados.

Em muitos casos, para a segunda fase são utilizados campos potenciais para gerar caminhos mais suaves e desviar de obstáculos não previstos na primeira fase. Nesse caso, os campos potenciais podem ser classificados como métodos locais (ver Capítulo 1). Porém, eles também podem ser usados como métodos globais permitindo a fusão das duas fases em uma única. Os obstáculos repelem os agentes, enquanto objetivos atraem, gerando a cada instante a melhor direção a ser seguida. No entanto, isto nem sempre é bem sucedido, pois em algumas configurações do ambiente o agente pode ficar preso em um mínimo local.

Para tanto, deve-se procurar um campo potencial que seja livre de mínimos locais. A melhor forma de se encontrar isto é através de soluções numéricas de uma equação diferencial parcial apropriada com condições de contorno convenientes, ou seja, um problema de valor de contorno (PVC).

Nesse sentido, a primeira proposta para o planejamento de caminho através de um PVC foi feita por Connolly e colaboradores (CONNOLLY; BURNS; WEISS, 1990) através do uso de funções harmônicas. Por sua definição, uma função harmônica em um domínio $\Omega \subset \Re^n$ , é uma função que satisfaz a equação de Laplace:

$$\nabla^2 \phi = \sum_{i=1}^{n} \frac{\partial^2 \phi}{\partial x_i^2} = 0 \tag{3.1}$$

A equação de Laplace é importante em muitas áreas da ciência, como gravitação, eletromagnetismo e dinâmica de fluídos, pois descreve campos de força usados em cada uma delas. Na expressão matemática dos campos de força, aparecem superfícies ou linhas eqüipotenciais, definidas pela equação de Laplace. Linhas de fluxo, normais a estas superfícies ou linhas, definem caminhos no qual uma partícula, massa, ou carga elétrica move-se, "caindo" de um potencial maior, para um menor.

Neste capítulo, será feita uma descrição do método de funções harmônicas e no capítulo seguinte, serão apresentadas extensões que permitem executar variações no campo potencial e fornecer trajetórias diferentes que definem determinados comportamentos ao

agente.

## 3.1 Definição do método

Seja $\phi_{x,y}$ uma equação que satisfaça a equação de Laplace e seja $u_{x_i,y_j}$ uma amostra discreta regular de $\phi$ em uma grade. Uma expansão em séries de Taylor para a segunda derivada pode ser usada para encontrar a versão discreta da equação de Laplace:

$$h^2\phi_{x_i,y_j} = u_{x_{i+1},y_j} + u_{x_{i-1},y_j} + u_{x_i,y_{j+1}} + u_{x_i,y_{j-1}} - 4u_{x_i,y_j} \tag{3.2}$$

onde $h$ representa a distância entre as células e pode ser considerada como tendo o valor 1 para este tipo de aplicação, simplificando a equação (CONNOLLY; GRUPEN, 1993). A Equação 3.2 define um sistema de equações lineares sobre a grade. As variáveis são os valores para as áreas livres na grade. Para resolver o problema digitalmente, podem ser usados métodos de relaxação como o método de Jacobi, Gauss-Seidel ou SOR (*Successive Over-Relaxation*). Para a equação de Laplace os métodos basicamente consistem em substituir cada elemento a ser calculado da grade por uma média ponderada de seus vizinhos. A equação acima representa o caso bidimensional, porém pode ser expandida de forma direta para dimensões maiores.

### 3.1.1 Métodos de relaxação

#### *3.1.1.1 Método de Jacobi*

Em sua forma mais simples, a utilização do método de Jacobi consiste em substituir o valor do potencial de cada célula $c_i \in C_{proibido}$ pela média simples de seus vizinhos *simultaneamente*. Sendo $p_{x_i,y_j}$ o potencial associado a cada célula $c_i$ no instante $k$, em um Espaço de Configurações bidimensional, o esquema pode ser representado por:

$$p^{k+1}_{x_i,y_j} = \frac{1}{4}(p^k_{x_{i+1},y_j} + p^k_{x_{i-1},y_j} + p^k_{x_i,y_{j+1}} + p^k_{x_i,y_{j-1}}) \tag{3.3}$$

Em uma máquina com arquitetura SIMD, uma iteração sobre a grade pode ser completada em um passo paralelo, onde cada elemento processado corresponde a uma variável da solução. Geralmente, o método de Jacobi necessita de um número maior de iterações para convergir do que Gauss-Seidel ou SOR. Por outro lado, são muito eficazes em máquinas SIMD. Uma solução que pode ser interessante é a adaptação do método para a utilização de Hardware Gráfico programável. Isto já foi usado em problemas como simulação de fluidos (SCHEIDEGGER; COMBA; CUNHA, 2004). Um protótipo preliminar para a solução da equação de Laplace utilizando harware gráfico foi testado e está brevemente descrito na Seção 5.1.

#### *3.1.1.2 Gauss-Seidel*

O método de Gauss-Seidel é similar ao de Jacobi, exceto que parte dos vizinhos utilizados pertencem a iterações diferentes. O caso bidimensional pode ser representado por:

$$p^{k+1}_{x_i,y_j} = \frac{1}{4}(p^k_{x_{i+1},y_j} + p^{k+1}_{x_{i-1},y_j} + p^k_{x_i,y_{j+1}} + p^{k+1}_{x_i,y_{j-1}}) \tag{3.4}$$

A Equação 3.4 fornece um algoritmo adequado para ser utilizado em máquinas monoprocessadas. Pode-se utilizar uma matriz e percorrê-la seqüencialmente, substituindo cada elemento pela média de seus vizinhos. Conforme a equação, haverão dois vizinhos

atualizados (iteração $(k+1)$) e dois pertencentes a iteração atual ($k$). Isto faz com que um valor mais recente, o produzido na iteração $k+1$, seja propagado, levando a uma convergência mais rápida. Enquanto for encontrada uma área livre o valor é propagado rapidamente. Porém quando é encontrado um obstáculo, esta propagação é interrompida. Uma forma de acelerar a convergência é alternar este sentido cada vez que a matriz for percorrida. Este processo é descrito em detalhes por Édson Prestes em sua tese de Doutorado (PRESTES, 2003).

#### *3.1.1.3 SOR*

De acordo com Armando de Oliveira (OLIVEIRA FORTUNA, 2000), o método SOR converge mais rapidamente para um estado de equilíbrio do que os métodos previamente apresentados e pode ser representado por:

$$p_{x_i,y_j}^{k+1} = p_{x_i,y_j}^{k} + \frac{w}{4}(p_{x_{i+1},y_j}^{k+1} + p_{x_{i-1},y_j}^{k+1} + p_{x_i,y_{j+1}}^{k} + p_{x_i,y_{j-1}}^{k} - 4p_{x_i,y_j}^{k}) \tag{3.5}$$

onde $w$ é a constante de aceleração. Apesar deste método convergir mais rapidamente, os resultados intermediários até a convergência sofrem mais oscilações do que os outros. Se forem utilizadas soluções parciais, como será visto adiante, esta pode não ser a melhor escolha, pois os resultados intermediários podem não ser confiáveis ou causarem uma maior oscilação na trajetória.

## 3.2 Aplicação das funções harmônicas no planejamento de caminhos

O método desenvolvido neste trabalho utiliza soluções parciais baseadas em Gauss-Seidel e condições de contorno de Dirichlet.

A escolha foi baseada no trabalho desenvolvido por Prestes *et al.* (PRESTES et al., 2002). Os autores utilizaram métodos baseados em um PVC para fazer com que um robô possuindo um sensor de proximidade do tipo sonar explorasse um ambiente desconhecido a fim de levantar sua estrutura. Em seu trabalho, eram utilizadas até 30 iterações a cada passo do robô. Eles perceberam que para um número pequeno de iterações, o método de Gauss-Seidel gera trajetórias mais suaves do que SOR (ver Seção 3.1.1.3).

A utilização de apenas resultados parciais para a equação de Laplace é importante para se conseguir controlar vários agentes em tempo real, conforme um dos objetivos deste trabalho. Também é importante em ambientes dinâmicos, onde o espaço de configurações muda constantemente (sendo alteradas as condições de contorno) e necessitando um novo cálculo para o campo potencial. Segundo Edson Prestes (PRESTES et al., 2004), este é um exemplo de algoritmo que fornece uma resposta a qualquer hora (*anytime algorithm*), cuja qualidade da saída é melhorada gradualmente a cada iteração.

### 3.2.1 Considerações quanto às condições de contorno

Connoly e Grupen (CONNOLLY; GRUPEN, 1993) enumeraram dois tipos de condições de contorno utilizados no planejamento de movimento: Dirichlet e Neumann.[1]

Condições de contorno de Dirichlet especificam o valor da função nos pontos do contorno enquanto que condições de Neumann especificam uma derivada normal à função nos pontos do domínio.

[1] Outras condições de contorno como Robin e Cauchy não foram enumeradas por Connoly e colaboradores na pesquisa sobre planejamento de movimento. Informações detalhadas sobre problemas de valores de contorno podem ser encontrados em livros de equações diferenciais ou em (OLIVEIRA FORTUNA, 2000)

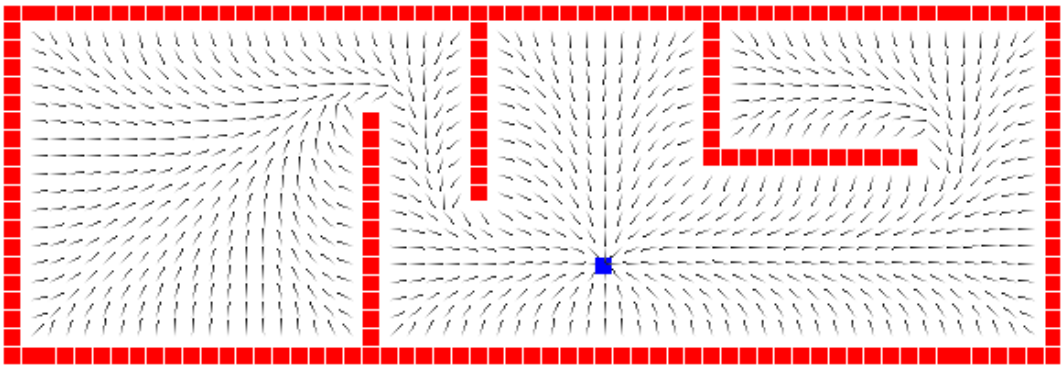


Figura 3.1: Campo Vetorial gerado pelas funções harmônicas

A utilização de condições de contorno de Dirichlet permite que se especifique, simplesmente, o valor das células contendo obstáculos com o valor de potencial mais elevado e os objetivos com o valor mais baixo. Na prática, as células pertencentes ao conjunto $C_{proibido}$, possuem o valor de potencial fixo em 1, enquanto que as células representando objetivos, o potencial fixo em zero. Como visto, o campo vetorial define linhas de fluxo que partem dos obstáculos em direção ao objetivo. O fluxo gerado através de condições de contorno de Dirichlet tende a seguir a normal externa a superfície do obstáculo, afastando os agentes para longe. Utilizando condições de Neumann, a tendência é que os agentes passem mais próximos dos obstáculos, pois o fluxo passa a ser tangencial aos obstáculos, uma vez que o valor do contorno está relacionado a derivada da função em relação a normal. Condições de Neumann também necessitam de uma implementação mais complexa e como os agentes podem passar mais próximos dos obstáculos, é necessário um cuidado maior para evitar colisões.

### 3.2.2 Geração da trajetória

A geração do caminho ou da trajetória pode ser feita de diversas formas, de acordo com a interpretação que se dá ao campo potencial gerado. Utilizando o gradiente do campo obtém-se linhas de força que guiam o robô até o seu objetivo. Pode-se, por exemplo, utilizar estas linhas simplesmente para gerar um caminho, ou seja, uma curva entre a posição inicial e a final, ou pode-se utilizar o gradiente para calcular uma força que será exercida sobre a partícula que representa o agente no espaço de configurações calculando a velocidade e direção do agente. Pode-se levar em conta o módulo do gradiente para fazer com que em determinadas regiões, afastadas do objetivo e com vários obstáculos, o agente se mova mais lentamente (gradiente menor) e quando estiver próximo ao objetivo, com o caminho livre, se mova mais rápido.

A Figura 3.1 mostra um campo vetorial composto pelo gradiente descendente ($dgrad$) normalizado, onde $dgrad$ é calculado por

$$dgrad = ((p_{x-1,y} - p_{x+1,y}), (p_{x,y-1} - p_{x,y+1})) \tag{3.6}$$

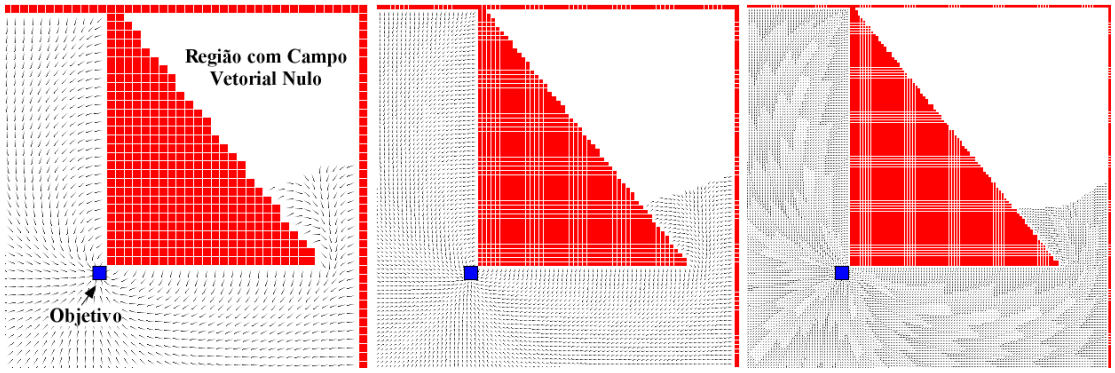
onde $p$ é o valor do potencial em cada célula.

Neste exemplo, em qualquer ponto do espaço, pode-se gerar uma trajetória seguindo-se a direção gerada pela Equação 3.6.

Apesar do método não possuir mínimos locais, existe um problema devido a precisão finita dos computadores chamado de achatamento (*flatness*). Em determinadas regiões, o potencial fica muito próximo do valor 1 de forma que o gradiente não é mais calculado

corretamente (em muitos casos, as células acabam sendo arrendodada para o valor 1). Este problema ocorre em regiões separadas da região onde está o objetivo por uma passagem estreita.

A Figura 3.2 mostra uma região onde ocorre achatamento (canto superior direito). Para o mesmo cenário foram utilizadas matrizes de tamanho $(60 \times 60)$, $(120 \times 120)$ e $(200 \times 200)$, demonstrando que este efeito não depende da resolução da grade utilizada, mas da geometria do ambiente.




(a) (b) (c)

Figura 3.2: Efeito do achatamento com diferentes resoluções: (a) $60 \times 60$; (b) $120 \times 120$; (c) $200 \times 200$

Outra questão que já foi comentada é o fato de que os caminhos gerados levam o robô a andar com o máximo afastamento dos obstáculos. Isto é profundamente adequado para problemas de robótica, entre outros, porém quando simula-se humanos virtuais, este comportamento nem sempre é desejado. Para tratar esta questão, o Capítulo 4 apresenta uma extensão a este método que permite modificar as trajetórias geradas, de forma que fiquem caracterizados comportamentos específicos como andar mais próximo a uma parede ou se afastar de outros agentes próximos.

# 4 EXTENSÕES

Como visto, o modelo de funções harmônicas faz com que um agente procure andar o mais longe possível dos obstáculos, procurando o centro da região livre. Isto fornece um comportamento estereotipado e pouco realístico para um humano virtual. Pode-se citar o exemplo de um corredor, onde é comum pessoas caminharem em sentidos opostos: o comportamento natural de uma pessoa é se antecipar a este encontro possivelmente caminhando mais próximo a uma das paredes.

Outra questão que leva a uma simulação pouco convincente é o fato de todos os personagens seguirem o mesmo caminho. Assim como ocorre no mundo real, é desejável que características individuais dos agentes, tais como constituição física, personalidade, humor e raciocínio influenciem na definição de sua trajetória.

Além dessas questões, existe o problema de achatamento devido a precisão limitada dos computadores na resolução de um PVC utilizando métodos numéricos (Seção 3.2.2).

Este capítulo irá descrever como é possível estender o modelo apresentado no capítulo anterior, possibilitando a criação de comportamentos variados para cada agente e como é possível resolver o problema de achatamento através da utilização de subobjetivos.

## 4.1 Planejamento de movimento além das funções harmônicas

A equação de Laplace não é a única equação diferencial que gera funções sem mínimos locais. Ela pode ser incluida em uma família de equações definidas por:

$$\nabla^2\phi + F(\nabla\phi) = 0 \tag{4.1}$$

tal que $F(v)$ é qualquer função contínua com $F(0) = 0$.

Prestes (PRESTES, 2003) mostrou que a solução gerada por esta equação é livre de mínimos locais.

Utilizando a função linear

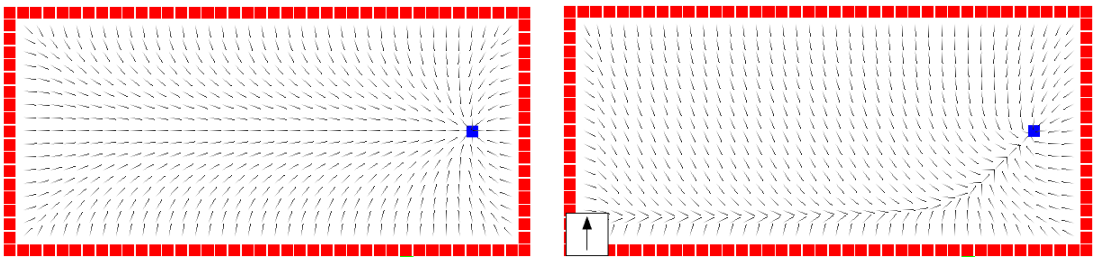
$$F(\nabla\phi) = \varepsilon\nabla\phi \cdot v \tag{4.2}$$

e substituindo na Equação 4.1 obtem-se o problema de Sturm-Liouville que dependendo das condições de contorno, pode ser resolvido analiticamente:

$$\nabla^2\phi + \varepsilon\nabla\phi \cdot v = 0 \tag{4.3}$$

Trevisan *et al.* (TREVISAN et al., 2006) apresentam um framework para navegação exploratória baseada na solução de equações diferenciais sem mínimos locais. Os autores sugerem a equação acima para tratar ambientes esparsos. A adição do termo $\varepsilon\nabla\phi \cdot v$ quebra a simetria do campo gerado pela equação de Laplace, aumentando o desempenho do sistema durante o processo de exploração (em ambientes esparsos).

A Figura 4.1 mostra a modificação que ocorre no campo vetorial em relação às funções harmônicas utilizando a Equação 4.3. Na direita é mostrado um campo modificado com a utilização de $\mathbf{v} = (0,1)$ e $\varepsilon = 0,8$.



(a) (b)

Figura 4.1: Campo vetorial gerado pelas funções harmônicas (a) e pela Equação 4.3 (b).

Neste trabalho, a principal contribuição é utilizar a Equação 4.3 para gerar comportamentos diferentes para vários agentes em um ambiente conhecido. O vetor $\mathbf{v}$, chamado de vetor comportamental, pode ser visto como um campo de força externo que age contra a tendência natural do agente de se afastar dos obstáculos. O parâmetro $\varepsilon$ pode ser entendido como a *força* ou *influência* em seguir a direção definida pelo vetor $\mathbf{v}$ em vez da direção produzida pelas funções harmônicas.

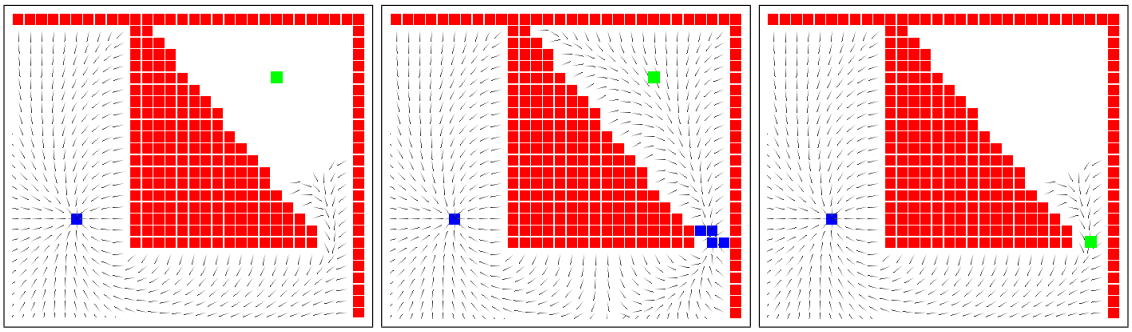
O Capítulo 5 trará detalhes sobre a arquitetura do planejador desenvolvido e alguns exemplos dos comportamentos obtidos mantendo fixo e variando dinamicamente os parâmetros $\varepsilon$ e $\mathbf{v}$ da Equação 4.3. O capítulo 6 descreve diversos experimentos e resultados obtidos com este método e analisa a influência dos parâmetros $\mathbf{v}$ e $\varepsilon$, além de outros descritos na implementação do planejador.

## 4.2 Solução para o problema de achatamento

Como visto, o problema de achatamento ocorre em regiões onde o campo potencial possui valores muito próximos de 1, formando um campo vetorial nulo ou incorreto devido a precisão limitada dos computadores. Uma solução simples para resolver o problema consiste em adicionar um sub-objetivo nas fronteiras entre a região onde o campo vetorial é nulo e a região com campo vetorial diferente de zero (que fornece um caminho até o objetivo).

A Figura 4.2 ilustra o processo: O algoritmo é disparado quando o robô está em uma região com gradiente nulo[1] (Figura 4.2 (a)). Então todas as células com valor de potencial $p_{min} = 1 - 10^{-5}$ são marcadas como objetivo e a matriz é relaxada (100 iterações). A Figura 4.2(b) mostra o novo campo vetorial gerado, no qual o robô passa a estar sob influência. Em seguida, remove-se o sub-objetivo e o robô passa a seguir o caminho fornecido pelo campo original (Figura 4.2(c)). O valor $p_{min}$ foi obtido experimentalmente e corresponde a um valor um pouco maior do que o encontrado na região de fronteira. Desta forma, garante-se que o robô estará sob a influência do campo original e são geradas trajetórias mais suaves durante a transição. É possível perceber na Figura 4.2(b) que o sub-objetivo é colocado um pouco além da fronteira entre as duas regiões, dentro da região com campo vetorial não nulo.

[1] Para tornar o processo mais confiável, o gradiente é considerado nulo se possuir módulo menor do que $10^{-8}$

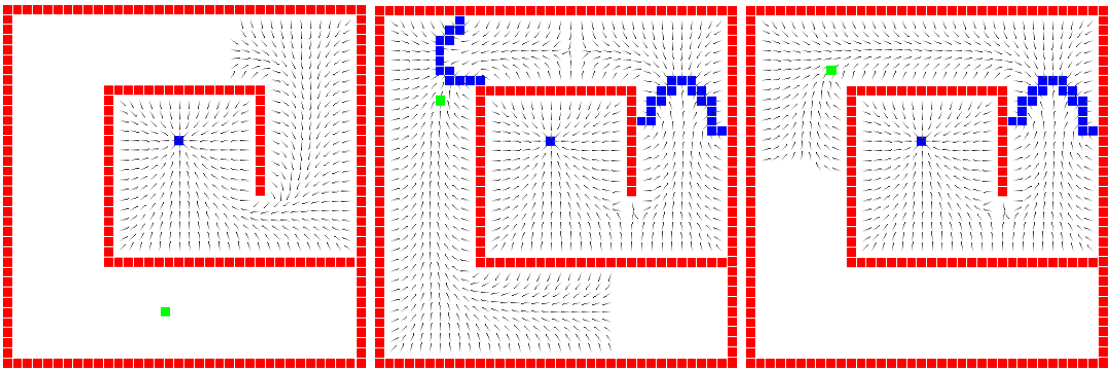


(a) (b) (c)

Figura 4.2: Fuga de uma região com achatamento. (a) Agente em uma região com campo vetorial nulo; (b) Adição de sub-objetivo nas células com valor igual a $p_{min}$; (c) Agente atinge o sub-objetivo que então é removido

Pode ocorrer que a colocação de um sub-objetivo não seja suficiente para que o robô esteja sob ação do campo vetorial que o leve até o objetivo. Dessa forma, é necessário repetir o procedimento anterior formando uma pilha de sub-objetivos. Cada vez que o robô atinge um sub-objetivo, este último poderá ser removido pois o robô estará sob influência do próximo e assim sucessivamente até o objetivo final ser atingido.

A Figura 4.3 demonstra o processo: Inicialmente, o robô (célula verde) está em uma região onde ocorre achatamento - Figura 4.3(a). Então são adicionados sub-objetivos de forma sucessiva, de acordo com o procedimento descrito acima, até que o robô esteja sob efeito do campo vetorial que o levará até o sub-objetivo mais próximo, como mostra a Figura 4.3(b). Quando este sub-objetivo for atingido, o robô estará sob ação do campo vetorial do seguinte (Figura 4.3(c)).



(a) (b) (c)

Figura 4.3: Utilização de uma pilha de sub-objetivos

# 5 IMPLEMENTAÇÃO

Como aplicação prática deste trabalho, foi desenvolvido um planejador de caminhos que pode ser usado para controlar a navegação de vários agentes em tempo real. Como o objetivo do trabalho foi investigar soluções para o problema de planejamento de caminhos para humanos virtuais, a arquitetura utilizada na implementação sofreu diversas alterações durante o desenvolvimento do estudo em função dos resultados encontrados, incluindo o objetivo de fornecer um comportamento individual para cada agente e a necessidade de um melhor desempenho. Neste capítulo será apresentada a arquitetura que se mostrou mais adequada a esses objetivos. Serão feitas considerações sobre a implementação e apresentado o algoritmo em passos gerais.

Para a implementação do planejador de caminhos, utilizou-se a linguagem de programação C++, com as bibliotecas OpenGL e GLUT.

## 5.1 Visão geral da arquitetura

O Capítulo 4 apresenta o método utilizado para alterar o comportamento do agente através da modificação do campo gerado pelas funções harmônicas. Quando existem vários agentes em um mesmo ambiente, a definição de um comportamento individual para cada um exige que esta *alteração no campo* também seja individual.

