Adriano Del Vigna de Almeida

B-Spline:CAEP – Algoritmos Culturais Para a Geração de Trajetórias B-Spline de Robôs Móveis

Curitiba

Outubro, 2005

Adriano Del Vigna de Almeida

B-Spline:CAEP – Algoritmos Culturais Para a Geração de Trajetórias B-Spline de Robôs Móveis

Dissertação submetida à Pontifícia Universidade Católica do Paraná como parte dos requisitos para a obtenção de Mestre em Engenharia de Produção e Sistemas, pelo Programa de Pós-Graduação em Engenharia de Produção e Sistemas – PPGEPS.

Orientador:
Leandro dos Santos Coelho

Pontifícia Universidade Católica do Paraná – PUCPR

Curitiba

Outubro, 2005

Dissertação de Mestrado defendida por Adriano Del Vigna de Almeida para a obtenção do grau de Mestre em Engenharia de Produção e Sistemas, Área de Concentração em Identificação de Sistemas e Controle, foi julgada adequada e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Produção e Sistemas (PPGEPS) da Pontifícia Universidade Católica do Paraná, pelos seguintes membros da banca examinadora:

Leandro dos Santos Coelho, Dr.
Orientador

Alcy Rodolfo dos Santos Carrara, Dr.
Pontifícia Universidade Católica do Paraná

Dante Augusto Couto Barone, Dr.
Universidade Federal do Rio Grande do Sul

Resumo

Pode se dizer que a geração de trajetórias é o mote principal da robótica móvel. Um robô móvel, incapaz de percorrer uma trajetória livre de obstáculos, de forma a atingir um objetivo pré-definido, poderia inclusive deixar de ser chamado de robô móvel. A geração de trajetórias se encontra em um nível de abstração superior em relação às outras tarefas que um robô móvel deve realizar, tais como o sensoreamento, e o controle de motores por exemplo. De fato, o planejamento de trajetórias fecha justamente este espaço entre o sensoreamento e o comando dos atuadores. Este trabalho apresenta a proposta da geração de trajetórias para um robô móvel por meio de curvas parametrizadas, especificamente *B-Spline*; faz uso de todas as características deste tipo de curva, em especial a definição da mesma por *pontos de controle*, para a definição de uma trajetória suave para o robô móvel, livre de obstáculos e dirigida a um ponto de destino no espaço. Esta curva, que remete à trajetória do robô é ajustada por meio de um *algoritmo cultural*. A questão principal deste trabalho é analisar o comportamento da definição de um caminho *B-Spline* sob a influência de um *algoritmo cultural*. É determinar qual comportamento o algoritmo assume quando o problema em questão é o planejamento de trajetórias para um robô móvel.

Orientador: Prof. Dr. Leandro dos Santos Coelho
Área de Concentração: Identificação e Controle de Sistemas
Palavras-chave: robôs móveis autônomos, navegação, planejamento de trajetória, algoritmos culturais, B-Spline
Número de páginas: 178

Abstract

It could be said that path planning is the main goal of mobile robotics. A mobile robot, which is unable to follow a collision-free path, in order to reach a goal point, should not even be named mobile robot. The path generation rests in an upper level in relationship with the other tasks that a robot must accomplish, such as sensoring and motors control. This work addresses the purpose of mobile robot path generation by means of parametric curves, specially B-Spline. It makes use of all the caracteristics of this kind of curve, in special the definition of such curves by means of control points, in order to define a smooth, collision-free path addressed to a goal point in space for the mobile robot. This curve, which in fact reflects the path of the mobile robot, is adjusted by a cultural algorithm. The purpose of this work is to analyze the behaviour of a B-Spline path definition under the influence of a cultural algorithm, and to determine which behaviour the algorithm assumes when the problem is the mobile robot path planning.

Advisor: Prof. Dr. Leandro dos Santos Coelho
Area of Concentration: Identification and System Control
Keywords: autonomous mobile robots, navigation, path planning, cultural algorithms, B-Spline
Number of Pages: 178

Prefácio

Informações do Projeto

Esta dissertação faz parte do Programa de Pós-Graduação em Engenharia de Produção e Sistemas (PPGEPS), mais especificamente da linha de pesquisa em Identificação e Controle de Processos, da Pontifícia Universidade Católica do Paraná (PUCPR). O projeto foi aprovado e supervisionado pelos membros diretores do PPGEPS. O período do mestrado se estendeu entre os anos 2003 e 2005.

Agradecimentos

Os primeiros agradecimentos vão para os meus pais, Eduardo e Sonia, que sempre me proporcionaram todos os meios para que eu trilhasse meu caminho acadêmico com segurança, além da prontidão para tudo aquilo que eu precisasse.Além de sempre priorizarem a educação dos próprios filhos, uma atitude das mais nobres, e que o tempo tem mostrado ser acertada.

Agradeço também à paciência, compreensão, persistência e amor de minha namorada, Camila. Foi ela quem me resgatou para a realidade muitas vezes após devaneios, introspecção e a solidão que este trabalho acabou exigindo. Sem ela, todo este período teria beirado o intolerável. Também peço humildemente desculpas a ela, pelos momentos em que me vi forçado a deixá-la, de certa forma, de lado. Espero que estes momentos preteridos possam ser prontamente recuperados ou compensados.

Minhas irmãs, Patrícia e Valéria, e meu grande amigo, Raul. Cujas conversas, sessões de vídeo, cinema, video-game, risadas, brincadeiras e outros momentos inestimáveis, preencheram, e continuam preenchendo minha vida com a maior riqueza disponível neste mundo, a amizade. Camila, o mesmo vale para você.

Aos meus amigos, Chip e Celsus. Celsus, guardo você em meu coração. Chip, agora poderemos brincar por muito mais tempo, assim espero.

Os últimos agradecimentos, em especial, vão para duas pessoas incríveis, que conheci

melhor ao longo da jornada que culminou neste trabalho. Me reservo ao direito de omitir seus títulos acadêmicos, pois estes agradecimentos vão diretamente para as pessoas. A Osiris Canciglieri Junior, cujas conversas, discussões e conselhos cairam como uma luva para nortear muitas de minhas ações. Muito obrigado, professor. A Leandro dos Santos Coelho, uma pessoa incrível, de uma bondade, compreensão, paciência, conhecimento, e mais uma infinidade de adjetivos positivos, que só fariam jus em seus respectivos superlativos, para descrever o mérito de tal pessoa. Posso dizer que o senhor encerra o puro arquétipo de professor. Muito obrigado, professor. Guardo, para estas duas pessoas, um sentimento especial, e espero não perder o contato.

São pessoas como estas que tornam os grandes obstáculos de uma vida, pequenos degraus. Meu obrigado a todos.

Dedico este trabalho à sociedade brasileira, para que um dia encontre seu caminho ótimo, que sua inocência e esperança não sirvam de obstáculos para este objetivo. Espero que, um dia, seja encontrada uma função de adequação tal que o ponto de destino de nosso país seja apenas a grandeza, em todos os seus aspectos.

Sumário

Lista de Figuras

Lista de Algoritmos

Lista de Acrônimos

ICC: *instantaneous center of curvature*, ou centro de curvatura instantânea;

ICR: *instantaneous center of rotation*, ou centro de rotação instantânea;

AD: conversor analógico-digital;

Sonar: *sound navigation and ranging* , navegação e medição por ondas sonoras;

Radar: *radio detecting and ranging*, detecção e medição por ondas de rádio;

GPS: *global positioning system*, sistema de posicionamento global;

Pixel: contração do termo em inglês *picture element*, elemento de figura;

Voxel: contração do termo em inglês *volume element*, elemento de volume;

3D: três dimensões, tridimensional;

BSP: *binary space partitioning*, particionamento binário do espaço;

2D: duas dimensões, bidimensional;

IA: inteligncia artificial;

SC: *soft computing*, termo traduzido para a língua portugesa como *inteligência computacional*;

HMIQ: *high machine inteligence quotient*, algo como quociente elevado de inteligência de máquina;

FL: *fuzzy logic*, lógica *fuzzy*, ou lógica nebulosa;

NC: *neurocomputing*, algo como computação neurológica;

PC: *probabilistic computing*, computação probabilística;

NN: *neural network*, rede neural;

BP: *backpropagation*, algo como retro-propagação;

RBF: *radial basis functions*, funções de base radial;

SOM: *self organizing maps*, mapas auto-organizáveis;

EC: *evolutionary computing*, que na língua portuguesa pode ser traduzido como computação evolutiva ou computação evolucionária;

ES: *evolutionary strategies*, estratégias evolutivas, ou estratégias evolucionárias;

EP: *evolutionary programming*, programação evolutiva ou evolucionária;

GA: *genetic algorithms*, algoritmos genéticos;

GP: *genetic programming*, programação genética;

CA: *cultural algorithms*, algoritmos culturais;

1 Introdução

A habilidade de navegar eficientemente é fundamental à maioria dos animais e organismos inteligentes. A importância da mobilidade espacial pode ser corroborada pela existência de uma pequena quantidade de organismos biológicos sofisticados que não podem se mover ou mesmo atingir tarefas distribuídas ao longo do seu ambiente de interação.

É esperado que a partir do desenvolvimento da tecnologia robótica surja uma mudança no contexto das aplicações do robô. A robótica móvel motiva este tipo de mudança.

Desde os primeiros protótipos de um robô móvel autônomo foi reconhecido que, independente do mecanismo usado para mover o robô ou os métodos para sentir o ambiente, os princípios computacionais que governam o robô são de vital importância.

O campo da robótica móvel é uma área de pesquisa relativamente nova, que lida com o controle de veículos autônomos ou semi-autônomos. O que faz os robôs móveis tão diferentes de outras áreas de pesquisa convencionais da robótica, como a robótica de manipulação, é a ênfase na solução de problemas relacionados ao entendimento do espaço em larga escala; regiões substancialmente maiores das que podem ser observadas por apenas um ponto de observação. O comportamento em ambientes de larga escala não se relaciona apenas na aquisição incremental de conhecimento, a estimação do erro posicional, a habilidade de reconhecer objetos ou localidades relevantes, e a resposta em tempo real. Mas requer que estas características sejam exibidas harmoniosamente, muitas vezes em paralelo. As tarefas de se mover através do espaço, sentir o espaço e deliberar sobre este espaço são alguns dos problemas fundamentais no estudo da robótica móvel.

Os robôs móveis apresentam pelo menos duas características importantes, que os diferenciam da robótica de manipuladores, por exemplo[1]:

Alcance: robôs móveis (e portanto mobilidade) são necessários se a área de atuação (envelope) do robô não está restrita a uma área suficientemente pequena.

Flexibilidade: se a posição de atuação do robô não é estática, o robô pode ter a carac-

terística de perseguir seu objetivo.

Uma das tarefas que caracteriza um robô móvel é a navegação, sua capacidade de se locomover da maneira mais inteligente possível em seu ambiente de trabalho[2]. A navegação está situada entre o controle dos atuadores (motores) e a execução da tarefa[3].

Para que um robô móvel apresente estas características, é comum definir o seguinte algoritmo para o robô[2]:

1. percepção do ambiente ao seu redor, geralmente com auxílio de sensores;
2. mapeamento do ambiente de trabalho, a partir das informações coletadas pelos sensores;
3. com base no mapa recém elaborado, e de posse de sua posição inicial e da posição final desejada, o robô calcula uma trajetória tendo como ponto inicial a posição atual, e como ponto final a posição desejada, buscando sempre desviar de obstáculos que se oponham à sua locomoção;
4. a trajetória planejada é corrigida para se adaptar às limitações do robô móvel, como raio de curvatura, velocidade e dimensões físicas do robô;
5. uma vez calculada a trajetória, esta deve ser executada o mais fielmente possível ao planejado. Faz uso dos atuadores do robô (como motores elétricos) submetidos a um mecanismo ou algoritmo de controle, que ajusta os parâmetros do robô em tempo real, para evitar qualquer desvio da trajetória.

É simplista dizer que robôs móveis são apenas uma coleção de algoritmos para o sensoreamento, deliberação e movimento através do espaço; eles são, na verdade, a materialização destes algoritmos e idéias que devem cooperar para superar os obstáculos do mundo real. Os robôs móveis são verdadeiros campos de prova para conceitos teóricos e algoritmos.

Robôs que assumem a mesma estrutura fisica humana, são denominados *robôs antropomórficos*. Existem várias razões para que os cientistas desenvolvam robôs antropomórficos. Em complemento ao desejo de criar um agente à sua própria semelhança, existem razões práticas. O ambiente operacional para muitos robôs móveis é o mesmo ambiente habitado por seres humanos, ambiente que já se encontra adaptado às nossas necessidades. "Mimetizando" a estrutura humana, pelo menos em termos funcionais, um robô já

se encontra de alguma forma adaptado a operar neste ambiente. Quando não há esta necessidade de mimetização, é conveniente dar as mais variadas formas aos robôs móveis, explorando ao máximo as peculiaridades de cada necessidade no projeto físico do robô.

O estudo da robótica móvel é um campo de pesquisa intrínsecamente interdisciplinar, que envolve áreas como:

Engenharia Mecânica – projeto do veículo e mecanismos de locomoção.

Ciência da Computação – representações lógicas, sensoreamento e algoritmos de planejamento e controle.

Engenharia Elétrica – integração de sistemas, motores, sensores e comunicações.

Biologia, psicologia cognitiva, percepção e neurociência – inspiração nas soluções da natureza para problemas similares.

Apesar da maioria dos sistemas robóticos móveis ainda possuirem um caráter puramente experimental, alguns sistemas já estão sendo implementados em ambientes industriais, ambientes militares, outros começam a habitar até mesmo nossos lares. Tais aplicações reais se caracterizam pelas seguintes características: ausência de um operador humano, um custo potencialmente alto e a necessidade a tolerância a ambientes com condições inaceitáveis para seres humanos. Desta forma, robôs são especialmente recomendados para as tarefas que exibem uma ou mais das seguintes características:

- um ambiente inóspito no qual o envio de um ser humano seria perigoso ou caro;
- um ambiente remoto no qual o envio de um ser humano seria difícil ou demorado;
- uma tarefa com uma demanda de desempenho ou fator fatigante elevados;
- uma tarefa degradante para seres humanos.

Schreiner[4] apresenta uma pesquisa que pretende incrementar a coleta de informações militares em ambientes urbanos. Um consórcio americano está desenvolvendo um pequeno robô móvel, capaz de conduzir o reconhecimento em áreas que são ou muito perigosas para soldados ou até mesmo inacessíveis. O consórcio é liderado pela NASA e recebe a colaboração de diversas outras instituições de tecnologia americanas. De acordo com a agência espacial americana, as atividades do pequeno robô móvel vão incluir o

reconhecimento de alvos militares, pessoas, materiais dentro de construções e agir em locais onde soldados não têm acesso, como cavernas. Como requisitos do projeto estão a portabilidade (o robô deve poder ser carregado e operado por apenas um soldado) e a robustez (aguentar quedas e os obstáculos dos campos militares). Deve também ser capaz de escalar escadas, obstáculos e ser operável tanto de dia quanto à noite.

Bay[5] defende o projeto *army-ant* (exército-de-formigas), idealizado como um sistema flexível de manipulação de materiais. Inseridos em um cenário, um conjunto de pequenos robôs poderia ser disponibilizado em grande número para, cooperativamente, erguer e transportar objetos.

Pequenos robôs poderiam se posicionar ao redor da periferia de um grande objeto, deslizar sob o objeto e então mover o objeto de forma coordenada, para transportá-lo.

Murphy[6] apresenta o projeto de um robô polimórfico que se aplica na busca e resgate urbanos (do inglês *Urban Search and Rescue* - USAR). O robô é projetado para localizar e extrair pessoas presas em estruturas destruídas ou danificadas. Uma tarefa que requer uma integração maior das técnicas de inteligência artificial, tais como: mobilidade em três dimensões, a habilidade do robô em mudar sua forma primária, autonomia ajustável, interação robô-humano, fusão de sensores e construção e localização em mapas com precisão[7].

O movimento sobre os destroços, ou a entrada em pequenos vãos por humanos, cães ou equipamentos pode causar desmoronamentos, ferindo ou até mesmo matando possíveis sobreviventes ou membros da equipe de resgate. Vazamentos de gases e explosões também são riscos constantes. Cães treinados também são usados para reduzir o risco humano, conseguindo entrar em pequenos vãos nos destroços onde um ser humano não poderia ir. Entretanto, em algumas situações eles não podem substituir uma câmera de vídeo ou um equipamento de resgate, nem mesmo atuar prontamente sobre uma vítima, caso necessário.

Pequenos robôs móveis poderiam auxiliar na busca e resgate urbanos, reduzindo o risco tanto para humanos quanto para os cães. Por exemplo, eles poderiam[6]:

- conduzir busca por sobreviventes com um nível de rigor que normalmente é fatigante para humanos;
- inserir e posicionar sensores especializados em meio aos destroços;
- coletar dados visuais e sismológicos para avaliar o estrago estrutural;

- depositar transmissores de rádio, pequenas quantidades de comida e água, ou ainda medicamento próximo aos sobreviventes;
- guiar a inserção de ferramentas;
- identificar a disposição de vítimas, prevenindo, por exemplo, os danos a braços e pernas durante a extração de vítimas soterradas.

De fato, todas estas caracterísitcas foram exploradas e corroboradas no atentado terrorista ao World Trade Center, como analisado em [8].

Um exemplo de até onde as pesquisas podem chegar pode ser dado pelo robô da Honda, Asimo[9]. Em 1986, a empresa Honda, conhecida mundialmente pelo desenvolvimento de automóveis e motores a combustão, iniciou a pesquisa e desenvolvimento de um robô humanóide. Tendo como lemas a "mobilidade" e a "inteligência", a empresa iniciou seus trabalhos tendo como conceito básico "que os robôs devem coexistir e cooperar com seres humanos, fazendo aquilo que uma pessoa não pode fazer e cultivando uma nova dimensão em mobilidade, trazendo benefícios para a sociedade". Isto serviu de guia para o desenvolvimento de um robô revolucionário que poderia ser usado no dia-a-dia, ao invés de um robô construído com um propósito puramente científico.

Durante um ano foi planejado o que o robô seria, com o que se pareceria, a fim de se criar o primeiro protótipo. O robô teria de ser capaz de se mover em ambientes mobiliados, subir e descer escadas, já que ele seria destinado ao uso doméstico. Ao mesmo tempo, o time do projeto decidiu que o robô teria como meio de locomoção duas pernas, de modo que o robô fosse compatível com a maior parte das construções humanas.

Com estas idéias em mente, os engenheiros da Honda iniciaram o programa de desenvolvimento, focados na locomoção sobre duas pernas, que corresponde à locomoção básica de um ser humano. Um sem número de desafios técnicos tiveram que ser superados antes da criação do primeiro protótipo. Grande atenção foi dada em como as pernas, tornozelos e pés humanos funcionam mecanicamente. O robô Asimo se encontra em sua terceira versão e é o maior exemplo de comunhão entre ficção e ciência.

Curiosamente, o termo *robô autônomo* cria uma contradição. A palavra *robô*, usada pela primeira vez por Karel Capek em sua peça de 1923 R.U.R., é derivada da palavra tcheca ou polonesa *robota* que significa "trabalho", "trabalhador". Já a palavra *autônomo* tem origem Grega, e tem significado "aquele com vontade própria".

1.1 Contexto Histórico

Os dispositivos autônomos têm lugar reservado em grandes papéis desde as lendas antigas à literatura moderna. Relatos de tais dispositivos remotam desde lendas mitológicas gregas e o folclore judaico. Das lendas à literatura, Mary Shelley escreveu Frankenstein, uma criatura formada a partir de partes humanas; Júlio Verne descreve um elefante mecânico. Isaac Asimov, considerado o pai da robótica, popula sua obra com máquinas maravilhosas, cria até um conjunto básico de leis a serem seguidas por tais máquinas, verdadeiros corolários.

A partir de década de 1940, os robôs móveis se tornam personagens comuns da literatura científica e do cinema. *Robbin*, de planeta Proibido, *Rosie*, dos Jetsons, *Robô*, de Perdidos no Espaço, *R2D2* e *C3PO*, de Guerra nas Estrelas a *Data*, de Jornada nas Estrelas.

Foi também em meados da década de 1940 que os robôs passaram do universo ficcional e aportaram na realidade. Norbert Wiener, o pai da cibernética, foi um matemático estudioso dos sitemas regulatórios e suas aplicações em controle. Durante a Segunda Guerra Mundial esteve envolvido no desenvolvimento de um dispositivo de controle de uma arma anti-aérea. Este trabalho resultou no primeiro sistema robótico de aplicação militar que se tem registro. Tal dispositivo integrava informações de sensoreamento (radar) via processamento (regras de controle simples, em um computador analógico).

Paralelamente ao trabalho de Wiener, a Alemanha nazista desenvolvia os projetos V1 e V2, foguetes auto-guiados, exemplos notórios de sistemas autônomos. Com o desenvolvimento dos computadores digitais veio a possibilidade do desenvolvimento de robôs móveis mais complexos. Desde meados do século XX, com o advento da computação digital, a robótica móvel se espalha pelo mundo, invade laboratórios universitários, parques industriais[10, 11], até o dia em que virá a fazer parte de nosso dia-a-dia.

1.2 Formas de Concepção

Os robôs podem ser considerados sob diversas perspectivas. Do ponto de vista físico, os robôs podem ser decompostos da seguinte forma[12]:

- uma combinação de mecanismos eletro-mecânicos, capaz de mover a si mesmo no ambiente;

- uma coleção de unidades de processamento, seja analógica ou digital, para o controle do robô;
- uma coleção de sensores com os quais o robô coleta informações do ambiente;
- um ou mais enlaces de comunicação, que possibilitam ao robô se comunicar com um operador remoto ou qualquer computador externo.

Sob uma perspectiva abstrata, pode-se considerar os robôs móveis dentro de uma esfera puramente computacional, de tal forma que sensores, comunicação, motores são vistos como módulos de *software* que permitem ao robô interagir com o ambiente. Os componentes típicos são[12]:

- sistema de controle de movimento;
- sistema de controle de sensores;
- sistema de interpretação de sensores.

Outras formas abstratas de representação podem ser formuladas. O termo *robótica cognitiva* é usado para indicar o uso de técnicas de *inteligência artificial* em um robô móvel que eventualmente assume uma abstração computacional idealizada do robô.

1.3 Formas de Operação

Existem poucos, senão nenhum, robôs completamente autônomos fora de ambientes de pesquisa altamente limitados. A maior parte dos robôs são projetados para operar com algum grau de controle humano. Até mesmo dos robôs autônomos espera-se que obedeçam sua programação, por exemplo[1, 3].

Quando um sistema robótico é descrito como sendo *completamente autônomo* o sistema é projetado para operar sem controle humano externo. Esta definição difere dos sistemas *semi-autônomos*, nos quais um operador é necessário, porém, o robô é capaz de tomar certas decisões por si só. Dentro do conjunto de sistemas semi-autônomos existem os sistemas *teleoperados*, que são controlados remotamente momento a momento, o operador comanda todos os movimentos do robô, em nível mais alto[3, 12], e os sistemas *telerobóticos*, nos quais os comandos de baixo nível são interpretados ou filtrados por complexas camadas de *software*. O operador indica um conjunto de macro comandos a serem executados e o robô os cumpre de forma indenpendente[1, 12].

Os robôs completamente autônomos devem realizar suas tarefas tomando suas próprias decisões. Eles recebem um conjunto de tarefas em alto nível, sabendo o que deve ser feito, mas não como deve ser feito. Devem selecionar as ações para o cumprimento de uma tarefa, decidindo apropriadamente sempre que encontrar situações não previstas, adaptando-se às mudanças do ambiente de forma a cumprir adequadamente suas metas.

De forma simplificada, o problema abordado pela navegação de robôs móveis compreende a determinação e o cumprimento de uma trajetória livre de obstáculos compreendida entre um ponto inicial e o ponto ou pontos objetivos, onde o robô deve estar no final de sua trajetória. Os principais empecilhos para a solução deste tipo de problema no mundo real são:

Inacessibilidade: os sensores podem fornecer leituras imprecisas e têm alcance limitado, não podendo fornecer uma leitura do ambiente completo.

Indeterminação: o mundo real é intrinsecamente não-determinístico. Um sem número de eventos podem desqualificar qualquer planejamento do robô, como o deslizamento de uma roda, ou a falha de um sensor.

Imprevisibilidade: as ações não podem ter todas as suas reações modeladas ou planejadas.

Dinamicidade: o mundo real é algo em constante transformação. Todas as formas de energia estão "à deriva".

Continuidade: o universo para o robô se altera de forma contínua, e não discreta. Nem sempre é possível "empacotar" eventos em intervalos de tempo de forma adequada[1].

Estes tipos de obstáculos – em conjunto com aqueles proporciondados pelo ambiente – devem ser contornados tanto pelos robôs quanto pelos pesquisadores.

1.4 Objetivos

Este trabalho visa a implementação de um sistema para o planejamento de trajetórias de robôs móveis em ambientes estáticos. Especificamente, implementa-se um algoritmo CAEP - *Cultural Algorithm with Evolutionary Programming* para a determinação de trajetórias livres de colisão em ambientes estáticos, fazendo uso de curvas *B-Splines*.

[1] "Empacotar" no sentido da busca por um intervalo de amostragem do ambiente.

Pretende-se como resultado maior deste trabalho, a implementação de um algoritmo *B-Spline:CAEP*, que faça uso de caracterísiticas como previsibilidade e controle de curvas *B-Splines*, e velocidade de convergência de algoritmos culturais para a determinação de trajetórias livre de colisão para robôs móveis.

Adicionalmente, é feita uma revisão dos principais problemas e soluções utilizados na robótica móvel. Também é apresentada uma revisão das principais técnicas de *soft computing* – inteligência computacional – bem como sua aplicação na robótica móvel. Com estas revisões almeja-se semear o terreno para a implementação do algoritmo *B-Spline:CAEP*.