O elevado custo computacional para a resolução numérica de um PVC, torna inviável a utilização de um campo potencial representando todo o ambiente para cada um dos agentes, em tempo real. Para tentar contornar este problema foram feitas três propostas:

1. Manter todos os agentes no mesmo campo global e modificar somente uma região ao redor de cada um;

2. Utilizar hardware gráfico programável para resolver o PVC;

3. Gerar um campo local para cada agente, que represente apenas parte do ambiente, de forma que possa ser alterado livremente sem influenciar o campo global.

A primeira proposta não apresentou resultados satisfatórios, pois a influência dos parâmetros sobre a região alterada não foi suficiente para modificar o comportamento do agente. Somente quando o agente se aproxima do seu objetivo a influência de $\mathbf{v}$ e $\varepsilon$ se tornam visíveis.

Para a investigação da segunda proposta, foi criado um protótipo para resolver a equação de Laplace em uma placa de vídeo programável, no caso uma nVidia FX5700, com a utilização da linguagem de programação Cg para implementar o algoritmo a ser processado pela GPU (método de Jacobi conforme Seção 3.1.1.1). Contrariando as expectativas,

o resultado deste experimento não foi satisfatório. Um dos problemas diz respeito a precisão de ponto flutuante do hardware (32 bits) que causou um aumento significativo no problema de achatamento (Seção 4.2). O outro diz respeito ao desempenho obtido pois a solução levou mais tempo para convergir do que no processamento via CPU. A investigação de um algoritmo mais adequado para GPUs, incluindo a utilização de hardware mais recente é sugerida como trabalho futuro estando correntemente em desenvolvimento por outro membro do grupo.

A terceira proposta mostrou ser mais eficiente em termos de desempenho e na geração de comportamentos. Assim foi utilizada como base para o protótipo, sendo que algumas adaptações tiveram que ser feitas, como será mostrado a seguir neste texto.

Dessa forma, o planejador consiste em: um mapa global para cada objetivo (ou espaço de configurações global) que define um fluxo a ser seguido até este objetivo e; um mapa local, que define alterações na trajetória normal possibilitando comportamentos individuais.

### 5.1.1 Mapa global

Em um ambiente com diversos atores virtuais, podem ser definidos diversos objetivos que serão compartilhados por mais de um ator. Por exemplo, pode-se imaginar um cenário como um pequeno parque em uma cidade[1]. Neste cenário, existe uma barraca de venda de pipoca e cinco saídas que levam o agente para fora do cenário, como mostra a Figura 5.1. Tanto a barraca de pipoca quanto as saídas são objetivos que podem ser compartilhados, de forma que podem haver vários agentes andando em direção a eles ao mesmo tempo. Por outro lado, um agente pode possuir mais de um objetivo, por exemplo, ele pode caminhar até a barraca de pipocas e depois seguir para uma das saídas.

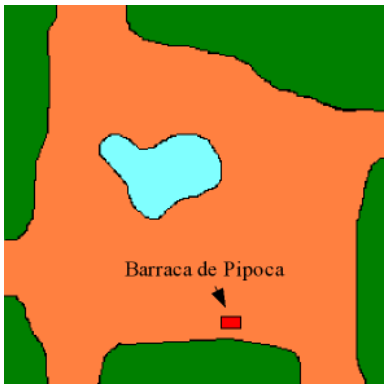



Figura 5.1: Pequeno parque em uma cidade contendo cinco saídas e uma barraca de pipoca

Dessa forma, faz sentido definir o ambiente global como um conjunto de campos potenciais, denominados de mapas globais, um para cada objetivo. Diversos agentes podem compartilhar um mesmo mapa global, e durante a simulação, um agente pode ter seu mapa global alterado se for desejável alterar o seu objetivo.

Assim, o ambiente global é representado por um *conjunto* de matrizes $\{m_k\}$, onde cada matriz $m_k$ está associada a um objetivo $o_k$ e possui $L_x \times L_y$ células, denotadas por $\{c^k_{i,j}\}$. Cada célula $c^k_{i,j}$ corresponde a uma região quadrada centrada nas coordenadas $r = (r_i, r_j)$ do ambiente e armazena um valor particular de potencial $p^k_{i,j}$.

[1] Este cenário é apresentado com maiores detalhes em um experimento na Seção 6.3.5

Cada matriz $m_k$ é relaxada independentemente através da Equação 3.1, conforme é descrito no Capítulo 3. Após a convergência, o mapa $m_k$ contém o campo potencial que é usado para se atingir o objetivo $o_k$. Este procedimento é realizado em um pré-processamento, pois o campo global é fixo. Considera-se ainda que o ambiente é cercado por obstáculos de forma a delimitar o espaço de navegação dos agentes.

### 5.1.2 Mapa local

Cada agente $a_k$ possui um mapa $am_k$ que armazena a informação corrente, isto é, obtida a cada ciclo do algoritmo, a respeito do ambiente global. O mapa é centrado na posição atual do agente no ambiente global e representa uma pequena fração deste ambiente, ou seja, em torno de 10% da área total coberta pelo ambiente global. Pode ser pensado como o campo de ação do agente.

Uma figura geométrica adequada para representar o mapa local poderia ser um círculo, que possui uma simetria perfeita em volta do agente, no entanto, a utilização de um quadrado simplifica muito a implementação do algoritmo.

Assim, o mapa $am_k$ possui $l_n^k \times l_n^k$ células, denotadas por $\{ac_{i,j}^k\}$ e pode ser dividido em três regiões: uma zona de atualização (*zona-A*); uma zona livre (*zona-L*) e uma zona de fronteira (*zona-F*), como mostrado na Figura 5.2. A região de fronteira (*zona-F*) delimita

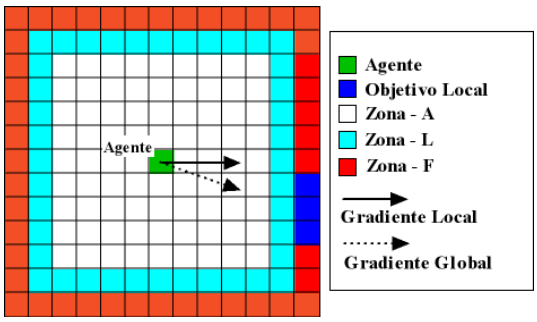



Figura 5.2: Mapa Local

o espaço de navegação e possui todas as células com potencial fixo em 1, portanto, podem ser consideradas como obstáculos, ou seja, pertencem a $C_{proibido}$. A excessão ocorre com o objetivo local que é mapeado sobre a *zona-F* e, como visto na definição do método, possui potencial igual a zero. A existência da *zona-F* faz com que o campo vetorial seja direcionado para o objetivo local, ou seja, uma partícula colocada em qualquer ponto de um mapa contendo apenas a região de fronteira e um objetivo local, sempre será levada até este objetivo.

A zona de atualização define a região onde os obstáculos do ambiente são mapeados para o espaço de configurações conforme descrito na Seção 5.1.3, enquanto que as células pertencentes a *zona-L* são sempre consideradas livres de obstáculos. A inexistência desta última pode fazer com que o objetivo local seja bloqueado por um obstáculo. Desta forma não existe conexão (células livre) entre o centro do mapa e o objetivo mapeado na borda. Quando isto ocorre o agente fica perdido pois não há informação vinda do objetivo intermediário para produzir um caminho até ele. A Figura 5.3 ilustra esta questão, sendo que as células pertencentes a $C_{proibido}$ são mostradas em vermelho: na Figura 5.3(a) a existência da *zona-L*, delimitada pelo quadrado interno, permite um gradiente diferente de zero no centro do mapa, enquanto que na Figura 5.3(b), o campo vetorial é nulo devido a existência de obstáculos mapeados na frente do objetivo.

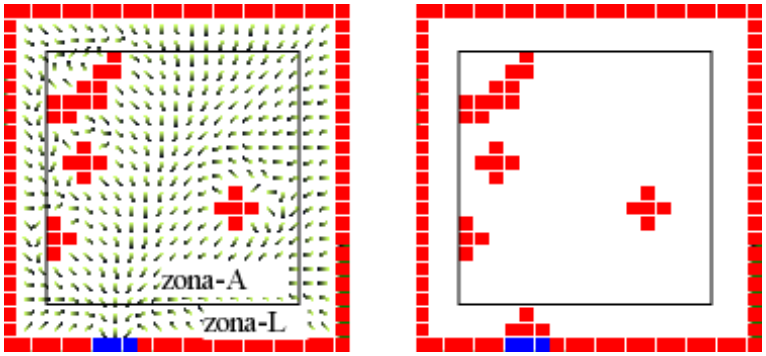



Figura 5.3: Objetivo Local bloqueado por Obstáculo

A Figura 5.2 mostra ainda dois vetores, o gradiente global e o local. O gradiente global é o gradiente calculado na célula onde está o agente no mapa global e é utilizado para definir a posição do objetivo localizado como referência. O gradiente local é calculado no centro do mapa local (posição do agente no mapa local). Dessa forma, o gradiente global é utilizado para gerar o objetivo local, sendo que o passo do agente será determinado em função do gradiente local, como será explicado adiante neste capítulo.

De uma forma similar ao mapa global, cada célula corresponde a uma região quadrada centrada em uma coordenada do ambiente $r = (r_i, r_j)$ e armazena um valor de potencial $ap^k_{i,j}$.

A área associada a cada célula do mapa global pode ser maior do que a área associada a célula do mapa local. Isto porque o mapa global define o caminho de forma ampla, como um mapa indicando as ruas a serem seguidas, enquanto que o mapa local define a trajetória real seguida pelo agente, considerando um movimento refinado. Assim, quanto menor o tamanho da célula, melhor a qualidade do movimento. Pode-se inclusive utilizar outro método de planejamento de movimento para gerar o caminho global, como os citados no Capítulo 1, em cada caso haverão variações nas trajetórias conforme o método escolhido.

### 5.1.3 Atualização do mapa local através do mapa global

Quando um agente percorre o ambiente, ele captura a informação ao seu redor atualizando o mapa local. A Figura 5.4 mostra um mapa global com três agentes. O mapa local de cada agente é mostrado à esquerda individualmente, sendo que $am_k$ corresponde ao mapa local do agente $a_k$. A área mapeada para o ambiente local é exibida no mapa global como um quadrado centrado na posição de cada agente.

Na versão apresentada por Dapper *et al.* em (DAPPER et al., 2006), somente os obstáculos que estivessem na região determinada por um triângulo formando o campo de visão do agente eram mapeados no mapa local. O objetivo era assegurar que obstáculos atrás do agente não interferissem em seu futuro movimento. Na prática, esta situação não apresentou melhores resultados em algumas situações, como no encontro de diversos agentes, pois quando um agente procura desviar de um obstáculo a sua frente, acaba se deparando com um obstáculo um pouco atrás, ocorrendo as vezes, um movimento de vai e vem.

Esta situação não ocorre quando é feito um mapeamento completo dos obstáculos dentro da região determinada pelo mapa local. Este mapeamento funciona como uma memória imediata do agente, ou seja, se ele ultrapassa um obstáculo, ele espera que o mesmo esteja atrás dele no momento seguinte, o que é condizente com nossa percepção da realidade. Também torna a implementação um pouco mais simples, pois pode-se mapear

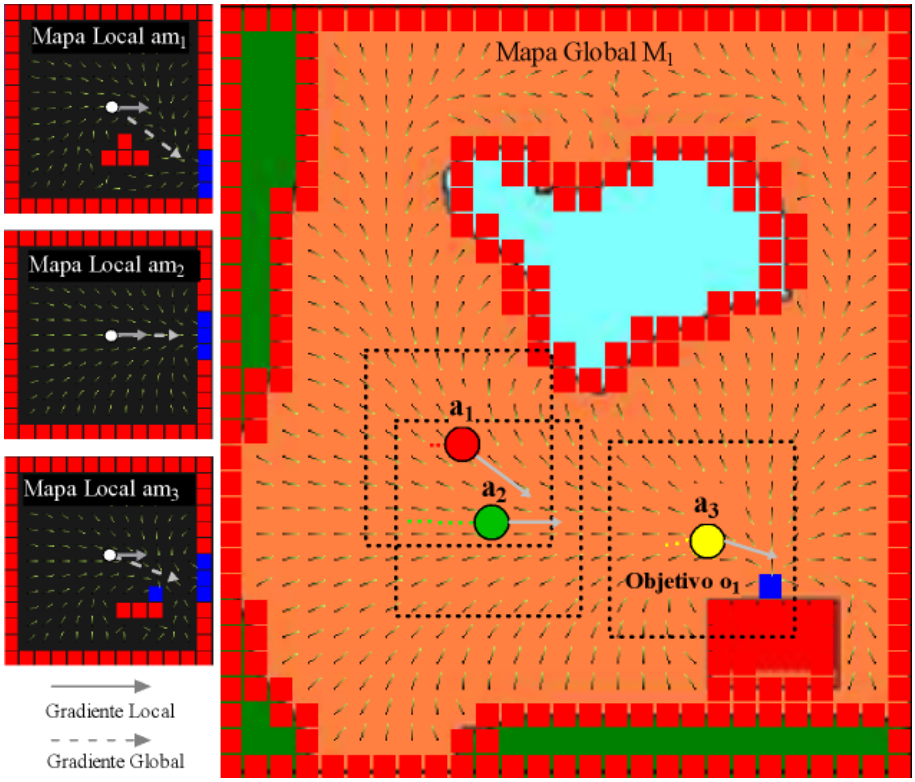



Figura 5.4: Atualização do Mapa Local através do Mapa Global. Na esquerda, um mapa local denotado por $am_k$ pertence a um agente $a_k$ representado no mapa global

diretamente os obstáculos estáticos do ambiente global para o local.

A Figura 5.5 mostra o mapa global e uma instância do mapa local para o agente $a_1$. Nesta figura, a região mapeada para a *zona-A* do mapa local está representada no mapa global de forma ilustrativa.

Após o mapeamento dos obstáculos, é gerado o objetivo local conforme a Figura 5.5. O objetivo é colocado em uma das bordas do mapa local segundo a direção indicada pelo gradiente global.

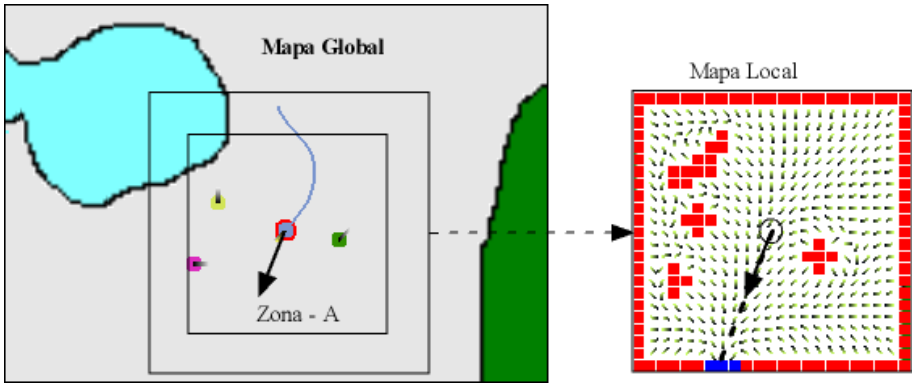



Figura 5.5: Mapeamento de obstáculos e criação de objetivo local.

### 5.1.4 Relaxamento da matriz

Após a etapa de mapeamento, o agente calcula o campo potencial no mapa local usando uma versão discreta da Equação 4.3 implementando o método de Gauss-Seidel

(ver seção 3.1.1.2):

$$ap_{i,j}^{k} = \frac{1}{4}(ap_{i-1,j}^{k+1} + ap_{i+1,j}^{k} + ap_{i,j-1}^{k+1} + ap_{i,j+1}^{k}) + \frac{\varepsilon^{k}}{8}((ap_{i+1,j}^{k} - ap_{i-1,j}^{k+1})v_{x}^{k} + (ap_{i,j+1}^{k} - ap_{i,j-1}^{k+1})v_{y}^{k}) \quad (5.1)$$

onde $\mathbf{v}^k = (v_x^k, v_y^k)$ e $\varepsilon^k \in [-2,+2]$(ver nota[2]) são os parâmetros comportamentais do agente $a_k$ atualizados a cada passo conforme Seção 5.2.1.

O campo potencial é parcialmente relaxado conforme comentado na Seção 3.2. O número de relaxações pode ser alterado, sendo o valor padrão igual a 30 para um mapa local de tamanho até 17x17. Observou-se nos experimentos que um número de iterações menor do que este pode fazer com que o campo potencial não forneça um caminho até o objetivo ou gere trajetórias pouco suaves. Para janelas maiores é recomendado um aumento no número de iterações, sendo que foram feitos testes com janelas com tamanho máximo de 35x35, onde 100 iterações foram suficientes para gerar trajetórias suaves e livres de colisão até o objetivo e permitir a geração de comportamentos individuais para os agentes.

### 5.1.5 Agente

O agente é implementado através de uma classe contendo atributos que se referem aos parâmetros e a outros objetos utilizados no algoritmo que controla sua trajetória (em tempo real). A Tabela 5.1 apresenta os atributos que são relevantes para a compreensão do algoritmo que será apresentado na Seção 5.2.

## 5.2 Algoritmo

Esta seção apresenta o algoritmo em passos gerais e descreve o seu funcionamento, bem como comenta o motivo da utilização das principais funções apresentadas a seguir, responsáveis pela geração do movimento dos agentes:

1. calcula o ambiente global $\{m_k\}$. (Uma matriz para cada objetivo $o_k$ conforme Seção 5.1.1)
2. instancia os agentes definindo seu comportamento inicial (parâmetros do agente) e objetivos a serem alcançados
3. repete para cada agente
   (a) se agente está em movimento

[2]O segundo termo da expressão 5.1, possuía originalmente o divisor igual a 4 (de acordo com a Equação 4.3) em vez de 8. Por isso, o intervalo de $\varepsilon$ está entre -2 e 2. Esta foi uma escolha meramente arbitrária. A Seção 6.1.2 apresenta experimentos relativos a variação deste parâmetro e demonstra qual é o intervalo que fornece melhores resultados

[3]No espaço de trabalho, o agente é visto como um disco de raio $r$. No espaço de configurações, onde é calculado cada nova posição do agente, ele é visto como um único ponto. Para garantir que não ocorra colisões, é necessário *expandir* o tamanho dos obstáculos por um raio igual ao raio do agente. No método utilizado, com uma matriz discretizando o espaço, as células que contêm obstáculos representam uma área um pouco maior do que a ocupada, permitindo uma margem maior de segurança.

[4]A variável *ma* (massa do agente) só é utilizada por uma técnica específica de geração da trajetória (determinada pela variável $tipoDeslocamento$) conforme apresenta a Seção 5.2.2

Tabela 5.1: Atributos do agente

| Atributo | Descrição |
|---|---|
| $p$ | Posição do agente no ambiente global |
| $\theta_{a_k}$ | Orientação do agente |
| $s$ | Tamanho do passo |
| $r$ | Raio do agente (no espaço de trabalho)[3] |
| $\mathbf{v}$ | Vetor Comportamental: altera o campo potencial local |
| $\varepsilon$ | Influência de $\mathbf{v}$ no Campo Potencial Local |
| $n$ | Ordem da Matriz Local (mapa local) |
| $at_{cel}$ | Tamanho da célula do mapa local |
| $am_k$ | Mapa Local: matriz de tamanho $n \times n$ |
| $p(m_k)$ | Mapa Global: ponteiro para uma matriz do conjunto $\{m_k\}$ |
| $\{o_k\}$ | Lista de Objetivos (cada objetivo se refere a um mapa global $m_k$ |
| $num_{rel}$ | Número de relaxações executadas a cada passo |
| $tipoComp_i$ | Conjunto de comportamentos pré-programados para cada agente |
| $tipoDeslocamento$ | Determina como é calculado o vetor deslocamento $\mathbf{d}$ (Seção 5.3) |
| $d_{vis}$ | Distância a frente do agente em que um obstáculo é detectado |
| $ma$ | Massa do Agente[4] |

i. executa função calculaPróximoPasso()
ii. executa função verificaObjetivo()

O usuário do aplicativo pode alterar as configurações de cada agente durante a execução do passo 3 fazendo com que ele altere seu comportamento. É possível colocar um agente em movimento ou fazer com que ele pare de andar. A função *calculaPróximoPasso* é responsável pela geração da trajetória determinando a próxima configuração do agente (posição e orientação):

**Função calculaPróximoPasso()**

1. captura o Ambiente Global
2. mapeia outros agentes e verifica se estão "visíveis"
3. calcula objetivo local
4. aplica comportamento
5. relaxa mapa local
6. captura gradiente local
7. calcula a direção do agente em função do gradiente
8. calcula a nova posição do agente

Os primeiros três itens se referem ao processo de criação do mapa local a partir do mapa global conforme descrito na Seção 5.1.3. O item 4 está descrito na Seção 5.2.1 e mostra como podem ser gerados determinados comportamentos a partir de propriedades

do agente, principalmente do vetor $\mathbf{v}$ e $\varepsilon$. O item 5 consiste em uma relaxação parcial do mapa local como já foi descrito na Seção 5.1.4. Os três últimos itens se referem a definição da trajetória utilizando o gradiente local conforme Seção 5.2.2.

Além da função $calculaPróximoPasso()$, o passo 3 do algoritmo possui a função $verificaObjetivo()$. Como cada agente possui uma fila de objetivos que devem ser alcançados, esta função é usada para testar se o objetivo atual foi atingido e em caso positivo selecionar o próximo. Assim, o agente passa a navegar através do novo mapa global ($m_{k+1}$). Quando o último objetivo é alcançado o agente pára de se mover ou é removido da aplicação (e conseqüentemente do ambiente virtual) se assim for configurado.

### 5.2.1 Gerando comportamentos

O algoritmo desenvolvido, baseado na Equação 4.3, permite gerar diferentes tipos de comportamentos variando adequadamente os parâmetros $\mathbf{v}$ e $\varepsilon$. Para cada agente, é possível aplicar uma função a estes parâmetros ou mantê-los com um valor constante. Também é possível trocar os comportamentos dinamicamente em função de algum critério arbitrário.

No protótipo criado, cada agente possui 2 tipos de comportamentos pré-programados dados pelo atributo $tipoComp_i$, onde $i \in \{1,2\}$ (ver Tabela 5.1). Quando o agente navega por uma região onde não existem obstáculos a sua frente, ele executa o comportamento determinado pelo primeiro tipo. Quando um obstáculo é detectado a uma distância $d < d_{vis}$, onde $d_{vis}$ é o raio de visão do agente, conforme a Tabela 5.1, ele executa o segundo.

Com esse mecanismo, o agente pode se antecipar a uma determinada situação e alterar sua trajetória. Se um agente detecta um outro vindo em sua direção, ele pode decidir se afastar de um possível contato ou mesmo se aproximar.

O critério de alteração de comportamento em função da aproximação de obstáculos é apenas um exemplo de fácil visualização, porém o tipo de comportamento poderia ser alterado segundo outros critérios a serem implementados. Por exemplo: a existência de muitos ou poucos agentes a sua frente, a detecção de um determinado objeto, ou a situação emocional do agente (que poderia se alterar por algum motivo como cansaço, irritação por alguma situação, dentre outros).

No protótipo também é possível configurar o agente para que ele execute somente um tipo de comportamento, ou, alterar o valor de $d_{vis}$. Como exemplo, diminuindo o valor de $d_{vis}$ é possível simular situações onde o agente possui um raio de ação muito curto, ou seja, só consegue alterar seu comportamento quando está próximo dos obstáculos. Isto é comum quando uma pessoa caminha em uma região com muitos agentes e só consegue desviar quando está muito próxima de uma possível colisão. Outro exemplo pode ser um agente cego que caminha com auxílio de uma guia e só consegue desviar quando o obstáculo está muito próximo.

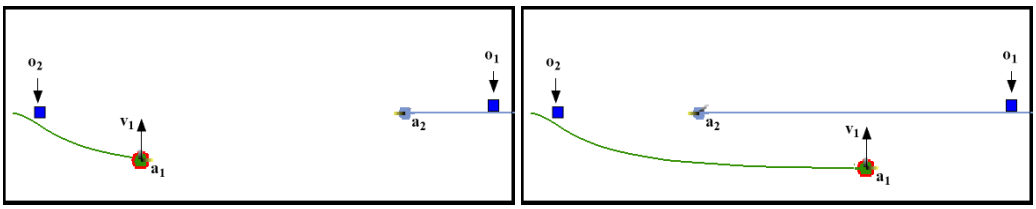
Cada tipo de comportamento corresponde a uma função que é aplicada sobre os parâmetros $\mathbf{v}$ e $\varepsilon$. Poderiam ser utilizadas qualquer função definida experimentalmente ou gerada a partir de um caminho pré-definido pelo usuário (sugestão de trabalho futuro). Como exemplo, foram implementadas quatro formas de atualizar estes parâmetros conforme descrito abaixo.

- Utilizar somente o valor individual de $\mathbf{v}$ e $\varepsilon$ do agente
- Utilizar somente funções harmônicas
- Atualizar $\mathbf{v}$ através de uma senóide

- Definir **v** como um vetor colinear ou ortogonal à reta que une o agente a um outro ponto

#### 5.2.1.1 *Utilizar o valor individual de* **v** *e* $\varepsilon$ *do agente*

Neste caso, o campo local é relaxado utilizando-se um valor constante para **v** e $\varepsilon$. Diferentes valores destes parâmetros sempre geram trajetórias diferentes entre as mesmas posições, inicial e final. A Figura 5.6 mostra um exemplo com dois agentes em sentidos contrários, sendo que $a_1$ possui o vetor comportamental $\mathbf{v}_1$ fixo, apontando para parede, enquanto $a_2$ utiliza funções harmônicas. A figura mostra também os objetivos individuais de cada agente, denotados por $o_1$ e $o_2$, respectivamente para os agentes $a_1$ e $a_2$. O próximo capítulo apresenta outros experimentos em relação a **v**.

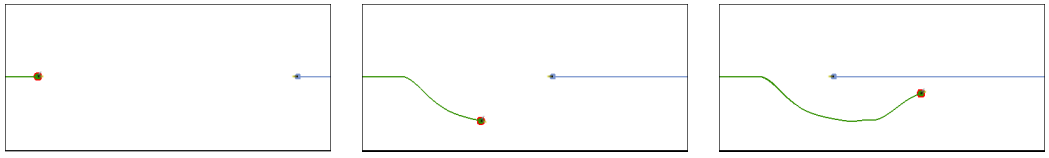



(a) (b)

Figura 5.6: Utilizando valores fixos para **v** e $\varepsilon$

#### 5.2.1.2 *Utilizar somente funções harmônicas*

Neste caso, $\varepsilon = 0$, ou seja, a Equação 4.3 se reduz à Equação de Laplace e portanto, tem-se funções harmônicas controlando os agentes. Como comentado antes, este método faz com que o agente busque sempre caminhos com afastamento máximo das *paredes* do ambiente, o que pode parecer muito estereotipado se for utilizado de forma isolada. No entanto, ele pode fazer parte de um conjunto de comportamentos que podem ser selecionados em situações específicas. Por exemplo, pode-se imaginar um agente caminhando pelo centro de um corredor (função harmônica) e quando ele avista um outro agente vindo em direção contrária ele passa a caminhar mais próximo de uma das paredes (configurando v para apontar para uma delas). Isto é mostrado na Figura 5.7. Na primeira parte, o agente segue o caminho gerado pelas funções harmônicas, quando ele avista o outro agente a uma distância $d_{vis}$ ele passa a caminhar pelo lado direito.



(a) (b) (c)

Figura 5.7: Alteração de comportamento

#### 5.2.1.3 *Atualizar* **v** *através de uma senóide*

Em vez de manter os parâmetros **v** e $\varepsilon$ fixos durante a realização de sua tarefa, como nos casos anteriores, um agente pode variá-los dinamicamente gerando comportamentos

complexos e interessantes como o mostrado na Figura 5.8. No exemplo apresentado nesta figura, o agente segue da esquerda para a direita através de uma trajetória oscilatória, formando uma senóide.

Quando passa próximo de obstáculos fixos, ele altera a amplitude do movimento para evitar a colisão, mas procura manter o comportamento definido pela função sempre que possível. No exemplo apresentado na figura, $\mathbf{v} = (1, \sin(w * t))$ onde $w = \pi/18$ e $t$ é o passo de simulação corrente. De acordo com essa função, a componente horizontal de **v** é mantida fixa em 1, enquanto a vertical oscila entre [-1,1]. A componente horizontal de **v** é colinear ao fluxo do mapa global que empurra o agente da esquerda para a direita. Essa condição é necessária para gerar o comportamento condizente com a função que se está aplicando, ou seja, a função deve ser aplicada sobre o eixo definido pelo fluxo do mapa global, na região em que o agente se encontra.

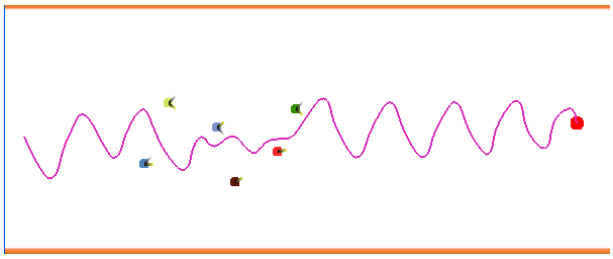


Figura 5.8: Aplicação de uma senóide ao vetor **v**

#### *5.2.1.4 Vetor colinear ou ortogonal a reta que une o agente a um outro ponto*

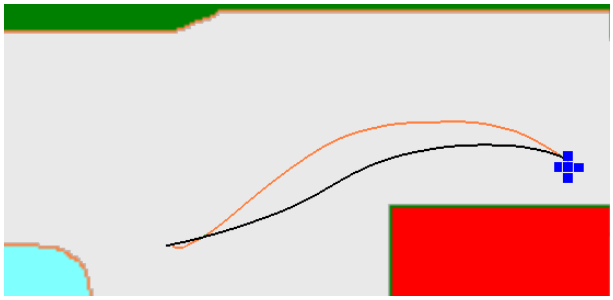
Como visto, **v** tende a *empurrar* o agente para uma direção em particular. Quando **v** aponta diretamente para o objetivo, o agente tende a seguir o caminho mais curto até ele. Quando ele aponta para um outro agente, o primeiro tende a se aproximar deste último. Estas idéias são ilustradas na Figura 5.9. A trajetória mais clara exibe o comportamento com funções harmônicas (agente $a_1$), enquanto a mais escura (agente $a_2$) ilustra o comportamento gerado pelo vetor apontando, respectivamente, para o objetivo (a) e para um outro agente parado (b).

Uma outra forma de utilizar a reta entre dois pontos é executar uma rotação no vetor gerado. Aqui é apresentado um exemplo no qual o vetor sofre uma rotação de 90 graus. Isto gera mais tipos de comportamentos básicos que podem ser utilizados em movimentos mais complexos, porém sua aplicação direta em alguma situação realística não foi endereçada.

A Figura 5.10(a) exibe o comportamento gerado por esta técnica quando utilizada desde o ponto de partida, enquanto a Figura 5.10(b) exibe um comportamento no qual os agentes seguem funções harmônicas e ativam o comportamento quando avistam o outro agente a uma distância $d_{vis}$.

### 5.2.2 Cálculo do próximo passo do agente

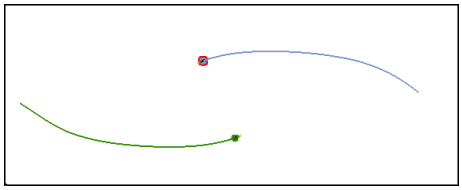
O aplicativo desenvolvido trabalha em um espaço de configurações 2D, portanto o planejamento de movimento se refere à translações no plano (ver Capítulo 1). Esta forma de trabalhar é utilizada por diversos autores que desenvolvem animações com o auxílio de planejadores em camadas (Seção 2.2) e consiste em uma boa solução em termos de desempenho considerando o custo elevado da resolução de um PVC. Além disso, a



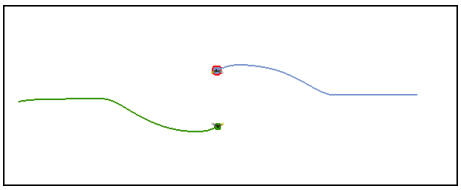
(a)

(b)

Figura 5.9: Vetor apontando para o objetivo (a) e para um outro agente (b)



(a)



(b)

Figura 5.10: Vetor Ortogonal a Reta entre dois Agentes

construção adequada do espaço de configurações 3D, para uma aplicação que simula a navegação de pedestres, pode ser significativamente complexa.

Assim, no trabalho desenvolvido, a nova posição do agente pode ser definida da seguinte forma:

Seja $p$, a posição do agente em um determinado momento. A nova posição $p'$ é dada por

$$p' = p + \mathbf{d}.s \tag{5.2}$$

onde $\mathbf{d}$ é o vetor deslocamento que representa a direção do próximo passo e possui módulo igual a 1 e $s$ é um multiplicador representando o tamanho do passo do agente.

Em um ambiente estático, a equação acima pode ser usada para gerar uma curva no ambiente, ou seja, realizar planejamento de caminhos (ver Capítulo 1). A orientação do agente (ou seja, para onde ele está voltado em cada momento) pode ser calcula em uma segunda fase onde o agente percorre o caminho gerado (*path following*). Uma forma de fazer isto é simplesmente utilizar a reta tangente a curva como direção do agente. Tratamentos e filtros também podem ser feitos sobre a curva antes de iniciar a segunda etapa.

Em um ambiente parametrizado pelo tempo (*planejamento de movimento*), é preciso criar uma forma de definir a orientação do agente a cada passo, junto com o deslocamento do agente, de forma que o planejador consiga controlar o personagem em tempo real.

O vetor **d** pode ser usado para representar a orientação do agente a cada passo, ou seja, as trajetórias são geradas através de uma rotação do agente (alinhando-o com o vetor **d**) e então realizando-se um deslocamento através da multiplicação do vetor pelo tamanho do passo.

Foram propostas três formas de se definir o vetor **d** em função dos resultados encontrados (como visto este vetor pode ser utilizado para definir a orientação do agente, ou somente como deslocamento, enquanto que a orientação é definida de outra forma):

- Utilizar diretamente o gradiente descendente;
- Utilizar a média entre o gradiente e o valor de **d** anterior;
- Através de uma função que considera a variação de **d** e uma média ponderada com o gradiente atual.

As três técnicas serão descritas a seguir enquanto que no Capítulo 6 serão apresentados resultados da utilização das três em experimentos específicos e feitas algumas conclusões a respeito. Como visto, todas utilizam o gradiente atual, denotado por $\mathbf{dgrad}^k$, onde $k$ é o número do agente. O valor de $\mathbf{dgrad}^k$ é calculado da seguinte forma:

$$\mathbf{dgrad}^k = \left( (ac^k_{p_x+1,p_y} - ac^k_{p_x-1,p_y})/2, (ac^k_{p_x,p_y+1} - ac^k_{p_x,p_y-1})/2 \right) \tag{5.3}$$

onde $p_x = \lceil l^k_x/2 \rceil$ e $p_y = \lceil l^k_y/2 \rceil$ representam a posição do centro do mapa local(Seção 5.1.2).

#### *5.2.2.1 Utilizar diretamente o gradiente descendente*

Esta forma foi utilizada no primeiro protótipo onde foram testados comportamentos para o agente em um único campo global. Neste caso, apesar dos caminhos gerados se mostrarem suaves, a variação do gradiente entre um passo e outro pode ser muito brusca para que seja considerada como a orientação do agente. Quando foi introduzida a arquitetura com mapa local para cada agente, mesmo os caminhos gerados passaram a sofrer muitas oscilações de forma que este método teve que ser modificado como descrito a seguir. O motivo disso ocorrer é a natureza discreta do campo vetorial, quando o tamanho das células diminui (no mapa global e local), os caminhos se tornam mais suaves.

#### *5.2.2.2 Utilizar a média entre o gradiente e o valor de **d** anterior*

Neste método o vetor **d**, no instante $t$ é definido por

$$\mathbf{d}^t = \frac{(\mathbf{d}^{t-1} + \mathbf{dgrad}^k)}{\|(\mathbf{d}^{t-1} + \mathbf{dgrad}^k)\|} \tag{5.4}$$

onde $\mathbf{dgrad}^k$ é o gradiente descendente calculado pela Equação 5.3. Esta interpolação torna os caminhos mais suaves e como **d** sofre menos variações, ele pode ser utilizado como vetor de orientação do agente. Porém, no caso da simulação de vários agentes caminhando próximos, onde não há possibilidade de desviar com antecendência, a variação de **d** é maior fazendo com que os resultados não pareçam muito naturais. Para melhorar um pouco este problema, o valor de $s$ (passo do agente) foi diminuido quando o agente está sob influência de um gradiente menor que um determinado parâmetro (normalmente

isto ocorre em locais com muitos obstáculos). Assim a Equação 5.2 pode ser redefinida por

$$p' = p + \mathbf{d}.s.F(\mathbf{dgrad}^k) \tag{5.5}$$

onde $F(\mathbf{dgrad}^k) \in [0,1]$ retorna um multiplicador que diminui o tamanho do passo do agente em função do gradiente e $\mathbf{d}^t$ é calculado pela Equação 5.4.

Esta técnica foi utilizada por Dapper *et al.* (DAPPER et al., 2006) apresentando bons resultados na geração de caminhos. Um aprimoramento desta técnica é apresentado a seguir, incluindo a substituição da função $F$ por outra que considera a variação do gradiente local durante a trajetória.

#### *5.2.2.3 Função que considera a variação de **d** e uma média ponderada*

Esta técnica é semelhante a utilização da Equação 5.5 porém com uma nova definição para a função que multiplica $s$, e com o vetor $\mathbf{d}^t$ calculado por

$$\mathbf{d}^t = \frac{\mathbf{d}^{t-1} \cdot ma + \mathbf{dgrad}^k}{\|\mathbf{d}^{t-1} \cdot ma + \mathbf{dgrad}^k\|} \tag{5.6}$$

Na Equação 5.6, *ma* determina um peso para a antiga direção do agente e pode ser associada a massa de um objeto determinando sua tendência a continuar seu movimento anterior (inércia).

Esta média ponderada torna as trajetórias mais suaves do que as técnicas citadas anteriormente, porém ela deve ser usada com cuidado, pois não garante a inexistência de colisões. A medida que *ma* aumenta, a chance de colisões também aumenta, o que é aceitável e condizente com princípios físicos (*leis de Newton*). Quanto maior a massa de um objeto, mais dificilmente ele conseguirá desviar de um obstáculo próximo (e de forma proporcional ao quadrado de sua velocidade).

Na prática, é possível impedir que ocorram colisões determinando que o agente pare de se mover quando for detectada uma situação de risco. Isto é feito através da multiplicação de $s$ por uma função da variação do gradiente. Assim, pode-se redefinir a Equação 5.5 por

$$p' = p + \mathbf{d}.s.F_a(\cos\theta_{dir}) \tag{5.7}$$

onde $\theta_{dir}$ é o ângulo entre o gradiente atual e a orientação anterior dada por $\mathbf{d}$,

$$\cos\theta_{dir} = \frac{\nabla \cdot \mathbf{d}}{\|\nabla\|\|\mathbf{d}\|}$$

A definição da Equação 5.7 foi baseada na observação de pedestres e no método de planejamento desenvolvido.

Quando um pedestre está em rota de colisão com outro e não consegue se antecipar a este fato (o que ocorre em regiões muito densas), ele diminui a sua velocidade e procura um caminho livre. Quando a possibilidade de colisão é muito grande ele pode inclusive parar de caminhar e então trocar de direção retomando o movimento.

No método implementado, a avaliação do risco de colisão pode ser feita através da variação do gradiente. Quando um agente caminha em uma região livre, sua direção tende a se manter constante. Quando um agente em sentido contrário se aproxima, o campo local é alterado fazendo com que o agente procure se afastar do obstáculo mapeado. O ponto máximo de variação do gradiente em relação ao movimento do agente ocorre quando ele estiver seguindo na mesma linha de um outro, em sentido contrário.

Tabela 5.2: $F_a(\cos\theta_{dir})$

| $\cos\theta_{dir}$ | $\theta_{dir}$ | $F_a$ |
|---|---|---|
| $[-1, 0.01)$ | 180 - 90.0 | 0 |
| $[0.01, 0.30)$ | 90.0 - 72.5 | 0.05 |
| $[0.30, 0.60)$ | 72.5 - 53.1 | 0.2 |
| $[0.60, 0.71)$ | 53.1 - 44.8 | 0.5 |
| $[0.71, 0.92)$ | 44.8 - 23.1 | 0.8 |
| $[0.92, 1]$ | 23.1 - 0.0 | 1 |

A Figura 5.11 mostra o processo de atualização de **d**. O agente $a_1$ inicia com o gradiente $\mathbf{dgrad}^1$ alinhado com $\mathbf{d}^t$ e igual a $\mathbf{d}^{t-1}$, quando encontra o agente $a_2$, o novo valor de $\mathbf{dgrad}^1$ possui um grande desvio (mudança de ângulo) em relação ao anterior. O novo valor de **d** é então calculado levando em consideração $ma_{a_1}$ e $\mathbf{d}^{t-1}$ conforme Equação 5.6.

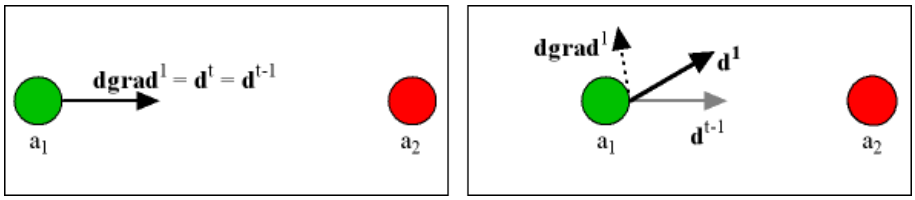



Figura 5.11: Deslocamento do agente

A Tabela 5.2 mostra um exemplo de definição para a função $F_a(\cos\theta_{dir})$. Quando $\theta_{dir}$ for menor do que 23.1 graus, o agente se move com a sua velocidade normal. A medida em que este ângulo aumenta, sua velocidade é reduzida de acordo com $F_a$. Quando o ângulo entre o deslocamento anterior e o gradiente for maior do que 90 graus, o agente não se move. Neste caso, somente **d** é atualizado, conforme mencionado anteriormente.

A técnica descrita nesta seção será considerada padrão, sendo utilizada nos demais experimentos quando não for explicitado o uso de uma das anteriores.

# 6 EXPERIMENTOS E RESULTADOS

Durante este trabalho, foram desenvolvidos diversos experimentos para avaliar as técnicas e testar sua utilidade na geração de caminhos para humanos virtuais. Foram propostas algumas situações realísticas, como vários agentes em um ambiente externo, agentes caminhando em um corredor em sentidos opostos e outras não muito naturais, mas que exemplificam a influência dos parâmetros sobre a definição da trajetória do agente.

A próxima seção mostrará a influência dos parâmetros **v**, $\varepsilon$ e do tamanho do mapa local sobre a trajetória gerada, dado um mesmo mapa global. Em seguida, será apresentada uma avaliação das técnicas descritas na Seção 5.2.2 usadas para gerar o vetor **d**. Por último, serão mostrados os resultados de algumas simulações realizadas para diferentes cenários e comportamentos para os agentes.

## 6.1 Influência dos parâmetros na geração das trajetórias

### 6.1.1 Variando o vetor comportamental v

Este experimento tem como objetivo demonstrar a influência do vetor **v** na geração das trajetórias. A alteração dinâmica de **v**, através de funções, já foi exemplificada no capítulo anterior (Seção 5.2.1) e o seu uso em situações realísticas é apresentado na Seção 6.3.

#### *6.1.1.1 Exemplos de comportamentos gerados pelo vetor **v***

O cenário apresentado aqui é simplesmente um corredor contendo uma posição inicial e outra final. São utilizados diferentes valores para **v**, mantendo os demais parâmetros com um valor constante conforme a Tabela 6.1.[1] Neste caso, o agente possui um único comportamento definido por $tipoComp_1$.

Tabela 6.1: Parâmetros dos agentes para o Experimento 6.1

| Atributo | Descrição |
|---|---|
| $\varepsilon$ | 0.7 |
| $n$ | 15 |
| $num_{rel}$ | 30 |
| $tipoComp_1$ | Vetor individual constante |
| $tipoDeslocamento$ | Média Ponderada |
| $ma$ | 4 |

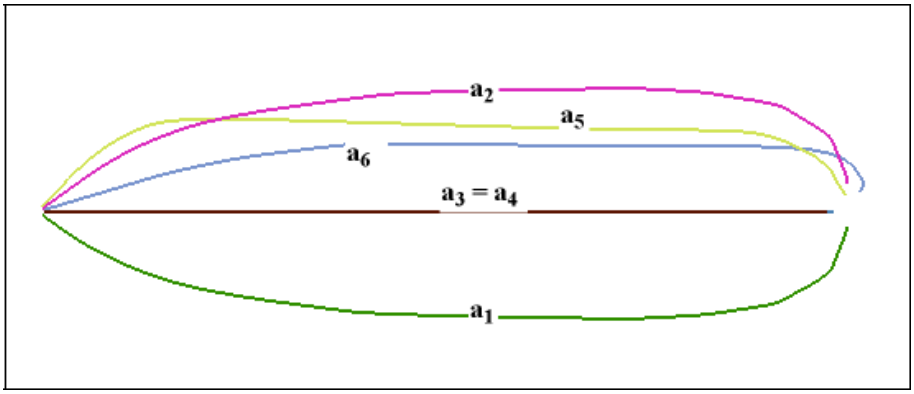
[1] Uma descrição dos atributos dos agentes pode ser encontrada na Tabela 5.1

Na Figura 6.1(a) são exibidos diferentes caminhos gerados pela utilização dos seguintes valores para o vetor **v**:

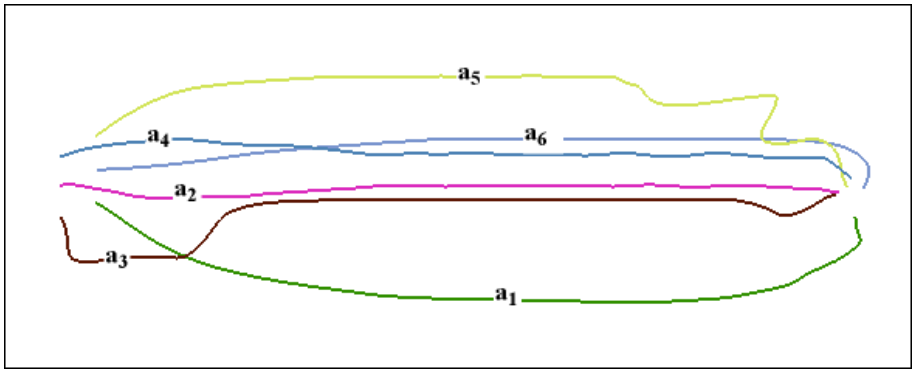
$$\{\mathbf{v}_1 = (0,1), \mathbf{v}_2 = (0,-1), \mathbf{v}_3 = (1,0), \mathbf{v}_4 = (-1,0), \mathbf{v}_5 = (0.7,-0.7), \mathbf{v}_6 = (-0.7,-0.7)\}$$

Conforme estes valores, os agentes $a_1$ e $a_2$ caminham o mais próximo possível de uma das paredes, cada um de um lado, pois seus vetores possuem sentido inverso um do outro. Já os agentes $a_3$ e $a_4$ possuem vetores alinhados com o sentido do gradiente global (no centro do corredor) e apresentam o mesmo caminho (estão sobrepostos). Os agentes $a_5$ e $a_6$ possuem **v** formando um ângulo de 45 graus em relação a direção do corredor, conforme os valores $\mathbf{v}_5$ e $\mathbf{v}_6$. O agente $a_5$ inicia com uma virada mais brusca em direção a parede (ver Subseção 6.1.1.2) enquanto $a_6$ executa um desvio mais suave, sem se afastar muito do centro do corredor.

Na Figura 6.1(b), utilizou-se os mesmos parâmetros, porém colocando um agente para cada valor de **v** do experimento anterior e deixando todos andar ao mesmo tempo. Na Figura 6.1(b) os agentes procuram seguir a trajetória definida pelo vetor **v**, porém alguns não conseguem porque são bloqueados por outros agentes caminhando ao seu lado com a mesma velocidade.




(a)




(b)

Figura 6.1: Diferentes valores para **v**

### *6.1.1.2 Análise das modificações causadas no mapa local*

A Figura 6.2 mostra o campo vetorial e potencial do mapa local para cinco valores diferentes de **v**, indicados pelas setas no canto inferior esquerdo de cada campo vetorial. A Figura 6.2(a) mostra o campo gerado por funções harmônicas para fins de comparação.

É possível perceber que o campo sofre uma distorção formando uma *linha* que parte do objetivo local e segue em sentido inverso a direção apontada por **v**. O campo vetorial resultante, tende a empurrar os agentes para esta linha (em direção ao objetivo).

Quando um agente está sobre esta linha, suas trajetórias podem sofrer uma certa *turbulência* fazendo com que sua trajetória sofra oscilações. Isto ocorre porque o agente passa de um lado para o outro da linha, gerando um movimento de ziguezague. Este efeito pode ser amenizado através das técnicas de interpolação da trajetória descritas no capítulo anterior.

Quando o vetor está apontando para uma direção contrária ao objetivo local, como mostram as Figuras 6.2(c,e), a linha gerada fica fora do mapa local. Nestes casos, o campo local é mais suave, porém é possível perceber que o campo vetorial não consegue envolver todo o mapa (problema de achatamento).

### 6.1.2 Variando $\varepsilon$

Neste experimento o vetor comportamental foi mantido constante em $\mathbf{v} = (0,1)$ e utilizou-se os seguintes valores de $\varepsilon$ para cada agente: $\{\varepsilon_1 = -0.1,\ \varepsilon_2 = 0,\ \varepsilon_3 = 0.1,\ \varepsilon_4 = 0.3, \varepsilon_5 = 0.7, \varepsilon_6 = 1.0, \varepsilon_7 = 1.2, \varepsilon_8 = 2, \varepsilon_9 = 3, \varepsilon_10 = 4\}$, onde $\varepsilon_i$ é o valor de $\varepsilon$ para o agente $a_i$. Os demais parâmetros são os mesmos da Tabela 6.1. As trajetórias geradas são mostradas na Figura 6.3. Para $\varepsilon_2 = 0$ o agente percorre uma trajetória retilínea entre a posição inicial e a final, o que equivale a utilização de funções harmônicas. A medida em que o valor do seu módulo aumenta, aumenta o afastamento em relação a esta trajetória retilínea até um determinado valor em que o campo fica *corrompido* de forma que não é mais gerado um caminho correto.

Para o valor $\varepsilon_9 = 3$ a trajetória fornecida não é mais segura. Como pode ser visto na figura, o agente inicia caminhando na direção contrária ao objetivo e só então retoma o caminho. Quando ele se aproxima do objetivo, a trajetória sofre oscilações consideráveis. Um valor seguro para $\varepsilon$ está aproximadamente no intervalo $[-2;2]$, considerando testes realizados com o mapa local tendo $n \in [11;31]$. A medida que $n$ cresce, *epsilon* possui uma influência maior, como é mostrado no Capítulo 3. Assim, pode-se usar um *epsilon* menor a medida que $n$ cresce.

É possível observar ainda que as trajetórias para os valores entre 1 e 2 estão quase sobrepostas, o que sugere que a maior variação de $\varepsilon$ ocorre no intervalo $[-1;1]$. Assim, o valor recomendável para $\varepsilon$ está em $[-1.2;1.2]$.