1.5 Organização da Dissertação

No capítulo 2, são apresentadas algumas das formas mais comuns de locomoção de robôs móveis, o que, de alguma forma, define algumas restrições e dificuldades na modelagem dos robôs. O capítulo 3 contém um resumo de algumas tecnologias que a robótica móvel usa para perceber o ambiente circundante ao robô móvel. Unindo os conteúdos dos capítulos 2 e 3 busca-se fornecer recursos suficientes para a compreensão das formas e principais problemas que um robô móvel encontra na interação com seu ambiente. Busca-se também esclarecer como estas dificuldades também são resultados das próprias escolhas do projetista.

No capítulo 4, são abordadas formas de representação espacial, algumas estruturas e técnicas que auxiliam a tomada de decisões por parte do sistema de controle do robô, especialmente no que tange ao planejamento de trajetórias. O capítulo 5 discute três das formas mais comuns de arquiteturas de controle, que fazem uso da representação espacial para deliberar sobre as ações do robô móvel.

No capítulo 6, é apresentada a teoria fundamental de curvas paramétricas, e busca-se relacionar esta teoria com a descrição de trajetórias de robôs móveis. Procura-se apresentar uma forma de representar uma ou mais trajetórias "suaves" de um corpo em movimento por meio de pontos de referência (*pontos de controle*). O capítulo 7 apresenta as diversas manifestações da Inteligência Artificial e Inteligência Computacional (*Soft Computing*), bem como algumas de suas problemáticas e características marcantes. Almeja-se com estes dois capítulos, a provisão da base teórica para a aplicação de ferramentas de Inteligência Computacional e curvas paramétricas para a solução de problemas do planejamento de trajetórias de robôs móveis, que se seguem nos capítulos subseqüentes.

O capítulo 8 descreve em detalhes a implementação do algoritmo *B-Spline:CAEP*, bem como a estrutura adjacente à ele, como a interface gráfica e classes especializadas. O capítulo 9 apresenta os resultados dos experimentos do trabalho descrito e implementado no capítulo 8. O capítulo 9 também resume alguns do possíveis trabalhos que porventura possam ser derivados pelo apresentado nesta dissertação.

2 Locomoção

Independentemente da tecnologia utilizada, a robótica, inclusive a móvel, precisa lidar com dois problemas fundamentais[10, 13]:

- dados os estímulos de controle, como o robô deve se mover. Este é conhecido como problema *cinemático*;
- dado um movimento desejado, qual estímulo de controle deve ser aplicado ao robô. Este é conhecido como problema de *cinemática inversa*.

Um robô móvel é uma combinação de vários componentes físicos (*hardware*) e computacionais (*software*). Em termos de componentes de hardware, um robô móvel pode ser considerado uma coleção dos seguintes subcomponentes[12]:

Locomoção: meios de como o robô se movimenta (atua) ao longo do ambiente;

Sensoreamento: meios de como o robô mede as propriedades de si mesmo e do ambiente;

Deliberação: como o robô mapeia, processa ou transforma estas medidas em ações;

Comunicação: como o robô se comunica com um operador externo ou supervisor.

A locomoção é o processo que proporciona ao robô autônomo, ou veículo, movimento[12]. Para produzir este movimento, forças devem ser aplicadas ao veículo. A *cinemática* trata do estudo matemático do movimento, desconsiderando as forças que afetam este movimento. Ela lida com os relacionamentos geométricos que governam o sistema, enquanto a *dinâmica* lida com a modelagem de forças, energia e velocidades envolvidas no movimento.

Levando em consideração apenas o domínio de aplicação de um robô móvel, estes dispositivos podem ser classificados em quatro categorias principais [12, 14, 15, 16, 17]:

Terrestres: aqueles robôs que se locomovem sobre uma superfície, exercendo algum tipo de contato com ela. Fazem uso de forças como gravidade, tração e atrito no relacionamento com esta superfície.

Aquáticos: aqueles que operam em contato com a água, tanto na sua superfície quanto submersos.

Aéreos: veículos robóticos que mimetizam aeronaves existentes.

Espaciais: veículos projetados para operar sob a microgravidade do espaço, geralmente destinados à manutenção de dispositivos espaciais ou o transporte de carga.

2.1 Robôs Móveis Terrestres Com Rodas

Dentre a grande variedade de estratégias de locomoção terrestre, certamente o uso de rodas é a mais comum[11, 17]. Ela faz uso do atrito com a superfície de deslocamento para proporcionar movimento ao robô. Considerando uma roda idealizada, ela é livre para girar em torno do seu eixo (geralmente transversal) o que propicia um movimento de rotação, perpendicular a este eixo de rotação. Eventualmente, devido a fatores como velocidade de rotação, velocidade de deslocamento do corpo do robô e forças laterais – como centrífuga e centrípeta no caso de movimentos curvilíneos – pode existir certo deslizamento paralelo aos eixos de rotação e/ou de deslocamento do robô (Figura 1).

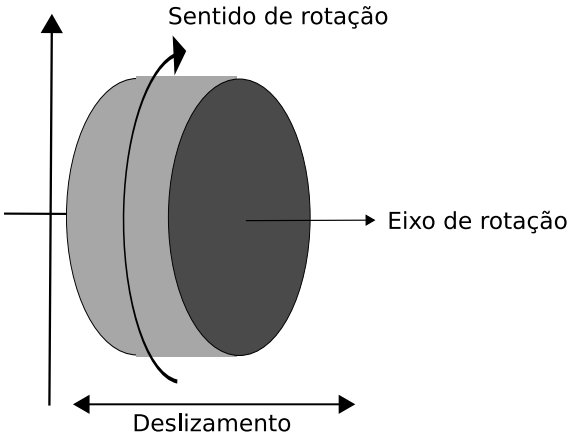

Figura 1: Eixo de rotação.

Uma das técnicas mais utilizadas para a estimativa do movimento de rodas é a *odometria*[15, 16, 17]; a estimativa da distância percorrida pela medida do número de rotações efetuadas por uma roda. No caso de uma roda ideal, isto implica que uma distância

de $2\pi r$ foi percorrida por cada rotação de uma roda com raio r (Figura 2). Adicionalmente ao deslizamento lateral – causado pelas forças centrífuga e centrípeta – tração insuficiente também pode levar ao deslizamento perpendicular ao eixo de rotação, ocasionando o giro "em falso" da roda, fatores que tornam a estimativa da distância percorrida imprecisa[18].Os trabalhos de Borenstein e seus colegas contemplam estes e outros tipos de problemas, bem como soluções relacionados à odometria[19, 20, 21, 22].

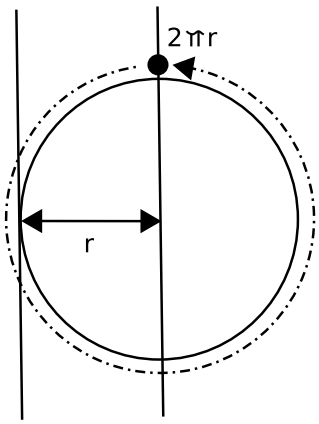

Figura 2: Exemplo de matemática envolvida na odometria.

Considere um veículo com várias rodas em contato com a superfície de deslocamento. Cada uma dessas rodas possui seu próprio movimento de rotação, perpendicular ao seu eixo de rotação. Desta forma, para um veículo impulsionado por rodas exibir um movimento curvilíneo ou de rotação em torno de um eixo transversal ao plano de deslocamento deve existir um ponto no ambiente, de tal forma que cada roda siga uma tajetória circular. Este ponto é denominado *centro de curvatura instantânea*, do inglês *instantaneous center of curvature* (ICC), ou ainda *centro de rotação instantânea*, do inglês *instantaneous center of rotation* (ICR). Para que o veículo altere seu ICC, alguma propriedade das rodas, como a orientação individual das rodas com respeito ao seu eixo de rotação, deve ser alterada. Na prática, é relativamente simples identificar o ICC, por que ele é definido pelo ponto de intersecção de cada eixo de rotação de cada roda em contado com a superfície de deslocamento (Figura 3). Se todas as rodas possuem contato com esta superfície, então não apenas existe um ICC, como a velocidade de cada roda deve ser consistente com a rotação de todo o veículo ao redor desse ICC, possuindo diferentes velocidades angulares (em função do raio de curvatura do movimento)[12, 18].

Um veículo localizado em um plano de deslocamento idealmente deve possuir pelo menos três graus de liberdade: uma posição sob as coordenadas planas (x, y) e um ângulo

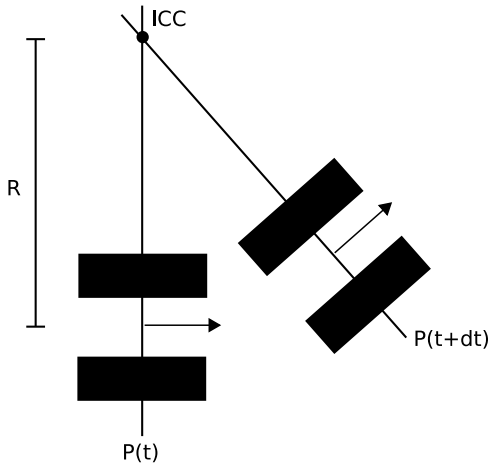

Figura 3: ICC da propulsão diferencial.

de orientação θ. A tupla (x, y, θ) é também referenciada como *pose*[1] ou *estado* do robô no plano (Figura 4).

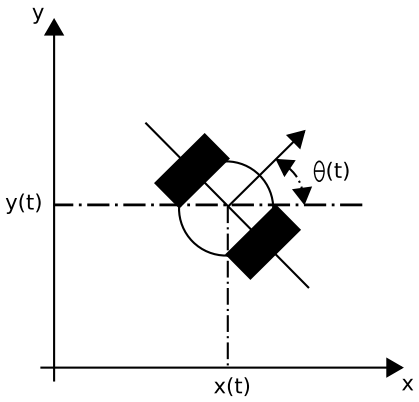

Figura 4: Pose do robô, em função da tupla (x, y, θ).

Os robôs móveis geralmente não possuem o controle independente e completo sobre estas três variáveis, e em certos casos devem realizar manobras complexas para atingir uma determinada posição. Um exemplo é estacionar um carro paralelamente à calçada. O motorista deve realizar um conjunto mínimo de manobras complexas de forma a atingir a posição desejada, paralela à calçada. Estes movimentos mínimos são dependentes da natureza do ambiente e da configuração do veículo. Ainda sobre o exemplo do carro, não é possível alterar arbitrariamente sua posição, nem mesmo existem mudanças de posição indenpendentes de sua orientação. Este tipo de restrição é chamada de *restrição holonômica*[12, 23], restrições que não envolvem velocidades ou acelerações[14].

[1] Do próprio inglês *pose*.

2.1.1 Propulsão Diferencial

A tração diferencial é o mecanismo de propulsão mais simples para um robô móvel em contato com o solo que faz o uso de rodas para seu movimento[12, 13, 16, 18]. Um robô com propulsão diferencial consiste de duas rodas montadas em um eixo comum, controladas por motores distintos[13], vide Figura 5.

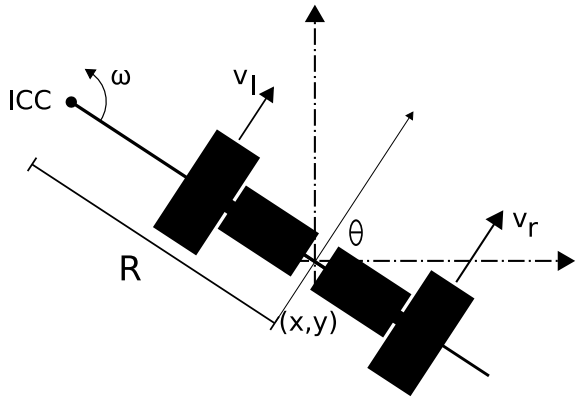

Figura 5: Parâmetros envolvidos na propulsão diferencial.

A cinemática lida com o relacionamento dos parâmetros de controle e o comportamento do sistema no espaço de estados[10, 13]. Sob propulsão diferencial, para cada uma das duas rodas produzir um movimento final de rotação diferencial, o robô deve rotacionar em relação a um ponto comum ao eixo das duas rodas. Pela variação das velocidades relativas destas duas rodas, o ponto de rotação (ICC ou ICR) pode ser variado, e diferentes trajetórias serem traçadas[16]. A cada instante do tempo, o ponto no qual o robô rotaciona deve ter propriedade tal, que as rodas esquerda e direita sigam um caminho que se move ao redor do ICC sob a mesma velocidade angular ω, desta forma

$$\omega(R + l/2) = v_r \tag{2.1}$$

$$\omega(R - l/2) = v_l \tag{2.2}$$

onde l é a distância ao longo do eixo entre os centros das duas rodas, a roda esquerda se move com uma velocidade v_l e a roda direita com uma velocidade v_r, ambas em relação ao solo. R é a distância a partir do ICC ao ponto mediano entre as duas rodas. Note que v_l, v_r, ω e R são funções no tempo. A qualquer instante no tempo, isolando R e ω resulta em:

$$R = \frac{l}{2}\frac{(v_l + v_r)}{(v_r - v_l)} \tag{2.3}$$

$$\omega = \frac{v_r - v_l}{l} \tag{2.4}$$

Alguns casos especiais são relevantes. Se $v_r = v_l$, então o raio R é infinito e o robô se move em uma linha reta. Se $v_r = -v_l$, então o raio é zero e o robô gira ao redor do ponto mediano entre as duas rodas. Isto faz a propulsão diferencial atrativa para a navegação em ambientes com espaços livres estreitos. Para qualquer outro valor de v_r e v_l o robô não se desloca em linha reta, pelo contrário, segue uma trajetória curvilínea ao redor de um ponto (ICC) a uma distância R do centro do robô, alterando tanto a posição do robô quanto sua orientação.

A estrutura cinemática do veículo proibe certos tipos de movimento. Por exemplo, não existe combinação de v_r e v_l tal que o robô se movimente paralelamente ao longo do eixo de rotação das rodas, de forma explícita (o deslizamento, na Figura 1).

Um veículo sob tração diferencial é muito sensível às velocidades relativas das duas rodas. Pequenos erros na velocidade proporcionada a cada roda pode resultar em trajetórias igualmente errôneas, e não apenas em um robô mais rápido ou lento.

Supondo que um robô esteja em uma posição (x, y), apontado para uma direção θ em relação ao eixo x, através da manipulação dos parâmetros de controle v_l e v_r, o robô pode assumir diferentes posições. A determinação da posição atingida dados os parâmetros de controle, *a priori*, é conhecido como problema de *cinemática direta*, do inglês *forward kinematics*[12, 10, 13]. Dados que v_l, v_r, R e ω são funções do tempo, é possível demonstrar que se o robô possui uma dada posição (x, y, θ)[13] em um dado instante t, e se as rodas direita e esquerda possuem velocidades v_r e v_l, respectivamente, durante o período $t \rightarrow t + \delta t$, então o ICC é dado por

$$ICC = [x - R\sin(\theta), y + R\cos(\theta)] \tag{2.5}$$

e no tempo $t + \delta t$ a posição do robô é dada por[2, 12]

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \tag{2.6}$$

A equação (2.5) descreve o movimento de um robô girando a uma distância R de seu ICC com uma velocidade angular ω. Os diferentes tipos de robôs vão gerar diferentes expressões para as variações de R e ω.

Pela integração da equação (2.5) a partir de uma condição inicial (x_0, y_0, θ_0), é possível determinar onde o robô estará para qualquer valor de t baseado nos parâmetros de controle $v_l(t)$ e $v_r(t)$, isto é, a solução do problema de cinemática direta para o veículo. Em geral, para um robô capaz de se mover com uma dada direção $\theta(t)$ a uma velocidade $V(t)$

$$x(t) = \int_0^t V(t)\cos[\theta(t)]dt \tag{2.7}$$

$$y(t) = \int_0^t V(t)\sin[\theta(t)]dt \tag{2.8}$$

$$\theta(t) = \int_0^t \omega(t)dt \tag{2.9}$$

e para o caso de um veículo com tração diferencial

$$x(t) = \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]\cos[\theta(t)]dt \tag{2.10}$$

$$y(t) = \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]\sin[\theta(t)]dt \tag{2.11}$$

$$\theta(t) = \frac{1}{l}\int_0^t [v_r(t) - v_l(t)]dt \tag{2.12}$$

Outra questão de igual relevância é a seleção dos parâmetros de controle de tal forma que o robô possa assumir uma posição final ou percorrer uma trajetória específica. Este é denominado de problema de *cinemática inversa*, do inglês, *inverse kinematics*[12, 10]. Trata-se da inversão do relacionamento entre as entradas de controle e o comportamento do robô. Este problema também está intimamente relacionado ao problema de planejamento da trajetória[10].

As equações de (2.7) a (2.12) descrevem uma limitação na velocidade do robô que não pode ser integrada em uma limitação de posição, o que é chamado de *limitação não-holonômica*[14], e tem soluções complexas. Entretanto, existem soluções facilmente demonstráveis para alguns casos especiais das variáveis de controle $v_l(t)$ e $v_r(t)$. Por

exemplo, se é assumido que $v_l(t) = v_l$, $v_r(t) = v_r$ e $v_l \neq v_r$, as equações (2.7) a (2.12) se resumem a

$$x(t) = \frac{l}{2}\frac{v_r + v_l}{v_r - v_l}\sin\left[\frac{t}{l}(v_r - v_l)\right] \tag{2.13}$$

$$y(t) = -\frac{l}{2}\frac{v_r + v_l}{v_r - v_l}\cos\left[\frac{t}{l}(v_r - v_l)\right] + \frac{l}{2}\frac{v_r + v_l}{v_r - v_l} \tag{2.14}$$

$$\theta(t) = \frac{t}{l}(v_r - v_l) \tag{2.15}$$

onde $(x, y, \theta)_{t=0} = (0, 0, 0)$. Dado um tempo objetivo t e uma posição objetivo (x, y). As equações (2.13) a (2.15) resolvem valores para v_l e v_r, mas não provêm controle independente para o controle do parâmetro θ. De fato, existem infinitas soluções para os parâmetros v_l e v_r, mas todos correspondem ao movimento do robô ao redor do círculo que passa pelo ponto $(0, 0)$ em $t = 0$ e (x, y) em $t = r$. A única diferença para cada solução é o sentido e o número de voltas ao longo deste círculo.

Ao invés de tentar inverter as equações (2.10) a (2.12) para a resolução dos parâmetros de controle, a fim de posicionar o robô em uma posição específica, pode-se considerar dois casos especiais de movimento para tração diferencial. Se $v_r = v_l = v$, o robô se move em linha reta, então o movimento do robô se resume a

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x + v\cos(\theta)\delta t \\ y + v\sin(\theta)\delta t \\ \theta \end{pmatrix} \tag{2.16}$$

e quando $-v_l = v_r = v$, as equações (2.10) a (2.12) se simplificam a

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + 2v\delta t/l \end{pmatrix} \tag{2.17}$$

Neste caso o robô gira ao redor de si mesmo. Assim, para mover o robô a um determinado objetivo (x, y, θ) ele pode girar até apontar para o ponto (x, y) e se mover em linha reta até atingir este ponto, e finalmente girar novamente para atingir a orientação determinada por θ. Obviamente estas não são as únicas soluções possíveis para a tração diferencial, mas se tratam dos casos mais simples e facilmente modelados matematica e

algoritmicamente.

2.1.2 Outras Formas de Propulsão com Rodas

Adicionalmente à propulsão diferencial, existem outras formas de propulsão com rodas para robôs móveis terrestres. Cada uma delas tenta explorar uma característica específica do ambiente ou do problema a ser resolvido em termos puramente de locomoção.

2.1.2.1 Propulsão Síncrona

Em um robô com propulsão síncrona, cada roda em contato com o solo é capaz de ser controlada e direcionada[13]. Configurações típicas envolvem três rodas direcionáveis dispostas nos vértices de um triângulo equilátero[12]. Uma roda direcionável é uma roda cuja orientação do eixo de rotação pode ser controlada.Uma vez que todas as rodas permanecem em paralelo, o robô andará em linha reta. Se os eixos de rotação de cada roda formarem um ICC sobre o centro do corpo do robô, o robô é capaz de girar sobre seu próprio eixo transversal, a exemplo da propulsão diferencial. Isto porporciona o controle direto do parâmetro θ.

A habilidade de controlar tanto a rotação quanto a velocidade do robô independentemente, simplifica o controle total e permite que tal tipo de propulsão sirva como um modelo idealizado para os robôs pontuais, robôs omnidirecionais[13, 18]. Em contraponto à facilidade de controle, existe a dificuldade de construção do mecanismo que proporcione a propulsão síncrona, geralmente por meio do uso de correias e da conexão física de todas as rodas juntas.

2.1.2.2 Rodas Direcionáveis

Os robôs que não fazem uso da propulsão diferencial (nenhuma roda diretamente direcionável) ou da propulsão síncrona (em que todas as rodas são diretamente direcionáveis) geralmente fazem uso de uma ou mais rodas que podem ser direciondas, e uma ou mais rodas cujas direções permanecem fixas. Para este tipo de robô, o cálculo do ICC pode se tornar extremamente complexo. Por exemplo, considere o caso do cálculo do ICC de uma bicicleta. O ICC deve se restringir à intersecção das linhas determinadas pelos eixos de rotação de cada roda. O ICC deve permanecer à linha que passa pelo eixo de rotação da roda traseira, sempre perpendicular ao corpo da bicicleta (Figura 6). A roda da frente pode ser direciona, desta forma o ICC é determinado exatamente pela intersecção da

linha determinada pelo eixo da roda traseira com a linha determinada pelo eixo da roda dianteira[12].

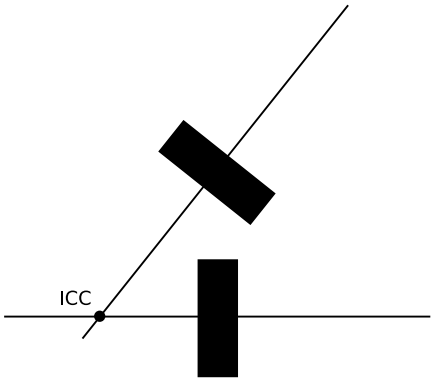

Figura 6: Posicionamento do ICC de uma bicicleta.

Configurações de triciclo – duas rodas com direção fixa e uma roda direcionável – e bicicleta possuem características cinemáticas semelhantes, embora a configuração de triciclo encontre um uso mais difundido na robótica móvel, basicamente devido à sua estabilidade estática[12, 15, 16, 18].

2.1.2.3 Propulsão de Carro - Direção de Ackerman

O direção de Ackerman é o tipo de direção encontrado nos automóveis. Neste tipo de direção, as rodas anteriores direcionáveis giram em eixos separados, de forma a proporcionar diferentes posições para o ponto de ICC[18], o qual sempre pertence à linha determinada pelos eixos das rodas traseiras (Figura 7). A direção de Ackerman é a preferida para veículos de maior porte que devem operar nas estradas existentes ou carregar grande quantidades de carga em superfícies não pavimentadas[12, 13, 16].

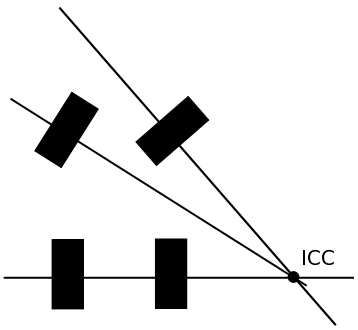

Figura 7: Posicionamento do ICC de um carro.

2.1.3 Propulsão com Esteiras

A propulsão com esteiras é semelhante à propulsão diferencial em termos de cinemática, porém, é mais robusta para variações de terreno[11] – embora sejam mais sensíveis a efeitos de refração, a mudanças no tipo de terreno. As duas rodas diferenciais são estendidas pelo uso de esteiras que proporcionam um maior contato com o solo[15, 16], se apoiando na capacidade de deslisamento para mudar de direção. De fato, quanto maior a área de contato das esteiras com o solo, maior a capacidade de sobrepor pequenas falhas do solo e até mesmo escalar pequenos obstáculos.

O fato dos veículos com esteiras fazerem uso do deslizamento entre suas esteiras e o solo para mudarem de direção, introduz sérias complicações na cinemática do veículo, especialmente no cálculo de sua posição (odometria). Geralmente tais tipos de veículos fazem uso de mecanismos externos ou passivos para determinar seu movimento, ao invés do uso exclusivo do movimento das esteiras. O uso de uma roda castor, ou uma roda omnidirecional passiva, pode ser utilizada para esta finalidade.

2.2 Robôs Terrestres com Pernas Articuladas

Mesmo com a grande variedade e sofisticação das propulsões que fazem uso de rodas e esteiras, ainda existem motivos e situações que justifiquem outras formas de contato com a superfície de deslocamento. Tais como:

Contato Permanente com o Solo – Embora o contato com o solo pareça uma premissa básica para um robô terrestre, existem casos em que essa premissa não é sempre adequada. No caso de terrenos acidentados[11], como os encontrados em florestas, desertos pedregosos, ou na exploraçao planetária, nem sempre é possível ou desejável garantir o contato permanente das rodas ao longo de todo o caminho a ser percorrido. Variações abruptas no terreno podem se mostrar intransponíveis até mesmo para veículos com esteiras. Mas, uma vez que partes discretas deste caminho se mostrem suficientes para o progresso do robô ao longo do ambiente, um robô articulado, apropriadamente projetado, pode se mostrar capaz de transpor estes obstáculos com relativa facilidade[17].

Ambientes Estruturados – Os ambientes estruturados, especialmente aqueles adequados aos seres humanos, apresentam uma topologia limitante para grande parte das soluções com rodas ou esteiras. Um veículo com rodas encontraria dificuldade para

subir uma escada, ao mesmo tempo que um veículo com esteiras poderia transpor este obstáculo com certa desenvoltura. Porém, existiriam restrições quanto a anatomia deste veículo (para evitar o tombamento, por exemplo), conseqüentemente restrições quanta a aplicação do mesmo. Um veículo melhor talhado para tal ambiente seria idealizado fazendo uso dos mesmos mecanismos usados para a estruturação deste ambiente *a priori*. No caso dos ambientes humanos, nada mais natural do que a escolha do uso de pernas articuladas para tais mecanismos.