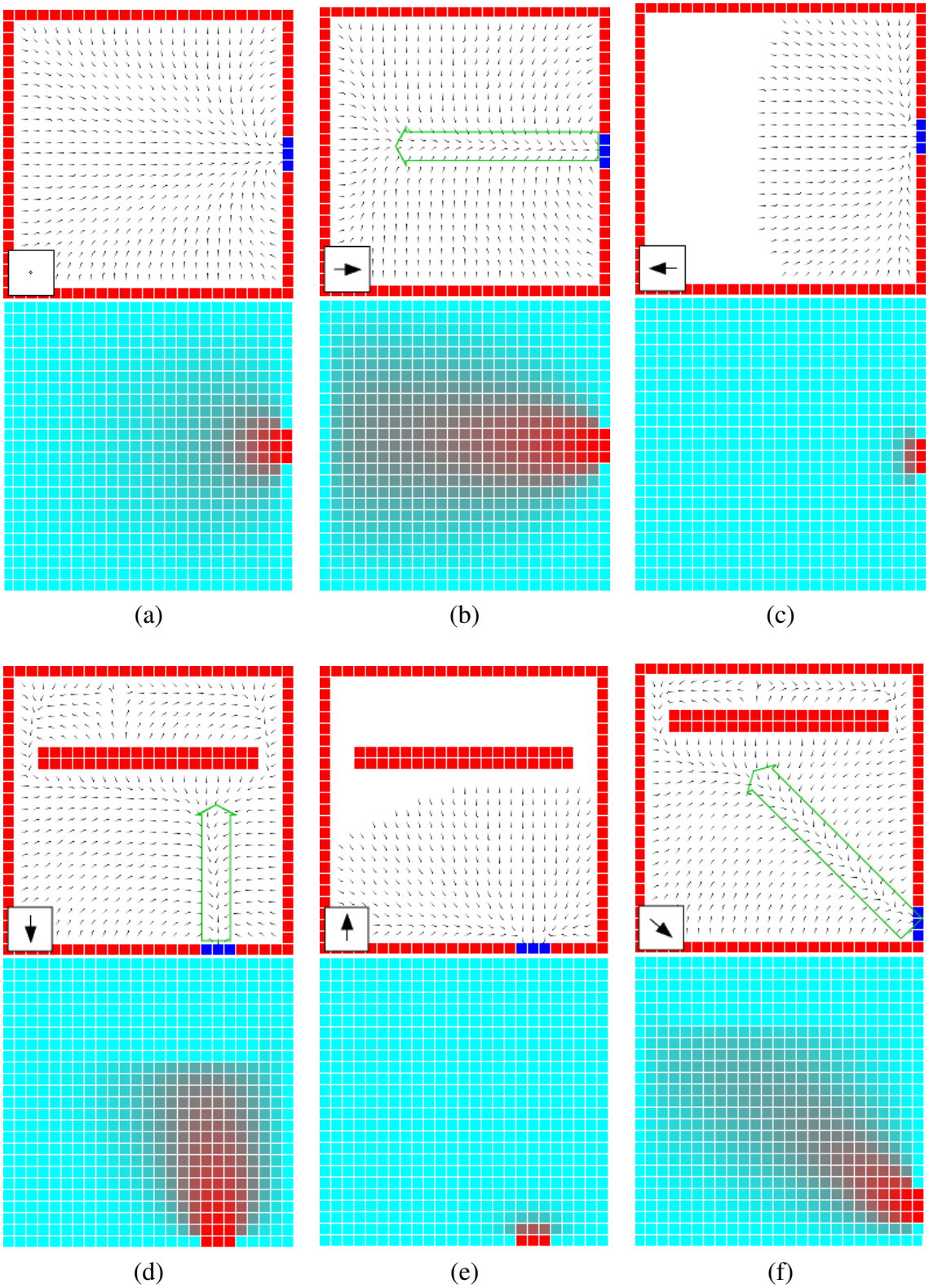
Outra questão que pode ser observada na figura é que para os valores $\varepsilon_1 = -0.1$ e $\varepsilon_3 = 0.1$ o caminho gerado é simétrico em torno do centro ($\varepsilon_2 = 0$). Isto porque, $\varepsilon$ é multiplicado pelo vetor **v** durante o relaxamento (assim, pode ser interpretado como $\|\mathbf{v}\|$).

### 6.1.3 Tamanho da célula

O tamanho da célula no mapa local está diretamente relacionada à qualidade das trajetórias geradas, sendo que quanto menor o seu tamanho, mais suaves serão os caminhos.

O experimento mostrado na Figura 6.4 demonstra esta situação. Foram mantidos fixos todos os parâmetros conforme a Tabela 6.2, com excessão do tamanho da célula $at_{cel}$ e de $n$. Para o cálculo de **d**, utilizou-se diretamente o gradiente descendente local para que as oscilações causadas ficassem mais evidentes.

O valor de $n$ foi alterado para que área coberta pelo mapa local seja sempre a mesma, sendo a largura do mapa local= $at_{cel} \times n$. Assim, quanto menor o tamanho da célula, maior deve ser o valor de $n$ para que o mapa local cubra a mesma área no ambiente global. Pode-se optar no entanto por janelas menores, isto é, fazer com que o mapa local corresponda



(a) (b) (c)

(d) (e) (f)

Figura 6.2: Alteração do Mapa Local pelo parâmetro **v**

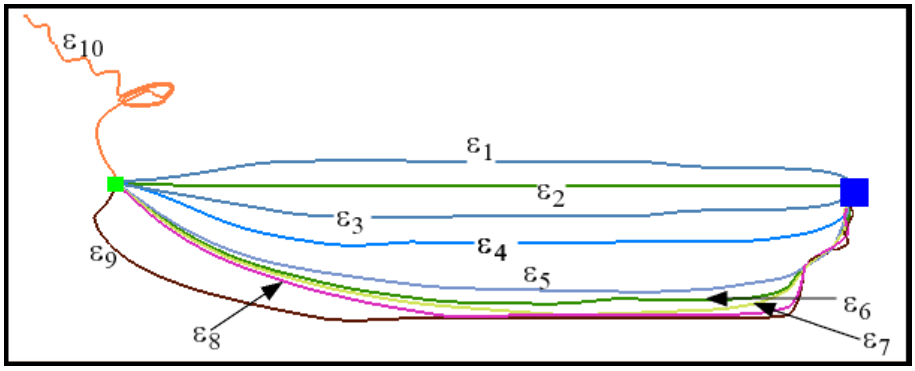



Figura 6.3: Variando $\varepsilon$

Tabela 6.2: Parâmetros do agente para a Figura 6.4

| Atributo | fig(a) | fig(b) | fig(c) |
|---|---|---|---|
| $at_{cel}$ | 0.8m | 0.59m | 0.42m |
| $n$ | 11 | 15 | 21 |
| $\varepsilon$ | 1 | 1 | 1 |
| $\mathbf{v}$ | $(0.7, 0.7)$ | $(0.7, 0.7)$ | $(0.7, 0.7)$ |
| $num_{rel}$ | 60 | 60 | 60 |
| $tipoDeslocamento$ | $dgrad^k$ | $dgrad^k$ | $dgrad^k$ |

a uma área menor no mapa global. A alteração desta área influencia na geração das trajetórias e pode ser útil em algumas situações como descrito na Seção 6.1.4.

Deve-se considerar ainda que, para matrizes maiores, são necessárias mais iterações para a convergência da solução. Nos experimentos realizados foram utilizadas entre 30 e 60 iterações para se obter bons resultados. A utilização de um $n$ maior, exigirá um número maior de iterações. Além disso, deve-se estar ciente do crescimento quadrático do número de células (no caso bidimensional).

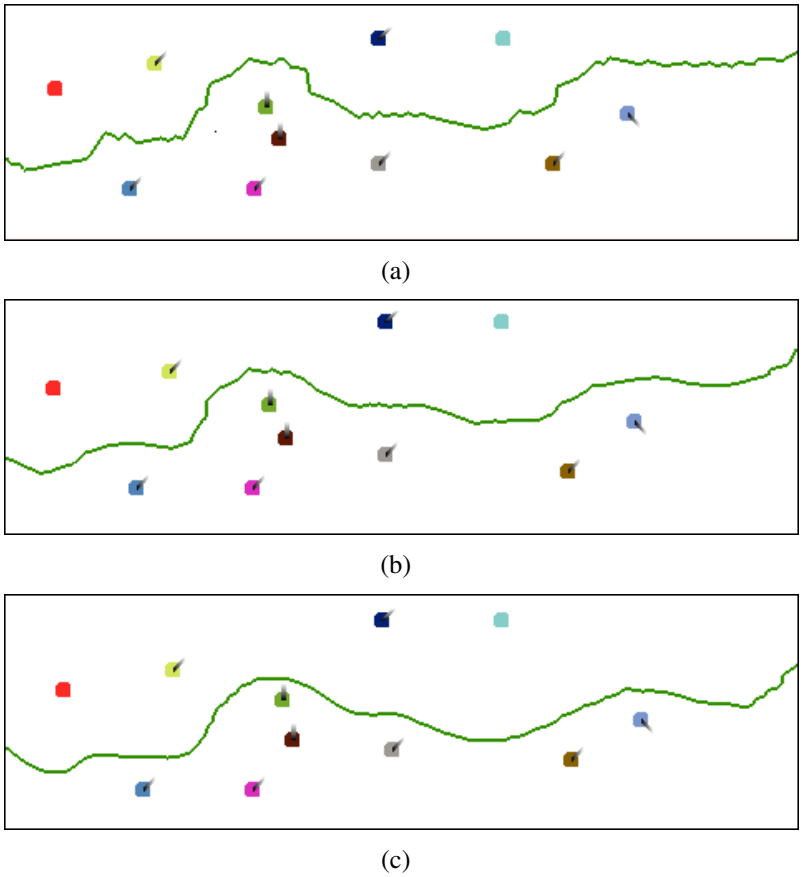
### 6.1.4 Tamanho do mapa local

Esta variação corresponde a aumentar a área coberta pelo mapa local em relação ao mapa global. Dessa forma, mais informação pode ser percebida pelo agente fazendo com que ele escolha a trajetória mais livre. Isto pode ser desejável ou não, de acordo com o comportamento que se queira atribuir ao agente.

A Figura 6.5 mostra um exemplo onde um agente encontra um grupo parado a sua frente e segue uma trajetória utilizando funções harmônicas. Na Figura 6.5(a), o mapa local possui largura aproximadamente igual a metade da largura do corredor, enquanto que na Figura 6.5(b), o mapa possui largura igual a do corredor. Com a janela pequena, o agente passa entre os agentes, enquanto que na segunda, quando ele possui conhecimento de todo o espaço diponível (largura do corredor), ele desvia do grupo mantendo a máxima distância entre os obstáculos.

## 6.2 Avaliação dos métodos de geração do deslocamento do agente

Como visto na Seção 5.2.2, foram propostos três métodos para definir o vetor **d** (*deslocamento do agente*). O terceiro método demonstrou fornecer trajetórias mais suaves em



(a)

(b)

(c)

Figura 6.4: Variando o tamanho da célula.

todas as situações testadas e foi escolhido como sendo a técnica padrão do planejador criado. Nesta seção será apresentado um experimento onde os três métodos foram aplicados a um mesmo cenário e depois serão avaliadas as influências dos parâmetros *ma* e $F_a$.

### 6.2.1 Comparação das três técnicas

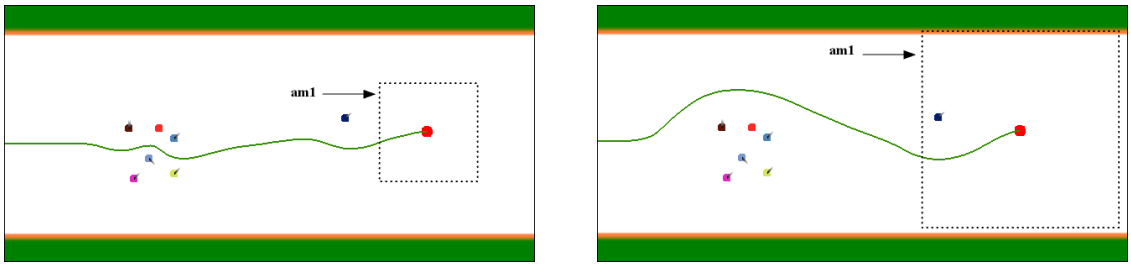
Neste experimento foi montado um cenário que simula uma corrida onde os agentes partem ao mesmo tempo de uma região (em posições próximas) e precisam chegar ao mesmo objetivo. Os agentes possuem um único tipo de comportamento determinado pelo valor individual de **v**.

A Figura 6.6 mostra o resultado da execução do experimento variando apenas o método de atualização de $d$. Na Figura 6.6(a) foi utilizado apenas o gradiente, na Figura 6.6(b) a média com o deslocamento anterior, e na Figura 6.6(c) a técnica padrão (média ponderada).

### 6.2.2 Variando o parâmetro *ma*

Este experimento consiste em variar o parâmetro *ma* utilizando a técnica de média ponderada. Como comentado anteriormente, a utilização de simulação física do movimento está além do escopo deste texto, sendo que cada configuração do agente é definida pela Equação 5.2. No entanto, pode-se fazer algumas interpretações livres que não contrariam princípios físicos. Assim, a variável *ma* pode ser interpretada como a massa do agente e define a tendência do mesmo em continuar seu percurso.

A Figura 6.7 mostra um exemplo onde *ma* é aumentada progressivamente. Na Figura




(a) (b)

Figura 6.5: Tamanho da janela

6.7(a) $ma = 0$, o que equivale a utilizar a Equação 5.2 com $d = \nabla p_{local}$, assim é possível verificar uma pequena oscilação na trajetória (quando são utilizadas funções harmônicas as trajetórias sofrem menos oscilações do que quando são definidos valores para $\mathbf{v}$ e $\varepsilon$).

A medida que aumenta *ma*, o agente têm certa dificuldade em executar uma rotação, executando trajetórias mais sinuosas e às vezes só desviando de um agente quando está bem próximo ($m = 24$).

Na Figura 6.7(e), *ma* é tão elevada que o agente acaba não conseguindo atingir o objetivo e acaba entrando em *órbita*, executando uma trajetória elíptica em torno do objetivo (uma análise matemática do resultado está além do escopo deste trabalho).

### 6.2.3 Variando $F_a$

Como visto na Seção 5.2.2, a definição de uma função que altera a velocidade do agente em regiões próximas a obstáculos torna as trajetórias seguras. Esta seção mostra um exemplo onde a função definida na Tabela 5.2 é comparada com outra onde $s$ não é modificado ($F_a = 1$).

Na Figura 6.8(a), os dois agentes em rota de colisão não alteram sua velocidade de forma que, para não colidirem, acabam realizando um grande desvio de sua rota, o que não é muito natural para um pedestre. No exemplo, um deles acaba realizando um caminho em forma de *laço*.

Já na Figura 6.8(b), os agentes utilizam a função $F_a$ mencionada acima, de forma que quando estão próximos de uma colisão, diminuem a velocidade. É possível perceber que um deles pára de se mover temporariamente, enquanto o outro cruza a sua frente. Esta situação decorre de um equilíbrio natural, sem a definição de qualquer prioridade.

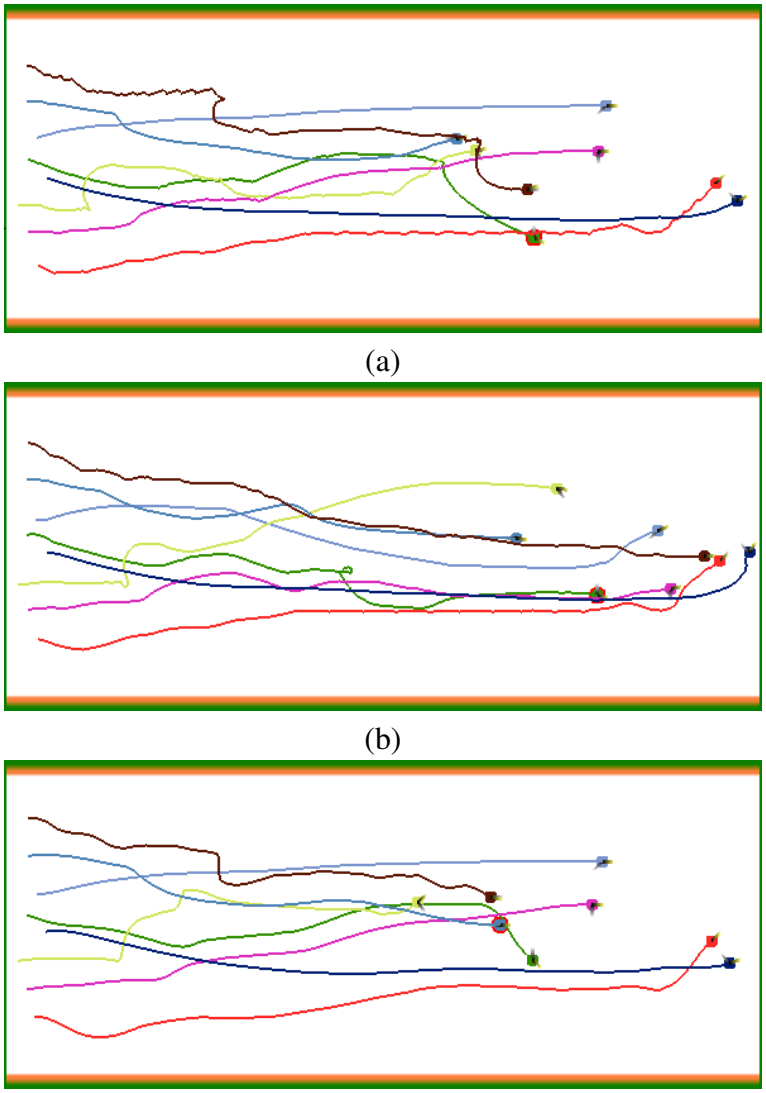
Um estudo mais profundo sobre a escolha da função $F_a$ através de comparações com dados reais de pedestres pode trazer melhorias no nível de realismo dos caminhos gerados, mas está além do escopo deste trabalho.

## 6.3 Aplicando as técnicas pesquisadas a diferentes cenários

Esta seção apresenta diferentes situações onde são aplicadas as técnicas desenvolvidas com o objetivo de testar suas potencialidades e avaliar suas aplicações em algumas situações realísticas.

### 6.3.1 Agentes em um corredor

Dois agentes caminhando em direções contrárias é uma situação comum que pode ser usada para avaliar as técnicas desenvolvidas. Estudos psicológicos (RYMILL; DODG-



(a) (b) (c)

Figura 6.6: Comparação das técnicas de atualização de **d** em uma situação com vários agentes

SON, 2005) mostram que uma pessoa nesta situação evita a possibilidade de colisão com alguma antecendência, simplesmente procurando caminhar mais próxima de uma das paredes. Em ambientes com muito movimento de pedestres o afastamento ocorre com menos antencedência e em menor intensidade. Estas situações podem ser simuladas através do ajuste adequado dos parâmetros **v** e $\varepsilon$. Além destes, outros comportamentos podem ser simulados, por exemplo, em vez de se afastar, um agente pode querer passar mais próximo de um outro.

Nos exemplos abaixo, os agentes iniciam caminhando pelo centro do corredor e quando detectam um agente a uma distância arbitrária $d_{vis}$ alteram **v** para um valor constante definido para cada caso.

A Figura 6.9 mostra dois agentes em rota de colisão utilizando funções harmônicas. Quando detectam a presença de outro agente, o vetor comportamental é ativado de acordo com a direção das setas na Figura 6.9(b). Após se cruzarem, eles retomam o seu comportamento anterior.

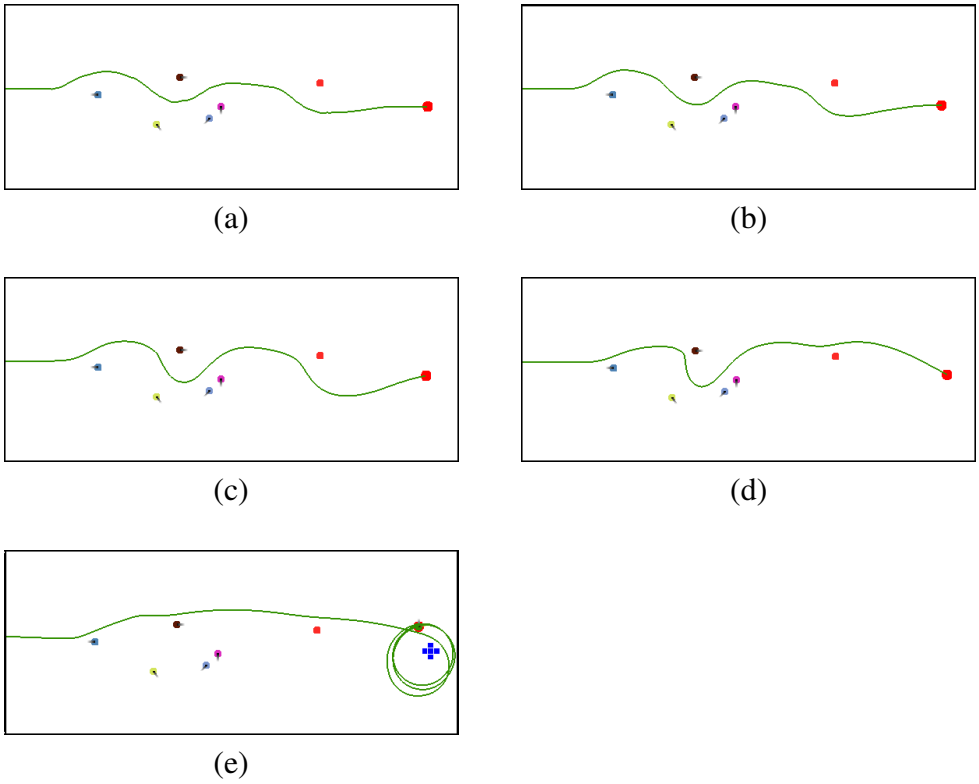



Figura 6.7: Variando *ma* utilizando funções harmônicas. (a) $ma = 0$; (b) $ma = 4$; (c) $ma = 12$; (d) $ma = 24$; (e) $ma = 100$;

Na Figura 6.10, existe uma outra situação, onde o agente $a_2$ tenta se afastar quando avista $a_1$, enquanto este último, tenta se aproximar. Este comportamento foi gerado através da alteração dinâmica de $\mathbf{v}$ para $a_1$ de forma que ele aponte na direção de $a_2$ conforme mostra a figura. O agente $a_2$, utiliza seu vetor comportamental apontando para a parede enquanto estiver com $a_1$ dentro de seu raio de visão ($d_{vis}$).

### 6.3.2 Corrida

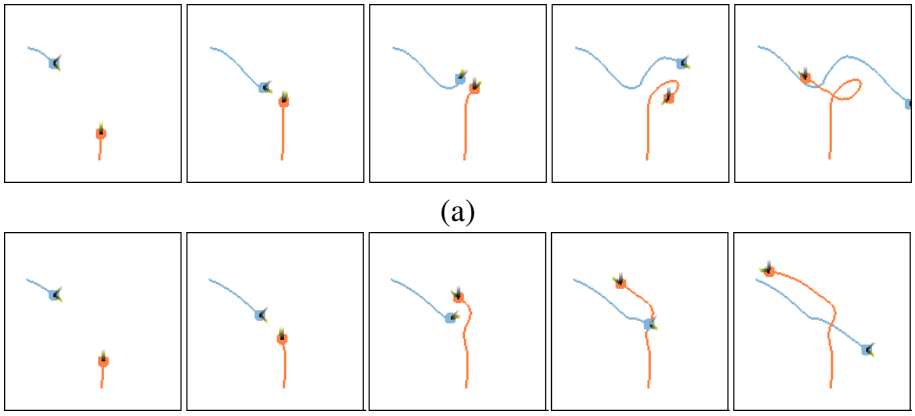
Neste experimento, todos os agentes possuem os mesmos parâmetros e devem atingir o mesmo objetivo ao mesmo tempo. A diferença nas trajetórias ocorre devido a interação entre eles. Considerando que todos procuram andar pelo lado direito, o agente que conseguiu andar mais rápido foi o que saiu mais próximo à parede e encontrou um caminho livre pela frente.

### 6.3.3 Comportamento exploratório

O planejador de movimentos desenvolvido pode ser usado para exploração de um ambiente desconhecido de forma similar ao trabalho desenvolvido por Prestes *et al.* (PRESTES et al., 2002). Para tanto, o mapa global é iniciado com todas as células livres tendo potencial igual a zero, e a cada passo do agente são executadas um número fixo de relaxações em uma área correspondente ao mapa local. Como o agente sempre procura regiões de menor potencial, ele segue naturalmente para regiões desconhecidas.

Também é possível adicionar um objetivo *desconhecido*, ou seja, o agente explora o ambiente até encontrar a célula com potencial fixo em zero.

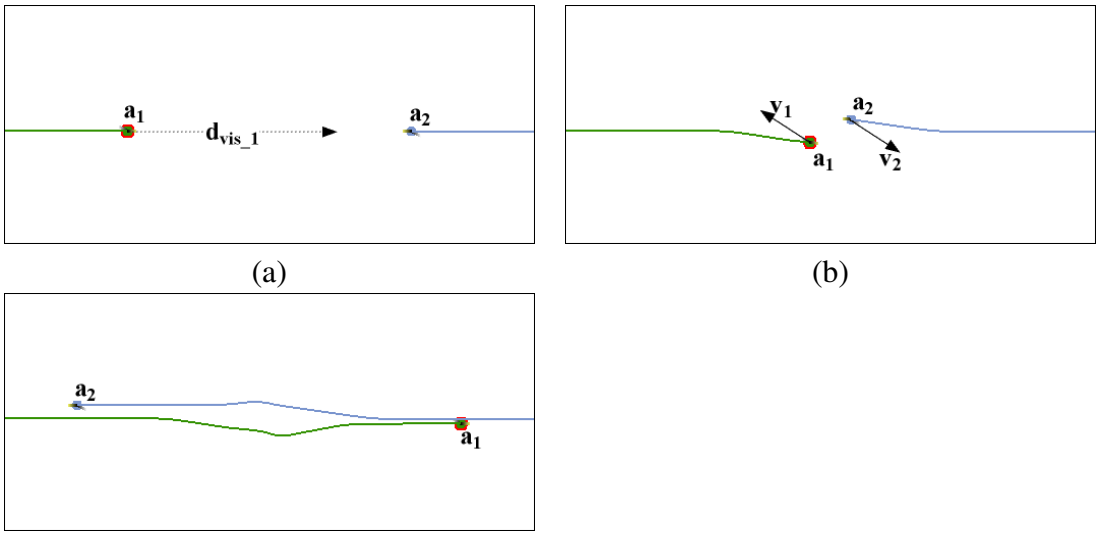
Na Figura 6.12, o agente $a_1$ vasculhou o ambiente até encontrar uma célula objetivo $o_1$



(a)

(b)

Figura 6.8: Variando a velocidade do agente através de $F_a$




(a) (b)

(c)

Figura 6.9: Agentes em rota de colisão: (a)agentes seguindo funções harmônicas; (b)vetor comportamental fixo; (c)agentes retornam a utilizar funções harmônicas.

ao lado de $a_2$. Este cenário pode ser usado para representar uma festa com várias pessoas em uma casa onde uma pessoa procura por outra que está em uma das salas. A pessoa que está sendo procurada possui uma célula marcada como livre ao seu lado e pode se mover livremente, desde que o objetivo seja alterado dinamicamente. Neste experimento, a cada passo do agente são executadas 30 iterações no ambiente global e 30 no ambiente local.

É possível perceber que as trajetórias executadas pelo agente não são tão naturais quanto as que são geradas sobre um campo potencial já definido. A pesquisa de técnicas específicas para este tipo de comportamento, utilizando este método na tentativa de simular pedestres, ainda não foi realizada.

### 6.3.4 Problema da emboscada

Este problema ocorre quando existe um *corredor* de agentes $\{a_i\}$, onde $i \in [2,n]$ bloqueando a passagem natural do agente $a_1$ e se assemelha a um processo de emboscada,

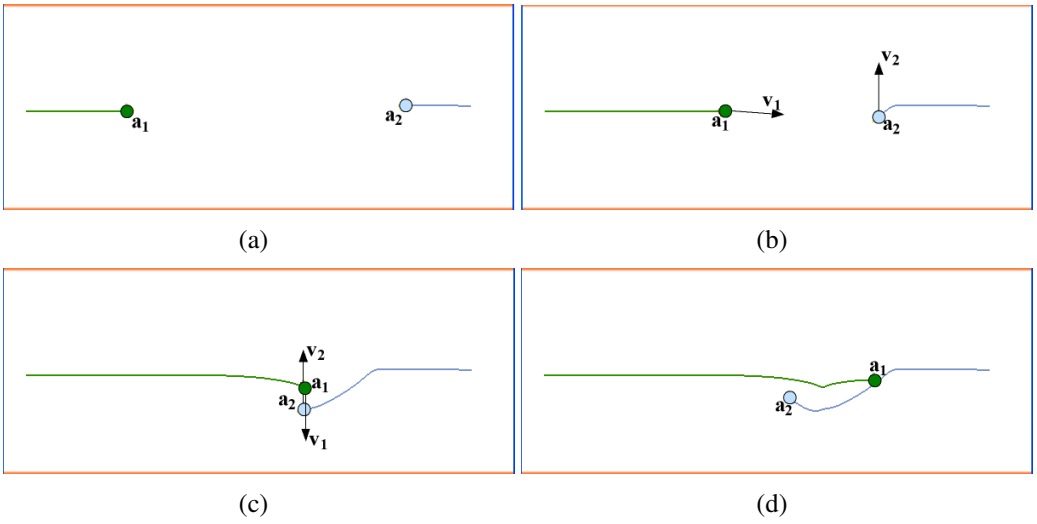



Figura 6.10: Aproximação: (a) os agentes $a_1$ e $a_2$ se avistam alterando $\mathbf{v}_1$ e $\mathbf{v}_2$. (b) $a1$ procura se aproximar de $a_2$, enquanto este tende a andar mais próximo a parede; (c) os agentes se cruzam e voltam aos seus comportamentos normais, seguindo em direção ao objetivo como mostra (d).

onde uma pessoa segue seu caminho de forma natural e quando percebe está cercada por inimigos (Figura 6.13). Configurações como a mostrada na figura costumam ocasionar mínimos locais em campos potenciais tradicionais, mesmo quando os obstáculos são mapeados em um mapa global.

No planejador de movimento criado, esta situação pode fazer com que um agente fique preso até que os obstáculos sejam removidos. Isto ocorre porque quando o agente entra no início do corredor, ele recebe a informação do mapa global de que este é um caminho válido e somente quando atinge a região final (quando o mapa local abrange esta região) ele percebe que está bloqueado. Em determinadas condições, um agente poderá escapar utilizando as técnicas já implementadas conforme descrito abaixo. É importante ressaltar que, se a janela local tiver tamanho suficiente para envolver toda a região onde existam agentes bloqueando a passagem, esta situação não ocorrerá (desde que exista um caminho alternativo).

Na pior das hipóteses, não é possível escapar de uma configuração deste tipo. Porém, quando se está simulando uma situação realística, isto não se constitui em um problema.

No planejador de movimento criado, esta situação foi utilizada para testar diversas configurações de parâmetros e variações no algoritmo de cálculo da trajetória. Algumas situações serão descritas abaixo.

#### *6.3.4.1 Fuga utilizando* $\mathbf{v}$

Uma forma aparententemente simples do agente $a_1$ conseguir escapar da situação descrita acima consiste em modificar seu vetor comportamental $\mathbf{v}$ assim que detectar uma situação de risco.

A Figura 6.14(a) mostra um exemplo onde $a_1$ altera o seu vetor comportamental para $\mathbf{v} = (0, -1)$ fazendo com que ele procure seguir próximo ao lado esquerdo assim que avistar o primeiro agente a uma distância $d_{vis} = 20$. Na Figura 6.14(b), são utilizados os mesmos parâmetros, porém com uma nova configuração para alguns dos agentes $\{a_i\}$. É possível observar que o agente $a_1$ procura ficar do lado esquerdo, porém detecta que a

única saída está à direita.

Em um outro exemplo, mostrado na Figura 6.15, o vetor **v** permite ao agente fugir da região em que ficaria preso se fossem utilizadas apenas funções harmônicas.

#### *6.3.4.2 Fuga aumentando o tamanho do mapa local*

Neste exemplo, o mapa local possui dimensões maiores do que as normalmente utilizadas, cobrindo uma área do mapa global conforme é mostrado na Figura 6.16. Assim o agente consegue detectar a região livre ao lado do grupo de agentes estáticos. É possível perceber que mesmo a janela não cobrindo toda a região, o caminho livre é encontrado.

### 6.3.5 Parque

Diferentemente de um corredor, onde a direção de movimento tende a ser uma só (com dois sentidos possíveis) este cenário, apresentado por Dapper *et al.* (DAPPER et al., 2006), possui grandes áreas livres onde os comportamentos dos agentes não sofrem influência direta de obstáculos próximos. Trata-se de um pequeno parque em uma cidade. Ele possui cinco acessos, um lago no centro e uma barraca de pipoca ao sul. Os agentes na simulação podem simplesmente cruzar o parque ou parar para comprar pipoca e continuar sua caminhada. Ele também se constitui em um cenário familiar para a maioria das pessoas, onde o comportamento de pedestres reais poderá ser comparado de forma subjetiva com os comportamentos gerados e em uma próxima etapa, através de dados reais coletados através de, por exemplo, uma *webcam* (em um ambiente similar).

Para as simulações apresentadas a seguir, foram construidos 6 mapas globais (um para cada objetivo: 5 para as saídas e outro para a barraca de pipoca). Para os mapas globais foram utilizadas matrizes de $80 \times 80$ células e $15 \times 15$ para os mapas locais.

Dado o cenário descrito acima, foram executados dois tipos de simulações. Na primeira, cinco agentes deveriam partir do acesso oeste, ir até a barraca de pipoca e depois sair pelo acesso norte. Utilizou-se $\varepsilon = 0.8$ e $s = 0.5$ para todos os agentes com os seguintes valores do vetor comportamental: $\mathbf{v}_1 = (1,0)$, $\mathbf{v}_2 = (0.707, 0.707)$, $\mathbf{v}_3 = (0,1)$, $\mathbf{v}_4 = (-1,0)$, $\mathbf{v}_5 = (-0.707, -0.707)$ respectivamente para os agentes $a_1$ até $a_5$.

A Figura 6.17 mostra o resultado desta simulação. Na Figura 6.17 (a), cada agente executou a tarefa individualmente, sendo o único objeto móvel no ambiente. Na Figura 6.17(b), os cinco agentes partiram ao mesmo tempo, precisando competir para atingir os objetivos. Isto fica claro nos caminhos formados próximos a barraca de pipoca. A trajetória do agente $a_2$ foi significativamente alterada, sendo que ele acabou utilizando um caminho pelo outro lado do lago devido a interferência dos outros agentes. As outras trajetórias, em geral, também sofreram alterações devido a necessidade de evitar a colisão entre eles.

A última simulação consistiu em incluir vários agentes no cenário sem a preocupação de definir um comportamento específico. Foram utilizados 100 agentes, todos com $\varepsilon$=0.8. O valor de **v** foi escolhido aleatoriamente, entre -1 e 1, assim como o tamanho do passo dos agentes, escolhido no intervalo 0.3 e 1. Cada agente parte de uma posição no ambiente e procura chegar a um dos seis objetivos definidos. O objetivo do experimento foi fazer uma demonstração da capacidade do algoritmo em utilizar vários agentes e analisar como eles negociam espaço e se locomovem num ambiente externo. A simulação trouxe resultados que parecem realísticos de forma subjetiva, tendo sida submetida a estudantes e professores de computação gráfica e apresentada em uma conferência da área[2]. Uma

[2]COMPUTER GRAPHICS INTERNATIONAL 2006

melhor análise dos resultados pode ser obtida com a comparação com dados reais, sendo sugerida como trabalho futuro. Um passo desta simulação é mostrado na Figura 6.18.

## 6.4 Desempenho do algoritmo

Em uma simulação de pedestres em um mundo virtual, para cada novo passo do agente, o planejador de movimento deve fornecer a nova posição e orientação do agente. Assim, para avaliar se o algoritmo pode ser utilizado em tempo real, é necessário saber a freqüência de passos que uma pessoa pode atingir. Segundo Mazarakis e Avaritsiotis (MAZARAKIS; AVARITSIOTIS, 2005), esta freqüência varia de 0,9 - 1 Hz para alguém andando muito devagar; até 3,5 Hz para alguém correndo muito rápido; sendo que a média para uma pessoa caminhando é de 2 Hz. Portanto, o processamento deve ser suficiente para a renderização do ambiente virtual, incluindo a animação dos agentes, e para calcular até 3,5 passos por segundo do algoritmo de planejamento de movimento.

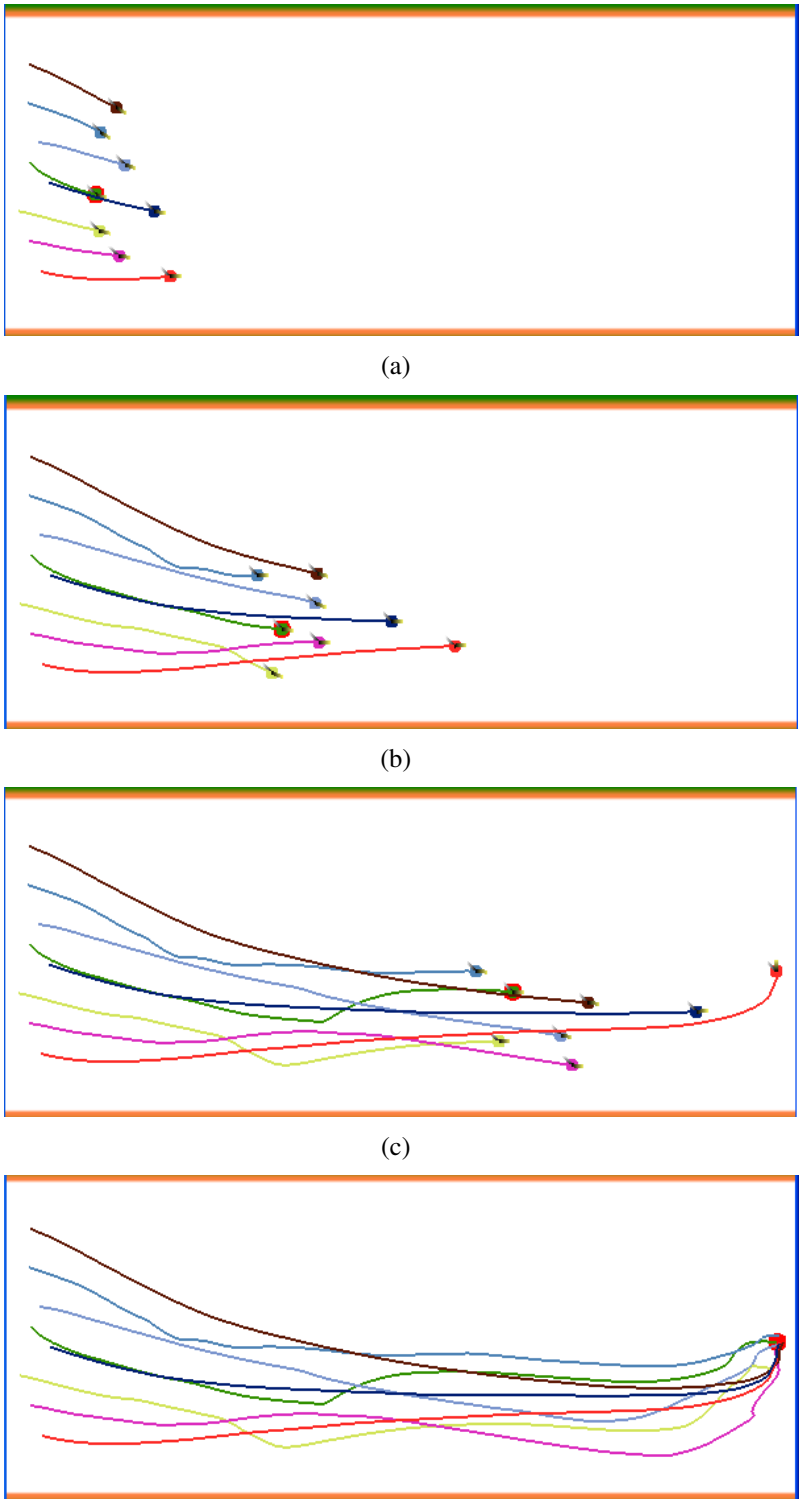
Com o objetivo de aumentar o número de agentes na simulação, em muitas aplicações, onde a maioria das pessoas caminha no ambiente, pode-se considerar a média de 2 passos por segundo. Para animações, quando existe foco em determinadas regiões ou personagens, pode-se optar por utilizar o algoritmo apenas entre os personagens em destaque e para os demais, utilizar outras técnicas como as utilizadas em simulações de multidões.

Para estimar o número de agentes que o algoritmo pode controlar, foi realizado um experimento medindo a freqüência de passos em função do número de agentes. O ambiente utilizado é o mesmo dos demais experimentos deste trabalho e possui 2 dimensões. A utilização deste ambiente se aproxima do custo real do algoritmo, sendo que se for utilizado um ambiente 3D, o custo deste último deverá ser avaliado.

Nos exemplos tratados neste trabalho, deve-se ressaltar que a complexidade do algoritmo depende somente do tamanho da grade, independentemente do número de obstáculos. Assim teremos uma complexidade quadrática. Se o algoritmo for utilizado em espaços de configurações com mais dimensões (ver Introdução), deve-se considerar além do processamento do algoritmo, o custo de gerar este espaço.

A Figura 6.19 mostra o resultado da simulação em uma máquina com processador ATHLON 64 3500+ 2.21GHz, 2.0 GB de RAM e placa gráfica nVidia GeForce 7800 GTX. Foram utilizadas 60 relaxações da matriz local a cada passo. Neste caso, se for considerada a média de 2 passos por segundo, é possível utilizar até 300 gentes. Para 3,5 passos por segundo, até 200. No entanto, se forem utilizadas somente 30 iterações a cada passo, conforme outros experimentos descritos neste trabalho, o número de agentes que o algoritmo consegue controlar é aproximadamente o dobro. Como visto, essa é a capacidade máxima, sem considerar o custo de animação de agentes em ambientes tridimensionais.

A integração com ambientes tridimensionais está sendo realizada por outro membro do grupo.



(a)

(b)

(c)

(d)

Figura 6.11: Agentes com mesmos parâmetros disputando uma corrida

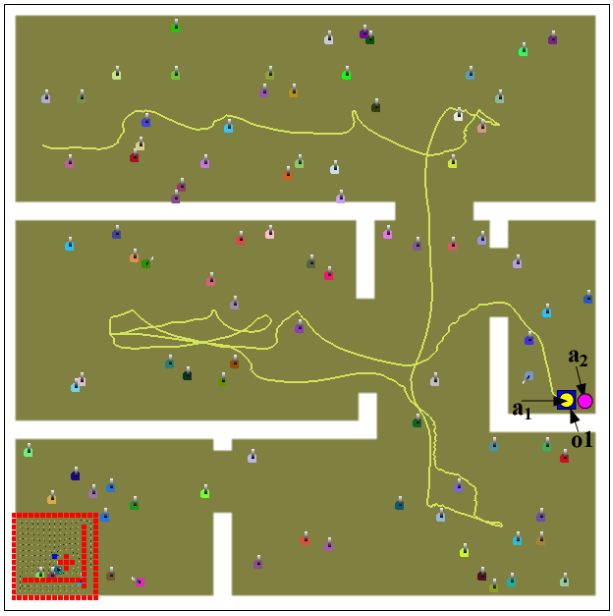



Figura 6.12: Comportamento exploratório. O agente $a_1$ explora o ambiente até encontrar o objetivo $o_1$.

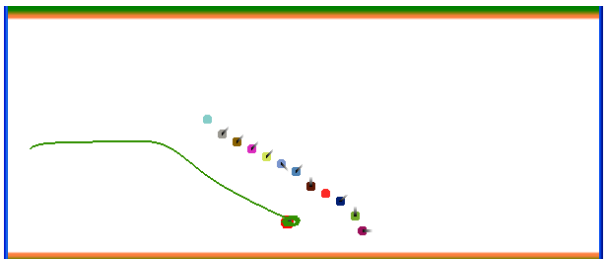
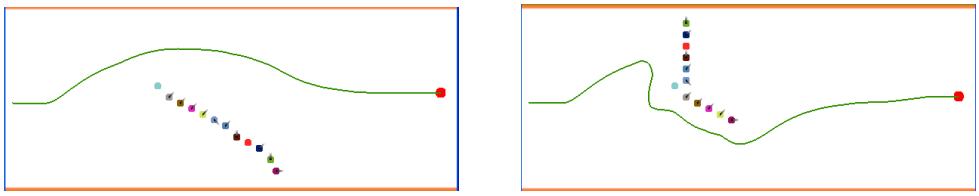


Figura 6.13: Problema da emboscada



(a) (b)

Figura 6.14: Evitando uma emboscada. O agente possui os mesmos parâmetros nas duas figuras.

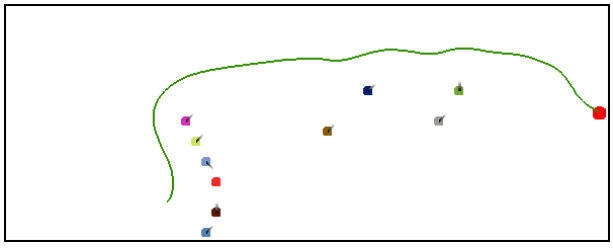
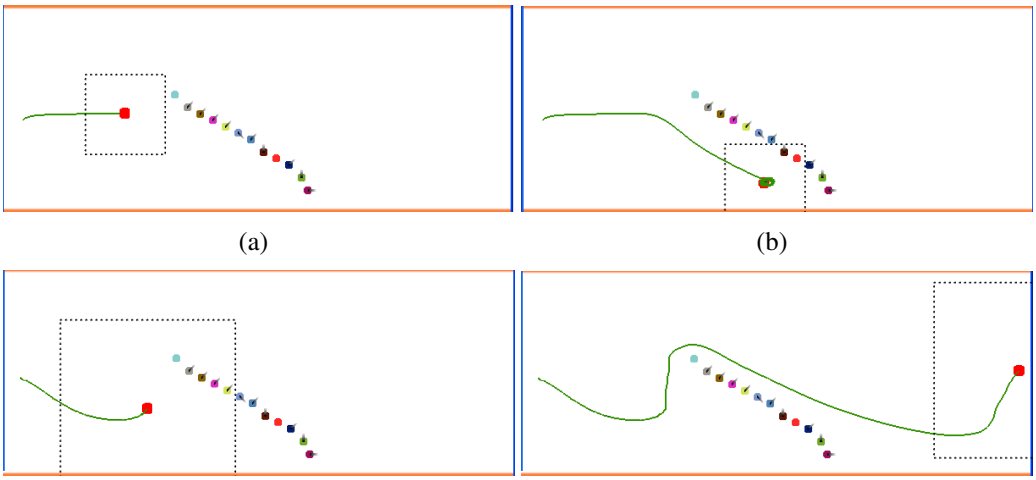


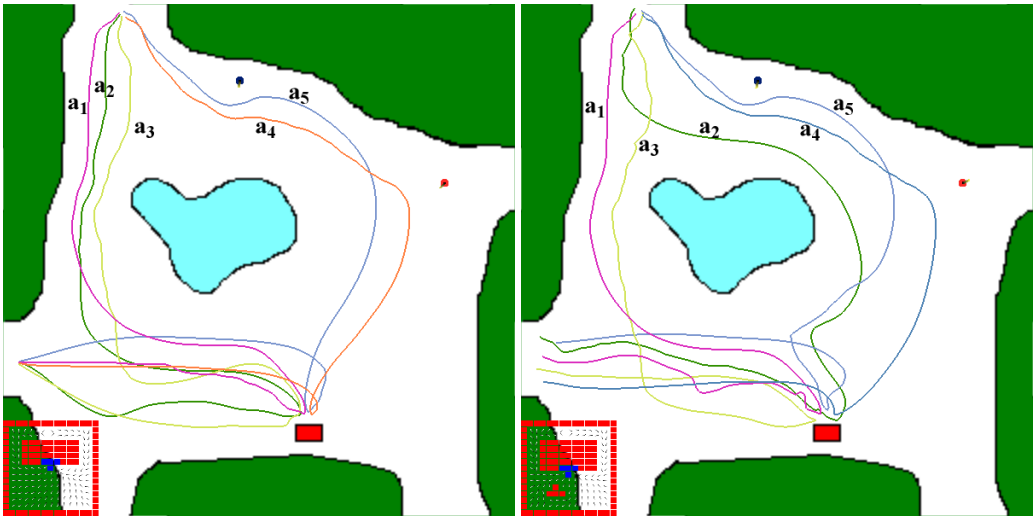
Figura 6.15: Fuga utilizando o vetor $\mathbf{v} = (0, -1)$



(a) (b)

(c) (d)

Figura 6.16: Fuga através do aumento do mapa local




(a) (b)

Figura 6.17: Agentes com comportamento diferentes executando a mesma tarefa

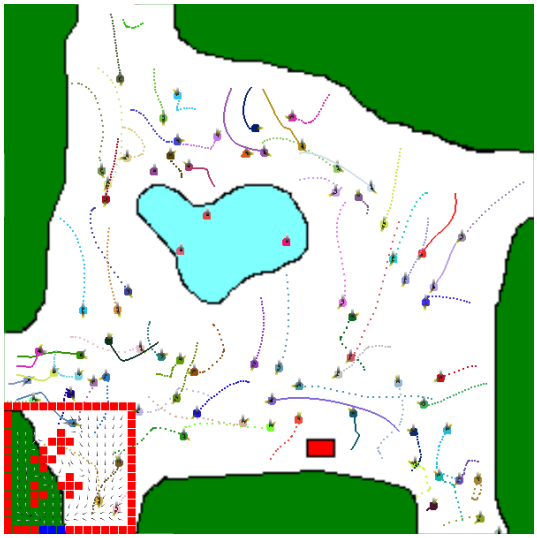


Figura 6.18: Diversos agentes com comportamento aleatório

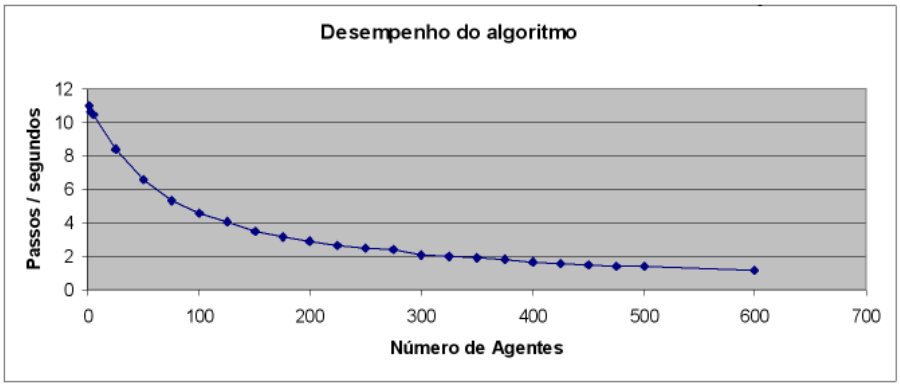



Figura 6.19: Número de passos por segundo

# 7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou um método de planejamento de movimento que permite a definição de comportamentos individuais para os vários agentes presentes na simulação, e que podem ser modificados dinamicamente, em tempo real. As trajetórias geradas são suaves e levam em consideração a presença de obstáculos dinâmicos, representados pelos *agentes em movimento*. Ao mesmo tempo em que um agente segue em direção ao objetivo, desviando dos obstáculos, ele procura seguir o padrão definido pelas suas características individuais.

Os comportamentos gerados podem tanto ser usados de forma isolada, como serem combinados em movimentos complexos. É possível utilizar funções que definem comportamentos, como por exemplo, um agente seguindo uma senóide, ou quantificar um desvio a esquerda ou a direita quando o agente avista um obstáculo a sua frente. A definição da personalidade do agente baseada em comportamentos de baixo nível e a validação dos caminhos gerados são sugeridas como trabalhos futuros, conforme será comentado mais adiante. Os capítulos 5 e 6 mostraram diversos resultados e experimentos em diferentes cenários.

Como visto, o planejamento de caminhos é baseado na utilização de campos potenciais gerados a partir de soluções numéricas para problemas de valores de contorno (PVC), que por sua vez, possuem como vantagem a inexistência de mínimos locais, a geração de trajetórias suaves e a relativamente fácil implementação.

Os comportamentos gerados se baseiam principalmente na alteração do campo potencial através de um parâmetro $\mathbf{v} = (x_k, y_k)$, que foi denominado aqui de vetor comportamental, e de um escalar $\varepsilon$, que determina a sua influência no campo. Também foi criada uma função $F_a$ que altera a velocidade do agente de acordo com condições como a variação do gradiente.