2.2.1 Estabilidade de Veículos com Pernas

Se um robô com pernas articuladas é projetado de tal forma que o equilíbrio seja mantido por todo o tempo, mesmo que suas pernas permaçam imóveis em uma determinada posição, este robô é dito possuir *estabilidade estática*. Formalmente, a estabilidade estática é mantida enquanto o centro de gravidade do robô permancer dentro do polígono convexo definido pelas pernas em contato com o solo[12, 16, 24], vide Figura 8.

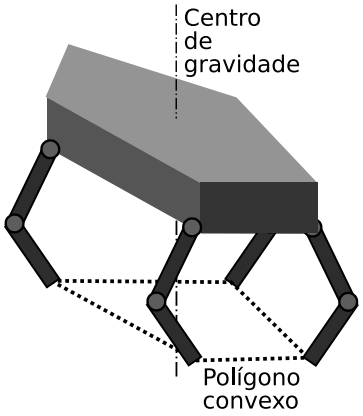

Figura 8: Exemplo de estabilidade estática.

Se é permitido ao centro de gravidade se mover para fora deste polígono convexo e o robô, mesmo assim age de forma controlada, o robô passa a exibir *estabilidade dinâmica*. Sob estabilidade dinâmica o robô mantém sua estabilidade fazendo o uso de movimentos controlados e de sua inércia[11].

2.2.2 Número de Pernas de um Veículo Articulado

Os robôs articulados têm sido construídos com uma grande variedade de número de pernas. Desde robôs com apenas uma perna, duas pernas, quatro pernas, seis ou oito pernas, sendo os três últimos casos mais comuns, devida à estabilidade estática[12, 16, 24].

Como pelo menos uma perna deve estar livre para se movimentar, de forma a mudar a posição do robô, para um robô exibir estabilidade estática ele deve possui pelo menos quatro pernas articuladas, sendo três sempre em contato com o solo, isto é, formando um polígono convexo, o mais simples deles, um triângulo. Já para o robô exibir estabilidade dinâmica, ele deve possuir pelo menos uma perna articulada.

2.2.3 Outros Fatores para o Projeto de Robôs com Pernas

Adicionalemente aos fatores de estabilidade e o número de pernas, outros fatores construtivos são relevantes para o projeto de robôs móveis fazendo o uso de pernas. O número de juntas determina o número de graus de liberdade desejado ou necessário para dada aplicação. O tipo destas juntas também é determinante para a flexibilidade do robô e suas articulações. Finalmente, a combinação do número e de diversos tipos de juntas determinam a flexibilidade de uma perna articulada.

Outro fator sutil é o passo usado para o movimento das pernas articuladas e conseqüentemente para o movimento do robô como um todo. Curiosamente, praticamente todo o conhecimento obtido quanto ao passo de robôs móveis articulados provém do estudo biológico. Eadweard Muybridge (1830-1904), fotógrafo norte-americano, é detentor do pioneirismo deste estudo. Muybridge, na tentativa de vencer uma aposta, provou que as quatros patas de um cavalo, em um determinado momento, não teriam contato com o solo. Para vencer a aposta, o fotógrafo posicionou centenas de câmeras ao longo do caminho a ser percorrido pelo cavalo. Cada câmera era disparada por uma corda, rompida pelo cavalo em seu movimento, e a partir de 1877 Muybridge passou a fotografar as primeiras seqüências de movimento de animais e humanos. Involuntariamente Muybridge provou que um cavalo exibe estabilidade dinâmica.

Da mesma forma que a cadência do passo possui uma importância fundamental para a velocidade e estabilidade do cavalo, a cadência do passo é importante também para um robô móvel com pernas.

2.3 Outras Formas de Interação para Robôs Móveis

Além da interação terrestre outras três formas de interação atraem o intresse da pesquisa em robótica móvel. Veículos aquáticos usam a água para propulsão. Duas abordagem são preferidas. A primeira faz uso de hélices propelentes para proporcionar o movimento a diante, enquanto superfícies móveis de controle são usadas para controlar a

direção do movimento (Figura 9). Tais dispositivos possuem uma estrutura semelhante a um torpedo. A segunda alternativa faz uso de jatos propulsores, onde o próprio fluxo de água ao redor do robô é manipulado a fim de proporcionar propulsão. A direção do robô pode ser controlada tanto pelo uso de superfícies móveis de controle quanto da manipulação da direção deste jato propulsor, ou até mesmo ambos. Um exemplo de veículo manualmente operado que faz uso deste tipo de propulsão aquática é o *jetski*.

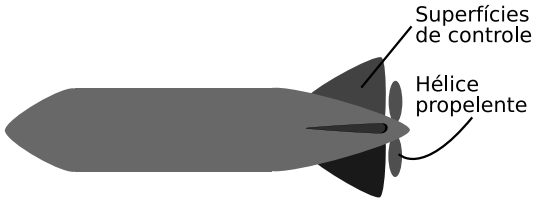

Figura 9: Exemplo de propulsão por hélices de um torpedo.

Entre as aplicações realizadas por esses robôs, podem-se citar[11]:

- investigação visual de partes submersas de navios, pontes e plataformas;
- monitoramento de colônias de peixes;
- controle de poluição subaquática;
- localização de depósitos de combustíveis nucleares;
- execução de operações tecnológicas em plataformas marítmicas;
- inspeção visual de estruturas subaquáticas de oleodutos e gasodutos;
- inspeção de barragens;
- explorações marítmias para fins de pesquisa.

Os robôs aéreos passaram dos pilotos automáticos das aeronaves comerciais a veículos completamente autônomos. Da mesma forma que os veículos aquáticos, os veículos aéreos fazem uso do controle do fluxo do fluído circundante, no caso o ar, para proporcionar propulsão, sustentação e controle de direção. As estratégias dos robôs aéreos podem ser classificadas em três tipos principais. A primeira, com asas fixas, faz o uso de superfícies de controle móveis para manipular a direção do robô, a forma de propulsão pode ser tanto a de hélices ou a de turbojatos (Figura 10). Exemplos de tal estratégia são os aviões convencionais. A segunda estratégia, com asas móveis, a mesma usada em helicópteros, faz o

uso de de hélices com inclinação controlável para variar a direção do robô. Adicionalmente uma ou mais hélices se fazem necessárias para controlar a tendência de rotação descontrolada do robô (devido ao momento de inércia originado pelo motor principal) (Figura 11). A terceira e última estratégia faz uso das propriedades de densidade relativa entre dois fluídos. São os balões e *zeppelins*, que no uso de combinções de gases – ou propriedades de gases – proporciona a flutuação atmosférica do veículo (Figura 12). Este último caso demanda uma forma adicional de propulsão para o controle total de direção e movimento, além da sustentação.

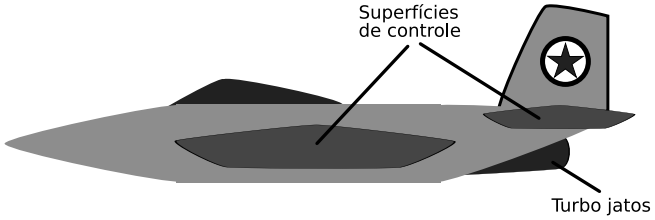

Figura 10: Exemplo de avião a jato.

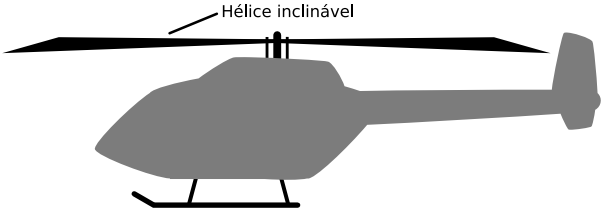

Figura 11: Exemplo de helicóptero.

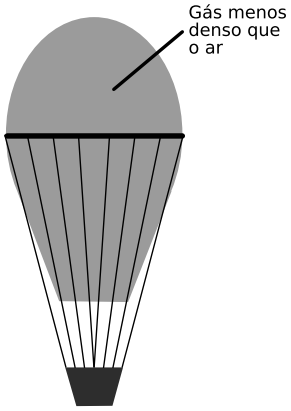

Figura 12: Exemplo de balão.

Entre as aplicações realizadas por esses robôs, podem-se citar[11]:

- sobrevôo de inspeção de florestas, estradas, fazendas, cidades e cursos de rios;

- controle de incêndios;
- detecção de áreas minadas e armadilhas militares;
- proteção de tropas militares;
- exploração de áreas contaminadas por ameaças químicas, biológicas ou nucleares.

Os robôs espaciais habitam o limite exterior físico conhecido pelos seres humanos. São idealizados para a assistência na construção, reparo e manutenção de estações espaciais e satélites. Devido à micro-gravidade, à pequena resistência ao movimento e à inexistência de qualquer corpo ou fluído para o aproveitamento fácil das leis de ação e reação, os robôs espaciais confiam basicamente a jatos propulsores e à sua própria inércia para o movimento e controle de direção.

2.4 Conclusões

Existem diversas formas de locomoção empregadas na robótica móvel. O principal interesse neste trabalho reside na locomoção terrestre, devido à sua fácil implementação, em consideração às outras formas. Fica evidente também, que a forma de locomoção, e seus conseqüentes parâmetros, representam dificuldades consideráveis por si só no estudo da robótica móvel. Dificuldades estas que se somam às de outras tarefas maiores, como o planejamento de trajetórias, o desvio de obstáculos, ou a realização de certa tarefa.

Desta forma, deve-se minimizar o impacto da solução de locomoção empregada, aliando suas características às características do ambiente de operação e às características das tarefas a serem realizadas. Deve-se ter em mente, desde o início do projeto, a sinergia entre a forma de locomoção, o ambiente e as tarefas.

3 Sensores

A percepção do ambiente é um requisito chave para o mais simples comportamento móvel[11]. Os sensores fundamentalmente associados com a mobilidade são aqueles capazes de mensurar a distância percorrida pelas rodas em relação ao solo, sensores que medem alterações inerciais, e aqueles que percebem estruturas externas no ambiente[11]. Para que um robô saiba onde ele está ou como ele chegou lá, ou ainda ser capaz de determinar onde ele esteve, sensores e algoritmos de sensoreamento se fazem necessários[12, 18].

Os sensores e os algoritmos envolvidos na interpretação destes parâmetros podem ser altamente complexos. Mesmo com a existência de uma grande variedade e complexidade de sensores, duas classes distintas de sensores se estabeleceram para a robótica móvel[12]:

Sensores Visuais: sensores que fazem uso da luz refletida por objetos no ambiente para determinar sua estrutura.

Sensores Não-visuais: sensores que fazem uso de propriedades como acústica, inércia, e outras manifestações de energia para o sensoreamento do ambiente.

Provavelmente os conceitos mais básicos associados à robótica móvel são os sensores de *estado interno* e sensores de *estado externo*. O primeiro tipo de sensores provê dados dos parâmetros internos do robô, tais como o nível de carga da bateria, a posição das rodas, ou os ângulos das juntas de uma articulação. Os sensores externos lidam com a observação de aspectos externos ao robô propriamente dito, parâmetros como umidade, cor de um objeto, contato com um obstáculo e assim por diante. Os sensores externos que apontam o contanto físico com objetos externos também são chamados de *sensores de contato*, todas as demais modalidades de sensores, que não envolvam o contato físico do robô, são denominados de *sensores de não-contato*.

Outra classificação relevante dos sensores é:

Sensores Ativos: sensores que agem pela emissão de energia ao ambiente ou modificando este ambiente para obter suas leituras. Por exemplo, um sonar ou ainda o

tato, no caso dos seres humanos, são sensores ativos.

Sensores Passivos: sensores que obtém suas leituras apenas pela recepção da energia do ambiente, sem qualquer emissão ou alteração do ambiente. Um microfone ou a visão humana são exemplos de sensores passivos.

O sensoreamento passivo é, muitas vezes, preferido em razão de sua mínima alteração do ambiente, além de possuir um consumo de energia menor em relação aos sensores ativos. O sensoreamento passivo também é preferido quando o robô deve ser discreto, por exemplo, em casos militares ou de vigilância, ou ainda quando múltiplos robôs ocupam o mesmo ambiente, a fim de reduzir ao máximo a interferência. O sensoreamento ativo, na medida que é menos econômico em termos de energia, tende a ser mais robusto quanto às interferências provenientes do ambiente e em alguns casos, interferências internas.

As seguintes observações podem ser feitas no que se refere a sensores no contexto da robótica móvel:

- sensores reais estão sempre sujeitos a interferências;
- sensores reais sempre retornam uma descrição incompleta do ambiente;
- sensores reais nem sempre podem ser modelados completamente.

O uso de sensores para inferências sobre o ambiente é geralmente descrito como um problema de recuperação e reconstrução do ambiente. Uma variada gama de tecnologias de sensores se encontra hoje disponível. E toda esta variedade pode ser classificada em três classes principais, quanto a robótica móvel:

Sensores de posição absoluta: retornam a posição do robô (ou qualquer outro elemento do vetor posição) em termos absolutos, por exemplo latitude e longitude, ou distância de um obstáculo ou até mesmo de sua posição objetivo.

Sensores ambientais: retornam propriedades do ambiente. Propriedades como temperatura, umidade ou outros parâmetros como a cor de um obstáculo a frente do robô.

Sensores inerciais: retornam propriedades diferenciais da posição do robô, como acelerações.

Outras caracterísitcas também devem ser consideradas para classificar as diferentes tecnologias de sensores, alguns de cunho pragmático, por exemplo[12, 18]:

Velocidade de operação: referente à taxa de medições que o sensor pode retornar quando em uso contínuo, ou em outras palavras, o intervalo de tempo entre amostragens subseqüentes.

Custo: um par de emissor-detector infra-vermelho pode custar pelo menos cem vezes menos do que um giroscópio de alta precisão.

Taxa de erros: estão incluídos o erro médio, o número de erros fora da linha padrão e o número de amostragens perdidas.

Robustez: até que ponto o sensor tolera alterações no ambiente a partir de suas condições ideais de operação. Fatores como ruídos ambientais e elétricos também são relevantes.

Requisitos Computacionais: uma simples chave de contato (*switch*) não requer qualquer recurso computacional, enquanto conversores analógico-digital (AD) ou algoritmos de visão podem exigir recursos computacionais específicos para operarem a uma velocidade aceitável.

Potência, peso e dimensões: alguns sistemas requerem potência contínua para operarem de forma completamente funcional, enquanto outros sensores podem ser desligados até quando se mostrarem necessários. As dimensões e o peso dos sensores também se tornam relevantes especialmente na robótica móvel, quando se dispõem de recursos limitados como espaço e força dos motores, por exemplo.

3.1 Tipos de Sensores

Além das características genéricas a todos os sensores, algumas caracterísitcas peculiares determinam sua forma de funcionamento e sua aplicação principal. De certa forma, são estas características que determinam o uso final de um sensor.

Um sensor pode ser modelado como o relacionamento entre uma propriedade física e de interesse no ambiente e uma leitura de sensor r, isto é, $r = f(e)$ (Figura 13). O simples fato de transformar ou relacionar uma propriedade física (e), manifestada sob qualquer forma energia, em outra propriedade ou manifestação física (r) oficialmente denomina os sensores como *transdutores*. A princípio, o modelo do sensor também deve incluir um

modelo para o ruído interno ao próprio sensor bem como o ruído proveniente de fontes externas. Porém, isto nem sempre é feito, com a finalidade de simplificar a modelagem matemática e computacional.

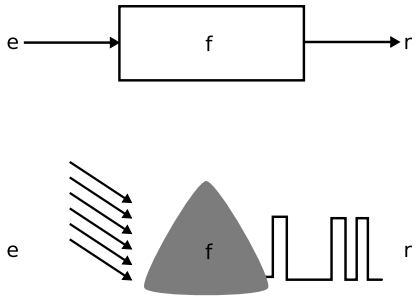

Figura 13: Sensor como elemento de transformação de formas de energia.

Na robótica móvel – e em todos os campos de aplicação que usam a eletricidade, eletrônica e computação – o tipo da propriedade r, da leitura do sensor, tende a ser o mesmo: sinais elétricos analógicos ou digitais. O que passa a variar são os fatores e, obviamente o tipo de propriedade física a ser mensurada, e o parâmetro $f(\bullet)$, que determina como esta propriedade e é transformada na leitura r. São estas variações em $f(\bullet)$ e e que determinam a finalidade e o tipo do sensor.

3.1.1 Sensores de Contato

Os *Sensores táteis* são aqueles usados para criar a "noção de toque"[11]. Estes são praticamente sensores de contato, com as exceções daqueles que assumem o conceito de toque pela passagem de perto, como os sensores capacitivos.

Sensores de contato são geralmente empregados nos pára-choques dos robôs móveis[12, 17]. Os dispositivos mais simples são os pequenos interruptores (*microswitches*) que retornam uma leitura binária: aberto ou fechado[14] (Figura 14). Sensores de contato também são utilizados como indicadores de final de curso de atuadores e manipuladores[11].

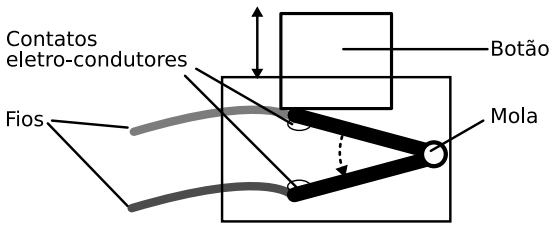

Figura 14: Exemplo de uma chave, que pode ser usada como sensor de contato.

Sensores mais sofisticados conseguem retornar uma leitura proporcional à força aplicada sobre eles. Geralmente, fazem uso de molas acopladas a potenciômetros e dispositivos baseados em materiais que variam suas características capacitiva ou resistiva em resposta à compressão.

Como as informações de contato são tratadas varia em função da arquitetura do robô. Geralmente os sensores de contato são os últimos recursos para se evitar danos severos, mesmo quando os sensores são realmente utilizados já não há mais possibilidade para evitar a colisão. De qualquer forma, eles podem ser tratados como sensores ou ser conectados diretamente a um sistema de controle de baixo nível, como os nervos constituíntes do arco reflexo nos seres humanos.

Mesmo que o objetivo dos sensores de contato seja evitar a colisão, se a velocidade do impacto for alta o suficiente, a mera detecção da colisão pode não ser suficiente para evitar qualquer dano. Por isso, é recomendada a associação de mais de um tipo de sensor para a detecção e o desvio de obstáculos.

3.1.2 Sensores Internos

Sensores inerciais pertencem à classe de sensores internos que fazem leituras derivativas das variáveis de posição de um robô[11]. Especificamente, o termo sensores inerciais é usado para referir acelerômetros e giroscópios, que medem as derivadas segunda da posição, a aceleração e a aceleração angular do veículo[11].

3.1.2.1 Acelerômetros

Os *acelerômetros* são essencialmente massas associadas a molas cuja posição variante por meio de aceleração pode ser mensurada (Figura 15). Usando a segunda lei de Newton, $F = ma$, e a relação massa-mola, $F = kx^2$, isolando a obtém-se

$$a = \frac{kx^2}{m} \tag{3.1}$$

onde a é a aceleração, m é a massa, e k é a constante de elasticidade da mola. Na prática, alternativas ao conceito massa-mola são utilizados. Cada acelerômetro pode medir aceleração em apenas uma direção, pela disposição de três dispositivos ortogonais um ao outro, um sensor de aceleração omnidirecional pode ser construído.

Acelerômetros são vistos como sensores absolutos, não relativos, uma vez que não

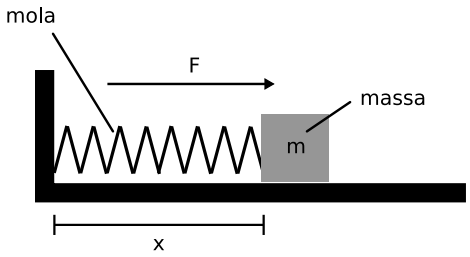

Figura 15: Diagrama massa-mola, que constituí um acelerômetro simples.

fazem qualquer referência ao mundo externo[12]. Embora esta consideração seja quase correta, para a determinação precisa da aceleração é necessário o conhecimento do vetor gravidade, tanto em termos de direção e amplitude, de forma que ele possa ser fatorado das leituras de aceleração. Isto é, os acelerômetros são sensíveis a "forças ruído". Os acelerômetros podem ser considerados sensores versáteis e robustos, uma vez que podem ser utilizados em qualquer ponto da Terra, até mesmo fora do campo gravitacional, no espaço.

3.1.2.2 Giroscópios

Assim como os acelerômetros, os *giroscópios* medem a aceleração angular fazendo uso da mecânica newtoniana, a conservação do momento angular[13]. Uma forma simples de giroscópio é o modelo mecânico, uma massa suspensa girando rapidamente. Como o momento angular é conservado, qualquer alteração na orientação do giroscópio resulta em uma força que, se não impedida, leva à precessão. Pela medição desta força é possível determinar a alteração de orientação. Desta forma, os giroscópios são usados para a medição de orientação diferencial ou relativa[12, 17].

3.1.2.3 Compassos e Clinômetros

Um *clinômetro* é um dispositivo simples que indica a orientação do vetor da força de gravidade (Figura 16). Os clinômetros digitais são baseados em chaves (*switches*) de mercúrio ou chaves de inclinação eletrolíticas[13, 17].

Embora pareça um parâmetro trivial, a importância de integrar um clinômetro em um robô móvel não pode ser subestimada[13]. Um robô que dispensa qualquer indicação de inclinação não pode determinar seu estado de equilíbrio e não pode responder a rampas ou depressões no ambiente, o que pode levar a tombamentos ou outros acidentes (Figura 17).

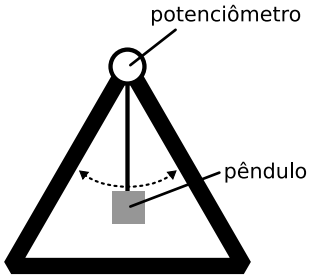

Figura 16: Exemplo de um clinômetro rudimentar, em que um potenciômetro pode ser usado para a medida de inclinação.

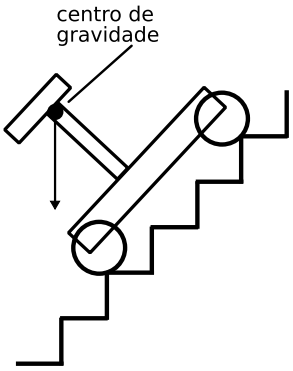

Figura 17: Exemplo de quando um clinômetro é importante, como quando o robô desce uma escada.

Enquanto clinômetros medem o desvio em relação ao vetor gravitacional, *compassos ou bússolas* medem a orientação em relação ao campo magnético terrestre[13, 17]. Compassos magnéticos usam o campo magnético da Terra para orientar o compasso pela rotação de um magneto em relação ao plano horizontal, alinhando este magneto com o campo magnético terrestre[12, 17, 18] (Figura 18). Compassos digitais de fluxo de porta, *flux-gate compass*, usam um magneto toroidal suspenso, sujeito ao campo magnético da Terra[18].

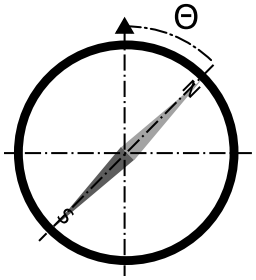

Figura 18: Bússola usada para a indicação da direção de um robô, por exemplo.

Os sensores que usam o campo magnético local para determinar o pólo norte estão suscetíveis a variações locais do campo magnético do ambiente. Grandes estruturas metá-

licas, como em alguns casos o próprio robô, motores elétricos, têm um efeito considerável nos campos magnéticos locais. É essencial calibrar estes sensores, levando em consideração o campo magnético do veículo e do ambiente e levar em consideração que desvios não previstos podem ocorrer.

3.1.3 Sensores Infra-vermelho

Sensores de proximidade *infra-vermelho* são uns dos sensores de proximidade mais rápidos e baratos[11, 12, 14, 17]. O conceito básico se restringe à emissão de um pulso infra-vermelho e à detecção da sua reflecção[11] (Figura 19). A distância ao obstáculo é estimada usando a força do sinal refletido recuperado e considerações de reflexividade das superfícies do ambiente. Para evitar interferências causadas por outras fontes infra-vermelho do ambiente, o sinal emitido é codificado, e qualquer outro sinal detectado é ignorado até que uma forma de onda reconhecida seja lida. Sensores simples, sem pulsos codificados, estão sujeitos a interferência, como da luz do Sol ou de lâmpadas fluorescentes. Mesmo um sinal robusto pode ser completamente dissipado em ambientes fortemente iluminados[13, 15, 16].

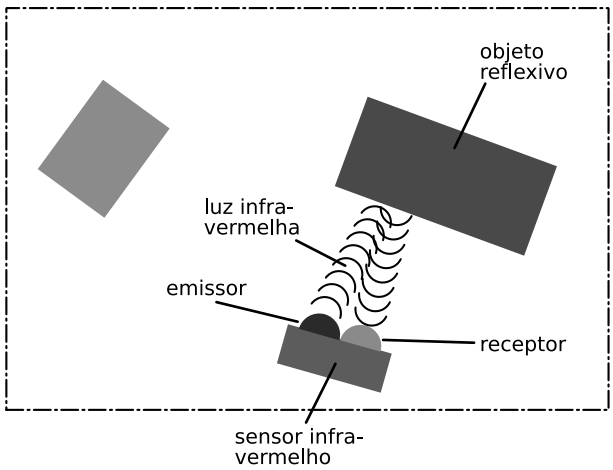

Figura 19: Exemplo de emissão e reflexão de um sinal infra-vermelho.

O processo de inferir precisamente a informação de distância de um sensor infra-vermelho não é trivial, por que é impossível estimar *a priori* as propriedades reflectivas da superfície, então a intensidade do sinal retornado é uma função da distância e das propriedades refletivas da superfície[14]. Por outro lado, o mecanismo é barato e compacto, e é relativamente ortogonal a outros mecanismos de sensoreamento que podem ser empregados no robô. Sensores de infra-vermelho são utilizados como sensores de não-contato para detectar obstáculos e medir distâncias[12, 15, 16].