A desvantagem do método acima é o alto custo computacional. Dada a necessidade de se gerar um campo potencial para cada agente, a solução encontrada foi utilizar janelas pequenas que cobrem apenas parte do ambiente global. Esta idéia deu origem a uma arquitetura flexível, que permite inclusive a utilização de métodos tradicionais de planejamento de *caminhos* para gerar o mapa global. Como visto, o mapa global é utilizado pelo mapa local para indicar a posição de um *objetivo local*. Devido ao fato dos agentes serem mapeados somente na janela local, pode ocorrer que um agente não consiga passar por um caminho bloqueado por outros agentes estáticos (problema da emboscada). No entanto, em uma situação realística, é possível que este seja o comportamento esperado, ou de outra forma, quando uma pessoa está cercada por outras ela acaba sem ter como escapar.

Um outro problema é o *achatamento* do potencial em regiões distantes do objetivo devido à precisão numérica dos computadores. Neste caso, não é gerado um caminho até

o objetivo. Isto foi resolvido através da colocação de sub-objetivos nas regiões achatadas.

Em síntese, o trabalho apresentado pode servir de base para a construção de modelos mais completos onde serão consideradas questões como estudos psicológicos na tentiva de definir a personalidade dos agentes e a validação com dados reais. Alguns exemplos de trabalhos relacionados ao comportamento de pedestres são comentados abaixo:

- Definição de comportamentos (de baixo nível) baseados em estudos psicológicos. Consiste em definir os parâmetros adequados que implementem comportamentos executados por pedestres em diferentes situações, como desviar de outras pessoas considerando características individuais e as condições do ambiente. Como características individuais pode-se levar em consideração aspectos como idade, sexo, altura dos pedestres além de questões como estado de humor. Como condições do ambiente estão a densidade de pedestres e o tipo de ambiente (fechado ou aberto) ou do clima (chuva, frio). Este estudo inclui a definição dos parâmetros $\mathbf{v}$ e $\varepsilon$ e da função $F_a$ para cada situação;

- Definição de movimentos complexos parametrizados. A definição de movimentos complexos pode ser executada através de diversas pequenas ações de nível mais baixo. Dessa forma, pode-se pensar em definir um conjunto de parâmetros que seriam utilizados de forma sucessiva para gerar comportamentos. Isto é equivalente a alterar determinados parâmetros através de uma função adequada. Este trabalho apresentou exemplos onde os parâmetros foram alterados dinamicamente quando um obstáculo é visto a frente ou em função do número de passos;

- Utilização de raciocínio cognitivo para determinar a alteração dos parâmetros. Alguns movimentos podem ser disparados a partir do desejo do agente. Por exemplo, quando uma pessoa avista outra à sua frente, ela pode decidir se ela quer caminhar mais distante ou mais próxima dessa pessoa, ou ativar outro comportamento qualquer;

- Permitir que o usuário desenhe um comportamento (uma curva no espaço 2D) e este seja transformado de forma automática ou semi-automática para uma determinada função;

- Desenvolvimento de um algoritmo eficiente para GPUs. A definição de um algoritmo mais adequado para ser usado em placas gráficas pode diminuir o problema de desempenho, possibilitando a utilização de matrizes maiores para o mapa local ou o aumento do número de agentes na simulação. Esta idéia considera a tendência de evolução do hardware gráfico maior do que as dos processadores convencionais.

Por último, este trabalho permitiu a escrita de dois artigos: *Simulating Pedestrian Behavior with Potential Fields* e *Generating Steering Behaviors for Virtual Humanoids using BVP Control*. O primeiro foi apresentado na conferência Computer Graphics International 2006 (DAPPER et al., 2006) e publicado em seus *Proceedings*. O segundo foi apresentado na edição de 2007 da mesma conferência e igualmente publicado nos *Proceedings*.

Os dois artigos estão disponíveis em anexo. Algumas animações referentes a experimentos apresentados nos artigos estão disponíveis no endereço *www.inf.ufrgs.br/~fdapper*

# REFERÊNCIAS

BARRAQUAND, J.; LATOMBE, J.-C. Robot motion planning: a distributed representation approach. **Int. J. Rob. Res.**, Thousand Oaks, CA, USA, v.10, n.6, p.628–649, 1991.

BERG, M. de; KREVELD, M. van; OVERMARS, M.; SCHWARTZKOPF, O. **Computational Geometry**: algorithms and applications. [S.l.]: Springer-Verlag, 2000. ISBN 3-540-65620-0.

CHESTNUTT, J.; KUFFNER, J. A Tiered Planning Strategy for Biped Navigation. In: IEEE - RAS / RSJ CONFERENCE ON HUMANOID ROBOTS, 2004. **Proceedings...** [S.l.: s.n.], 2004.

CHOI, M. G.; LEE, J.; SHIN, S. Y. Planning biped locomotion using motion capture data and probabilistic roadmaps. **ACM Trans. Graph.**, New York, NY, USA, v.22, n.2, p.182–203, 2003.

CHOSET, H. et al. **Principles of Robot Motion**: theory, algorithms, and implementations. [S.l.]: MIT Press, 2005. ISBN 0-262-03327-5.

CONNOLLY, C.; GRUPEN, R. On the Applications of Harmonic Functions to Robotics. **International Journal of Robotic Systems**, [S.l.], v.10, p.931–946, 1993.

CONNOLLY, C. I.; BURNS, J. B.; WEISS, R. Path Planning Using Laplace's Equation. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1990. **Proceedings...** [S.l.: s.n.], 1990. p.2102–2106.

DAPPER, F.; PRESTES, E.; IDIART, M. A. P.; NEDEL, L. P. **Simulating Pedestrian Behavior with Potential Fields**. [S.l.]: Springer Verlag, 2006. 324-335p. (Lecture Notes in Computer Science, v.4035).

FOSKEY, M. et al. **A Voronoi-Based Hybrid Motion Planner for Rigid Bodies**. Chapel Hill, NC, USA: [s.n.], 2000.

FOSKEY, M. et al. A Voronoi-Based Hybrid Motion Planner. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, 2001. **Proceedings...** [S.l.: s.n.], 2001. p.55–60.

FRAICHARD, T.; AHUACTZIN, J.-M. Smooth path planning for cars. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS & AUTOMATION, 2001. **Proceedings...** [S.l.: s.n.], 2001. p.3722–3727.

HOFF, K. E. et al. Fast computation of generalized Voronoi diagrams using graphics hardware. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 26., 1999. **Proceedings...** ACM Press, 1999. p.277–286.

KAMPHUIS, A.; OVERMARS, M. H. Finding paths for coherent groups using clearance. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION, SCA, 2004. **Proceedings...** New York: ACM Press, 2004. p.19–28.

KAVRAKI, L. et al. Probabilistic roadmaps for path planning in high-dimensional configuration space. **IEEE Trans. on Robotics and Automation**, [S.l.], v.12, n.4, p.566–580, 1996.

KOGA, Y. et al. Planning motions with intentions. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 21., 1994. **Proceedings...** New York: ACM Press, 1994. p.395–408.

KUFFNER, J. J. Goal-Directed Navigation for Animated Characters Using Real-Time Path Planning and Control. In: INTERNATIONAL WORKSHOP ON MODELLING AND MOTION CAPTURE TECHNIQUES FOR VIRTUAL ENVIRONMENTS, 1998. **Proceedings...** Springer-Verlag, 1998. p.171–186.

KUFFNER, J. J.; LATOMBE, J.-C. Interactive Manipulation Planning for Animated Characters. In: PACIFIC CONFERENCE ON COMPUTER GRAPHICS AND APPLICATIONS, 8., 2000, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2000. p.417.

LATOMBE, J.-C. **Robot Motion Planning**. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

LATOMBE, J.-C. Motion Planning: a journey of robots, molecules, digital actors, and other artifacts. **The International Journal of Robotics Research**, [S.l.], v.18, n.11, p.1119–1128, 1999.

LAVALLE, S. **Rapidly-exploring random trees**: a new tool for path planning. [S.l.]: Computer Science Dept., Iowa State University, 1998.

LENGYEL, J. et al. Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware. **Computer Graphics**, [S.l.], v.24, n.4, p.327–335, 1990.

LINGELBACH, F. Path planning using probabilistic cell decomposition. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS & AUTOMATION, 2004. **Proceedings...** [S.l.: s.n.], 2004.

MAZARAKIS, G. P.; AVARITSIOTIS, J. N. A prototype sensor node for footstep detection test. In: EUROPEAN WORKSHOP ON WIRELESS SENSOR NETWORKS, 2., 2005. **Proceedings...** [S.l.: s.n.], 2005. p.415–418.

METOYER, R. A.; HODGINS, J. K. Reactive pedestrian path following from examples. **The Visual Computer**, [S.l.], v.20, n.10, p.635–649, 2004.

OLIVEIRA FORTUNA, A. de. **Técnicas Computacionais para Dinâmica dos Fluidos**: conceitos básicos e aplicações. [S.l.]: EDUSP, 2000. ISBN 85-3140-5262.

PETTRE, J.; LAUMOND, J.-P.; SIMEON, T. A 2-stages locomotion planner for digital actors. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION, SCA, 2003. **Proceedings...** Eurographics Association, 2003. p.258–264.

PETTRE, J.; SIMEON, T.; LAUMOND, J. Planning Human Walk in Virtual Environments. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEM, 2002. **Proceedings...** [S.l.: s.n.], 2002. v.3, p.3048 – 3053.

PRESTES, E. **Navegação Exploratória baseada em Problemas de Valores de Contorno**. 2003. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

PRESTES, E.; ENGEL, P. M.; TREVISAN, M.; IDIART, M. A. Exploration Method using Harmonic Functions. **Robotics and Autonomous Systems**, [S.l.], v.40, n.1, p.25–42, 2002.

PRESTES, E.; IDIART, M. A.; TREVISAN, M.; ENGEL, P. M. Autonomous Learning Architecture for Environmental Mapping. **Journal of Intelligent and Robotic Systems**, [S.l.], v.39, p.243–263, 2004.

RYMILL, S. J.; DODGSON, N. A. Psychologically-based simulation of human behaviour. In: THEORY AND PRACTICE OF COMPUTER GRAPHICS, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.35–42.

SALOMON, B. et al. Interactive navigation in complex environments using path planning. In: SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, SI3D, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.41–50.

SCHEIDEGGER, C. E.; COMBA, J. L. D.; CUNHA, R. D. da. Navier-Stokes on Programmable Graphics Hardware Using SMAC. In: BRAZILIAN SYMPOSIUM ON COMPUTER GRAPHICS AND IMAGE PROCESSING, SIBGRAPI, 27., 2004. **Proceedings...** [S.l.: s.n.], 2004. p.300–307.

SHAO, W.; TERZOPOULOS, D. Autonomous Pedestrians. In: ACM SIGGRAPH/EUROGRAPH SYMPOSIUM ON COMPUTER ANIMATION, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.19–28.

SIMEON, T.; LAUMOND, J.; LAMIRAUX, F. Move3D: a generic platform for motion planning. In: IEEE INTERNATIONAL SYMPOSIUM ON ROBOTICS ASSEMBLY AND TASK PLANNING, 2001. **Proceedings...** [S.l.: s.n.], 2001. p.25–30.

STILMAN, M.; KUFFNER, J. Navigation Among Movable Obstacles: real-time reasoning in complex environments. **International Journal of Humanoid Robotics**, [S.l.], v.2, n.4, p.479–504, Dec. 2005.

TANG, X. et al. Using motion planning to study RNA folding kinetics. In: RESEARCH IN COMPUTATIONAL MOLECULAR BIOLOGY, RECOMB, 8., 2004. **Proceedings...** ACM Press, 2004. p.252–261.

TOMBROPOULOS, R.; ADLER, J.; LATOMBE, J. CARABEAMER: a treatment planner for a robotic radiosurgical system with general kinematics. **Medical Image Analysis**, [S.l.], v.10, p.931–946, 1998.

TORRES, J. A. R.; NEDEL, L. P.; BORDINI, R. H. Using the BDI Architecture to produce autonomous characters in virtual worlds. In: INTERNATIONAL WORKSHOP ON INTELLIGENT VIRTUAL AGENTS, 4., 2003, Kloster Irsee, Germany. **Proceedings...** Heidelberg: Springer Verlag, 2003. p.197–201.

TREVISAN, M.; IDIART, M. A.; PRESTES, E.; ENGEL, P. M. Exploratory Navigation based on Dynamic Boundary Value Problems. **Journal of Intelligent and Robotic Systems**, [S.l.], 2006.

YAMANE, K.; KUFFNER, J. J.; HODGINS, J. K. Synthesizing animations of human manipulation tasks. In: ACM SIGGRAPH PAPERS, 2004. **Proceedings...** ACM Press, 2004. p.532–539.

# ANEXO: ARTIGOS PUBLICADOS

Conforme comentado no capítulo de conclusões, este trabalho permitiu a geração de dois artigos apresentados neste anexo: *Simulating Pedestrian Behavior with Potential Fields* e *Generating Steering Behaviors for Virtual Humanoids using BVP Control*. O primeiro foi apresentado na conferência Computer Graphics International 2006 (DAPPER et al., 2006) e publicado em seus *Proceedings*. O segundo foi apresentado na edição de 2007 da mesma conferência e igualmente publicado nos *Proceedings*.

# Simulating Pedestrian Behavior with Potential Fields

Fábio Dapper[1], Edson Prestes[1], Marco A.P. Idiart[2], and Luciana P. Nedel[1]

[1] Instituto de Informática, Universidade Federal do Rio Grande do Sul,
[2] Instituto de Física, Universidade Federal do Rio Grande do Sul,
Porto Alegre – RS – Brazil
{fdapper,prestes,nedel}@inf.ufrgs.br, idiart@if.ufrgs.br

**Abstract.** The main challenges of realistically simulating the displacement of humanoid pedestrians are twofold: they need to behave realistically and they should accomplish their tasks. Here we present a field potential formalism, based upon boundary value problems, that allows a group of synthetic actors to move negotiating space, avoiding collisions, attaining goals in prescribed sequences while at same time producing very individual paths. The individuality of each pedestrian can be set by changing its inner field parameters. This leads to a broad range of possible behaviors without jeopardizing its task performance. Simulate situations as behavior in corridors, collision avoidance and competition for a goal are presented and discussed.

## 1 Introduction

The use of synthetic actors able of acting as autonomous agents in applications involving virtual environments is becoming more and more common [1]. Suitable skills for those actors (often simulating human beings) include: a realistic appearance, the ability to produce natural movements, and the aptitude to reasoning and act in an unforeseeable way.

To simulate the behavior of human beings, it is usual to consider system architectures implemented in layers. The lowest one deals with the rotation of each body joint. The intermediary level is responsible for encapsulating composed movements that bring together a set of single joint motions. These movements can represent simple tasks (e.g. stand-up, sit-down, take something, give a step) that used together can provide a higher abstraction level, so called behaviors (e.g. open the door, walk from one position to another one, etc). Finally, the higher abstraction motion layer (cognitive) involves a reasoning mechanism that makes decisions and commands actions in view of the context information (e.g. position, orientation, and distance to target) and humanoids intentions, beliefs and desires.

In a previous work [2] we presented a well succeeded proposal for the implementation of the cognitive level using the BDI (beliefs, desires and intentions) architecture to simulate autonomous agents reasoning. However, good solutions for lower level behaviors may also be investigated. Such solutions should preview

not just a handy manner to specify complex tasks based on simple ones, but also to consider the addition of expressiveness on those tasks.

The simulation of virtual humans moving into a synthetic world involves mainly the environment specification, the definition of the agent initial position as well as its target position in the world (also called *goal*). By setting those parameters, a path-planning algorithm can be used to find a trajectory to be followed. However, in a real world, if we consider several persons (all in the same initial position) looking for achieving the same target position, each individual path followed will be different. Even if we have the same task, the strategy used for each one to reach his/her goal will depend on his/her physical constitution, personality, mood and reasoning.

In this paper we propose a path-planning approach based on boundary value problems to find paths between an initial and a target position in a dynamic environment. The paths found by our algorithm are smooth and variable, depending on the individual characteristics of each agent, which can be dynamically changed.

The paper is organized as follows. In Section 2 we presented some related work on path-planning techniques for virtual humans. Section 3 describes our path planner based on harmonic functions and Section 4 presents our main contribution, involving the extensions for the basic method. In Section 5 we deeply explain the way we implemented the method and in Section 6 we present our results. Finally, in Section 7 we present the conclusions and point out some future work.

## 2 Related Work

Motion planning methods have been largely studied by the robotics community. As in this paper our focus is on its use for simulating human beings behaviors while walking, we limit this Section scope for the works involving humanoids animation.

Lengyel *et al.* [3] have published one of the first articles on the subject of motion planning as a computer graphics problem. Their work presented a solution for the classical *Piano Movers* problem based on the use of standard graphics hardware to rasterize obstacles and generates the configuration space. The motion path produced by the planner is minimal with respect to the Manhattan distance metric and includes rotations and translations.

In order to generate more realistic results and allows its use in real-time applications, several authors proposed motion planning solutions based on two steps. In general, the first step is dedicated to define a valid path, while the second adapts this path in order to generate a more realistic movement. Kuffner [4] proposed a technique with the first step dedicated to the path-planning and the second to the path-following. The 3D scenario is projected in 2D and the humanoid treated as a disc, reducing the planning problem to a 2D problem.

Metoyer and Hodgings [5] proposed a similar technique also based on two steps. In their method, the characters have a pre-defined path to follows and

this path is smoothed and slightly changed to avoid collisions by using force fields.

The development of randomized path-finding algorithms, specially the PRM (Probabilistic Roadmaps) [6] and RTT (Rapidly-exploring Random Tree) [7], allow the use of large and most complexes configuration spaces, generating paths most efficiently. In this way, the challenge becomes more the generation of realistic movements than finding a valid path. Choi *et al.* [8] proposed the use of a captured movements library associated to the PRM to generate realistic movements in a static environment. Despite the fact the path maps should be generated in a pre-processing phase, the results are very realistic.

Thanks to the researches in robotics, the path-planning problem is almost solved. However, in the computer graphics domain, to find a natural and realistic way to move a character is as important as to find a path between two points. The most part of the works developed since now propose methods based on two separate phases. In the next sections we present our own proposal for generating realistic paths based on a single phase.

## 3 Harmonic Functions Path Planner

Whether it is a human being, a robot or a synthetic actor the action of moving from an initial position to a goal position in space consists of at least two stages: a planning stage when a path is devised; and an implementation stage when the path is followed by the moving agent. The first stage deals with a combination of concepts like efficiency, risk avoidance, computability, etc. To the second stage belongs the series of routines or corrections that the agent has to perform to adapt its motion when the predefined path cannot be followed due to unpredictable changes in the agent's surrounds, or in case of robotics, due to machine limitations.

In a seminal work in the field of robotics, Khatib [9] proposed a method that fuses these two stages in a very elegant way. He considered that instead of looking for a good path and trying to control the agent's movement around it a good planner should provide a potential field, or a force field (its gradient), that expanded the whole region of manoeuvre, producing a continuum of alternative paths. The potential field is devised to incorporate obstacles and goals, and should guide agent at all times indicating the best direction to follow. Its most straightforward implementation is a simple superposition of fictitious forces: obstacles forces that repel the agent to prevent collisions; and target forces that attract the agent. Such superposition is not always successful since for some environment configuration the agent can end up trapped in local minima before reaching the target.

Up to this date, the best way to produce a potential field that is free from local minima is through the numerical solution of a convenient partial differential equation with boundary conditions - a boundary value problem (BVP). The boundary conditions are central to the method indicating which regions in the environment are obstacles and which are targets.

The first proposal in this direction was made by Connolly and Grupen [10] and it is called the method of the harmonic functions. In their method the potential fields are the solutions of the Laplace's equation - whose solutions are called harmonic functions

$$\nabla^2 \, p(\mathbf{r}) \; = \; \sum_i \; \frac{\partial^2 p(\mathbf{r})}{\partial x_i^2} \; = \; 0 \tag{1}$$

where $\mathbf{r}$ is the environment coordinates. The Laplace's equation does not present local minima, and that is why it was chosen. They also proposed boundary conditions such that the potential should be one in the contours of the obstacles and zero in the region of the target. Setting up the value of the function in the boundaries is called a Dirichlet boundary condition in the language of a BVP.

The agent uses the gradient descent of this potential to determine the path that connects its current position to the target. As there is only a minimum defined in target position, it exists exactly one path from any point to the potential to the target. This method is formally complete, i.e., if there is a path that connects the agent position to the target it will be found. The resulting path is smooth and safe and it minimizes the collision probability with the obstacles.

## 4 Beyond Path Planner based on Harmonic Functions

Laplace's equation is not the only partial differential equation that generates functions without local minima. In [11], Trevisan *et al.* came up with a framework for exploratory navigation based on a family of potential field functions that does not possess local minima. The authors suggest the following equation

$$\nabla^2 \, p(\mathbf{r}) + \epsilon \mathbf{v}.\nabla p(\mathbf{r}) \; = \; 0 \tag{2}$$

for handling sparse environments, where $\mathbf{v}$ is a bias vector and $\epsilon$ is a scalar. The addition of the term $\epsilon \mathbf{v} \nabla p$ breaks the symmetry of vector field generated by Laplace's equation increasing the system performance in sparse environments during the exploration process.

The central contribution of this paper is to use the Equation 2 for generating different behaviors (illustrated in this work through the path followed by each agent) for several agents in a known environment. As discussed before, if the agent is controlled by a vector field produced by harmonic functions, it will always tend to follow a path that minimize the collision probability with the obstacles, i.e., in an indoor environment the agent will tend to follow a path equidistant to the walls, as shown in Figure 1(a). This behavior is not always adequate to simulate humanoid motion since it looks very stereotyped.

The adjustment of the vector $\mathbf{v}$ can produce a path close to the walls, as shown in Figures 1(b) and (c). The vector $\mathbf{v}$, also called *behavior vector*, can be seen as an external force field that counteract the natural tendency of agent moving away from the obstacles. The parameter $\epsilon$ can be understood as the

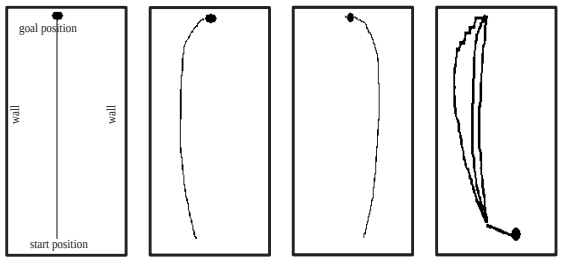



**Fig. 1.** Different paths followed by agents using Equation 2: (a) path produced by harmonic potential, i.e., with $\epsilon = 0$; (b) with $\epsilon = 0.8$ and $\mathbf{v} = (1, 0)$; (c) with $\epsilon = 0.8$ and $\mathbf{v} = (-1, 0)$; and using the same vector $\mathbf{v} = (1, 1)$ and different values to the parameter $\epsilon$ (0.4, 0.8, 1.2, 1.6, 2.0, 2.4 and 7.2).

*strength* or *influence* in following the direction defined by vector $\mathbf{v}$ instead of the direction produced by harmonic functions.

Figure 1(d) shows the results obtained in several experiments that use different $\epsilon$ and the same vector $\mathbf{v} = (1, 1)$. This flexibility allows to develop different and interesting behaviors to generate realistic humanoid motion during the navigation process. In our case, we simulated several agents with different $v$ and $\epsilon$ and put them into a known environment to perform a couple of navigation tasks.

## 5 Implementation

In this section, we present the global environment representation, the structure of the agents that act on the environment, as well as the mechanisms used to control each agent behavior.

### 5.1 Environment Global Map

The environment is represented by a *set* of homogenous meshes $\{m_k\}$, where each mesh $m_k$ is associated to a target $o_k$ and has $L_x \times L_y$ cells, denoted by $\{c^k_{i,j}\}$. Each cell $c^k_{i,j}$ corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $p^k_{i,j}$. These maps are used by the harmonic path planner (see Section 3) to assist the agent to reach a specific target.

Each mesh $m_k$ has the potential values of its cells relaxed independently using the Equation 1. After the convergence, the map $m_k$ stores a potential field that is used to reach the target $o_k$. This procedure is performed before the simulation starts and we consider that the environment is surrounded by obstacles in order to delimit the agent navigation space.

### 5.2 Agent Local Map

Each agent $a_k$ has one map $am_k$ that stores the current local information about the environment obtained by its sensors. This map is centered in the current agent position and represents a small fraction of the global map, nearly 10% of the total area covered by the global map.

The map $am_k$ has $l_x^k \times l_y^k$ cells, denoted by $\{ac_{i,j}^k\}$ and can be divided in three regions: the update zone (*u-zone*); the free zone (*f-zone*) and the border zone (*b-zone*), as shown in Figure 2(a). In a similar way, each cell corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $ap_{i,j}^k$.

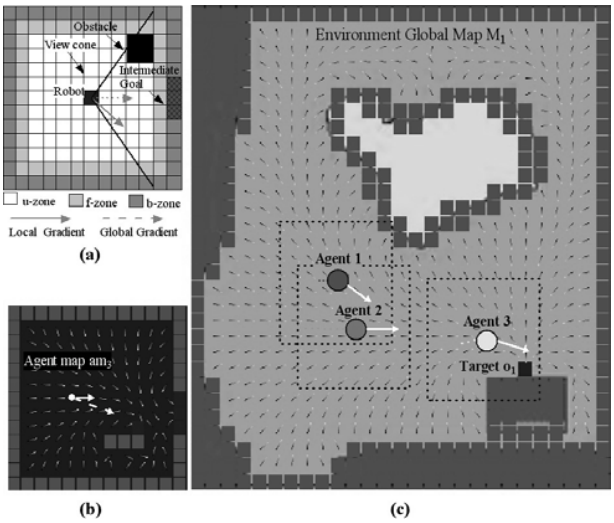



**Fig. 2.** (a)Agent Local Map. The white, light gray and dark gray cells comprise the update, free and border zones, respectively. (b,c)Agents acting in the real environment. Each agent senses the environment, updates its local map (b) and navigates towards the target $o_1$ (c).

The area associated to each agent map cell is smaller than the area associated to the global map cell. The main reason is that the agent map is used to produce a refined motion, hence, the smaller cell size the better the quality of motion; while the global map is used only to assist the long-term agent navigation.

### 5.3 Updating Local Maps using Global Maps

Each agent $a_k$ has a well determined goal $o_{goal(k)}$ (the function *goal* maps the agent number $k$ into its current target number. In this description, we will consider that each agent must reach only one target. The extension to multiple

targets is straightforward and will be commented in Section 5.5), a particular vector $\mathbf{v}_k$, that controls its behavior, and a $\epsilon_k$ that determines the influence of $\mathbf{v}_k$. The same goal, $\mathbf{v}$ and $\epsilon$ can be designated to several agents.