3.1.4 Sonares

O *sonar* (*sound navigation and ranging*) usa sinais acústicos, sonoros, para medir distâncias[18]. Os sonares são tecnologias de sensoreamento ativa, que usam um sinal sonoro, ou pulso sonoro emitido, e sua reflexão recebida. O tempo de viagem, a variação de fase, e a atenuação do sinal[1] como uma função da freqüência são os aspectos do sinal refletido que são explorados por diferentes tipos de sonares. Os sonares têm uma longa história na localização de obstáculos e alvos. Já em 1918, ondas acústicas de alta freqüência foram usadas para determinar a posição, velocidade e orientação de objetos submersos. A grande maioria dos sonares terrestres faz uso de sinais no intervalo ultra-sônico, com uma freqüência alta demais para ser sentida pela audição humana.

Unidades de sonar são instaladas em robôs móveis na combinação de emissor-receptor em posições fixas do veículo. Uma das estratégias é dispor vários sonares a um intervalo angular regular da circunferência do robô. Uma estratégia alternativa é montar o par emisso-rececptor em uma plataforma rotativa, que podem ser direcionadas mecanicamente.

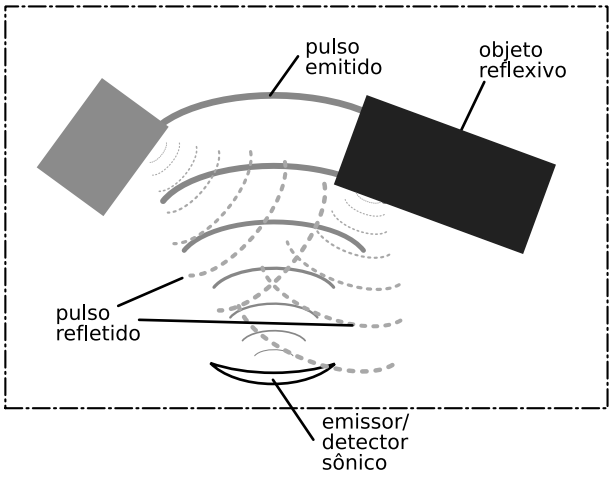

Figura 20: Exemplo de emissão e reflexão de um sinal ultra-sônico.

Os sonares usados em robótica móvel podem fazer uso de estratégias ainda mais simples. Como o uso uso preemptivo do transdutor sônico para a emissão e a recepção de um curto pulso e para medir o tempo até a recepção do eco (Figura 20).

A freqüência acústica empregada em sensores sonares comerciais é a ultrasônica – no intervalo de 40-50kHz[15, 16, 17]. Elevadas freqüências são atenuadas mais rapidamente, mas provêm uma melhor resolução espacial. O sinal emitido possui um perfil de amplitude

[1] Analogamente ao Efeito Doppler.

complexo, como uma função do ângulo a partir da normal do transdutor. O feixe sonar possui basicamente uma forma cônica tridimensional, com uma distribuição não-uniforme de energia dentro deste cone. Uma implicação direta destas características é que o eco retornado pode ser a reflexão de uma parte da onda que deixa o transdutar sob um ângulo oblíquo. Assim, um sinal de eco retornado pode não necessariamente resultar em uma localização precisa do objeto que refletiu o sinal usando apenas métodos que consideram o tempo de reflexão.

O maior impedimento para a inferência da localização de uma superfícia acústico-reflexiva é que, a freqüências ultrasônicas, objetos comuns se tornam refletores especulares [14]; isto é, uma onda acústica oblíqua será refletida de um emissor ao invés de refletida de volta para ele. Se um eco retorna ao robô, não há certeza em se afirmar que este eco não seja resultado de uma complexa série de reflexões ao longo do ambiente do que a simples reflexão do objeto mais próximo na direção apontada pelo transdutor. Além do mais, objetos com uma seção transversal menor que o comprimento da onda acústica usada vão retornar uma mínima quantidade de energia, e vão se comportar praticamente como "invisíveis" acusticamente[12] (indetectáveis).

Enquanto a reflexão especular e as dificuldades para localizar a fonte refletiva são consideradas rúidos, elas são de alguma forma características repetitivas do sensor e podem ser modeladas com certa facilidade. Considerados todos estes aspectos idiossincráticos do sensoreamento acústivo, complexos modelos do transdutor e da interpretação dos dados podem ser elaborados para o uso aceitável deste tipo de sensor.

Alguns dos trabalhos de Borenstein e Koren[25, 26, 27, 28] são notórios nas problemáticas do uso de pulsos ultrasônicos na robótica móvel, especialmente na detecção e desvio de obstáculos.

3.1.5 Radares

O *radar* (*radio detecting and ranging*) opera sob os mesmos princípios dos sonares. Ondas de rádio de alta freqüência são emitidas e suas reflexões são analisadas para se obter medições de distância e outas propriedades[18]. Até recentemente esta tecnologia se mostrou muito custosa e desajeitada para robôs móveis, mas avanços tecnológicos já permitem o uso de tais dispositivos em grandes veículos móveis. Os radares ganham sua audiência na robótica móvel pelo fato de sua rapidez nas medidas e na capacidade de prover informação sobre as propriedades do espaço e sua geometria. O radar pode ser usado para determinar o tipo do solo, ou ainda pode penetrar a superfície de objetos,

revelando até certo grau sua estrutura interna. Como faz uso de sinais de rádio, ao invés de ondas acústicas, pode ser usado em ambientes sem atmosfera, como a superfície de outros planetas[12].

Os robôs móveis se baseiam na detecção de freqüência e fase[2] para medir a distância até o alvo. Micro-ondas e milimetro-ondas têm caracterísitcas operacionais adequadas ao uso em robôtica móvel. Da mesma forma que os sonares, os radares sofrem dos mesmos problemas de especularidade e à própria interferência do sinal de rádio.

3.1.6 Lasers

Uma das tarefas chave do sensoreamento é obter estimativas de distância em relação a objetos no ambiente. Uma das tecnologias proeminentes no sensoreamento são os medidores laser[12, 18]. Os medidores são operados sob os seguintes métodos[18]:

Triangulação: uso das relações geométricas entre o feixe luminoso emitido, o raio refletido, e sua posição no plano.

Tempo de vôo: a medida do tempo para um feixe luminoso emitido atingir um alvo cuja distância é medida e retornada.

Fase: uso da diferença de fase do feixe de luz emitido e o refletido.

O princípio de triangulação faz uso de relações geométricas para medir a distância até um determinado alvo. Os métodos de tempo de vôo e fase, de outra forma, exploram o atraso da propagação do sinal. O uso de um feixe curtamente colimado permite uma maior resolução e maior eficiência de potência. Isto é possível com lasers em particular porque eles são fontes coerentes que podem produzir pulsos curtos que permanecem colimados ao longo de grandes distâncias. A tecnologia de sensoreamento se baseia na medição do atraso ou na diferença de fase entre o sinal laser emitido e o refletido.

Os sistemas de baixa potência possuem uma faixa operacional de alguns metros, enquanto os de alta potência podem operar a distâncias de quilômetros. É a mesma tecnologia usada para medir a distância da Terra à Lua com elevada precisão. Como os raios laser divergem com o aumento da distância, a localização da distância recuperada se torna cada vez mais imprecisa com alvos mais distantes.

[2] Analogamente ao Efeito Doppler

Os sistemas lasers são dispostos sobre uma plataforma rotacional ou como uma montagem de espelhos que permitem o direcionamento do feixe luminoso a diferentes partes do ambiente. Em ambientes fechados um ponto de consideração de um sistema de sensoreamento laser é o risco de danos aos olhos dos observadores. Para tanto, os sistemas em tais ambientes devem fazer uso de saídas de baixa potência. Outro ponto relevante é a reflexibilidade do feixe luminoso em certas superfícies, como espelhos, poças de líquidos, o que pode resultar em desvios do sinal e sinais enganosos.

3.1.7 Posicionamento Baseado em Satélites - GPS

Em 1973 o Departamento de Defesa dos Estados Unidos iniciou um programa para o desenvolvimento de um sistema de estimativa de posição baseado em sinais transmitidos a partir de satélites orbitando a Terra. Em 1995, o sistema foi considerado operacional[12].

O sistema desenvolvido pelos Estados Unidos é conhecido como Sistema de Posicionamento Global, do inglês *Global Positioning System - GPS*. É baseado em 21 satélites artificiais orbitando a Terra (Figura 21). O sistema permite a um receptor apropriadamente equipado determinar sua posição absoluta, velocidade e tempo com precisão elevada, em qualquer ponto da Terra. Embora o objetivo principal tenha sido o bélico, o sistema GPS é amplamente difundido e utilizado em aplicações civis.

O sistema opera basicamente pela medição da diferença do tempo de deslocamento de um sinal de rádio que é originado de uma combinação de satélites. Como estes satélites se encontram em esferas orbitais distintas, estão espalhados em seis planos esféricos, é possível obter estimativa de latitude, longitude, e elevação de qualquer ponto da superfície da Terra[3]. Para a obtenção da estimativa da posição é necessária a "visibilidade" de pelo menos quatro satélites, e esta informação pode ser obtida a uma freqüência média de 2Hz.

Uma das grandes vantagens do sistema GPS é que ele não necessita de nenhuma outra observação do ambiente externo. Ele também é um sistema intrínsecamente passivo, não envolve qualquer emissão de energia.

Infelizmente o sistema GPS não apresenta um perfeito funcionamento em grande parte dos ambientes de interesse da robótica móvel. O GPS não pode ser usado em ambientes onde ondas de rádio não possam ser recebidas diretamente da linha de visibilidade dos satélites (no mínimo quatro), ou onde a velocidade da luz é diferente daquela do ar. Isto é, o sistema GPS não é funcional em ambientes fechados, subaquáticos ou subterrâneos[3].

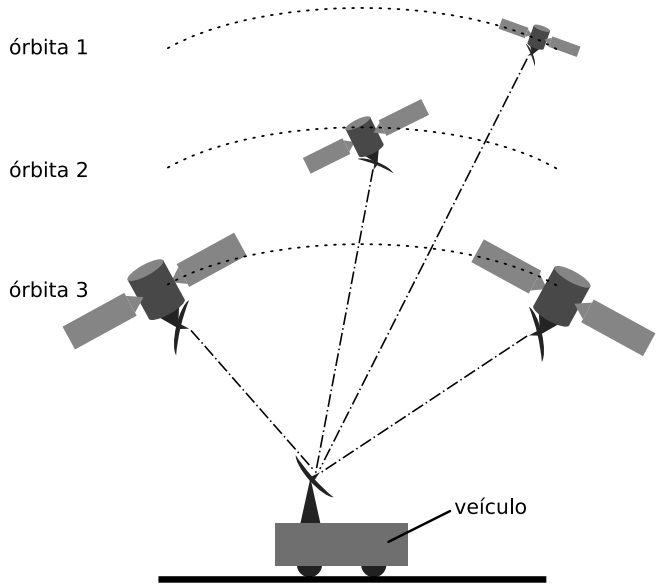

Figura 21: Esquemático da triangulação entre satélites no sistema GPS.

3.2 Conclusões

Existe uma grande variedade de sensores aplicados na robótica móvel. A maioria deles teve sua origem a partir de outras aplicações, desde a eletrônica convencional à automação industrial. Outros, tiveram sua função primordial nos campos de batalha. Independente de suas aplicações iniciais, cada tipo de sensor pode dar aos robôs móveis capacidades que vão muito além dos nossos cinco sentidos.

É este potencial de percepção estendida do mundo, além de sua interação com o próprio mundo, que tornam a robótica, em todos os seus campos, uma tecnologia tão promissora, como podem atestar diversas aplicações contemporâneas.

Entretanto, analogamente às formas de locomoção, os sensores acabam embutindo complexidade no projeto e modelagem de robôs móveis. Da mesma forma, os sensores devem ser elementos catalizadores da tarefa principal do robô. Devem ser portanto considerados em função de seu funcionamento e limitações, como qualquer outro elemento restritivo. Para uma referência completa, recomenda-se a leitura de [18, 29, 30, 31].

4 *Representação e Interpretação do Espaço*

Embora seja possível realizar grande número de tarefas complexas sem uma representação interna do ambiente, muitas tarefas requerem tal representação. A representação provavelmente mais difundida na robótica móvel é o uso de mapas. De fato, este é o metodo mais usual para nós seres humanos. Adicionalmente à representação de localidades de um ambiente, um mapa para robótica móvel pode incluir outras informações, tais como a refletância dos objetos, regiões perigosas ou de travessia difícil, ou ainda informações de experiências anteriores. Uma representação interna do espaço pode ser usada pelo robô para planejar e pré-executar as tarefas que serão executadas posteriormente.

Uma representação interna do ambiente atende às seguintes finalidades:

- estabelecer quais partes do ambiente são livres para navegação. Estas regiões são chamadas de *espaços livres*;
- reconhecer regiões ou localidades do ambiente;
- reconhecer objetos específicos dentro do ambiente.

Existem basicamente duas formas de se obter os elementos para a representação do ambiente. A primeira é através de uma representação fornecida *a priori*, usualmente gerada pelo conhecimento humano. A segunda, e teoricamente a mais intuitiva, seria permitir ao ambiente representar a si mesmo, não construindo uma representação interna de todo o ambiente. Uma dificuldade desta segunda alternativa é que, como não existe uma representação interna de todo o ambiente (em algumas aplicações não há nenhuma representação), todo o planejamento e manipulação do ambiente devem ser feitas no ambiente real, em tempo real. De acordo com esta estratégia, os planejamentos só podem ser baseados em leituras instantâneas dos sensores. O planejamento a longo prazo é

impreciso, o *planejamento reativo* se mostra uma estratégia atraente para a interações em tempo-real e planejamento de baixo nível.

Se uma representação interna é desejada, é preciso definir quais primitivas[1] irão compor a construção dessa representação. Representações baseadas em objetos, características ou entidades simbólicas, ocupação espacial, localidades, rotas, e outras informações têm sido propostas. Em geral, as representações espaciais podem ser divididas em duas categorias principais: aquelas que se baseiam em uma representação métrica e aquelas em representações topológicas.

4.1 Decomposição Espacial

A *decomposição espacial* se baseia na amostragem discretizada do ambiente, de duas ou mais dimensões, a ser representado. A idéia principal consiste em representar o espaço livre por si mesmo[32], ao invés de representar os objetos individualmente. Parte do pressuposto da capacidade de identificação ou discriminação dos objetos individuais. Esta amostragem pode ser realizada de várias maneiras usando diferentes métodos de subdivisão, seja baseado nas formas dos objetos, ou pela definição de uma forma padrão de amostragem, amostrando (ou subdividindo) o ambiente recursivamente.

Uma vez decomposto o espaço, pode-se construir um grafo não-dirigido, chamado de *grafo de visibilidade*, onde os vértices são as células e as arestas a representação da adjacência entre elas[3]. De posse das células inicial e final, é realizada a busca de um caminho que as relacionem indiretamente, utilizando algum método de busca neste *grafo de conectividade*. O caminho real pode ser determinado pelas propriedades geométricas de cada célula, por exemplo, passando pelo *centro de gravidade* de cada nó.

4.1.1 Decomposição Uniforme

A forma mais simples de decomposição espacial é a amostragem espacial fazendo o uso de células convexas de tamanho uniforme sem qualquer recursão[3] (Figura 22). Amostras tomadas deste procedimento expressam o grau de ocupação de cada ponto amostrado: *espaço vazio*, *ocupado* ou *parcialmente ocupado*. Se as amostras forem binárias, então de uma amostragem bidimensional extrai-se um *mapa do bits*, do inglês *bitmaps*. Outras denominações são *mapa de pixels*[2] (*pixel maps*) ou *matrizes de ocupação* (*occupancy grids*).

[1] Elementos constituíntes e básicos à representação.

[2] *pixel* é uma contração do termo em inglês *picture element*, ou elemento de figura.

Em três dimensões os elementos de amostragem são chamados de *voxels*[3].

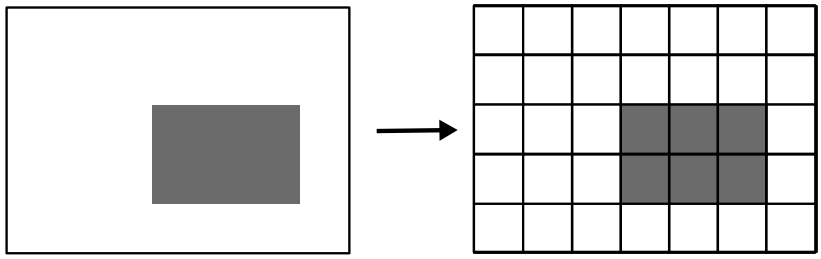

Figura 22: Exemplo de decomposição espacial uniforme.

Uma das vantagens da representação por decomposição uniforme é sua generalidade, uma vez que nenhuma pressuposição é necessária quanto às formas dos objetos individuais presentes no ambiente. Em contrapartida, as principais desvantagens deste tipo de representação são que a resolução da matriz, ou sua fidelidade, é limitada pelo tamanho padrão[3, 32] de célula, e que o espaço de memória computacional utilizada é indiferente mesmo que o ambiente esteja relativamente pouco ocupado. Por exemplo, para um ambiente com $15 \times 15 \times 15m^3$, amostrado segundo uma precisão de 3 metros, serão necessárias 125 células, enquanto que para um volume de $100 \times 100 \times 100m^3$, com uma precisão de $1cm$, 10^9 células são necessárias, independente da ocupação do espaço. A representação baseada em *voxels* pode ser mostrar inviável computacionalmente para grandes volumes de espaço.

Adicionalmente à representação do nível de ocupação de localidades no espaço, aos elementos da matriz podem ser associados outros atributos do ambiente ou da representação, como o grau de confiança no valor de ocupação, questões de segurança, informações do terreno, por exemplo.

4.1.2 Decomposições Hierárquicas

Dada a simplicidade da representação por decomposição uniforme, mas levando em conta os excessivos recursos computacionais necessários para armazenar cada célula, pode-se tomar proveito do fato de que muitas células, especialmente aquelas correspondentes a porções do espaço adjacentes, apresentam estados semelhantes. Fazendo uso desta heurística, duas estratégias se destacam, a primeira é a representação do espaço fazendo uso de células com formas não uniformes em tamanho o forma, a segunda é uma representação hierarquica do espaço, chamada de *quadtree*[3].

[3] *voxel* é uma contração do termo em inglês *volume element*, ou elemento de volume.

A *quadtree*[4] é uma estrutura de dados recursiva[3], para a representação de uma região bidimensional. É uma representação hierárquica que pode potencialmente reduzir o espaço de armazenamento computacional e facilitar em certos tipos de computação[32]. O método é iniciado com uma grande região quadrada que engloba todo o espaço necessário. As células que não estão nem totalmente ocupadas ou totalmente livres são subdivididas em quatro subpartes iguais. As subpartes são divididas recursivamente até que elas estejam uniformemente livres ou ocupadas ou até que um limite de resolução pré-determinado seja alcançado (Figura 23).

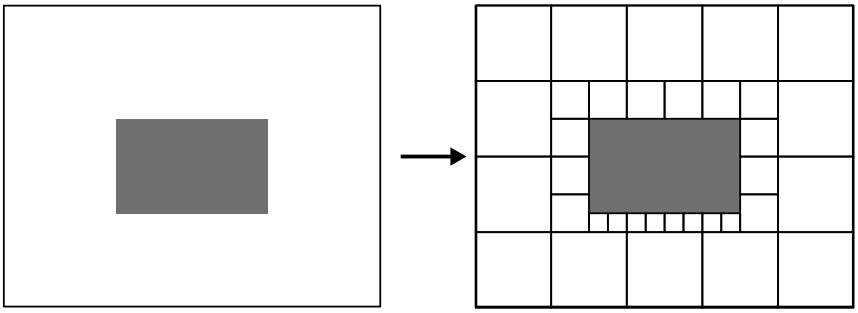

Figura 23: Exemplo de decomposição espacial *quadtree*.

O análogo tridimensional (3D) da *quadtree* é a *octree*[5]. Neste caso, ao invés de um quadrado, inicia-se com um cubo, recursivamente dividido até se atingir a homogeneidade de ocupação ou um limite de resolução.

No pior dos casos, ocorre a completa subdivisão em minúsculas células, quando a representação em *quadtrees* é pior do que a subdivisão uniforme, adicionado o peso computacional na representação da *quadtree*. Em geral o número de células varia em função da área ou a superfície dos obstáculos sendo descritos. Assim, para ambientes onde a maior parte do espaço está ocupada ou livre, as representações por *quadtree* se mostram adequadas.

Sistemas de representação hierárquica baseados em uma decomposição de potência de dois são populares em parte pela natureza binária do processo de busca. Infelizmente nem sempre o espaço pode ser caracterizado apropriadamente pela representação de potência de dois. Imagine um obstáculo triangular. Diferentes representações da mesma forma podem ser obtidas. Na busca da solução deste problema, duas soluções foram desenvolvidas para contornar esta restrição na representação por *quadtrees*. As árvores de *particionamento binário do espaço* (*binary space partitioning trees - BSP trees*) e o *método de decomposição*

[4]Em 2D existem $2^2 = 4$ sub-células, daí o nome "quad"tree.

[5]Em 3D existem $2^3 = 8$ sub-células, daí o nome "oct"tree. Versões N-dimensionais são baseadas em hipercubos subdivididos em 2^N subpartes.

exata.

Uma árvore BSP é uma estrutura de representação hierárquica comumente usada em computação gráfica que tem sido explorada com cada vez mais freqüência na representação do espaço – esta técnica é um dos fundamentos dos jogos eletrônicos tridimensionais, por exemplo. É uma representação hierárquica na medida em que uma célula retangular é tanto uma folha ou é dividida em duas novas árvores BSP por um plano paralelo a um limite externo do ambiente (Figura 24). Cada nova divisão divide uma célula em duas, não necessariamente no seu centro. Árvores BSP proporcionam o mesmo particionamento binário das *quadtrees*, mas não exigem o mesmo rigor do particionamento exato. Uma representação em BSP de uma porção de espaço não é necessariamente única, é dependente da escolha dos planos de divisão, gerando representações radicalmente diferentes da mesma porção de espaço.

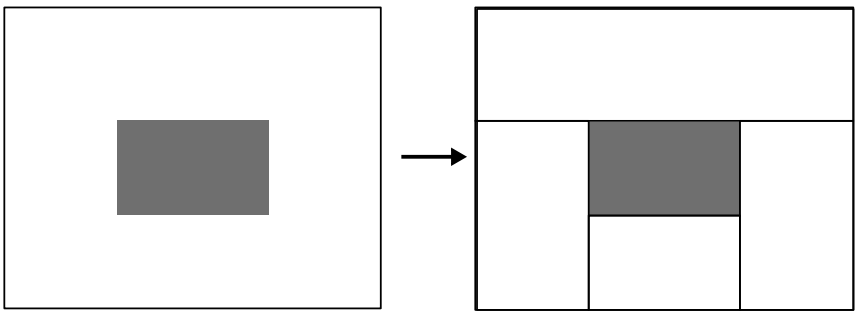

Figura 24: Exemplo de decomposição espacial BSP.

4.1.3 Decomposição Exata

Outra alternativa para a subdivisão do espaço é a *subdivisão exata*. O espaço livre é repartido em regiões não sobrepostas por planos, de tal forma que a união destes planos compõem exatamente o espaço livre[3, 1] (Figura 25).

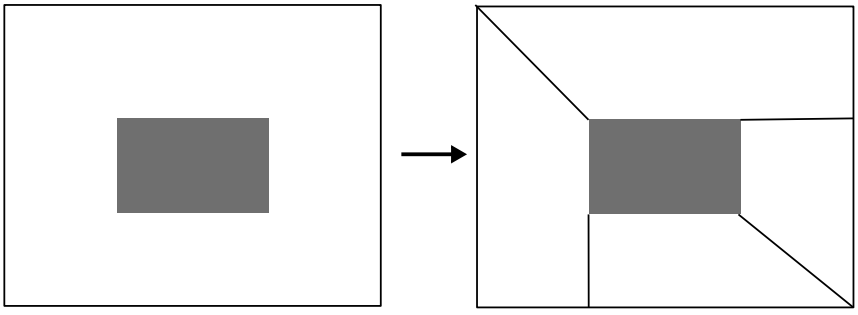

Figura 25: Exemplo de decomposição espacial exata.

As células geradas pela decomposiação devem possuir um formato simples, de modo a facilitar a busca por um caminho entre duas células adjacentes. Em ambientes bidimensionais e com obstáculos poligonais é conveniente utilizar uma *decomposição polinomial convexa* ou *decomposição trapezoidal*[32].

A principal vantagem da subdivisão exata é que ela é exata. Porém, da mesma forma que as árvores BSP, diferentes formas de planos de subdivisão podem gerar representações radicalmente diferentes do mesmo espaço.

4.2 Representações Geométricas

Mapas geométricos são aqueles formados por primitivas geométricas, como linhas, polígonos, poliedros, pontos, funções polinomiais, por exemplo. Tais mapas possuem o diferencial de serem eficientes quanto a representação do espaço, por exemplo, uma grande região de espaço livre pode ser representada por poucos parâmetros geométricos. Adicionalmente, representações geométricas podem assumir dados de ocupação espacial com uma resolução arbitrária, sem representar problemas de armazenamento computacional.

Os mapas geométricos usados na robótica móvel são compostos pela união de primitivas geométricas, e baseado neste fato podem ser caracterizados pelas seguintes propriedades:

- o conjunto de primitivas geométricas usado na descrição de objetos;
- o conjunto de composição e operadores de deformação usado para manipular estes objetos.

As primitivas como pontos, retas, segmentos de retas e polinômios são utilizadas para as presentações bidimensionais (2D). Já para mapas tridimensionais (3D) são empregadas primitivas bidimensionais e elementos como superfícies planas e poliedros[32].

O mesmo formalismo matemático que suporta os espaços bi e tridimensionais fornece também os operadores para a manipulação destas primitivas, os operadores mais comuns são:

- transformações rígidas, como rotação e translação;
- transformações padronizadas, que preservam as formas dos objetos;

- transformações de distorção;
- operadores Booleanos.

Operadores estes geralmente empregados na forma matricial. Idealmente, os modelos geométricos podem possuir características como[3, 32]:

Simplicidade: o número de parâmetros descrito pode ser mínimo.

Não-ambiguidade: cada primitiva geométrica tem apenas uma representação.

Completude: cada primitiva geométrica tem pelo menos uma representação.

Diferenciável: permite a linearização das equações de medida.

A principal dificuldade na aplicação de modelos baseados em representação geométrica se refere ao fato de que é difícil obter leituras confiáveis apenas dos sensores. Além desta limitação, outras três dificuldades são intrínsecas a esta abordagem. Estas dificuldades emergem na estimativa imprecisa de parâmetros e devido a diversidade das classes de modelos, o que dificulta a associação de modelos específicos a diferentes conjuntos de leituras. Estas dificuldades são:

Falta de Estabilidade: a representação geométrica pode variar drasticamente dada uma pequena variação na entrada dos sensores. Está associada ao fato de que modelos geométricos gerados como resultado de um conjunto de leituras pode variar rapidamente em ressonância a pequenas variações nos dados lidos.

Falta de Unicidade: diferentes ambientes podem ser mapeados pela mesma representação, e vice-versa, diferentes representações são possíveis para o mesmo ambiente. A falta de unicidade de representação advém da possibilidade que um conjunto de observações pode ser expressado em mais de uma forma. Isto é comum em sistemas que aproximam suas entradas ao invés de usar valores exatos. Por exemplo, modelos extraídos a partir do princípio de menor caminho de descrição, que associa um custo a cada modelo usado, seus erros de aproximação e tentativas de encontrar um modelo único. A minimização deste custo pode levar à falta de unicidade.

Falta de Força Expressiva: pode ser difícil a representação de saliências no ambiente dentro do modelo; em alguns casos pode ser até mesmo impossível.

4.2.1 Representações Geométricas Estocásticas

Os modelos baseados na representação geométrica geralmente são definidos *a priori*, baseados no conhecimento previamente adquirido do ambiente e possuem forma estática. O modelo de *representação geométrica estocástica* não é definido *a priori*, e a incerteza é adicionada ao modelo[32]. Este tipo de modelo é construído durante a operação do robô, sendo utilizado na exploração e geração de mapas[3, 32].

Para o uso deste tipo de modelo o robô deve ser capaz de construir representações geométricas do seu ambiente de operação. Por exemplo, pelo uso de pontos construir retas ou planos[32]. A representação tende a ser sempre local, dadas as restrições de alcance dos sensores, a não ser que o robô seja provido de algum mecanismo de armazenamento de informações para armazenar o histórico das representações.

4.3 Representações Topológicas

As representações geométricas se baseiam em primitivas geométricas e em dados métricos como fundamento de sua abordagem. Para contornar todas as dificuldades e deficiências da abordagem geométrica, pode-se optar pela representação topológica. As representações topológicas se mostram especialmente adequadas em ambientes de larga escala, como podemos observar em nosso dia-a-dia. Por exemplo, é muito mais simples, tanto para o comunicador quanto para o receptor, o conjuto de instruções "siga esta rua, vire na próxima esquina à direita, no segundo cruzamento, vire à esquerda..." do que "ande mais 50 metros na direção Sul, vire 90^o no sentido horário em relação ao seu vetor direção, ande mais 350 metros, depois vire 90^o no sentido anti-horário...". Desta forma, é relativamente simples estabelecer e representar o relacionamento entre entidades[3].

A principal característica da representação topológica é a representação da conectividade entre regiões e/ou objetos. Em sua manisfestação idealizada pode abdicar de qualquer dado métrico. Esta representação faz uso de abstrações para localidades discretas, com vértices conectores, de forma semelhante a um grafo (Figura 26), $G = (V, E)$, onde V é um conjunto de nós ou vértices (ou localidades) e E é o conjunto de segmentos que conectam as localidades (ou caminhos). Pode ser apropriado associar ao grafo G dados métricos, como o comprimento ou custo dos segmentos, ou a orientação dos vértices em função dos respectivos nós. O ambiente do robô pode ser representado como um grafo cujos vértices correspodem a marcas visuais únicas (*landmarks*) dispostas no ambiente do robô. A representação deve conter todas as informações necessárias para o

robô conduzir sua locomoção ponto-a-ponto. De fato, esta é uma das representações mais utilizadas devido à aplicabilidade das técnicas de teorias de grafos e de ser extremamente compacta[32].

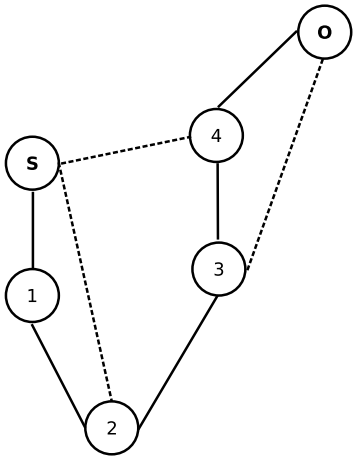

Figura 26: Exemplo de representação espacial em grafos.

4.3.1 Representação Topológica Por Marcas - *Landmarks*

Uma marca (*landmark*) pode ser qualquer objeto ou conjunto de objetos que integre o ambiente[3] (Figura 27). As marcas, ou marcações, podem ser classificadas em dois grupos: (*i*) naturais e (*ii*) artificiais. As marcas naturais geralmente não podem ser modificadas ou removidas, como uma parede, uma porta, uma montanha ou árvore). Já as marcas artificiais podem ser movidas ou modificadas, uma vez que fazem parte da estruturação do ambiente, como placas e sinalizadores. As marcas podem ser utilizadas para localização do robô, planejamento de ações e controle do robô[32].

4.4 Representação do Robô

Os robôs móveis são projetados e construídos nas mais variadas formas e tamanhos, encerrando as mais variadas características. A representação da forma, tamanho e características se torna relevante na mesma proporção da necessidade da representação do ambiente que cerca o robô, em termos representativos, matemáticos e/ou computacionais, será a representação do ambiente que interagirá com a representação do robô.

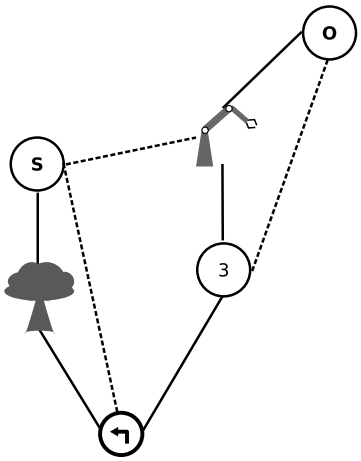

Figura 27: Exemplo de representação espacial por *landmarks*.

4.4.1 Espaço de Configurações

O *espaço de configuração* é um conceito chave no formalismo para o planejamento de movimento[33]. Uma configuração q de um robô A é a especificação do estado fisico de A em relação a uma representação fixa do ambiente F_w, como um sistema de coordenadas cartesianas, por exemplo. Considere, por exemplo, um robô A capaz de transladar e rotacionar sobre um plano. A configuração do robô A pode ser representada pela tupla $q = [x\ y\ \theta]$, onde q define comletamente a configuração do robô em um instante do tempo. Para robôs mais complexos, como os robôs com membros articulados, a estrutura q se torna mais complexa, na medida que contém o estado de cada característica relevante do robô como, por exemplo, os ângulos de rotação das articulações. O espaço de configuração de A é o espaço $\mathcal{C}$ de todas as configurações possíveis de A. O espaço $\mathcal{C}$ contém todas as configurações válidas do robô A em um ambiente, considerando-se as restrições impostas pelos obstáculos[1, 32].

A construção física de um robô impõe certas limitações de configuração, da mesma forma que a presença de obstáculos. Por exemplo, um robô circular com raio r pode se mover a uma distância não menor que r de um objeto, restringindo as configurações possíveis do robô e reduzindo o tamanho da região de configurações acessíveis pelo robô. As restrições desta natureza podem ser definidas na forma de

$$G(q) = 0 \tag{4.1}$$

onde q é a configuração do robô, restrições desta mesma natureza estática são denominadas de *restrições holonômicas*. Para qualquer obstáculo real, pode-se transformá-lo em um

conjunto de pontos em um espaço de configuração e produzir uma relação na forma de (4.1). A região determinada inacessível por um obstáculo real é chamada de *espaço de configuração de obstáculo* – $\mathcal{C}_{obstaculo}$. O planejamento de trajetórias envolve determinar uma trajetória que satisfaça um ou mais requisitos de otimização e que evite qualquer espaço de configuração de obstáculo. A porção do ambiente que é acessível pelo robô, o *espaço livre*, é representada por uma região análoga de espaço de configurações denotada por $\mathcal{C}_{livre}$, e é definda por

$$\mathcal{B} = \bigcup_{i=1}^{q} \mathcal{C}_{obstaculos} \tag{4.2}$$

$$\mathcal{C}_{livre} = \{q \in \mathcal{C} | A(q) \cap \mathcal{B} = \oslash\} \tag{4.3}$$

Um caminho é o mapeamento contínuo das configurações do robô que o conduzem do ponto de origem até o ponto objetivo[32]:

$$\tau : [t_o, t_f] \rightarrow \mathcal{C}_{livre} \tag{4.4}$$

onde $\tau(t_o)$ é o ponto de origem e $\tau(t_f)$ é o ponto objetivo.

Restrições nas *derivadas* do movimento do robô que não podem ser integradas (reduzidas a restrições holonômicas) são conhecidas como *restrições não-holonômicas*. Estas tomam a forma de

$$G\left(q, \frac{dq}{dt}, \frac{d^2q}{dt^2}, \ldots\right) = 0 \tag{4.5}$$

As restrições não-holonômicas incluem restrições em quais tipos de velocidades são permitidas. Restrições não-holonômicas reduzem o intervalo de movimentos permitidos e adicionam complexidade ao planejamento de trajetórias. A essência do problema causado por restrições não-holonômicas advém do fato de que para que um robô possa se mover de um estado permitido para outro, mesmo que estes estados sejam próximos um do outro, uma trajetória de complexidade arbitrária pode ser necessária. Exemplos de restrições não-holonômicas são as encontradas em carros e veículos com reboques.

Um problema clássico de restrição não-holonômica em robótica móvel é a restrição no *raio de curvatura* que um determinado trecho de trajetória pode ser executado. Na presença de tal restrição, quando o comprimento do caminho deve ser otimizado, pode

ser necessário decompor um movimento curvilíneo em uma seqüência de pequenos arcos circulares.

Para tornar o relacionamento das restrições holonômicas e não-holonômicas mais palpável, eis um exemplo: consideremos um robô A com propulsão diferencial e cuja posição seja determinada pela tupla $q = [x\ y\ \theta]$. Suponha que o raio do robô é r e que existe um obstáculo de tamanho infinitesimal na origem do plano de coordenadas cartesiana. Este obstáculo resulta em uma restrição holonômica na forma

$$x^2 + y^2 > r^2 \tag{4.6}$$

Isto é, o centro do robô coincide com a posição de um obstáculo. Se o robô se move a uma velocidade v, então

$$\frac{dx}{dt} = v\cos(\theta) \tag{4.7}$$

$$\frac{dy}{dt} = v\sin(\theta) \tag{4.8}$$

e $dx\ \sin(\theta) - dy\ \cos(\theta) = 0$; isto é, o robô se move em linha reta na direção de sua frente. Esta é uma restrição não-holonômica como dado na forma de (4.5), uma vez que envolve q e q', e q' não pode ser eliminado. O planejamento de trajetórias de robôs que sofre de restrições holonômicas é conhecido como *planejamento de trajetórias holonômico*, que é significativamente diferente do *planejamento de trajetórias não-holonômicas.* Na presença de restrições não-holonômicas o mais simples problema de planejamento de trajetória pode se mostrar extremamente complexo.

Controlabilidade se refere ao fato de ser capaz de se mover para diferentes posições no espaço de estados. Sistemas não-holonômicos são localmente controláveis, embora sejam em apenas alguns casos globalmente controláveis.

4.4.2 Simplificações

O maior problema associado com a representação de um robô e seu espaço de configurações é que a representação de obstáculos no espaço de configurações e a dimensionalidade resultante do espaço de busca podem ser tornar computacionalmente complexos. Várias simplificações devem ser desenvolvidas para reduzir a complexidade do espaço de confi-

gurações e custo associado à representação dos objetos e o planejamento de trajetória.

A primeira simplificação é assumir a representação do robô como um ponto, capaz de movimentação omnidirecional. O problema inerente a esta simplificação é que robôs autônomos não são pontos, e muitos projetos introduzem restrições não-holonômicas. Se o robô é simplificado a um ponto, e todo o processamento subseqüente faz uso dessa simplificação para o planejamento de trajetórias, será necessário introduzir metainformações sobre as reais dimensões e restrições do robô. Uma alternativa simples para contornar o fato de que robôs móveis não são pontuais é assumir que o robô possui uma secção transversal circular, e então dilatar todos os obstáculos do ambiente pela quantidade igual ao raio dessa secção transversal (Figura 28). O próprio ambiente passa a embutir, em sua representação, metainformações acerca do tamanho do robô. A operação de dilatação – conhecida como soma de Minkowski, é muito utilizada no processamento de imagens – de um objeto pode ser computada, aproximadamente, como a união da forma original do objeto com um conjunto de círculos (ou esferas quando a representação é 3D) dispostos sobre cada ponto dos limites físicos do objeto. Porém, a dilatação de objetos pode resultar em uma representação do ambiente mais complexa do que o próprio ambiente original, uma vez que limites retos (como cantos angulosos) serão dilatados em formas circulares.

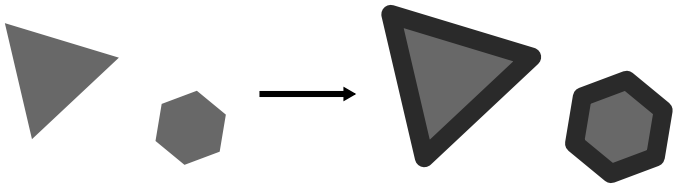

Figura 28: Exemplo de dilatação de obstáculos.

4.5 Planejamento de Trajetórias

O planejamento de movimento sujeito a vários tipos de restrições é uma tarefa que, embora à primeira vista possa parecer simples e intuitiva, pode se tornar simbólica e computacionalmente complexa. O problema básico no planejamento de trajetórias se refere em determinar um caminho dentro do espaço de configurações entre uma configuração inicial e a configuração final do robô (ver seção 4.4), de tal forma que o robô não venha a colidir com qualquer obstáculo no ambiente ($\mathcal{C}_{obstaculo}$) e que o movimento planejado seja consistente com as restrições cinemáticas do veículo. A configuração inicial é chamada de *posição de início* e a localização final de *objetivo*. O planejamento de trajetória deve superar os seguintes desafios:

- Eventualmente é de interesse considerar o caminho com o menor comprimento;
- Formulações alternativas de um caminho de custo mínimo podem ser relevantes. Um caminho que minimiza o tempo decorrido é diferente do caminho de menor comprimento. Por exemplo, dois caminhos são diferentes quando a máxima velocidade do robô é uma função da curvatura do caminho. Conseqüentemente o caminho de menor comprimento pode envolver menores velocidades do que um caminho mais longo, porém, com curvaturas mais suaves e velocidades maiores;
- Ambientes com obstáculos móveis representam desafios adicionais;
- Pode ser necessário escolher caminhos que satisfaçam outras restrições além de atingir o objetivo. Por exemplo, a escolha de um caminho seguro que permita ao robô detectar marcas no ambiente, ou passagem por algum ponto importante do espaço livre.

O planejamento de movimento deve considerar as habilidades do robô e a estrutura do ambiente. Encontrar um caminho τ, que a partir de um estado inicial q_0 leva o robô ao seu objetivo. Os algoritmos para tal tarefa são desenvolvidos fazendo-se uso de diferentes teorias e requisitos:

Ambiente e o robô: a estrutura do ambiente, as capacidades do robô, como sua forma, tamanho.

Corretude: o caminho planejado é garantidamente correto, livre de colisões e dirigido ao ponto objetivo.

Completude: o algoritmo é sempre capaz de encontrar um caminho, mesmo na existência de apenas uma solução.

Complexidade de Tempo e Espaço: o espaço de armazenamento ou tempo computacional necessário para se obter uma solução.

Para tornar o problema de planejamento de trajetórias tratável, muitas vezes é necessário assumir uma série de simplificações com respeito ao ambiente. Uma vez desenvolvido um algoritmo baseado em um conjunto de pré-suposições, é preciso verificar e validar sua aplicabilidade no mundo real. Algoritmos idelizados para o planejamento de trajetórias devem ser corroborados para lidar com vários obstáculos reais como obstáculos móveis,

restrições variáveis em função do tempo, definições complexas do objetivo (até mesmo objetivos móveis), critérios de otimização ou condições colaterais e as incertezas do mundo real.

4.5.1 Busca em um Espaço de Estados Discreto

Algoritmos de busca formam um dos componentes fundamentais do planejamento de trajetórias de muitos robôs. Dado um espaço de busca, um conjunto de possíveis estados do problema, e uma função de transição de estados para determinar os estados adjacentes a partir de qualquer estado, um *método de busca* é um algoritmo para o controle da exploração do espaço de estados para a identificação de um caminho a partir de um estado inicial até o objetivo.

4.5.1.1 Busca em Grafos

Uma completa literatura existe para a busca em grafos, mas o algoritmo de busca em grafos mais simples consiste na expansão dos nós em seqüência, até que o nó objetivo seja alcançado. Como aprensentado pelo Algoritmo 1.

Algoritmo 1: Algoritmo genérico de busca em grafos.

entrada: s, objetivo
saída : caminho

$ABERTO \leftarrow \{s\}$;
$FECHADO \leftarrow \{\ \}$;
$encontrado \leftarrow falso$;
enquanto *(*$ABERTO \neq \oslash$*) e (não encontrado)* **faça**
  Selecionar nó n de $ABERTO$;
  $ABERTO \leftarrow ABERTO - \{n\}$;
  $FECHADO \leftarrow FECHADO \cup \{n\}$;
  se $n \in objetivo$ **então**
    $encontrado \leftarrow verdadeiro$
  senão
    $M \leftarrow$ conjunto de nós acessíveis apartir de $n | (n \notin FECHADO)$;
  fim
fim

Dada uma função de transição de estado, um grafo G com nó inicial s e um conjunto

de *nós objetivos*, o algoritmo genérico de busca determina se existe um caminho apartir de s até um objetivo. O algoritmo mantém uma lista de nós visitados ($FECHADOS$) e uma lista de nós ($ABERTOS$) que foram visitados mas que levam, direta ou indiretamente ao objetivo. O algoritmo permanece em execução até que um objetivo seja alcançado (quando *encontrado* é verdadeiro), ou até que o conjunto $ABERTO$ esteja vazio, isto é, não existem nós que levem a soluções direta ou indiretamente (e *encontrado* é falso). O algoritmo de busca apenas determina se existe um caminho do nó de início até um dos nós objetivos. Diferentes estruturas para $ABERTO$ podem resultar em diferentes algoritmos. Por exemplo, se $ABERTO$ for uma pilha, então é realizada uma busca em profundidade. Se $ABERTO$ for uma fila então é realizada uma busca em amplitude. Outra alternativa é ordenar os elementos de $ABERTO$ pelo uso de uma função de avaliação $f(n)$. Valores pequenos para $f(n)$ indicam que n tende a pertencer ao caminho mais curto.

Por exemplo, considere uma formulação para a função de avaliação $f(n)$. Supondo que nenhuma ligação no grafo possui peso zero ou negativo, então $f(n)$ pode ser usada para determinar o custo do caminho mais curto a partir do nó de início até o nó n. O processo de avaliar $f(n)$ pode ser realizado na linha 11 do algoritmo genérico de busca, de tal forma que $f(n) = PESO + f(n_{anterior})$ onde $n_{anterior}$ é o nó expandido para se obter o nó n, e $PESO$ é o peso associado com o movimento do $n_{anterior}$ ao nó n.

A formulação anterior é dita *desinformada*, uma vez que só pode determinar o custo do nó inicial até o nó atual, não há qualquer informação quanto a direção ou distância do conjunto de nós explorados em relação ao nó objetivo. Porém, supondo que $f(n) = g(n) + h(n)$ é a função de custo, onde $g(n)$ é o custo estimado do nó de início até o nó atual, e $h(n)$ é o custo estimado do nó atual até o nó objetivo. Se $g(n)$ é o limite inferior de $g^*(n)$, o caminho de custo mínimo do nó de início até o nó atual n, e se $h(n)$ é o limite inferior de $h^*(n)$, o caminho de custo mínimo entre o nó atual n até o nó objetivo, o algoritmo passa a ser capaz de achar o caminho ótimo (minimizando os custos para cada elo no grafo) do nó inicial até o nó objetivo[6].

Se nenhum caminho existir entre o nó de início até o nó objetivo, então o algoritmo de busca genérica degenera, avaliando cada estado alcançável, antes de retornar sem nenhum caminho. Desta forma, na pior das hipóteses, quando não há um caminho entre o nó de início e o de objetivo, pode haver uma demora considerável antes da notificação do fracasso da busca. Embora eficiente, a busca em grafos tende a se mostrar computacionalmente lenta e muitas vezes inadequada para o planejamento de espaços amplos e em tempo real.

[6]Este algoritmo é conhecido como A*.

4.5.1.2 Programação Dinâmica

A programação dinâmica é outra estratégia de propósito geral que pode ser usada para o planejamento de trajetórias. Pode ser descrita como um procedimento recursivo para a avaliação de caminhos de custo mínimo a partir de um ponto de origem até um ponto de destino. Para o uso da programação dinâmica, o problema deve possuir a característica de que a partir de quaisquer três pontos A, B e C, o caminho ótimo de A até B via C pode ser determinado pelo caminho ótimo de A até C e de C até B; o que é chamado de princípio de Bellman. Para isso, o ambiente deve ser discretizado, e ao movimento de uma localidade até outra adjacente é atribuído um custo específico que pode ser espacialmente variável mas nunca uma função do tempo. O espaço discretizado passa a ser chamado de *tabela de custos* e indica o custo associado à transição entre as células da tabela. O princípio é encerrar o processamento no cálculo incremental da transição entre células adjacentes previamente conhecidas. Eventualmente, de forma a atenuar a necessidade de uma busca exaustiva entre toda a tabela, funções heurísticas podem ser usadas para guiar a direção de transição para células específicas, evitando a busca por força bruta.

4.5.2 Alternativas à Construção de um Espaço de Busca Discreto

Exploradas algumas das técnicas mais comuns para a busca de trajetórias em espaços discretos, resta apresentar alguns dos métodos mais utilizados para a discretização de um ambiente.

Possivelmente, a estratégia mais simples é tomar uma representação geométrica do ambiente e discretizá-la usando, por exemplo, métodos de discretização geométrica ou topológica, como apresentados anteriormente. Depois um grafo é construído, e as células adjacentes são conectadas. As células que eventualmente representariam um obstáculo para o robô podem ser excluídas da representação discretizada. O grafo é então alvo de um método de busca. Ambientes amplos, e discretizados com fina granularidade podem resultar em espaços de busca grandes e complexos, inviabilizando a aplicação de tal técnica em um robô real.

Porém, outras estratégias foram desenvolvidas na busca de simplificar as dificuldades e complexidades da estrutura em grafo.

4.5.2.1 Planejamento por Grafos de Visibilidade

O *grafo de visibilidade* é uma técnica que produz o caminho de menor comprimento em relação ao nó de início até o nó objetivo pela resolução de um algoritmo de travessia de grafo. O grafo de visibilidade $G_{visibilidade} = (V, E)$ é definido de tal forma que o conjunto de vértices é formado pela união de todos os vértices dos obstáculos poligonais presentes no ambiente, bem como dos pontos de partida e objetivo (Figura 29). Os vértices do grafo conectam todos os vértices visíveis uns aos outros, isto é, uma linha reta que nunca intersepta qualquer obstáculo. Concluída a construção do grafo, este contém um subconjunto dos vértices dos obstáculos e dos pontos de partida e objetivo, além da conexão das regiões que podem ser percorridas livremente sem atingir qualquer obstáculo[32].

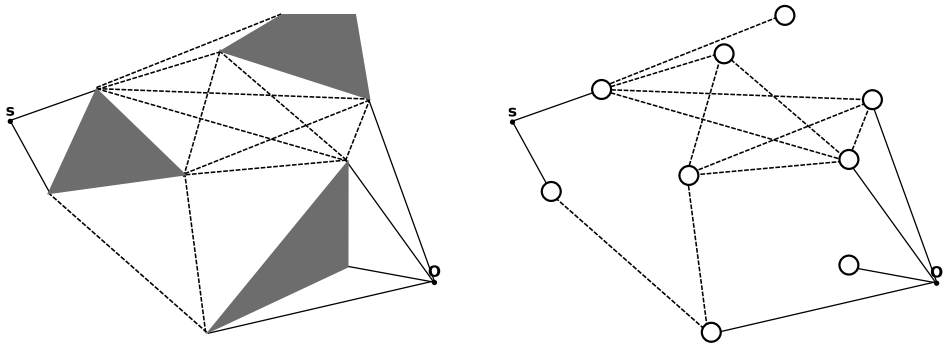

Figura 29: Exemplo de grafo de visibilidade.

Como nenhum caminho no grafo resultante pode passar através de um vértice de um objeto côncavo, apenas vértices de obstáculos convexos devem ser considerados.

Uma característica que pode se tornar depreciativa à escolha do grafo de visibilidade é que o caminho passa literalmente através dos vértices e é arbitrariamente coincidente com estes pontos dos obstáculos. Devido a esta caracterísitica, os grafos de visibilidade definem caminhos *semi-livres*. A técnica simplesmente não leva em consideração as dimensões do robô, faz a pressuposição de que estas dimensões podem ser ignoradas. Estes problemas podem ser parcialmente contornados pela dilatação dos obstáculos presentes no ambiente pelo raio máximo das dimensões do robô.