When the agent $a_k$ navigates the environment, it uses its sensors to perceive the environment and to update its local map with the information about the obstacles and other agents. The agent sensors set a view cone with aperture $\alpha$.

Figure 2(b) sketches a particular instance of the agent local map. The *u-zone* cells $\{ac^k_{i,j}\}$, inside the view cone, with obstacles or agents have their potential value set to 1. Obstacles are not considered in the *u-zone* out of the view cone. This procedure assures that dynamic or static obstacles behind the agent do not interfere in its future motion.

The agent $a_k$ calculates the global descent gradient on the cell in the global map $m_{goal(k)}$ containing its current position. The gradient direction is used to generate an intermediate goal in the border of the local map setting the potential values to 0 of a couple of *b-zone* cells. While the other *b-zone* cells are considered as obstacles having their potential values set to 1. In Figure 2(c), each agent calculates the global gradient in order to project an intermediate goal in its local map. As the agent local map is delimited by obstacles, the agent is pulled towards the intermediate goal using the direction of its local gradient. The intermediate goal helps the agent $a_k$ to reach its target $o_{goal(k)}$ while allowing it to produce its particular motion.

In some cases, the target $o_{goal(k)}$ is inside both the view cone and the *u-zone*, and consequently, the local map cells associated are set to 0. The intermediate goal is always projected even if the target is mapped onto the *u-zone* otherwise the robot can easily get trapped because the robot would be taking into consideration only the local information about the environment, in a same way as traditional potential fields [9].

The *f-zone* cells are always considered free of obstacles, even when there are obstacles there. The absence of this zone may close the connexion between the current agent cell and the intermediate goal due to the mapping of obstacles in front of intermediate goal. When this occurs, the robot gets lost because there is no information coming from the intermediate goal to produce a path to reach it. The *f-zone* cells handle the situation permitting always that the information about the goal is propagated to the cells associated to the agent position.

After the sensing and mapping steps, the agent updates the potential value of its map cell using a discrete version of Equation 2,

$$ap^k_{i,j} = \frac{1}{4}(ap^k_{i-1,j} + ap^k_{i+1,j} + ap^k_{i,j-1} + ap^k_{i,j+1}) + \frac{\epsilon^k}{8}((ap^k_{i+1,j} - ap^k_{i-1,j})v^k_x + (ap^k_{i,j+1} - ap^k_{i,j-1})v^k_y) \quad (3)$$

where $\mathbf{v}^k = (v^k_x, v^k_y)$ is the vector that controls the behavior of agent $a_k$ and $\epsilon^k \in [-2, +2]$ and represents the *influence* of vector $\mathbf{v}^k$. The local potential is partially relaxed [12] and the agent calculates the gradient descent of its position

in the local map $am_k$ by

$$\mathbf{dgrad}^k = \Big((ac^k_{p_x+1,p_y} - ac^k_{p_x-1,p_y})/2, (ac^k_{p_x,p_y+1} - ac^k_{p_x,p_y-1})/2\Big)$$

where $p_x = \lceil l^k_x/2 \rceil$ and $p_y = \lceil l^k_y/2 \rceil$, and it follows the direction $\theta^k$ calculated by $\theta^k = \arctan(dgrad^k_x, dgrad^k_y)$ where $\arctan(.,.)$ is the inverse tangent taken in the interval $[-\pi, +\pi]$.

### 5.4 Characterizing the Agent Behavior

In the real world, even if several people have the same goal, the strategy used for each one to reach it will depend on different factors as: physical constitution, personality, mood, and reasoning. In Figure 1 we shown we can simulate different behaviors by setting both the behavior vector $v$ and $\epsilon$ differently for each agent. In this first example we kept the variables constant during the animation, however we can produce more interesting behaviors dynamically changing vector $\mathbf{v}$ and $\epsilon$. For instance, the vector $\mathbf{v}$ can be controlled by a function defined by the user, as in Figure 3. Even with this new complex behavior, which simulates a *drunk* agent, the resulting potential guarantees that the robot reaches safely the target.

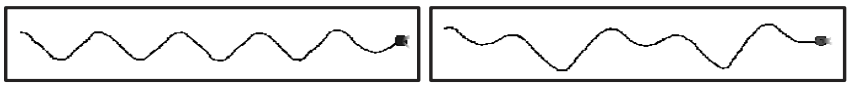



**Fig. 3.** Paths followed by agents using different equations that control the behavior vector $\mathbf{v}$: (a) $\mathbf{v} = (1, \sin(\omega * t))$; (b) $\mathbf{v} = (1, \sin(\omega * t) + \sin(\omega/2 * t))$, with $\omega = \pi/18$ and $t$ the current simulation step.

We can change $\mathbf{v}$ in a regular periodic fashion, as shown above, but it does not need always to be the case. We can consider an agent that randomly changes its behavior vector. Each new value of $\mathbf{v}$ is kept constant during an also random time interval.

### 5.5 Algorithm

In this section we present the algorithm that implements the concepts shown before and produce the humanoids simulation.

1. computes all the environment global maps (one for each possible goal $o_k$)
2. for each agent $a_k$, defines the behavior vector $\mathbf{v}_k$ and $\epsilon_k$. Each variable can be either static or dynamic. If a variable is chosen to be dynamic then the function that controls it must be specified.

3. for each agent $a_k$ do (asynchronously)
   (a) reads its sensors in order to detect static and dynamic obstacles
   (b) updates its map with local information about the obstacles and other agents
   (c) computes the global gradient descent and generates the intermediate goal
   (d) updates the potential field
   (e) computes the local gradient descent and follows the gradient direction
   (f) while not reaching the target $o_{goal(k)}$ repeat the steps from $(a)$ to $(f)$, otherwise stops moving

The first two steps are performed in a pre-process phase. In relation to the step 3, each agent executes independent and asynchronously the actions from $(a)$ to $(f)$. This algorithm considers each agent must reach only one target. However, the agent can be in charge of reaching several targets orderly. In this case, the step $(f)$ must be changed to

(f) while not reaching the target $o_{goal^i(k)}$ repeat the steps from $(a)$ to $(f)$, otherwise if $goal^i(k) = goal^{last}(k)$ then stops moving. Else repeats the process with the next target $o_{goal^{i+1}(k)}$

## 6 Results

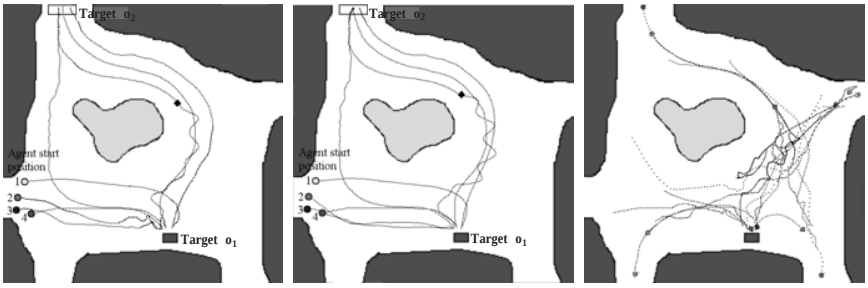
In order to illustrate the potentialities of our path-planning approach we made some experiments considering a realistic situation. Taking into account the scenario described bellow, we have induced some agents' behaviors to verify some considerations made before, as: how to accomplish the same task in different ways; or how different agents avoid collisions, for example. In another set of tests we have ran the algorithm considering a variable number of agents with random objectives, behaviors and velocities. Our goal with these experiments was to verify the motion diversity. Finally, we made some considerations about performance.

We consider a small park in a town (see Figure 4). It has five accesses, a lake in the middle and a popcorn-cart in the south. Characters in the simulation can simply cross the park or stop to buy a popcorn bag and continue their walking. It is a quite familiar real scenario; the large open area makes easy and clear the simulation of different agents behaviors that will not be constrained by an excessive amount of obstacles; by simulating a group of agents walking in the park it is easy to verify the collision avoidance with dynamic obstacles (here represented as other agents).

The set up for this scenario involves the statement of six possible goals, one for each park access and another in front of the popcorn-cart. We will need to compute 6 environment global maps. In our tests, we used a matrix with 60x60 cells to represent global maps and a matrix with 11x11 cells for the agent local maps.

The first situation induced by us consists in simulating the behavior of 4 agents while accomplishing the same task. The agents are initially disposed somewhere in the park access west and their task consists on go to the popcorn-cart and after, to quit the park by the access north. Figure 4 illustrates the results of animation. Each agent accomplishes its task individually without the intervention of the others. The small square specifies the moment where the agent 3 changes its behavior vector $\mathbf{v}$.

Figure 4(b) shows the same task of Figure 4(a), but in this case all the agents are moving at the same time, therefore they compete for the targets. The paths drawn in these two figures are slightly different and these differences are duo the collision detection and avoidance between the agents.




(a) (b) (c)

**Fig. 4.** Four agents individually accomplishing the same task (a) and accomplishing the same task concurrently (b). Agent 1: $\mathbf{v} = (-0.707, -0.707)$, $\epsilon = 0.8$ and $step = 0.6$ cells per frame; agent 2: $\mathbf{v} = (0.707, 0.707)$, $\epsilon = 0.8$ and $step = 0.5$ cells per frame; agent 3: initially $\mathbf{v} = (\sin(\omega * t), 1)$, changing to $\mathbf{v} = (0.707, 0.707)$ after some time, $\epsilon = 0.8$ and $step = 0.35$ cells per frame; agent 4: $\epsilon = 0$, and $step = 0.46$ cells per frame.(c) Simulation of a set of 12 agents walking around the park with random behaviors.

Figure 5 shows two frames of the animation of two agents. One agent walks from the north to the south while the other one walks from the south to the north. Using our algorithm we automatically avoid the collision between the two agents, since each agent is considered as a dynamic obstacle by the other. However, the final path definition can be more or less natural, depending on the parameters definition. In the sequence presented on Figure 5(a), we set $\epsilon$ as 0. In this way, the behavior vector $\mathbf{v}$ is not considered. For the animation shown in Figure 5(b), both agents begins the animation with $\epsilon = 0.0$. When the proximity is detected, the behavior vector $\mathbf{v}$ of each agent is oriented orthogonally to the collision direction, forcing the movement to its right direction. At the same time, the $\epsilon$ becomes equal to 0.6.

Finally, we generated some animation sequences without searching to reproduce any specific behavior. In those sequences we used 12 agents, $\epsilon = 0.8$ for all agents and the components of $\mathbf{v}$ randomly varying between -1 and 1. The

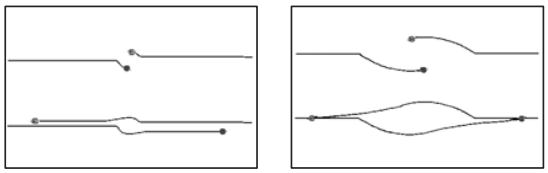



**Fig. 5.** Two collision avoidance animation sequences produced with different values for the behavior vector and $\epsilon$.

agents step size are also randomly defined between 0.3 and 1.0 cells per frame. The initial and final positions for the agents are arbitrarily chosen. The agents can begin its movement from any valid position in the environment and its goal is one of the 6 possible target positions described before. Figure 4(c) shows a frame of one of the animation generated by us.

## 7 Conclusions and Future Work

This article presents a new approach for generating pedestrian behavior using path planning based on numerical solution of boundary value problems. We demonstrate that the correct adjustment of behavior vector and the parameter $\epsilon$, that determines the vector influence, can produce interesting behaviors, as illustrated in Figures 1 and 3. These behaviors can be interchanged to produce complex motions, as shown in Figure 4, oriented to the agent personality. In this work, we do not implement the agent personality. This step is actually in progress and will be shown in our future submissions.

The guiding potential of Equation 2 is free of local minima what constitutes a great advantage when compared to the traditional potential fields. Furthermore, the method proposed is formally complete and generates smooth and safe paths that can be directly used in mobile robots. The local information gathered by agent sensors allows treating the dynamic obstacles, as other agents navigating in the environment.

We handle the usual costs associated to BVP calculations by using small local maps, instead of a large map that cover all the environment, for each agent. This permits to have several agents acting in the environments while keeping an acceptable running time. Even with only local information about the environment, the intermediate goals computed from the environment maps add global information about the agent target in order to treat conveniently local minima and to allow the agent to reach its target.

Another drawback is that the potential gets flat far from the target position due to numerical precision. In these regions, the gradient is very small to provide useful information to guide the robot. In this case, the robot can easily get lost.

We have successfully overcome this problem by using intermediate goals in the flat region.

In the future, we intend: to test different path planners to minimize the computational cost associated to the environment global map; to develop an architecture to be implemented into the GPU to reduce the potential time computation; and to develop an efficient data structure to compact the environment information, such as quadtree, and an efficient algorithm to access this information in real-time.

## Acknowledgments

We would like to thank FAPERGS and CNPq for financial support and Renato Oliveira for helping with the figures.

## References

1. Shao, W., Terzopoulos, D.: Autonomous pedestrians. In: ACM SIGGRAPH/Eurograph symposium on Computer Animation. (2005) 19–28
2. Torres, J.A., Nedel, L.P., Bordini, R.H.: Using the bdi architecture to produce autonomous characters in virtual worlds. In: Intelligent Virtual Agents. Volume 2792 of Lecture Notes in Artificial Intelligence., Springer Verlag (2003) 197–201
3. Lengyel, J., Reichert, M., Donald, B.R., Greenberg, D.P.: Real-time robot motion planning using rasterizing computer graphics hardware. Computer Graphics **24**(4) (1990) 327–335
4. James J. Kuffner, J.: Goal-directed navigation for animated characters using real-time path planning and control. In: International Workshop on Modelling and Motion Capture Techniques for Virtual Environments, London, UK, Springer-Verlag (1998) 171–186
5. Metoyer, R.A., Hodgins, J.K.: Reactive pedestrian path following from examples. The Visual Computer **20**(10) (2004) 635–649
6. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration space. IEEE Transactions on Robotics and Automation **12**(4) (1996) 566–580
7. LaValle, S.: Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University (1998)
8. Choi, M.G., Lee, J., Shin, S.Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Trans. Graph. **22**(2) (2003) 182–203
9. Khatib, O.: Commande dynamique dans l'espace opérational des robots manipulaters en présence d'obstacles. PhD thesis, École Nationale Supérieure de l'Aéronatique et de l'Espace, France (1980)
10. Connolly, C., Grupen, R.: On the applications of harmonic functions to robotics. International Journal of Robotic Systems **10** (1993) 931–946
11. Trevisan, M., Idiart, M.A., Prestes, E., Engel, P.M.: Exploratory navigation based on dynamic boundary value problems. accepted for publication in Journal of Intelligent and Robotic Systems (2006)
12. Prestes, E., Engel, P.M., Trevisan, M., Idiart, M.A.: Exploration method using harmonic functions. Robotics and Autonomous Systems **40**(1) (2002) 25–42

**The Visual Computer manuscript No.**
(will be inserted by the editor)

**Fábio Dapper · Edson Prestes · Luciana P. Nedel**

# Generating Steering Behaviors for Virtual Humanoids using BVP Control

**Abstract** One of the main challenges on animating embodied autonomous characters in real-time applications is the ability to generate believable behaviors, more precisely, actors capable of moving in a natural and improvisational manner. In this paper we propose an elegant and low cost solution based upon boundary value problems (BVP) to control steering behaviors of characters. We use a field potential formalism that allows synthetic actors to move negotiating space, avoiding collisions, and attaining goals, while producing very individual paths. The individuality of each character can be set by changing its inner field parameters leading to a broad range of possible behaviors. To illustrate the technique potentialities, some results exploring situations as steering behavior in corridors with collision avoidance and competition for a goal, and searching for objects in unknown environments are presented and discussed.

**Keywords** Humanoid Simulation · Path Planning · Steering behavior · Harmonic Functions · Boundary Value Problems

## 1 Introduction

In interactive applications such as games and virtual reality experiences, autonomous agents (also called *non-player characters*) are characters with the ability of playing a role into the environment with life-like and improvisational behavior [14]. Suitable skills for these characters (often simulating human beings) include: a realistic appearance, the ability to produce natural movements, and the aptitude to reason and act in an unforeseeable way. However, the high performance required for the algorithms used on real-time graphics applications frequently compel developers to look for better methods to generate more natural and unexpected simple movements. In this way, it is possible to improve the applications behavior quality avoiding the high cost frequently imposed by the use of AI methods.

The simulation of virtual humans moving into a synthetic world involves the environment specification, the definition of the agent initial position and its goal (target position). By setting these parameters, a path-planning algorithm can be used to find a trajectory to be followed. However, in the real world, if we consider several persons (all in the same initial position) looking for achieving the same target position, each path followed will be unique. Even for the same task, the strategy used for each person to reach his/her goal will depend on his/her physical constitution, personality, mood and reasoning. In this work we propose an algorithm to generate interesting behaviors for humanoids, considering that, from a single path, several behaviors can be explored to drive the agent from one position to another.

Despite *humanoid*, *autonomous agent*, and *behavior* are terms used in many different contexts, in this paper we will limit its use to match our goals. For sake of simplicity, we consider *humanoids* as a kind of embodied *autonomous agent* with reactive behaviors (driven by stimulus), represented by a computational model, and capable of producing physical manifestations in a virtual world. The term *behavior* will be used mainly as a synonymous of *animation* or *motion behavior* and intend to refer the improvisational and personalized action of a *humanoid*.

In a previous work [4] we proposed a method based on the numeric solution of the boundary value problem (BVP) to control pedestrians. We showed that a single principle can be used to generate interesting and complex human-like behaviors while humanoids move to achieve a navigational task. In this paper we propose some improvements for our initial algorithm, varying the motion speed and proposing new possibilities to follow a path. Some very first experiments towards to endow the humanoids the ability to explore unknown environments are also presented.

The remaining of this paper is structured as follows. Section 2 reviews some related work on path-planning techniques applied to virtual humans simulation. Section 3 describes the fundamentals of the path planning method proposed by us, as well as how we solve the BVP. In Section 4

F. Dapper, E. Prestes, L.P. Nedel
Instituto de Informática – Universidade Federal do Rio Grande do Sul
Av. Bento Gon calves, 9500 – Campus do Vale - Bloco IV
Porto Alegre – RS – Brasil 91501-970
E-mail: {fdapper,prestes,nedel}@inf.ufrgs.br

we detail the strategy used to handle the information about the environment and other agents and in Section 5 how the movement and velocity of the agent are managed. Finally, Section 6 presents our results and Section 7 conclusions and future works.

## 2 Related Work

Thanks to the researches in robotics, the path-planning problem is almost solved. However, in the computer graphics domain, to find a natural and realistic way to move a character is as important as to find a path between two points. In order to generate realistic results and allow its use in real-time applications, several authors proposed motion planning solutions based on two steps. In general, the first step is dedicated to define a valid path, while the second adapts this path to generate a more realistic movement.

Kuffner [5] proposed a technique with the first step dedicated to path-planning and the second to path-following. The 3D scenario is projected in 2D and the humanoid treated as a disc, reducing the dimension of the planning problem. Metoyer and Hodgings [10] proposed a similar technique also based on two steps. In their method, the characters have a pre-defined path to follows and this path is smoothed and slightly changed to avoid collisions based on force fields.

The development of randomized path-finding algorithms – specially the PRM (Probabilistic Roadmaps) [6] and RTT (Rapidly-exploring Random Tree) [8] – allow the use of large and most complexes configuration spaces, and generating paths most efficiently. Thus, the challenge becomes more the generation of realistic movements than finding a valid path.

Choi *et al.* [2] proposed the use of a library of captured movements associated to PRM to generate realistic movements in a static environment. Despite the fact the path maps should be generated in a pre-processing phase, the results are very realistic. Pettré *et al.* [11] used a PRM to identify a free of collisions path and Bézier curves to generate smooth paths associating it with a motion library. As in the previous works, the motion is also performed on a 2D environment.

Differently, Burgess and Darken [1] proposed a method to obtain very realistic paths through a terrain using properties of fluid simulation to produce human-like movements. The authors consider that a realistic path for a human is the one requiring the smallest amount of effort.

The most part of the works developed since now propose methods based on two separate phases. In next sections we present our own proposal for generating realistic paths based on a single phase. Our assumption is that realistic paths derive from human personal characteristics and internal state, thus varying from one person to another.

## 3 BVP-Path Planner

Recently, we proposed a framework for controlling virtual humanoids in navigational tasks. It is based on potential fields that do not have local minima [4,15] generated through the numeric solution of the BVP using Dirichlet boundary conditions and the following equation

$$\nabla^2 p(\mathbf{r}) + \varepsilon \mathbf{v}.\nabla p(\mathbf{r}) = 0 \tag{1}$$

where $\mathbf{v}$ is a bias vector and $\varepsilon$ is a scalar value.

The allowed values of the parameters $\varepsilon$ and $\mathbf{v}$ generate an expressive amount of action sequences that virtual humanoids (agents) can take to reach a specific target (goal position). Each action corresponds to a particular displacement that the agent performs at each step. Two sequences are not statically defined for a same pair $\varepsilon$ and $\mathbf{v}$. They vary according to the information gathered by the agent to allow it to react dynamically against unexpected events (e.g. dynamic obstacles). Satisfactory adjustments of parameters $\varepsilon$ and $\mathbf{v}$ generate realistic steering behaviors for agents [4].

The core of the Equation 1 is the vector $\mathbf{v}$, so called *behavior vector*, that acts as an external force pulling the agent to its direction always as possible. The parameter $\varepsilon$ can be understood as the *strength* or *influence* of this vector in the agent behavior. When $\varepsilon = 0$, Equation 1 can be reduced to

$$\nabla^2 p(\mathbf{r}) = 0$$

which is the Laplace's equation and the path planner is called harmonic functions path planner. It has been developed by Connolly and Grupen [3] and one of its features is to lead the agent to a path that minimizes the collision probability.

Figure 1 shows some paths produced using the equation of Laplace and the Equation 1. In Figure 1a, Laplace's equation conducts the agent through a path equidistant to the walls, which is not always adequate to simulate humanoid motion since it looks very stereotyped. In Figure 1b-c, we can observe that adding the term $\varepsilon\mathbf{v}.\nabla p(\mathbf{r})$ allows the generation of different kind of paths leading the agent, for instance, closer to the wall.

Our method starts with the discretization of the environment into a fixed homogeneous mesh with identical cells, like an occupancy grid. Each cell $(i, j)$ is associated to a squared region of the real environment and stores a potential value $p_{i,j}$. Dirichlet boundary conditions are such that, the cells with high probability of having an obstacle are set to 1 (*high potential*) while cells containing the target are set to 0 (*low potential*). The high potential value prevents the agent from running into obstacles whereas the low potential value generates an attraction basin that pulls the agent.

Solving the BVP thus consist in interpolating the potential values on the grid between the obstacles and the target. This can be done using the Gauss-Seidel algorithm which updates the potential cells according to the equation

$$\underbrace{\underbrace{p_{i-1,j} + p_{i+1,j} + p_{i,j-1} + p_{i,j+1} - 4p_{i,j}}_{\nabla^2 p(\mathbf{r})} + \underbrace{\varepsilon\left(\frac{(p_{i+1,j} - p_{i-1,j})}{2} v_x + \frac{(p_{i,j+1} - p_{i,j-1})}{2} v_y\right)}_{\varepsilon\mathbf{v}.\nabla p(r)} = 0}_{} \tag{2}$$

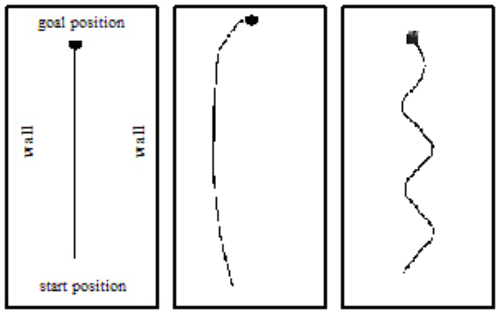



**Fig. 1** Different paths followed by agents using Equation 1: (a) path produced by Laplace's equation, i.e., with $\varepsilon = 0$; (b) with $\varepsilon = 0.8$ and $\mathbf{v} = (1,0)$; (c) with $\varepsilon = 0.8$ and $\mathbf{v} = (1, \sin(\omega * t))$.

that leads us directly to the update rule

$$p_{i,j} = \frac{1}{4}(p_{i-1,j} + p_{i+1,j} + p_{i,j-1} + p_{i,j+1}) + \frac{\varepsilon}{8}((p_{i+1,j} - p_{i-1,j})v_x + (p_{i,j+1} - p_{i,j-1})v_y) \quad (3)$$

where $\mathbf{v} = (v_x, v_y)$ and $\varepsilon \in [-2, +2]$.

$\varepsilon$ must be in the interval $[-2, +2]$, otherwise, the boundary conditions that assert the agent – repelling obstacles and attracting the target – are violated. Then the method generates oscillatory and unstable behaviors that do not guarantee the agent will reach the target.

The agent uses the gradient descent of this potential to determine the path to follows towards the target position. This method is formally complete, i.e., if there is a path connecting the agent position to the target, it will be found.

## 4 Environment Management

As explained in last section, our path planning method requires the environment discretization into a regular grid. In this section we present a strategy to implement it by using global environment maps (one for each target) and local maps (one for each agent) to enhance the algorithm performance, allowing the use of our method for real-time applications.

### 4.1 Environment Global Map

The entire environment is represented by a set of homogeneous meshes $\{m_k\}$, where each mesh $m_k$ is associated to an achievable target $o_k$ and has $L_x \times L_y$ cells, denoted by $\{c^k_{i,j}\}$. Each cell $c^k_{i,j}$ corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $p^k_{i,j}$. Each mesh $m_k$ stores a potential field computed by the harmonic path planner [3] that is used by agents to reach the target $o_k$.

In order to delimit the navigation space of agents, we consider the environment is surrounded by static obstacles. Global maps are built before the simulation starts.

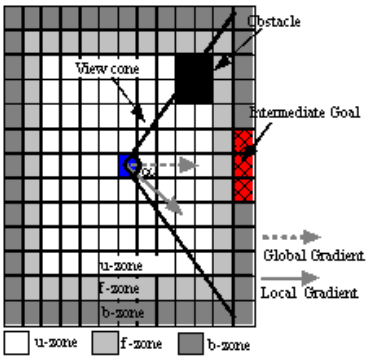



**Fig. 2** Agent Local Map. White, light gray and dark gray cells comprise the *update*, *free* and *border zones*, respectively. Red, black and blue cells correspond to the intermediate goal, obstacles and the agent position, respectively.

### 4.2 Agent Local Map

Each agent $a_k$ has one map $am_k$ that stores the current local information about the environment obtained by its sensors. This map is centered in the current position of the agent and represents a small fraction of the global map. The area associated to each agent map cell is smaller than the area associated to the global map cell. The main reason is that the agent map is used to produce refined motion, hence, the smaller cell size the better the quality of motion; while the global map is used only to assist the long-term agent navigation.

The map $am_k$ has $l^k_x \times l^k_y$ cells, denoted by $\{ac^k_{i,j}\}$ and is divided in three regions: the update zone (*u-zone*); the free zone (*f-zone*) and the border zone (*b-zone*), as shown in Figure 2. In a similar way, each cell corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $ap^k_{i,j}$.

### 4.3 Updating Local Maps from Global Maps

For each agent $a_k$, a goal $o_{goal(k)}$, – where the function *goal()* maps the agent number $k$ into its current target number – a particular vector $\mathbf{v}_k$, that controls its behavior, and a $\varepsilon_k$ that determines the influence of $\mathbf{v}_k$, should be stated. The same goal, $\mathbf{v}$ and $\varepsilon$ can be designated to several agents. Vector $\mathbf{v}_k$ and $\varepsilon_k$ can be either static or dynamic. If a variable is dynamic, then the function that controls it must be specified.

To navigate into the environment, an agent $a_k$ uses its sensors to perceive the world and to update its local map with the information about obstacles and other agents. The agent sensor set a view cone with aperture $\alpha$.

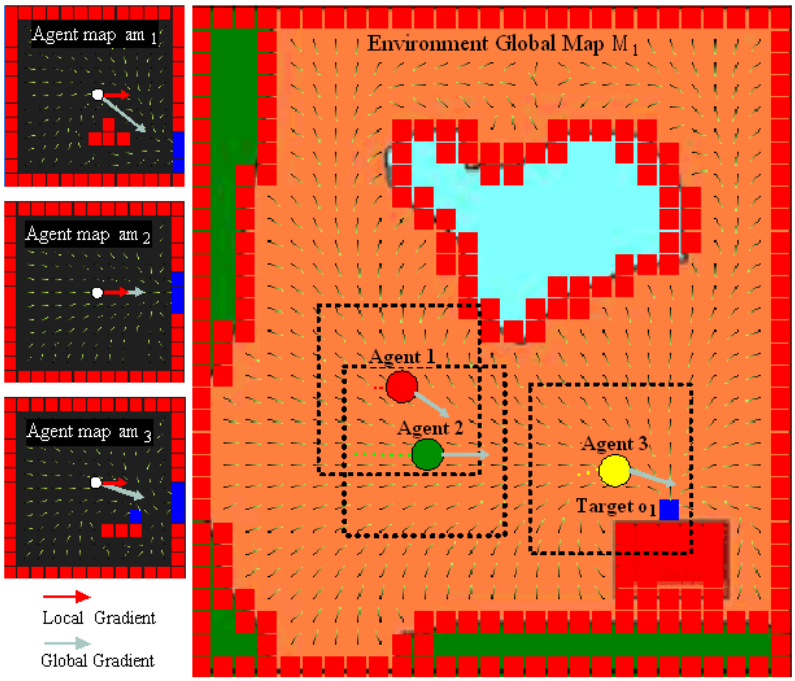



**Fig. 3** Agents acting in an environment. Each agent senses the environment, updates its local map and navigates towards the target $o_1$. Obstacles are represented as (*red squares*) in both global environment map and agent local map; the target $o_1$ is represented as a blue square in the global environment map and the intermediate goal generated by each agent is represented by a (*blue square*) in its local map.

Figure 2 sketches a particular instance of the agent local map. The *u-zone* cells $ac^k_{i,j}$ that are inside the view cone and correspond to obstacles or other agents have their potential value set to 1. In Figure 3, as the agent 1 is inside the *u-zone* of agent 2 local map but out of its view cone, it is not mapped as an obstacle into the local map of agent 2. This procedure assures that dynamic or static obstacles behind the agent do not interfere in its future motion.

For each agent $a_k$, the global descent gradient on the cell in the global map $m_{goal(k)}$ that contains its current position is calculated. The gradient direction is used to generate an intermediate goal in the border of the local map, setting the potential values to 0 of a couple of *b-zone* cells, while other *b-zone* cells are considered as obstacles, with their potential values set to 1. In Figure 3, each agent calculates its global gradient in order to project an intermediate goal in its own local map. As the agent local map is delimited by obstacles, the agent is pulled towards the intermediate goal using the direction of its local gradient. The intermediate goal helps the agent $a_k$ to reach its target $o_{goal(k)}$ while allowing it to produce a particular motion.

In some cases, the target $o_{goal(k)}$ is inside both the view cone and the *u-zone*, and consequently, local map cells associated are set to 0. The intermediate goal is always projected, even if the target is mapped onto the *u-zone*. Otherwise, the agent can easily get trapped because it would be taking into consideration only the local information about the environment, in a same way as traditional potential fields [7].

*F-zone* cells are always considered free of obstacles, even when there are obstacles inside. The absence of this zone may close the connection between the current agent cell and the intermediate goal due to the mapping of obstacles in front of the intermediate goal. When this occurs, the agent gets lost because there is no information coming from the intermediate goal to produce a path to reach it. *F-zone* cells handle the situation, always allowing the propagation of the information about the goal to the cells associated to the agent position.

After the sensing and mapping steps, the agent updates the potential value of all the cells of its map using Equation 3 with its pair $\mathbf{v}^k$ and $\varepsilon^k$. The local potential is partially relaxed [12] and the agent uses the gradient descent of its position defined by

$$\mathbf{dgrad}^k = \left( \frac{ap^k_{p_x+1,p_y} - ap^k_{p_x-1,p_y}}{2}, \frac{ap^k_{p_x,p_y+1} - ap^k_{p_x,p_y-1}}{2} \right)$$

to determine its displacement. In the local map $am_k$, $p_x = \lceil l^k_x/2 \rceil$ and $p_y = \lceil l^k_y/2 \rceil$.

## 5 Updating the Position and Speed of Agents

In our previous work [4], the agent position at time $t$ is computed using the following equation

$$\mathbf{pos}^t = \mathbf{pos}^{t-1} + step \frac{\mathbf{dgrad}}{||\mathbf{dgrad}||} \quad (4)$$

where *step* is a constant that corresponds to the maximum agent displacement[1]. However, during the experiments, we observed that, for several scenarios, this equation failed in producing realistic steering behaviors, as observed in real world. One of the reasons is that the agent changes its direction based solely on the gradient descent of its position. For instance, if the agent local map is small, its reaction time will be very short to treat dynamic obstacles. Then, these obstacles will produce a strong repel force that will change the agent direction abruptly. As we can see in Figure 4, if the agent uses only the gradient descent it will change its direction in nearly 90°.

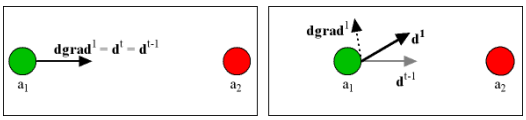



**Fig. 4** Agent displacement scheme

We handle this problem by changing Equation 4 into,

$$\mathbf{pos}^t = \mathbf{pos}^{t-1} + step\frac{\mathbf{d}^t}{||\mathbf{d}^t||} \tag{5}$$

with

$$\mathbf{d}^t = \eta\ \mathbf{d}^{t-1} + (1-\eta)\ \mathbf{dgrad}^t$$

where $\eta \in [0,1]$.

If $\eta = 0$, this equation reduces to Equation 4. If $\eta = 0.5$, the previous agent direction ($\mathbf{d^{t-1}}$) and the gradient descent ($\mathbf{dgrad}^t$) influence equally the computation of the new agent direction. Figure 4 shows the vector $d^t$ computed with $\eta = 0.5$. The parameter $\eta$ can be viewed as an inertial factor that tends to keep constant the agent direction insofar $\eta \to 1$. When $\eta \to 1$, the agent reacts slowly to unexpected events, increasing its hitting probability with obstacles.

Despite Equation 5 produces good results and smoother paths in environments with few obstacles, when the environment is cluttered with obstacles, the behavior of the agents are not realistic. To solve this problem, we incorporate the control of the speed in our model, allowing the simulation of agents mood through its magnitude. For instance, a tired agent will probably tend to move slowly whereas an agent that is anxious about its work will tend to move faster. Furthermore, the adjustment of the speed helps to prevent collisions and adds more realism to the simulation [2], e.g., when two pedestrians are in the eminence of collision, they will naturally change its speeds.

[1] This section presents the equations used by all agents. Therefore, to make the exposion clearer we supress the superscript of the terms that individualize each agent.

[2] Our formalism guarantees that collisions will not happen, however, because the sensor range or/and speed, the agent can perceive another one only when they are about to collide. To avoid abrupt changes in its direction or unnatural movements (see Section 6.3), it can alter its speed according to the collision risk

This consideration is incorporated in Equation 5, producing the equation

$$\mathbf{pos}^t = \mathbf{pos}^{t-1} + \upsilon_{max} f(\mathbf{dgrad}^k, \mathbf{d}^{t-1})\frac{\mathbf{d}^t}{||\mathbf{d}^t||} \tag{6}$$

where $\upsilon_{max}$ defines the maximum agent speed and function $f$ generates an output based on the cosine of the angle between vectors $\mathbf{d}^{t-1}$ and $\mathbf{dgrad}^t$, that stops the agent movement or reduces its speed when moving towards an obstacle. Function $f$ is defined as follows.

$$f(\mathbf{x},\mathbf{y}) = \begin{cases} 0 & \text{if } \cos(\mathbf{x},\mathbf{y}) < 0 \\ \cos(\mathbf{x},\mathbf{y}) & \text{otherwise} \end{cases}.$$

If the angle is higher than 180°, then there exists a high hitting probability and this function returns the value 0, doing the agent to stop. Otherwise, the agent speed will change proportionally to the collision risk defined by $f$. In regions cluttered with obstacles, agents will tend to move slowly. If a given agent is about to cross the path of another one, one of them will stop and wait until the other get through.

## 6 Results

In this section, the results obtained through the improvements proposed in our path planner are presented. In addition, we present a preliminary result of the extension of our framework for exploratory tasks using multiples humanoids.

### 6.1 Analyzing the Agent Displacement

Figure 5 shows some results using Equation 5 without considering variations in the agent speed. The figure shows different paths followed by an agent only varying the parameter $\eta$ in the interval $[0,1]$. We assume $\varepsilon = 0.7$ and $\mathbf{v} = (0.7,-0.7)$ constants for Equation 1.

As previously commented, the parameter $\eta$ acts as an inertial factor that tends to keep constant the agent direction insofar $\eta \to 1$. Hence, the bigger the $\eta = 1$ the smoother the path is. In Figure 5e, the influence of $\eta$ is so strong that the agent has not been able to reach the target position, passing by it and colliding against the wall at the end of the corridor.

### 6.2 Varying the Size of the Agent Map

Interesting results can be produced in the way agents interact with others in the environment by varying the size of the agent map. The more information on environment is available to the agent the more it will tend to change its behavior to avoid regions with plenty of obstacles. Figure 6 shows two situations where an agent finds a group in its path. In Figure 6a, the side length of the agent map is the half of the corridor width, while in Figure 6b, it corresponds exactly to the corridor width. In the first case, the agent passes in the

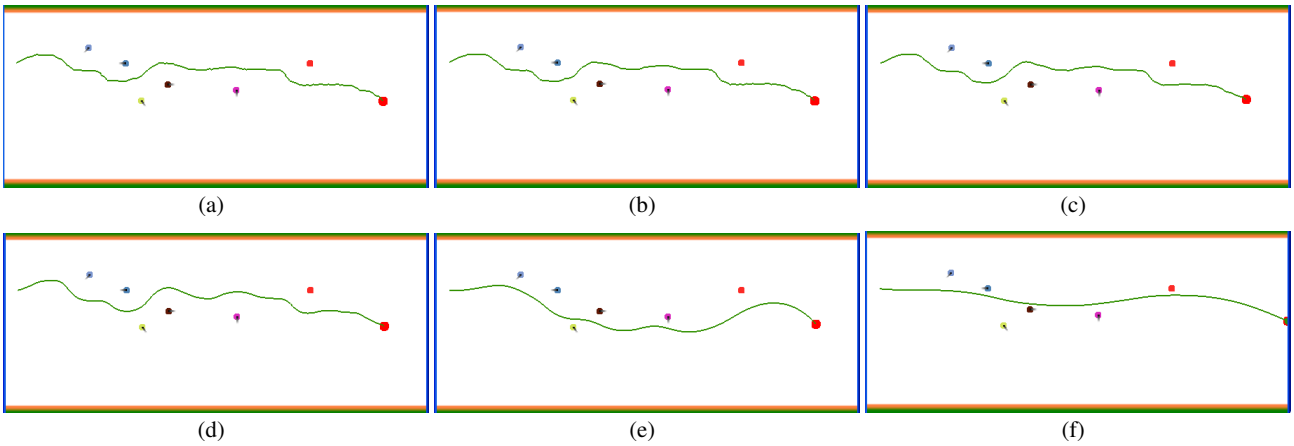



**Fig. 5** Varying the parameter $\eta$ in the interval $[0,1]$. From (a) to (e), $\eta$ is equal to 0, 0.25, 0.5, 0.75, 0.95 and 0.99, respectively.

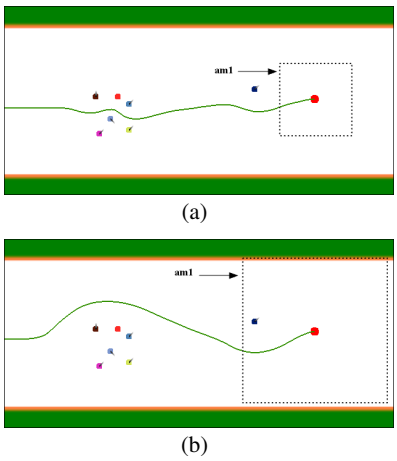



**Fig. 6** Varying the size of the agent map

middle of a group with other agents, whereas in the second case, the agent avoids the group.

The size of the local map can be controlled adaptively using, for example, information about the subjective state of agents. This idea is yet under development, but we obtained preliminary results in robotics context [13] where the robot dynamically adjusts its field of view using information coming from its sonar sensors.

### 6.3 Varying the Speed

As discussed before, the variation of the speed parameter is very important to generate not only more realistic simulations but also to refine the result of the collision avoidance between agents. Figure 7 shows two experiments that point out the importance of adjusting the agent speed.

In Figure 7a, both agents keep their speed constant during the simulation, tending to follow unnatural paths. For instance, the blue agent described a circular path. In Figure 7b, they vary their speed according to Equation 6, showing a natural balance in the negotiation of the space. The blue agent stops to allow the red one getting through. This reflects more adequately pedestrian behaviors found in real world.

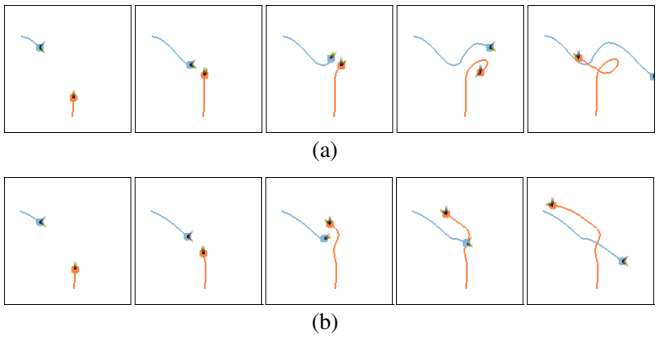
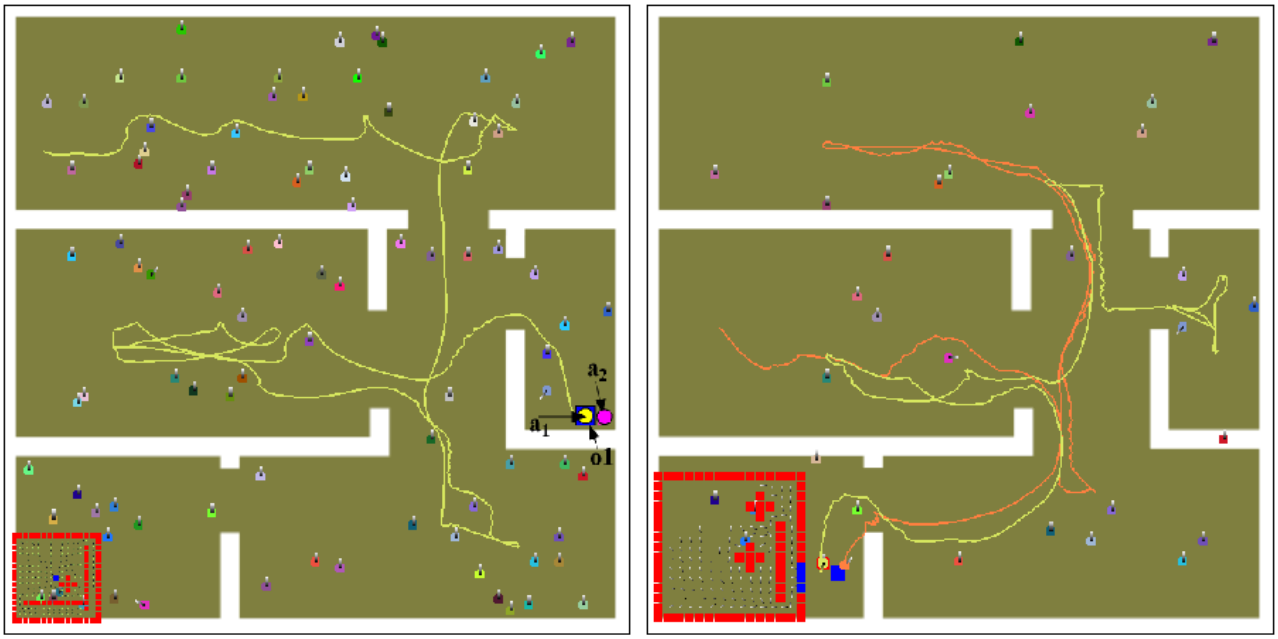



**Fig. 7** Simulation of two agents moving one against the other with constant speed (a), and varying the speed (b) to negotiate the space.

### 6.4 Exploring an Unknown Environment

Our framework is not limited to generate pedestrian behaviors in path planning tasks. It can be also used for more complex tasks as the discovery of targets in unknown environments. In a previous publication [15] we used a small version of this framework to endow a mobile robot Pioneer the ability to seek targets.

Initially, potential fields of the global map cells are updated with a low potential value, indicating that the agent does not know its environment. Then, at each step the agent gathers information using its sensors and adds it into the global map. The cells covered at least one time by the agent sensors have their potential values updated using the relaxation process. The other cells keep their potential value gen-




(a) (b)

**Fig. 8** Searching for an object in an unknown environment: Agent $a_1$ visits the environment until finding the searched object $o_1$ (a); two agents $a_1$ and $a_2$ are in charge of finding the same object $o_1$ (b).

erating an attracting force that pulls the robot towards them. Afterwards, the agent calculates the descent gradient on the cell associated to its current position in the global map and uses it to generate an intermediate goal, comparable to the process commented in Section 4.3. This intermediate goal leads the agent automatically towards the nearest region not visited. These steps are very similar to the algorithm proposed by us in a previous work [15].

Figure 8a shows the path followed by the agent $a_1$ during the search for the object $o_1$ and Figure 8b shows the case where two agents $a_1$ and $a_2$ are searching for the same object $o_1$. Both situations can be easily found in a party, where a person must find another in a large group of people spatially distributed.

We can perceive that the agent path seems unnatural when compared to the paths generated in the previous experiments. This happens because in the previous examples the global potential field has been computed before the simulation starts and, while in this experiment, it is calculated during the agent movement. Thus, when the agent identifies the presence of an obstacle, it updates the global map and relaxes its cells. After, the system dynamics makes the agent moves to the largest unvisited region, which can be in an opposite direction of the global gradient descent. We are currently treating this situation in order to make the exploratory behavior realistic.

### 6.5 Considerations about Performance

Considering the visualization of pedestrian simulations, for each new step the agent do, the motion planner should provide its new position and orientation. According to Mazarakis and Avaritsiotis [9], the frequency of human steps varies from 0.9 to $1Hz$ for someone walking slowly to $3.5Hz$ for someone walking very fast, with a mean of $2Hz$. Then, the performance of a real-time algorithm should be enough to calculate until 3.5 (2 as a mean) steps per second per agent, to animate it and to render the complete scene.

In our experiments, we were not yet concerned by the rendering quality, but only by the quality of the behaviors generated and the number of agents carried by the algorithm. Figure 9 illustrate the results obtained on an ATHLON 64 3500+ 2.21GHz computer with 2.0 GB RAM and graphics card nVidia 7800 GTX. For each step of each agent, considering the mean of 2 steps per second, we have done 60 relaxations of the matrix representing the local map. This allows the management of up to 300 agents concurrently. If we consider the max of 3.5 steps per second, 200 agents are allowed at the same time.

However, this performance evaluation is simplistic, since 3D animation and rendering is not being considered, as well as algorithm optimizations. Besides, a better compromise between rendering, animation and path planning algorithms can be obtained by reducing the number of relaxations for the local maps. In several examples presented in this section, we used 30 relaxations per step done, instead of 60.

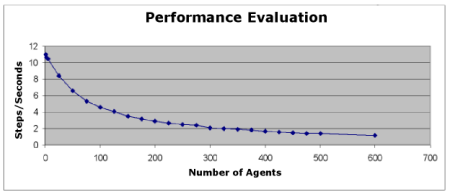



**Fig. 9** Performance evaluation.

## 7 Conclusions and Future Work

We presented a path planner based on a numerical solution for boundary value problems to produce realistic steering behaviors for virtual humans. In a previous work [4] we demonstrated that adjusting the behavior vector $\mathbf{v}$ and the parameter $\varepsilon$, that determines the vector influence, interesting behaviors could be produced.

In this paper, we introduced a new technique to update the position of agents during the simulation, also varying its speed. We proposed a new equation (see Equation 5) to update the agent position that includes the parameter $\eta$, representing the inertial factor, used to keep constant the agent direction during the movement. The possibility of varying the speed helps the agents to naturally negotiate the space to try to avoid eminent collisions, as shown in Figure 7. We have also demonstrated that changing the size of local maps (that can be dynamically changed) it is possible to produce different steering behaviors, as illustrated in Figure 6.

Finally, we performed some experiments involving the use of our method to explore unknown environments (see Section 6.4), which can be helpful for applications with very large environments where the topology is not completely known.

The method proposed is formally complete and generates smooth and safe paths. However, as it comes from harmonic functions path planner, it inherits their problems. We minimized the computational cost associated to the convergence of the potential field using, instead of large maps that cover all the environment, small local maps for each agent. Basically, we use several environment maps, one for each target, agent local maps and intermediate goals, as mentioned in Section 4. The size of local maps is a small fraction of the global map size and, therefore, the method has a small amount of cells to calculate the potential field. In this way, it is possible to have several agents acting in the environment still keeping an acceptable running time.

With the conclusion of this first part of the work, we are now exploring the adjustment of our algorithm to manage small groups of agents – while guaranteeing the individual personalities and mood – to be used in applications such as *RTS* (Real-Time Strategy) games. Some efforts will also be made to increase the algorithm performance, such as its adaptation to run into the GPU and the use of efficient data structures, e.g. quadtrees.

## References

1. Burgess, R.G., Darken, C.J.: Realistic human path planning using fluid simulation. In: Proceedings of Behavior Representation in Modeling and Simulation (BRIMS) (2004)
2. Choi, M.G., Lee, J., Shin, S.Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Trans. Graph. **22**(2), 182–203 (2003). DOI http://doi.acm.org/10.1145/636886.636889
3. Connolly, C., Grupen, R.: On the applications of harmonic functions to robotics. International Journal of Robotic Systems **10**, 931–946 (1993)
4. Dapper, F., Prestes, E., Idiart, M.A.P., Nedel, L.P.: Simulating pedestrian behavior with potential fields. In: Advances in Computer Graphics, *Lecture Notes in Computer Science*, vol. 4035, pp. 324–335. Springer Verlag (2006)
5. James J. Kuffner, J.: Goal-directed navigation for animated characters using real-time path planning and control. In: International Workshop on Modelling and Motion Capture Techniques for Virtual Environments, pp. 171–186. Springer-Verlag, London, UK (1998)
6. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration space. IEEE Transactions on Robotics and Automation **12**(4), 566–580 (1996)
7. Khatib, O.: Commande dynamique dans l'espace opérational des robots manipulaters en présence d'obstacles. Ph.D. thesis, École Nationale Supérieure de l'Aéronatique et de l'Espace, France (1980)
8. LaValle, S.: Rapidly-exploring random trees: A new tool for path planning. Tech. Rep. 98-11, Computer Science Dept., Iowa State University (1998)
9. Mazarakis G.P., A.J.: A prototype sensor node for footstep detection. In: Proceeedings of the Second European Workshop on Wireless Sensor Networks, pp. 415–418. IEEE Press (2005)
10. Metoyer, R.A., Hodgins, J.K.: Reactive pedestrian path following from examples. The Visual Computer **20**(10), 635–649 (2004)
11. Pettre, J., Simeon, T., Laumond, J.: Planning human walk in virtual environments. In: IEEE/RSJ International Conference on Intelligent Robots and System, vol. 3, pp. 3048 – 3053 (2002)
12. Prestes, E., Engel, P.M., Trevisan, M., Idiart, M.A.: Exploration method using harmonic functions. Robotics and Autonomous Systems **40**(1), 25–42 (2002)
13. Prestes, E., Trevisan, M., Idiart, M.A.P., Engel, P.M.: Bvp-exploration: further improvements. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2003)
14. Reynolds, C.: Steering behaviors for autonomous characters (1999). URL citeseer.ist.psu.edu/reynolds99steering.html
15. Trevisan, M., Idiart, M.A., Prestes, E., Engel, P.M.: Exploratory navigation based on dynamic boundary value problems. Journal of Intelligent and Robotic Systems **45**, 101–114 (2006)

**Acknowledgements** The authors thank CNPq and FAPERGS for the financial support.