4.5.2.2 Planejamento por Diagramas de Voronoi

O *diagrama de Voronoi* é o análogo do grafo de visibilidade. Enquanto o grafo de visibilidade propositadamente determina caminhos coincidentes a um subconjunto de vér-

tices dos obstáculos presentes no ambiente, o diagrama de Voronoi determina a localidade de pontos eqüidistantes dos limites de dois obstáculos diretamente próximos, incluindo os limites do ambiente. O conjunto de pontos assim determinados possui a caracterísitica de maximizar a distância dos obstáculos[32]. O diagrama resultante de um ambiente fechado e com obstáculos pequenos é composto de curvas contínuas, isto é, caminhos físicos factíveis. Desta forma, em qualquer ponto do diagrama de Voronoi, a distância a obstáculos próximos não pode ser incrementada por qualquer movimento local aos contornos do diagrama (Figura 30).

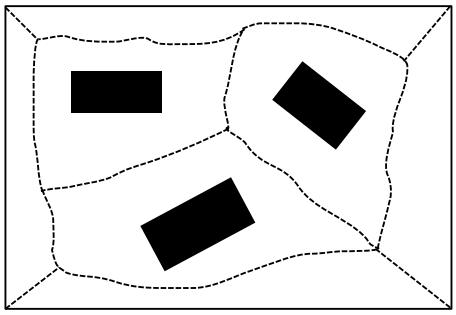

Figura 30: Exemplo de diagrama de Voronoi.

Se a um robô é determinado percorrer os caminhos definidos por um diagrama de Voronoi, então este não apenas evita os obstáculos, mas também maximiza localmente a distância em relação aos obstáculos. Dados os pontos de partida e objetivo, um robô poderia se afastar diretamente do obstáculo mais próximo, até que ele atingisse um ponto pertencente ao diagrama de Voronoi. A partir desta posição ele passa a percorrer os arcos e segmentos de reta que formam o diagrama, até atingir uma posição tal que seja posível se mover diretamente até a posição objetivo, mantendo máxima a distância de qualquer obstáculo.

Enquanto esta estratégia se mostra ótima em relação aos obstáculos existentes no ambiente, ela pode resultar em caminhos excessivamente longos.

4.5.3 Busca em um Espaço de Estados Contínuo

Alternativamente à busca em um espaço discreto que representa o estado do robô, é possível modelar o espaço de configurações do robô como um espaço contínuo e considerar o planejamento de trajetórias dentro desse espaço.

4.5.3.1 Campos Potenciais Artificiais

O planejamento que faz uso de *campos potenciais artificiais* é baseado na seguinte analogia: o robô passa a ser tratado como uma partícula que age sob a influência de um campo potencial[3] U, que é modulado para representar a estrutura do espaço livre. Geralmente os obstáculos são modelados de forma a possuir uma carga elétrica, e o campo potencial escalar resultante é usado para representar o espaço livre. O robô também é modelado com a mesma polaridade das cargas usadas na representação dos obstáculos, o que faz com que o robô desvie de qualquer obstáculo segundo a influência das forças repulsivas entre eles, o que em termos matemáticos é representado pelo negativo do gradiente do campo potencial. O mesmo é realizado para o ponto de partida, o que garante ao robô o ímpeto de abandonar este ponto[32] (Figura 31). Diferentemente de qualquer outro elemento modelado, o ponto objetivo possui carga de polaridade contrária a qualquer outro elemento, o que proporciona a força atrativa que guia o robô em direção a este ponto objetivo.

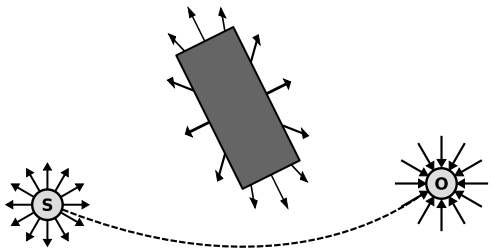

Figura 31: Exemplo simplificado de campo potencial artificial.

Formalmente, um campo potencial artificial $U(q)$ é construído a partir da associação dos campos gerados pelo ponto de partida $U_{partida}(q)$, pelo ponto objetivo $U_{objetivo}(q)$ e pelos campos gerados pelos obstáculos $U_{obstaculos}(q)$. O campo potencial percebido pelo robô pode ser determinado por[32]:

$$U(q) = U_{partida}(q) + U_{objetivo}(q) + \sum U_{obstaculos}(q) \tag{4.9}$$

Que pode ser usado para definir uma força de campo artificial definida por:

$$F = -\nabla U(q) = -\begin{pmatrix} \delta U/\delta x \\ \delta U/\delta y \end{pmatrix} \tag{4.10}$$

O movimento do robô pode ser executado a partir de pequenos passos segundo a direção e sentido da força local. A força repulsiva pode ser calculada em relação aos

obstáculos mais próximos (localmente) ou pela soma de todos os obstáculos do ambiente (globalmente).

Um ponto importante para o planejamento de trajetórias, segundo campos potenciais artificiais, é como lidar com mínimos locais, que podem aprisionar o robô (Figura 32). Devido a esta característica, é comum usar esta técnica apenas para o planejamento local[3].

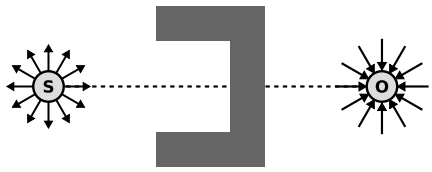

Figura 32: Exemplo simplificado de um mínimo local em um campo potencial artificial, formando uma "armadilha" para o robô.

4.5.3.2 Algoritmo do Inseto – *Bug Algorithm*

Em algumas situações não é possível dispor de um mapa global do ambiente enquanto o robô se move em direção a seu objetivo. O planejamento baseado em campos potenciais é um exemplo de método de planejamento que pode ser aplicado nestas situações. Porém, o planejamento local baseado em campos potenciais não pode garantir o resultado de um caminho até o ponto objetivo. Sob certas circunstâncias é possível aplicar métodos que se adequem à presença de incertezas na busca da melhoria de desempenho. O *algoritmo do inseto* pertence a esta classe de método. Este tipo de algoritmo é usado para o planejamento de trajetórias de posições de partida até a de objetivo que possuem coordenadas conhecidas, na pressuposição de um robô pontual holonômico com odometria perfeita, um sensor de contato ideal e uma memória infinita.

O algoritmo opera basicamente alternando entre dois comportamentos simples: (*i*) o movimento em direção do ponto objetivo e (*ii*) a circunavageção de um obstáculo. A forma básica do algoritmo do inseto é descrita por:

Algoritmo 2: Algoritmo do inseto.

Visualizar um caminho direto $\overline{SO}$ do ponto de partida S até o objetivo O;
enquanto *o objetivo O não é atingido* **faça**
  enquanto *o caminho* $\overline{SO}$ *permanece livre* **faça**
    Mover em direção ao objetivo;
    se *o caminho é obstruído* **então**
      Marque a posição atual como **p**;
      Circunavegue o obstáculo até que o robô;
      (1) atinja novamente a linha $\overline{SO}$ e possa se mover em direção de O, ou;
      (2) retorne a **p** onde O é inatingível;
    fim
  fim
fim

O algoritmo do inseto sempre vai encontrar um caminho até o objetivo se este estiver acessível. A trajetória, porém, pode ser pior do que o comprimento da trajetória ótima (Figura 33).

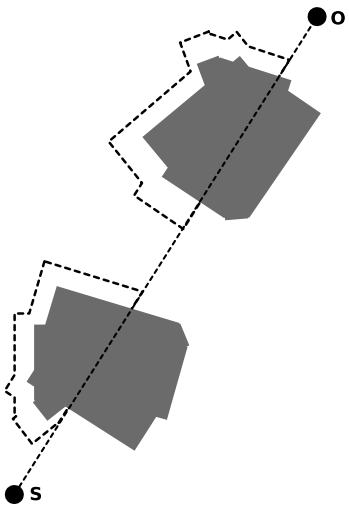

Figura 33: Exemplo do algoritmo do inseto.

Uma fonte de complexidade associada ao algoritmo do inseto é o retorno do robô à linha $\overline{SO}$ depois de circunavegar um obstáculo. Uma estratégia mais eficiente seria se mover na direção do obstáculo assim que uma linha reta livre seja viável. Outra alternativa seria o movimento em direção a um ponto da linha $\overline{SO}$ mais próximo de O do que o ponto de intersecção com os obstáculos interferentes. O algoritmo do inseto pode até mesmo ser estendido para realizar a exploração do ambiente.

4.5.4 Incerteza Espacial

A maioria das estratégias algorítmicas para a navegação enfatizam aspectos particulares do total do problema enquanto assumindo simplificações do que é deixado de lado. As habilidades do robô em identificar quando ele retornou a um ponto específico, identificar quando retornou a uma linha no espaço, e seguir o contorno de um objeto de forma precisa são essenciais para as abordagens algorítmicas. As falhas em uma ou mais destas características levarão à falha no planejamento da trajetória.

Um problema chave na aplicação de algoritmos idealizados de planejamento de trajetórias no mundo real é o fato da *incerteza espacial*. O problema em se determinar quando a posição atual (x_i, y_i) é a mesma de qualquer outro ponto (x_k, y_k) nem sempre é fácil de ser resolvido. Idealmente, a posição do robô em um espaço de configurações pode ser representada por uma *função de densidade de probabilidade* para denotar a probabilidade de que o robô esteja em um determinado ponto do espaço. Infelizmente nem sempre é viável computar este tipo de representação, devido à impossibilidade de modelar a evolução da função de densidade de forma precisa à medida que o robô se move ou leituras dos sensores são realizadas[32].

Para representações topológicas a incerteza aparece de duas formas distintas: (i) incerteza de que o robô está em um lugar representado por um nó, e (ii) incerteza em relação a qual nó o robô pertence, assumindo que ele esteja em um. Uma estratégia para lidar com a incerteza espacial em representações topográficas é o uso de processos Markovianos observáveis; as ações do robô são associadas com as mudanças na distribuição de probabilidades do robô. Por exemplo, o movimento adiante aumenta a probabilidade de que o robô esteja em um nó mais adiante do que aquele que ele estava ocupando. À medida que o robô se move e observa seu ambiente, esta distribuição de probabilidades pode convergir em uma elevada probabilidade em um único nó – uma junção de corredores onde os dados dos sensores são definitivos – ou pode divergir quando o robô encontra uma região com leituras ambíguas.

4.5.5 Ambientes Complexos

Os algoritmos de planejamento de trajetórias sempre fazem algum tipo de simplificação referentes ao ambiente do robô, simplificações que muitas vezes não podem ser utilizadas na prática.

4.5.5.1 Ambientes Dinâmicos

Ao invés do requisito de obstáculos estáticos, é permitido que os objetos se movam. Se o movimento dos objetos é conhecido *a priori*, então o problema é similiar ao planejamento com objetos estáticos, basta embutir este conhecimento, na forma de regras, no modelo do ambiente.

Mudanças no ambiente são expressadas na forma de regras que capturam o fato de que ações possuem diferentes custos sob diferentes condições. Por exemplo, o planejador de trajetórias pode aprender que uma determinada auto-estrada se mostra sempre congestionada no horário do *rush*, ou que um elevador sofre uma maior carga de requisições em algumas faixas de horários bem definidas. Uma vez que os padrões foram identificados e relacionados às características do ambiente, o planejador pode predizer e planejá-las quando condições similares ocorrerem no futuro. Um obstáculo móvel, com movimento previsível, pode ter seu comportamento modelado e embutido no conjunto de regras do planejador.

4.5.5.2 Ambientes Abertos

O espaço em ambientes abertos pode não ser atravessável de forma omnidirecional. Vegetação, ruas, direção do Sol, são elementos que podem restringir a travessia do terreno. O planejador deve tomar em consideração tais informações no momento do cálculo do caminho. Outro fator complicador em ambientes abertos são as irregularidades no terreno, como depressões, buracos, que geralmente não são associados com o preenchimento do espaço e são de difícil detecção.

4.5.5.3 Ambientes Desconhecidos

Um dos principais pontos fracos dos sistemas de planejamento de trajteória é o conhecimento do ambiente *a priori*. Em um exemplo clássico, o robô planeja sua tajetória sobre uma representação do ambiente e começa sua execução. Se o modelo do mundo é impreciso, então em algum momento da execução da trajetória planejada o robô pode encontrar um evento que torna a trajetória planejada inválida, exigindo o replanejamento. O planejamento inicial foi desperdiçado. Como é comum que o planejamento inicial não se adeque ao mundo real, uma alternativa ao esforço gasto para o planejamento inicial é a geração de um plano rápido aproximado para execução, que é refinado à medida que o robô se move.

Os métodos que lidam com a necessidade de replanejamento e podem reavaliar o caminho em execução são chamados de *algoritmos on-line*, e a trajetória produzida é referenciada como *plano condicional*. Um algoritmo on-line verdadeiro deve ser capaz de gerar uma trajetória preliminar até mesmo na ausência de mapas. O algoritmo do inseto, e o método de campos potenciais são exemplos de algoritmo on-line para o planejamento de trajetórias.

4.6 Conclusões

A representação do espaço é o elo de ligação entre a percepção do ambiente, por meio dos sensores, e a atuação do robô móvel, por meio de sua forma de locomoção. Representa um elemento essencial também para o sistema de controle, que hierarquicamente reside em um nível superior a estes três componentes.

A representação do espaço deve possuir um relacionamento direto e harmônico com a tarefa principal do robô. Caso contrário, o próprio robô pode se mostrar inviável. Os próprios parâmetros de cada representação são decisivos. Por exemplo, uma resolução de representação muito "grossa" pode ignorar certos obstáculos, e em alguns casos até mesmo o próprio objetivo do robô. Ou ainda, a desconsideração das dimensões do robô móvel em relação aos obstáculos existentes podem causar danos irreparáveis ao robô.

Para uma maior elucidação da seção 4.4.1, é recomendada a leitura de [2], que possui uma abordagem matemática completa bem como suas extensões para as técnicas de grafo de visibilidade, diagrama de Voronoi, decomposições celulares e campo potencial. Para o complemento das demais técnicas de representação do espaço, o material produzido por Moreno [32] consitui um ponto de partida adicional. O material de Johansson[1] estende o assunto para sistemas multi-robóticos.

5 *Arquiteturas de Controle*

Os sistemas robóticos, em especial os robôs móveis, são a materialização de um conjunto de processos complexos, sistemas mecânicos, sensores, e dispositivos de comunicação. Os problemas inerentes à integração destes componentes em um robô real são sempre desafiadores. Esta complexidade, proveniente de várias fontes, por vezes tem dificultado e distraído pesquisadores da real tarefa, seja ela da manipulação de objetos ou do planejamento de trajetórias, por exemplo.

A carência de padrões, as idiossincrasias de cada componente, acabam proporcionando novos ramos de pesquisa, sempre perseguindo formas eficientes de abstração de toda esta complexidade, proporcionando a integração adequada de todos estes componentes a fim de atingir um objetivo específico na robótica. A complexidade subjacente a estes componentes de *software* e de *hardware* acaba, de alguma forma, resultando em restrições no tipo de tarefas que podem ser realizadas por um robô.

Desdes os primeiros robôs móveis existe a preocupação com o processo de colocar um robô em movimento, contornando todas as limitações da arquitetrua de controle do veículo. Como abordagem à complexidade do problema, geralmente opta-se pela decomposição deste em problemas menores, ou em camadas, o que é chamado de *decomposição horizontal* ou *decomposição funcional*[32], onde o problema de mover o robô é quebrado em componentes ou funções separadas, cada qual processada em sua devida ordem, sendo que a saída de um módulo serve de entrada para o módulo subseqüente.

5.1 Decomposição Funcional

A decomposição funcional ou horizontal é uma metodologia clássica *top-down* usada para o projeto de grande parte dos sistemas robóticos autônomos. O ambiente é representado usando um conjunto de ações, temporizações e eventos discretos[34] (Figura 34), tais como[32]:

Percepção: um módulo com a função de coletar informações do ambiente.

Modelagem: módulo responsável por estabelecer um modelo do ambiente a partir da percepção.

Planejamento: módulo que constrói o plano de ação do robô, a partir do modelo do ambiente.

Execução: um módulo que move o robô baseado no plano estabelecido.

Controle de movimento: módulo de "baixo nível" que controla diretamente os componentes (atuadores) do robô.

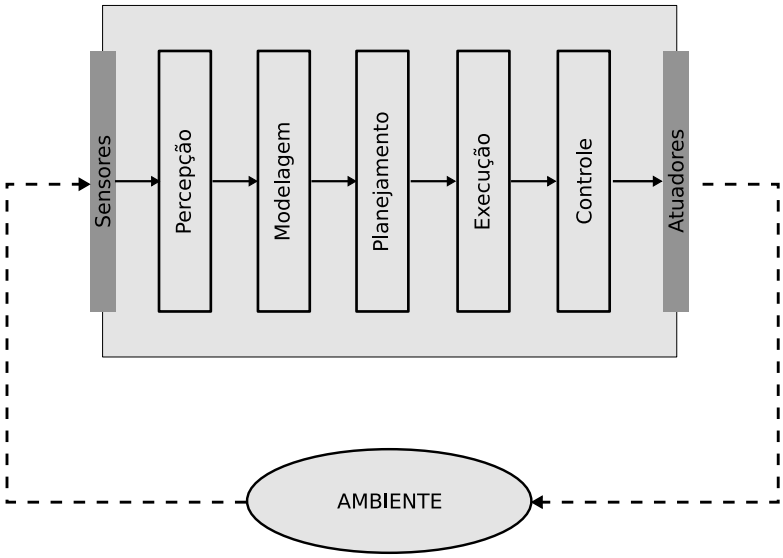

Figura 34: Decomposição operacional horizontal, tipicamente serial.

Dentro da decomposição funcional, cada módulo individual opera de uma forma determinista, formando o que é chamado de *ciclo de percepção-planejamento-ação*[12, 32].

A crítica a este tipo de decomposição geralmente converge na inaptidão da estrutura de reagir rapidamente, por exemplo, em qualquer caso de emergência. Para reagir a qualquer evento, seja ele planejado ou não, o robô deve perceber, modelar, planejar e só então agir, o que em alguns casos pode ser demorado demais[13]. Os sistemas decompostos horizontalmente tendem a ter tempos de latência maiores, além de geralmente considerarem um ambiente estático durante o intervalo do ciclo de percepção-planejamento-ação. Muitas vezes há a necessidade da pressuposição de que o robô é o único agente ativo no ambiente, e condições que violem estas premissas, como obstáculos móveis, podem causar problemas para a operação do robô.

A decomposição horizontal, ou funcional, acaba fazendo emergir dois novos problemas para o robô: como organizar a informação e representá-la de forma eficiente. Dois mecanismos básicos se destacam dentre os muitos existentes: sistemas hierárquicos e sistemas com quadro-negro.

5.1.1 Sistemas Hierárquicos

Sistemas hierárquicos decompõem o processo de controle em funções. Processos de "baixo nível" originam funções simples, que são agrupadas em processos de mais "alto nível", formando o controle do veículo. Este sistema decompõe a tarefa de controle em pequenas unidades, sempre minimizando a comunicação entre as unidades, com as tarefas sendo executadas de forma seqüencial.

5.1.2 Sistemas com Quadro-Negro

Sistemas com quadro-negro baseiam-se em uma área comum de comunicação (ou quadro-negro) compartilhada por cada processo computacional independente. Um aspecto fundamental deste sistema é estabelecer um mecanismo que suporte a comunicação eficiente entre os vários agentes computacionais fazendo uso do quadro-negro.

O sistema com quadro-negro proporciona um fraco acoplamento entre cada subsistema, permitindo a fácil troca de informações e uma consistente representação das informações que pode ser usada por cada subsistema.

Enquanto sistemas de quadro-negro porporcionam um mecanismo explícito de compartilhamento de informações, este recurso compartilhado pode levar a gargalos de processamento. A própria natureza assíncrona do sistema de quadro-negro o torna complexo.

A decomposição funcional se caracteriza por uma abordagem planejada, ou deliberativa, e isto é muito característico pelo ciclo de percepção-planejamento-ação. Alguns métodos que se caracterizam pela sua abordagem planejada são: a decomposição espacial, as representações geométricas, as representações topológicas, grafos de visibilidade e diagramas de Voronoi[3], dentre outros não citados. Todos eles apresentam duas características em comum, que os caracterizam como métodos planejados:

Pressuposição do ambiente estático: nenhum objeto, além do próprio robô, se move pelo ambiente, especialmente durante o ciclo de percepção-planejamento-ação.

Pressuposição do ambiente completamente conhecido: é necessário possuir o conhecimento global do ambiente, todos os obstáculos devem ter sua posição e formas conhecidos antes de qualquer planejamento, bem como a posição de todos os pontos objetivos.

Estas duas características trazem duas restrições inerentes a qualquer abordagem planejada[32]:

- A representação de ambiente não é completamente precisa, as leituras dos sensores podem apresentar ruídos.
- Quando aplicadas em ambientes com características dinâmicas, o sistema deve remodelar e pré-planejar todos os movimentos antes de executá-los, o que pode causar a prisão em uma espécie de *loop* infinito, onde o sistema permanecerá eternamente em planejamento, no caso de ambientes altamente dinâmicos.

5.2 Controle Reativo

Enquanto a decomposição horizontal ou funcional é baseada no ciclo de percepção-planejamento-ação, o controle reativo, a arquitetura de subsunção, ou o comportamento reflexivo conectam diretamente os sensores aos atuadores. O paradigma do controle reativo é baseado em modelos rudimentares de inteligência animal, onde a ação do robô é decomposta em comportamentos, ao invés de processos deliberativos de processamento. Mecanismos baseados em modelos comportamentais foram desenvolvidos na tentativa de contornar as limitações existentes na adaptação dos planejadores clássicos ao controle robótico de baixo nível. Em particular, os planejadores clássicos encontram problemas com[12]:

Objetivos Múltiplos: ao contrário dos planejadores clássicos, que buscam corresponder a apenas um objetivo geral, o controle de baixo nível deve corresponder a vários objetivos, simultaneamente. O controlador deve proporcionar ao veículo segurança enquanto executando um movimento, ao mesmo tempo que processa sinais de entrada, enquanto lida com tarefas de comunicação. Cada uma destas tarefas possui requisitos de tempo críticos e determinísticos.

Múltiplos Sensores: os sensores de um robô possuem restrições de tempo-real. Os dados devem ser lidos rapidamente, ou podem ser sobreescritos por leituras sub-

seqüentes. Adicionalmente, certo dado foi adquirido em um ponto particular do espaço-tempo e deve ser identificado como tal, ou corre o risco de ser analizado inapropriadamente. Diferentes sensores terão constantes de tempo diferentes; alguns retornando dados mais rapidamente, outros levando décimos de segundo para proporcionar qualquer leitura.

Robustez: sistemas de baixo nível devem ser relativamente insensíveis a dados conflitantes, confusos, ou dados incompletos obtidos de um ou mais sensores.

Extensibilidade: o sistema de controle deve ser facilmente modificável para lidar com mudanças em sensores, tarefas ou configurções de locomoção.

Comum a todos os sistemas reativos está o fato de que os objetivos não precisam ser representados explicitamente, podem ser codificados pelos comportamentos individuais que operam dentro do controlador. Comportamentos genéricos emergem das interações que tomam lugar entre os comportamentos individuais, leituras de sensores e o ambiente.

5.2.1 Arquitetura de Subsunção

Provavelmente uma das formas mais conhecidas de controle reativo é a *arquitetura de subsunção*[3], que consiste em um conjunto de módulos de comportamentos organizados hierarquicamente. Diferentes camadas na arquitetura são responsáveis por diferentes comportamentos[32, 35], onde as camadas de comportamento superiores controlam funções mais avançadas[12].

A arquitetura de subsunção introduz o conceito de *níveis de competência*[3, 12, 32] (Figura 35):

- evitar contato com obstáculos;
- navegar no ambiente sem atingir os obstáculos;
- "explorar" o ambiente;
- construir um mapa do ambiente;
- perceber alterações no ambiente;
- deliberar sobre o ambiente em termos de objetos identificáveis;
- formular e executar planos;

- deliberar acerca do comportamento e modificar os planos apropriadamente.

Cada nível de competência se constrói sobre os anteriores[32]. A arquitetura de subsunção consiste em um conjunto de comportamentos independentes que mapeiam diretamente o sensoreamento com a ação[13]. O mapeamento hierárquico, estaticamente estabelecido, dá prioridade implícita às camadas inferiores, em detrimento das superiores[35].

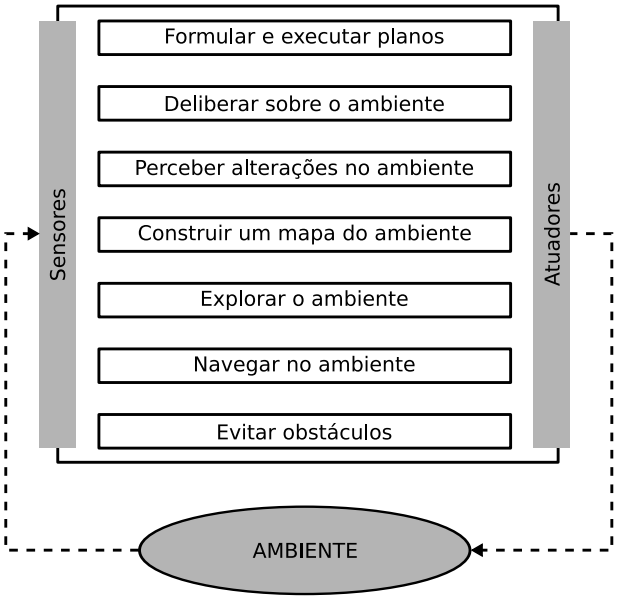

Figura 35: Decomposição da arquitetura de subsunção, tipicamente paralela.

O comportamento final emergente do sistema é um conjunto de reações resultantes de vários módulos providos para atingir cada um dos níveis de competência[35]. O modo incremental da construção da arquitetura de subsunção torna relativamente fácil implementar e permitir experimentação e flexibilidade no projeto do sistema. Entretanto, dada a aleatoriedade nas quais as condições que disparam os vários comportamentos ocorrem, o comportamento que emerge pode ser de difícil explicação, o que torna complexa sua validação em termos de desempenho, por exemplo.

As primeiras implementações da arquitetura de subsunção foram baseadas em uma coleção de processadores, cada um simulando uma máquina de estados finita. Esta estrutura evoluiu, porém, em diferentes formas de implementação, como um único processador e fazendo o uso da generalização das habilidades computacionais para cada comportamento.

5.2.2 Controle Contínuo

Muito do controle reativo é baseado em controladores discretos. Uma alternativa ao controle discreto é simplesmente considerar o desenvolvimento do controle de baixo nível

como o desenvolvimento de um controlador clássico, que transforma a entrada contínua em alguma outra forma de saída também contínua. Uma vantagem desta alternativa é a vasta bibliografia sobre teoria de controle hoje disponível. Porém, em alguns casos, a complexidade matemática inerente ao controle clássico pode tornar esta abordagem trabalhosa. Há a necessidade da completa modelagem matemática de todos os termos relevantes para controle.

5.3 Controle de Alto Nível

Ficou claro desde os primórdios da construção de robôs autônomos que a representação procedimental (do inglês *procedural*) não se mostra adequada para atingir objetivos complexos. Conseqüentemente, mecanismos alternativos, muitos deles baseados na lógica formal, têm sido propostos com a finalidade de prover controle de alto nível.

Provavelmente, a escolha mais intuitiva para a tarefa do controle de alto nível ou planejmento é tratar o robô como uma entidade abstrata e fazer uso de técnicas clássicas de inteligência artificial na especificação de tarefas complexas.

5.4 Conclusões

Pode-se dizer que hierárquica e lógicamente a arquitetura de controle reside um nível acima dos sistemas de percepção, atuação e representação do espaço. Ela faz uso das variáveis de saída do sistema de controle, das informações armazenadas na representação espacial para fornecer as entradas para o sistema de atuação, locomoção.

A arquitetura do sistema de controle, como qualquer outro componente do robô móvel, deve ser mais um catalizador para a realização da tarefa. Esta arquitetura também é a principal responsável pela abordagem de modelagem dos sistemas, ela é o fator principal do uso de modelos como caixa preta, onde apenas as variáveis de entrada (sensores) e saída (atuadores) são relevantes para a modelagem; caixa cinza, onde além das variáveis de entrada e saída, algumas das variáveis internas também são relevantes, ou caixa branca, onde todas as variáveis e parâmetros são relevantes.

6 Curvas Paramétricas

Uma importante aplicação da representação de curvas parametricamente é a descrição do caminho tomado por um objeto como seu movimento ao longo do tempo[36]. Este tipo de representação teve seu início especialmente na computação gráfica. Por exemplo, no projeto de uma animação, a trajetória de uma câmera ao longo de uma cena deve ser especificada para cada instante. Sua localização é dada por $P(t)$ para cada instante t. O projetista poderia então escolher uma função $P(t)$ que descreve a trajetória desejada da câmera.

A mesma analogia pode ser feita para a descrição de uma trajetória de um robô móvel (Figura 36). Seria desejável a elaboração de uma função parametrizada $P(t)$ que descrevesse completamente a trajetória de um robô móvel, inclusive desviando de qualquer obstáculo, e dirigida ao ponto objetivo.

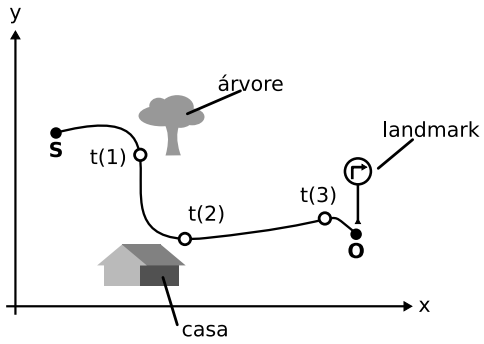

Figura 36: Exemplo de uma curva para a descrição do caminho de um objeto.

Além da especificação da posição do robô móvel em função do tempo, deve-se garantir que o robô se mova suavemente ao longo da trajetória definida por $P(t)$, sem qualquer movimento súbito, o que poderia dificultar ou prejudicar o controle do veículo. Esta imposição encerra certas restrições ao vetor velocidade $\mathbf{P}'(t)$.

6.1 Descrevendo Curvas por Meio de Polinômios

Os polinômios são objetos matemáticos fundamentais, que são freqüentemente usados na computação, devido ao seu comportamento previsível e eficiente aproveitamento computacional[36, 37].

Um *polinômio de grau L em t* é uma função definida por:

$$a_0 + a_1 t + a_2 t^2 + \cdots + a_L t^L \tag{6.1}$$

onde as constantes a_0, a_1, ..., a_L são os *coeficientes* do polinômio, cada qual associado a uma das potências de t. O *grau* do polinômio é a maior potência de t. Para que (6.1) seja de grau L, a_L deve ser diferente de zero. A *ordem* do polinômio é o número de coeficientes existentes, para (6.1), a ordem é $(L + 1)$. A ordem de um polinômio é sempre maior do que seu grau.

O grau de um polinômio também rege seu comportamento, por exemplo:

Polinômios de Grau 1: *lineares*, se comportam como uma linha reta.

Polinômios de Grau 2: *quadráticos*, parametricamente na forma:

$$x(t) = at^2 + 2bt + c \tag{6.2}$$

$$y(t) = dt^2 + 2et + f \tag{6.3}$$

sempre terão forma parabólica, para qualquer escolha de $a, b, \ldots, f$. Não é possível obter qualquer outra forma quadrática a partir de (6.2) e (6.3), tais como uma elipse ou uma hipérbole.

Indo em sentido contrário, na busca de formas quadráticas implícitas – polinômios de grau 2 para x e y – a partir da definição oriunda da geometria analítica para formas quadráticas implícitas

$$F(x, y) = Ax^2 + 2Bxy + Cy^2 + Dx + Ey + F \tag{6.4}$$

Para as constantes $A, \ldots, F$, a forma descrita pela curva $F(x, y) = 0$ é uma seção cônica. A cônica representada é uma função do *discriminante* $AC - B^2$:

- $AC - B^2 > 0$, a curva é uma *elipse*.
- $AC - B^2 = 0$, a curva é uma *parábola*.
- $AC - B^2 < 0$, a curva é uma *hipérbole*.

Assim, $x^2 + xy + y^2 - 1$ é a função implícita de uma elipse ($AC - B^2 = 0,5$), por exemplo.

Ao invés de definir uma curva diretamente por meio dos parâmetros que compõem seu polinômio, seria intuitivamente mais proveitoso definí-las através de uma coleção de *pontos de controle*, espalhados no espaço[1][36, 37, 38, 39], de tal forma que um algoritmo gerasse os pontos ao longo da curva, independemente da função implícita que ela satisfaz. Esta é uma abordagem mais natural do que o projeto de curvas pela abordagem puramente matemática[36, 38, 39].

Polinômios racionais mostram como a definição de alguns pontos podem definir a forma de uma curva. Inclusive, a seção cônica pode ser representada exatamente como uma razão de dois polinômios quadráticos.

Consideremos a parametrização na qual $x(\bullet)$ e $y(\bullet)$ são definidos como a razão entre dois polinômios. Para o caso do polinômio quadrático, em sua forma paramétrica particular

$$P(t) = \frac{P_0(1-t)^2 + 2\omega P_1 t(1-t) + P_2 t^2}{(1-t)^2 + 2\omega t(1-t) + t^2} \tag{6.5}$$

onde P_0, P_1 e P_2 são quaisquer três pontos no plano. Estes três pontos são chamados de *pontos de controle*, uma vez que eles controlam a forma da curva. Para o espaço bidimensional, a equação (6.5) é uma contração de

$$\begin{aligned} \alpha &= (1-t) \\ \beta &= \omega t(1-t) \\ (x(t), y(t)) &= \left(\frac{x_0\alpha^2 + 2x_1\beta + x_2 t^2}{\alpha^2 + 2\beta + t^2}, \frac{y_0\alpha^2 + 2y_1\beta + y_2 t^2}{\alpha^2 + 2\beta + t^2} \right) \end{aligned} \tag{6.6}$$

onde x_0 e y_0 são componentes de P_0, e similarmente para os demais dois pontos. Os coeficientes dos polinômios quadráticos no numerador são componentes dos pontos de controle. Os polinômios dos denominadores para $x(\bullet)$ e $y(\bullet)$ são iguais, e quadráticos,

[1]Espaço bidimensional para definição de curvas. Espaço tridimensional para a definição de superfícies.

mas não dependem dos pontos de controle. Eles dependem sim de um parâmetro de *peso*, ω. $P(t)$ é uma combinação linear dos pontos de controle. Para $t = 0$, o lado direito da equação (6.6) se resume a (x_0, y_0), isto é, a curva *interpola* o ponto P_0. Para $t = 1$, a curva interpola o ponto P_2. Para t entre $t = 0$ e $t = 1$, $P(t)$ é influenciado pelos três pontos.

A curva define seções cônicas em função do peso ω, da seguinte forma:

- Se $\omega < 1$, a curva é uma elipse.
- Se $\omega = 1$, a curva é uma parábola.
- Se $\omega > 1$, a curva é uma hipérbole.

Desta forma pode-se gerar uma seção cônica parametricamente (Figura 37), sem a necessidade de recorrer a funções trigonométricas.

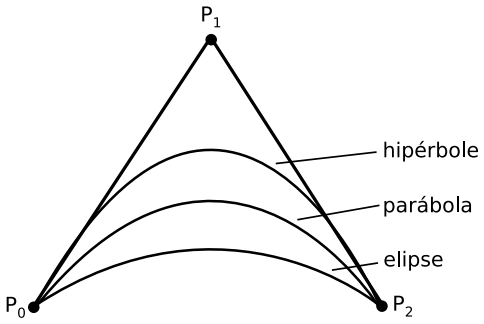

Figura 37: Geração de cônicas a partir de polinômios racionais quadráticos.

6.2 Curvas de Bezier

As *curvas de Bezier* foram independentemente desenvolvidas por Paul de Casteljau em 1959 e Pierre Bezier em meados de 1962. Elas foram incorporadas ao campo da computação no momento que duas montadoras automobilísticas francesas, a Renault e a Citröen, pioneiramente passaram a usar as curvas de Bezier para o projeto de automóveis.

6.2.1 O Algoritmo de Casteljau

O algoritmo de Casteljau usa uma seqüência de pontos, P_0, P_1, P_2,..., para determinar um valor do ponto $P(t)$ para t variando de 0 a 1. O algoritmo possiblita a criação de uma

curva a partir de um conjunto de pontos. Mudando os pontos deste conjunto, muda-se a forma da curva. A construção da curva é baseada em uma seqüência de passos de "interpolação" (*tweening*).

Por exemplo, a interpolação de três pontos – P_0, P_1 e P_2 – para a criação de uma parábola. Escolhe-se um valor para t entre 0 e 1, digamos $t = 0,3$. Localizamos um ponto A que está a uma fração t de distância no segmento orientado de P_0 a P_1. Localizamos outro ponto, B, a uma fração t de distância no segmento orientado de P_1 a P_2.

Os pontos A e B podem ser expressados por:

$$A(t) = (1-t)P_0 + tP_1 \tag{6.7}$$

$$B(t) = (1-t)P_1 + tP_2 \tag{6.8}$$

Repetindo o processo de interpolação linear nestes dois pontos, A e B, usando o mesmo valor de $t = 0,3$, encontramos um ponto $P(t)$ que se localiza a uma fração t do segmento orientado de A a B:

$$P(t) = (1-t)A + tB \tag{6.9}$$

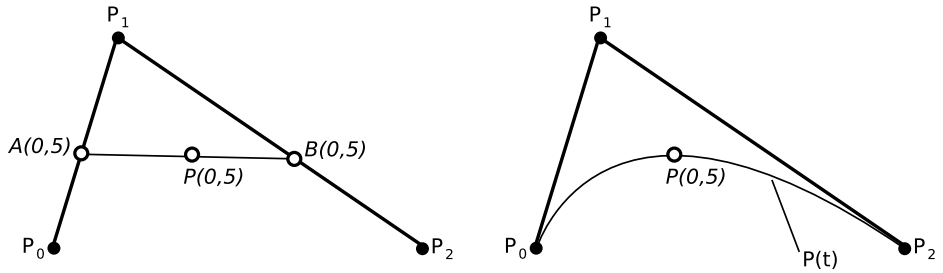

Figura 38: Exemplo do algoritmo de Casteljau aplicado a três pontos.

Se $t = 0,5$, $P(0,5)$ é o ponto mediano dos pontos medianos dos três pontos P_0, P_1 e P_2. Se este processo é realizado para todos os valores de t em [0,1], a curva $P(t)$ será gerada (Figura 38). A forma paramétrica para e equação (6.9) é obtida pela substituição da equação (6.7) e (6.8) diretamente na equação (6.9):

$$P(t) = (1-t)^2 P_0 + 2t(1-t)tP_1 + t^2 P_2 \tag{6.10}$$

A forma paramétrica de $P(t)$ é quadrática em t, então a curva tem uma forma parabólica.

O algoritmo de Casteljau pode ser usado para um número maior de pontos de controle, por exemplo quatro pontos, P_0, P_1, P_2 e P_3 (Figura 39). Para um dado valor de t A é posicionado a uma fração t do segmento P_0 a P_1; o mesmo para os pontos B e C em relação aos segmentos P_1 a P_2 e P_2 a P_3, respectivamente. Então, o ponto D é posicionado a uma fração t de distância do segmento A a B, o mesmo é feito para um ponto E, em relação ao segmento B para C. Finalmente, o processo de interpolação linear termina com a determinação do ponto P, localizado a uma fração t de distância do segmento D a E.

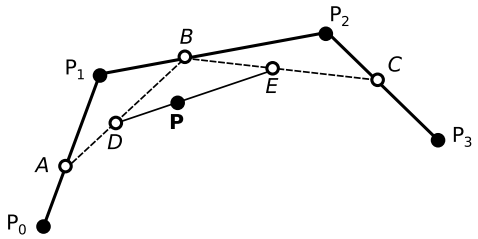

Figura 39: Exemplo do algoritmo de Casteljau aplicado a quatro pontos.

Se este processo é repetido para todo valor de t pertencente ao intervalo [0,1], a curva $P(t)$ se inicia no ponto P_0, é "atraída" em direção de P_1 e P_2 e termina em P_3. A forma paramétrica da curva de Bezier para quatro pontos é:

$$P(t) = P_0(1-t)^3 + P_1 3(1-t)^2 t + P_2 3(1-t)t^2 + P_3 t^3 \tag{6.11}$$

Que é um polinômio cúbico em t. Cada ponto de controle P_i é ponderado por um polinômio cúbico, e os pontos ponderados são adicionados. Os termos envolvidos são chamados de *polinômios de Bernstein*:

$$\begin{aligned} B_0^3(t) &= (1-t)^3, \\ B_1^3(t) &= 3(1-t)^2 t, \\ B_2^3(t) &= 3(1-t)t^2, \\ B_2^3(t) &= t^3 \end{aligned} \tag{6.12}$$

Que podem ser obtidos pela expansão da expressão $[(1-t)+t]^3$. $P(t)$ pode então ser

apresentado como:

$$P(t) = \sum_{k=0}^{3} P_k B_k^3(t) \tag{6.13}$$

Este racioncínio permite a generalização do algoritmo de Casteljau para um número arbitrário de pontos $L+1$, P_0, P_1, ..., P_L.

6.2.2 O Problema do Controle Local

A princípio, as curvas de Bezier fornecem todos os artifícios necessários para o projeto de qualquer tipo de curva. Um número infinito de curvas suaves pode ser projetado pela disposição de pontos de controle adequadamente no plano. Entretanto, o grau dos polinômios de Bernstein usados é diretamente proporcinoal ao número de pontos de controle. Uma curva de Bezier com $L+1$ pontos de controle possui grau L. Polinômios de elevada ordem são custosos computacionalmente e vulneráveis a problemas de arredondamento.

Uma outra restrição é a inflexibilidade das curvas de Bezier oferecer *controle local* sobre a forma da curva[36]. O problema reside no fato de que a alteração em um simples ponto de controle resulta em uma alteração de toda a forma da curva (Figura 40). Este relacionamento intrínseco surge da natureza dos polinômios de Bernstein, cada um dos polinômios é ativo no intervalo [0,1], itervalo este chamado *suporte*[2]. Devido à natureza dos polinômios de Bernstein, e ao fato de que a curva final é um resultado da influência destes polinômios, segue o fato de que cada ponto de controle tem influência durante todo o intervalo de t compreendido entre [0,1].

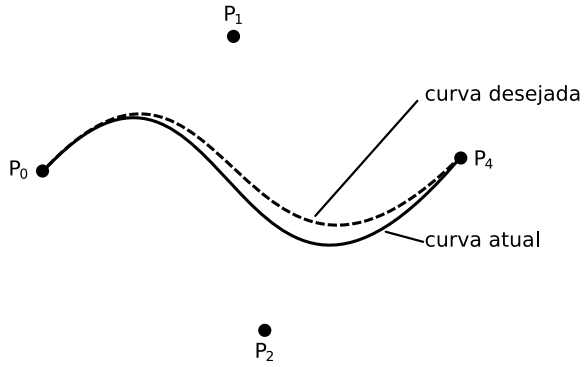

Figura 40: Edição de porções de uma curva.

Uma situação favorável ao controle local seria um conjunto de *funções de mistura*

[2]O intervalo em que uma função é diferente de zero.

(*blending functions*), cada uma, com suporte em apenas uma parte do intervalo de [0,1], facilmente computáveis numericamente, suaves a ponto de produzir as curvas desejadas (Figura 41).

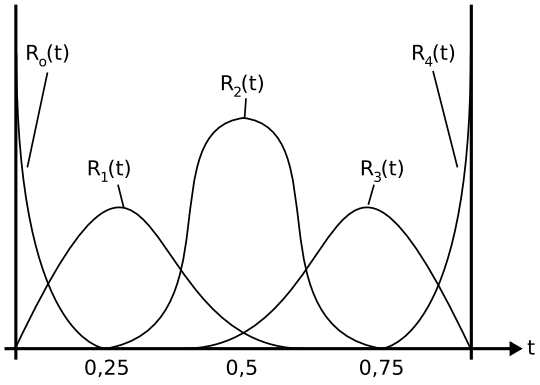

Figura 41: Funções de mistura e suporte concentrado.

6.3 *Splines*

A busca por funções de mistura começa por polinômios de baixa ordem. Por exemplo, a busca por um polinômio cúbico com a forma necessária (Figura 42) e que corresponda às necessidades de controle local, baixa complexidade numérica e computacional, definido pela função:

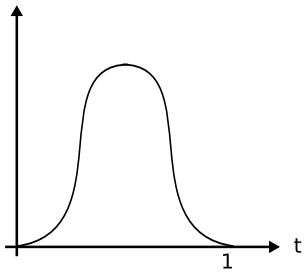

Figura 42: Polinômio cúbico desejado.

$$R(t) = at^3 + bt^2 + ct + d \tag{6.14}$$

passa pela verificação se existe uma escolha dos coeficientes de $R(t)$ e de sua derivada primeira zero em $t = 0$ e $t = 1$,

$$R(0) \;=\; d = 0$$

$$\begin{aligned} R(1) &= a+b+c+d=0 \\ R'(0) &= c=0 \\ R'(1) &= 3a+2b+c=0 \end{aligned} \tag{6.15}$$

Estas condições forçam que $a=b=c=d=0$, isto é, não existe uma tal curva para a forma cúbica. Não há flexibilidade suficiente na forma cúbica para a função de mistura.

Na busca por mais flexibilidade, pode-se compor vários polinômios de baixo grau, cada um definido em diferentes intervalos de t, para se obter uma curva final mais complexa. Estes polinômios são chamados de *polinômios compostos* (*piecewise polynomials*) (Figura 43). Por exemplo, uma função $g(t)$ definida pelos seguinte segmentos polinomiais

$$\begin{aligned} a(t) &= \frac{1}{2}t^2 \\ b(t) &= \frac{3}{4}-\left(t-\frac{3}{2}\right)^2 \\ c(t) &= \frac{1}{2}(3-t)^2 \end{aligned} \tag{6.16}$$

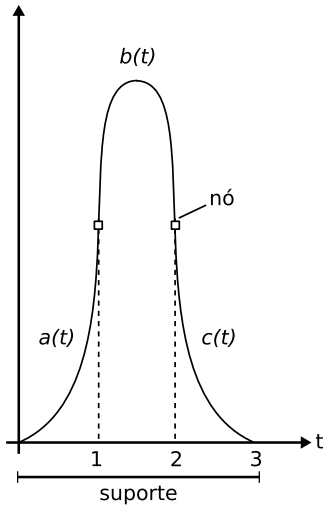

Figura 43: Componentes de um polinômio composto.

O suporte de $g(t)$ está contido ao intervalo [0,3]; $a(t)$ é suportada em [0,1], $b(t)$ em [1,2] e $c(t)$ em [2,3]. Os pontos em que cada par de segmentos individuais se encontram são chamados *juntas* (*joints*), e os valores de t aos quais as juntas existem são chamados de *nós* (*knots*). $g(t)$ possui quatro nós: 0, 1, 2 e 3.

Outra característica importante, e que é relevante para aplicações como trajetórias, é

a continuidade de $g(t)$ ao longo de seu suporte. Como $g(t)$ é composta por polinômios, a função certamente é contínua dentro dos limites de cada polinômio, resta checar a continuidade nas juntas. Isto pode ser verificado fazendo uso de (6.16); $a(1) = b(1) = 1/2$ e $b(2) = c(2) = 1/2$. A mesma continuidade pode ser verificada para a derivada primeira de $g(t)$.

A forma definida por $g(t)$ é um exemplo de resultado de uma *função spline*, uma função polinomial composta que possui "suavidade" suficiente[3]. O exemplo $g(t)$ é spline quadrática, é um polinômio composto de grau 2 e possui continuidade em sua primeira derivada.

6.3.1 Construíndo Um Conjunto de Funções de Mistura

Uma forma de construir um conjunto de funções de mistura nos moldes da equação (6.12) é o uso de versões deslocadas de $g(t)$, onde cada função de mistura $g_k(t)$ é formada pela translação da forma básica de $g(t)$ por uma certa quantidade (Figura 44):

$$g_k(t) = g(t-k), \quad para\ k = 0, 1, \ldots \tag{6.17}$$

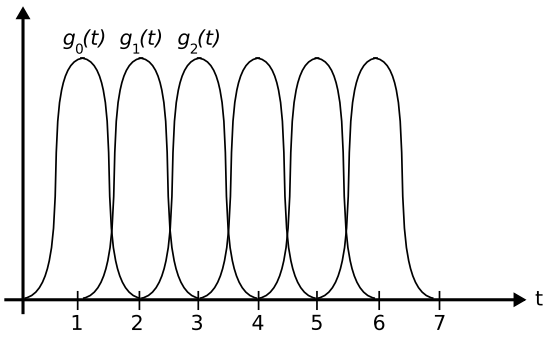

Figura 44: Curvas fechadas baseadas em *splines*.

Estas versões deslocadas formam um conjunto válido de funções de mistura. A curva é então determinada por:

$$V(t) = \sum_{k=0}^{L} P_k g(t-k) \tag{6.18}$$

para $L+1$ pontos de controle e funções de mistura.

[3]Continuidade suficiente, inclusive nas derivadas relevantes de cada aplicação.

6.3.2 Funções Base

O método envolvendo translações de $g(t)$ já fornece uma ferramenta robusta para o projeto de curvas, mas ainda é possível torná-lo ainda mais versátil. Pode-se desejar funções de mistura mais influentes ou continuidades em derivadas além da primeira. A busca segue por funções de influência genéricas que satisfaçam a um conjunto ainda mais exigente de requisitos. Continua-se com a forma paramétrica

$$P(t) = \sum_{k=0}^{L} P_k R_k(t) \tag{6.19}$$

baseada em $L+1$ pontos de controle e $L+1$ funções de mistura. Também é desejável o uso de polinômios compostos para as funções de mistura, mas estes polinômios podem ser definidos em uma seqüência genérica de nós, chamado *vetor de nós*,

$$\mathbf{T} = \{t_0, t_1, t_2, \ldots\} \tag{6.20}$$

Que é simplesmente uma lista de valores de nós, presumidamente não-descrescentes, isto é, $t_i \leq t_{i+1}$. Alguns dos nós podem possuir o mesmo valor.

Cada função de mistura $R_k(t)$ é um polinômio composto cujo suporte se inicia em t_k, é diferente de zero durante algumas variações de t, e depois retorna a zero (Figura 45).

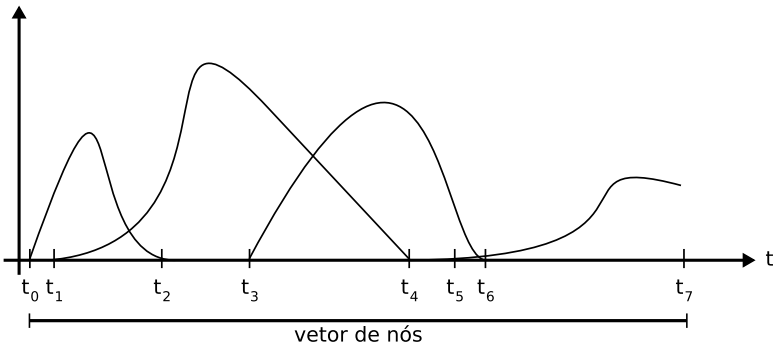

Figura 45: Generalização do vetor de nós e funções de mistura.

Dado um vetor de nós, existe uma família de funções de mistura que pode ser usada para gerar cada curva spline possível, definida pelo vetor de nós. Tal família é chamada de *base* para *splines*, significando que qualquer curva spline pode ser definida por uma soma no formato da equação (6.19) e a definição apropriada de um polígono de controle. Existe uma base em particular cujas funções de mistura possuem o menor suporte, oferecendo o maior controle local. São as *B-splines*, onde B é derivado da palavra *base*.

6.4 *B-Splines*

É desejável que as *B-splines* satisfaçam a todos os requisitos anteriormente apresentados, além de embutir forte sentido intuitivo e uma fácil implementação computacional[37, 36, 40]. Existe uma fórmula que define *B-splines* de qualquer ordem. É uma relação recursiva que é de fácil implementação computacional e numericamente previsível.

Cada função *B-spline* é baseada em polinômios de certa ordem m. Se $m = 3$, os polinômios serão de ordem 3 e de grau 2, assim, serão *B-splines* quadráticas. Se $m = 4$, os polinômios serão de grau 3, ou cúbicos. O caso quadrático e cúbico são os mais comuns na literatura, embora a formalização permita a exploração de *B-splines* de qualquer ordem. A k-ésima função de mistura *B-spline*, de ordem m, é definida por $N_{k,m}(t)$. Desta forma, a equação (6.19) assume a forma de:

$$P(t) = \sum_{k=0}^{L} P_k N_{k,m}(t) \tag{6.21}$$

A fórmula fundamental para a função *B-spline* $N_{k,m}(t)$ é

$$N_{k,m}(t) = \left(\frac{t - t_k}{t_{k+m-1} - t_k}\right) N_{k,m-1}(t) + \left(\frac{t_{k+m} - t}{t_{k+m} - t_{k+1}}\right) N_{k+1,m-1}(t) \tag{6.22}$$

para $k = 0, 1, \ldots, L$. Esta é uma definição recursiva, especificando como construir a função de m-ésima ordem a partir de duas funções *B-spline* de ordem $(m - 1)$. Devido ao caráter recursivo, a função de primeira ordem deve ser definida *a priori*, possuindo o valor 1 dentro de seu intervalo:

$$N_{k,1}(t) = \begin{cases} 1 & se \;\; t_k < t \leq t_{k+1} \\ 0 & caso\ contrario \end{cases} \tag{6.23}$$

Por exemplo, $N_{1,2}(t)$ é um pulso triangular iniciando em $t = 1$ e terminando em $t = 3$. Baseado em (6.22), a função $N_{k,m}(t)$ tem suporte no intervalo $[t_k, t_{k+m}]$. Um, ou os dois, denominadores da equação (6.22) podem se tornar zero para certas escolhas de nós. Sempre que isso acontecer, as funções correspondentes de menor ordem, $N_{k,m-1}(t)$ ou $N_{k+1,m-1}(t)$, também serão zero. Devido a isso é possível adotar a regra de que qualquer termo com denominador zero seja valorado como zero. O seguinte algoritmo apresenta a sugestão de pseudo-código para (6.22).

Algoritmo 3: Pseudo-código do algoritmo *B-spline*.

entrada: k, m, t, nós[]

saída : ponto

$denom1 \leftarrow 0.0$;

$denom2 \leftarrow 0.0$;

$sum \leftarrow 0.0$;

se $m = 1$ **então**
| retornar(t $\geq$ nós[k] $\wedge$ t $<$ nós[k+1]);

fim

denom1 $\leftarrow$ nós[$k + m$-1] - nós[k];

se $denom1 \neq 0.0$ **então**
| sum $\leftarrow$ (t - nós[k]) $\cdot$ B-spline($k, m-1, t$, nós[]);

fim

denom2 $\leftarrow$ nós[$k + m$] - nós[$k + 1$];

se $denom2 \neq 0.0$ **então**
| sum $\leftarrow$ sum + (nós[$k + m$]-t) $\cdot$ B-spline($k + 1, m - 1, t$, nós[]);

fim

retornar (sum);

6.4.1 Vetor de Nós Padrão

Uma escolha especial para vetor de nós se tornou um padrão *de facto* no projeto de curvas. Um vetor que permite a interpolação do primeiro e último pontos de controle, permitindo uma forma mais intuitiva da curva e podendo predizer onde a curva começará e terminará.

O *vetor de nós padrão* para uma *B-spline* de ordem m começa e termina com um nó de multiplicidade m, e usa espaçamento unitário para os nós restantes. Por exemplo, de posse de oito pontos de controle, e a necessidade de uma *B-spline* cúbica ($m = 4$). O vetor de nós padrão teria a seguinte forma:

$$\mathbf{T} = \{0, 0, 0, 0, 1, 2, 3, 4, 5, 5, 5, 5\} \tag{6.24}$$

O vetor de nós padrão para $L+1$ pontos de controle e *B-splines* de ordem m é definido pela seguinte regra:

1. Existem $L + m + 1$ nós, denotados por $t_0, \ldots, t_{L+m}$.

2. Os primeiros m nós, $t_0, \ldots, t_{m-1}$, compartilham o valor zero.

3. Os nós $t_m, \ldots, t_L$ crescem em incrementos unitários, a partir de 1 até $L - m + 1$.

4. Os m nós finais, $t_{L+1}, \ldots, t_{L+m}$, são iguais a $L - m + 2$.

6.5 Conclusões

As curvas paramétricas fazem uso da facilidade do controle do comportamento de polinômios para a descrição de curvas "suaves". Estas curvas podem representar desde superfícies a trajetórias de um corpo em movimento. É devido a esta facilidade de controle do comportamento destas curvas que elas podem ser usadas para o planejamento de trajetórias. Uma vez dispostos apropriadamente, os pontos de controle podem definir uma trajetória que inclua os pontos de origem e destino, além de desviar dos obstáculos existentes no ambiente.

Algumas características das curvas devem ser respeitadas em função de sua aplicação. Se elas, além da trajetória, devem agregar informações de velocidade, tais curvas devem ser contínuas após a primeira derivada. Se há a necessidade de agregar, além da velocidade, a aceleração do corpo em movimento, elas devem ser contínuas também após a segunda derivada. É esta continuidade nas diversas derivadas que define a "suavidade" de uma curva ou trajetória nos seus parâmetros de posição, velocidade e aceleração.

7 *Inteligência Artificial*

O campo da *inteligência artificial* (IA) busca compreender as entidades ou seres ditos inteligentes. Ao contrário da filosofia e da psicologia, que também se concentram na inteligência, a IA busca construir entidades inteligentes, além de entendê-las; é uma ciência com espírito de síntese, a exemplo da robótica.

A IA nasceu em 1956 em uma conferência de verão no Dartmouth College, Estados Unidos da América. Na proposta dessa conferência, escrita por John McCarthy (Dartmouth), Marvin Minsky (Harward), Nathaniel Rochester (IBM) e Claude Shannon (Bell Laboratories) e submetida à fundação Rockfeller, consta a intenção dos autores de realizar "um estudo durante dois meses, por dez homens, sobre o tópico denominado inteligência artificial". Ao que tudo indica, esta parece ser a primeira menção oficial da expressão "Inteligência Artificial"[41]. Desde então a IA gerou polêmica, a começar pelo seu próprio nome, considerado presunçoso por alguns, até a definição de seus objetivos e metodologias. O desconhecimento dos princípios que fundamentam a inteligência, por um lado, e dos limites práticos da capacidade de processamento dos computadores, por outro, levou periodicamente a promessas exageradas e às inevitáveis decepções.

A IA lida com um dos últimos grandes desafios da humanidade. Como é possível que um pequeno, por vezes lento, cérebro biológico seja capaz de perceber, entender, predizer e manipular um mundo muito maior e complicado quanto ele mesmo? Como proceder na construção de um dispositivo com tais características? Enquanto na física descobertas pertencem a nomes como Galileu, Newton ou Einstein, dentre outros expoentes, a IA ainda está à espera de seu grande gênio, da inteligência que vai compreender todas as outras. A inteligência artificial é o estudo de como fazer os computadores realizarem coisas que, até o momento, as pessoas fazem melhor[42], busca prover máquinas com a capacidade de realizar algumas atividades mentais do ser humano[43].

O estudo da inteligência tem sido uma das mais antigas disciplinas. Por mais de 2000 anos os filósofos têm tentado descobrir como ver, aprender, lembrar, relembrar e entender

pode ou deveria ser feito. Com o advento da computação em meados da década de 1950, o aprendizado deixa de ser alvo de especulação para algo passível de experimentação, uma ciência completa. São utilizadas máquinas com algum recurso computacional, de variadas arquiteturas, que permitem a implementação de rotinas não necessariamente algorítmicas. As atividades realizadas por essas máquinas podem envolver a senso-recepção (como tato, audição e visão), as capacidades intelectuais (como aprendizado de conceitos e de juízos, raciocínio dedutivo e memória), a linguagem (como as verbais e gráficas) e a atenção (decisão no sentido de concentrar as atividades sobre um determinado estímulo).

O estudo da inteligência artificial é algo intrigante, pois de certa forma possibilita aprender-se algo mais sobre nós mesmos. Como exemplo, basta que se mencionem as pesquisas no campo da linguagem natural, das redes neurais artificiais, da lógica, da robótica, enfim, qualquer área da IA. Em termos de tecnologia, a IA permite que máquinas possam realizar tarefas complexas no lugar do operador humano, liberando-o de atividades enfadonhas, insalubres ou inseguras. Também pode aumentar a eficiência do ser humano na sua interação com equipamentos sofisticados. Permite ainda que conhecimentos possam ser compartilhados por muitas pessoas, sem que haja necessidade de consultas a especialistas.

Mesmo com tanto potencial ainda é difícil definir o que é a inteligência artificial. A maior parte das definições variam substancialmente nas mais diferentes dimensões. Vão desde idéias como *processos pensantes* e *argumentação*, até *comportamento*. Também existem definições que medem o sucesso tomando como referência o desempenho humano. Outras definições medem o sucesso levando em consideração um conceito ideal de inteligência, algo que pode ser chamado de *racionalidade*. Eventualmente, todas as definições acabam convergindo em algum ponto ou momento. Nem mesmo a inteligência manifestada na natureza possui uma definição definitivamente aceita.

Algumas definições para a IA são:

- o estudo das faculdades mentais através do uso de modelos computacionais[44];
- o estudo de como fazer os computadores realizarem tarefas que, no momento, são feitas melhor por seres humanos[42];
- o estudo das idéias que permitem habilitar os computadores a fazerem coisas que tornam as pessoas inteligentes[45];
- o campo de conhecimentos onde se estudam sistemas capazes de reproduzir algumas das atividade mentais humanas[46];

- a parte da ciência da computação que compreende o projeto de sistemas computacionais que exibam características associadas, quando presentes no comportamento humano, a inteligência[47];
- tornar os computadores mais úteis e compreender os princípios que tornam a inteligência possível[48].

7.1 IA Aplicada à Robótica Móvel

Para o pesquisador, os problemas enfrentados pelo robô móvel podem parecer tão lógicos quanto aqueles enfrentados pelo computador que joga jogos de estratégia ou planeja rotas de transporte. Dado um objetivo, o robô deve formular uma solução baseada nos caminhos disponíveis. Metodologias de projeto *top-down* sugerem que decisões estratégicas tomadas pelo robô vão requerer a execução de segmentos individuais da trajetória, que por sua vez vão requisitar o controle dos motores à aquisição de dados. A operação de tal sistema pode ser descrita como um ciclo da consulta do modelo do mundo, a geração do caminho, a saída do atuador, leituras dos sensores e revisão do modelo do mundo. É o ciclo de percepção-planejamento-ação, descrito anteriormente[49].

Para robôs da indústria de manufatura, por exemplo, o modelo tradicional de percepção-planejamento-ação funciona a contento. Peças chegam em uma linha de montagem controladas por um computador e são posicionadas precisamente na frente do robô. O conjunto de tarefas de montagem é lido da memória e o robô realiza as tranformações cinemáticas inversas requeridas para determinar as trajetórias de cada grau de liberdade. *Encoders*[1] provêm o *feedback*[2] de posição à medida que o movimento do braço é realizado.

O sucesso das técnicas estabelecidas de robótica e inteligência artificial pode ser atribuído ao conjunto finito de aplicações às quais elas têm sido aplicadas. Tome-se como exemplo um jogo de xadrez de computador: em qualquer turno, o computador sabe a posição de cada peça no tabuleiro. Ele também dispõe de regras referenciando cada movimento possível, para cada peça. Isto permite ao computador realizar uma busca de cada configuração possível do tabuleiro dentro de uma distância a partir do estado presente, e calcular qual opção provê o maior ganho estratégico. O modelo do mundo é completo o suficiente para o computador considerar efetivamente as opções relevantes e as conseqüências de suas decisões.

[1] Codificadores digitais de posição.
[2] Realimentação.

7.1.1 Desempenho de Sistemas Simbólicos no Mundo Real

Infelizmente, o ambiente caótico ao redor de um robô móvel é virtualmente impossível de ser descrito como um modelo completo de mundo. Esta é uma característica que contrasta com o mundo dos robôs estacionários, comuns na indústria. Enquanto um braço manipulador pode manter um registro de sua posição com base em seus *encoders*, um para cada articulação, o sistema de mobilidade de um robô móvel está sujeito a deslizamento em relação à superfície de deslocamento, por exemplo, tornando impreciso qualquer forma de *feedback*. O ambiente de robôs industriais é projetado de forma a limitar qualquer variação, e é estruturado de tal forma que qualquer variação possa ser facilmente detectada pelos sensores. Entretanto, um robô móvel precisa lidar com um elevado número de situações imprevisíveis. Ambientes desestruturados e perigosos são situações comuns para robôs móveis. Nem sempre é possível remover ameaças dinâmicas ou manter um mapa preciso do mundo. O robô deve ser hábil para sentir e responder à qualquer circunstância que afete sua habilidade de completar sua missão.

A dificuldade com a abordagem de sistemas simbólicos, da inteligência artificial clássica, em robótica móvel se refere a três grandes obstáculos: (*i*) para o robô planejar e se movimentar em seu ambiente, ele precisa de uma representação simbólica do mundo. Este modelo do mundo precisa ser detalhado o suficiente para representar qualquer ameaça, objetivo ou característica do mapa que possa ser de interesse do robô. Mas uma construção precisa deste modelo a partir de dados ruidosos é difícil. (*ii*) o trabalho necessário para um computador construir e manter um modelo detalhado, mais o desenvolvimento de um plano de movimentação, consome grande quantidade de recursos computacionais, além de tempo. Isto geralmente torna o desempenho do robô insatisfatório em determinadas situações. (*iii*) as limitações do modelo, sensores e algoritmos de planejamento geralmente deixam aberta a possibilidade de existir algum tipo de ameaça inesperada no mundo real que é impossível de ser incorporada ao modelo. Tal ameaça será invisível ao planejamdo simbólico, e pode causar danos ao sistema robótico.

No momento que um programador desenvolve uma representação simbólica do mundo, suposições são feitas de forma a restringir severamente a utilidade da representação. Na robótica móvel, a pesquisa em planejamento e localização é freqüentemente realizada usando representação geométrica do ambiente[50, 51, 33] enquanto o mapeamento e o desvio de objetos inesperados é freqüentemente realizado usando mapas celulares, ou tabelas de ocupação[52, 53]. Os sistemas geométricos não são adequados a ambientes onde hajam muitos obstáculos não-mapeados. Os mapas celulares tornam computacionalmente cus-

tosas a manutenção da posição de referência e a compressão de dados com características abstratas. Por esta razão, sistemas comercialmente disponíveis realizam uma combinação de um mapa geométrico elaborado *a priori*, trajetórias explicitamente programadas, e alguma estruturação do próprio espaço de trabalho. Isto é adequado para uma planta industrial ou escritório, mas indesejável para a superfície de Marte ou o fundo do oceano.

A questão da representação apropriada tem sido um problema maior na IA do que as estratégias envolvidas na tomada de decisão. Por esta razão, as aplicações mais disseminadas em IA, como jogos de xadrez, estão restritas a espaços de estados finitos. Isto é ilustrado por[54]:

> "*Jogos geralmente não envolvem problemas complexos de representações. Um simples nó no espaço de estados é apenas a descrição de um tabuleiro e geralmente pode ser capturada de uma forma simples. Isto permite aos pesquisadores focar no comportamento da heurística, ao invés dos problemas de representação do conhecimento.*"

7.2 *Soft Computing* – Inteligência Computacional

Soft Computing (SC) é uma coleção de metodologias que almejam explorar a tolerância para imprecisão, incerteza e verdades parciais para obter robustez, tratabilidade e baixo custo. SC provê recursos para representar a ambiguidade do pensamento humano com as incertezas do mundo real.

A *Soft Computing* foi proposta para a construção de uma nova geração da inteligência artificial, máquinas com alto quociente de inteligência (*high machine inteligence quotient* – HMIQ)[55], com uma forma de processamento de informações semelhante às utilizadas pela natureza para a solução de problemas não-lineares e matematicamente não modelados. SC é a fusão ou combinação de técnicas como computação fuzzy, neural e evolutiva.

Soft Computing é um termo originalmente cunhado por Zadeh[56, 57] para denominar sistemas que "*...exploram a tolerância para imprecisão, incerteza, e verdade parcial para atingir tratabiliadde, robustez, baixo custo e melhor aderência com a realidade*". *soft computing* é "*uma associação de metodologias computacionais que incluem como membros principais sistemas fuzzy (fuzzy logic - FL), neurocomputação (neurocomputing - NC) e computação probabilística (probabilistc computing - PC)*".

A *Soft Computing* está causando uma mudança de paradigmas na engenharia e na ciência, uma vez que permite a solução de problemas que não foram totalmente resolvidos pelos métodos analíticos tradicionais. A *soft computing* permite[55]:

- rica representação de conhecimento (símbolos e padrões);
- aquisição flexível de conhecimento (aprendizado por máquinas - *machine learning* - a partir de dados empíricos ou pela entrevista de especialistas);
- processamento flexível de conhecimento (inferências pelo interfaceamento entre conhecimento simbólico e padrões).

Existe uma grande gama de problemas industriais e computacionais que requerem a análise de informação incerta e imprecisa[55]. Geralmente, um entendimento completo do domínio do problema estende a dificuldade da geração de modelos usados para explicar comportamentos passados ou predizer comportamentos futuros. Este tipo de problema é uma grande oportunidade para a aplicação de tecnologias de *soft computing*.

Devido à complexidade de muitas destas tarefas, a inteligência artificial, e em particular a SC, são usadas nas buscas por soluções. Recentemente a SC tem sido aplicada em problemas de detecção de anomalias e identificação, diagnóstico, prognóstico, estimativa e controle.

Uma das principais razões para o sucesso da SC é a sinergia derivada de seus componentes, a intrínseca capacidade para a criação de sistemas híbridos que são baseados, em maior ou menor grau, na integração das tecnologias constituintes. Esta integração provê métodos adicionais de argumentação e busca, permite combinar domínios de conhecimento e dados empíricos no desenvolvimento de ferramentas computacionais flexíveis e resolver problemas complexos. A *soft computing* tenta mimetizar soluções usadas pela natureza, que possuem mihões de anos de aprimoramento[58].

7.2.1 *Sistemas Fuzzy* – Nebulosos

Os *sistemas fuzzy* (*nebulosos*) lidam com o tratamento de imprecisão e informações vagas sustentado nos estudos da lógica multivalorada, baseada em mais de dois valores, para representar indeterminação, desconhecimento, ou outros valores-verdade intermediários possíveis entre os clássicos *verdadeiro* e *falso* da lógica Booleana. Enquanto conceitos vagos se relacionam com ambiguidade, nebulosidade (*fuzziness*) se relaciona com a ausência de limites definidos[59]. Antes de 1965, quando Zadeh propôs uma teoria completa

para conjuntos *fuzzy*[60], não éramos capazes de representar e manipular conceitos fracamente definidos.

Em um sentido amplo, os sistemas *fuzzy* têm quatro facetas[58]:

1. Uma faceta lógica, suportada por sua lógica multivalorada.
2. Uma faceta de teoria de conjuntos, cobrindo a representação de conjuntos com fronteiras fracamente definidas e seus relacionamentos.
3. Uma faceta relacional, focada na representação e uso das relações *fuzzy*.
4. E uma faceta epistêmica, cobrindo o uso da lógica *fuzzy* para "fuzzificar" sistemas baseados em conhecimento e banco de dados.

Os sistemas *fuzzy* nos dão uma linguagem com sintaxe e semântica, na qual é possível traduzir conhecimento qualitativo sobre um problema a ser resolvido. Em particular, esta característica permite o uso de variáveis linguísticas para modelar sistemas dinâmicos. Estas variáveis tomam valores *fuzzy* que são caracterizados por um nome (uma sentença gerada a partir de uma sintaxe) e um significado (uma função de pertinência determinada por procedimentos de semântica). O significado de uma variável linguística pode ser interpretado como um limite elástico de seu valor. Estes limites são propagados por operações de inferência *fuzzy*, baseado na generalização do *modus-ponens*. Este mecanismo de argumentação, com suas propriedades de interpolação dá aos sistemas *fuzzy* robustez com respeito às variações nos parâmetros de um sistema, distúrbios, etc, o que é uma das principais características dos sistemas *fuzzy*.

Os sistemas *fuzzy* têm sido empregados com sucesso em diversas áreas da robótica móvel, como em [61], onde é proposto um controlador de um robô móvel com dois graus de liberdade. Em [62], Arsene e Zalzala propõem um controlador *fuzzy* sintetizado por algoritmos genéticos. Proposta semelhante defendem Li *et al.*[63]. Shibata e Fukuda, em [64], usam um sistema *fuzzy* para coordenar o comportamento de um systema multi-agente. Juidette e Youlal, em [65], apresentam um planejador dinâmico para trajetórias. Kawanaka *et al.*[66] apresentam um sistema de aquisição de regras *fuzzy* para o controle de robôs móveis, por meio de algoritmo genético. Já Qiu e Walters[67] usam algoritmos genéticos para propiciar o aprendizado de comportamentos *fuzzy* de um controlador reativo de robôs móveis, algo semelhante ao trabalho de Kubota *et al.*[68] e em alguma extensão de [69]. Hagras *et al.*[70], propõem um sistema *fuzzy*-genético para o aprendizado e o controle de veículos agrícolas autônomos. [71, 72, 73], usam sistemas *fuzzy* para o desvio de

obstáculos. Jeong e Lee desenvolvem controladores *fuzzy* para o problema de perseguição. Estes, dentre outros trabalhos, tais como [74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85], cobrem praticamente todas as problemáticas inerentes à robótica móvel, desde o controle de sistemas simples, até com múltiplos agentes, a fusão de sensores, o controle reativo, o planejamento de trajetórias, por exemplo. Uma observação interessante, muitas das propostas apresentadas, é composta por sistemas híbridos, isto é, fazem uso de mais de uma técnica de inteligência computacional.

7.2.2 Redes Neurais Artificiais

A *neurocomputação* (*redes neurais artificiais*, do inglês *artificial neural networks* – NN, ou ANN) remonta a 1943, quando o neuropsicólogo McCulloch e o lógico Pitts concluíram que uma rede composta de unidades de decisão binária poderia implementar qualquer operação lógica[86], estabelecendo a *linhagem conexionista* da *soft computing*.

Arquiteturas de computação conexionaista são algoritmos matemáticos capazes de aprender mapeamentos entre estados de entrada e saída através de aprendizado supervisionado, ou organizar informações de entrada segundo aprendizado não-supervisionado. Suas características peculiares são o trabalho simultâneo de várias unidades de processamento independentes, e que o comportamento da rede não é determinado por qualquer elemento de sua arquitetura, mas emerge de todas estas unidades. Devido a sua habilidade em mapear sinais de entrada e saída, generalizar dados, interpretar dados sem supervisão, sua resistência a ruídos e robustez, a perda de uma unidade de processamento vai significar uma apenas uma degradação, e não uma perda total de desempenho ou funcionalidade[3].

Redes Neurais Artificiais podem ser usadas de diversas formas na robótica móvel. Por exemplo, no aprendizado de associações entre sinais de entrada (sinais de sensores) e sinais de saída (sinais de controle). Elas também podem ser usadas para determinar a estrutrua subjacente aos dados, o que pode ser útil para representações internas.

7.2.2.1 Neurônio de McCulloch and Pitts

A inspiração para redes neurais artificiais é originária dos neurônios naturais, os quais realizam complicadas tarefas como reconhecimento de padrões, aprendizado, atenção em determinado foco, controle de movimento e muitas outras tarefas de uma forma confiável

[3] O termo *degradação graciosa* descreve o fato que o desempenho de tais redes não é unicamente dependente de unidades individuais.

e robusta. Um neurônio biológico simplificado – o modelo para o neurônio artificial – consiste de um corpo celular, que realiza toda a computação, um número finito de entradas, os *dentritos*, e um ou mais sinais de saída, os *axônios*, que se conectam a outros neurônios (Figura 46). No modelo simplificado o sinal é codificado em ondas elétricas.

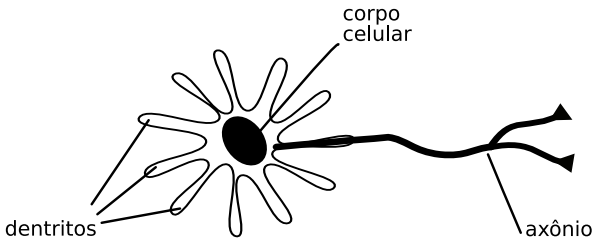

Figura 46: Neurônio simplificado.

As conexões entre os dentritos e o corpo celular, a *sinapse*, são modificáveis pelos *neurotransmissores*, isto é, sinais que podem ser amplificados ou atenuados, por exemplo.

O modelo simplificado assume que a taxa de disparo (a freqüência dos sinais de saída) seja proporcinal à atividade do neurônio, isto é, não há qualquer forma de atraso ou avanço do sinal. Em redes de neurônios artificiais o sinal de saída pode ser mantido analógico ou restrito a uma saída binária. Isto é dependente da aplicação.

McCulloch e Pitts propuseram um modelo computacional do neurônio biológico. Neste modelo, os sinais de entrada biológicos foram substituídos por um sinal contínuo; a "codificação química" da força sináptica foi substituída por um peso multiplicativo, a função de corte do neurônio artificial foi modelada usando um comparador, e o sinal de saída natural substituído por uma saída binária. O neurônio calcula uma soma ponderada k de todas as n entradas i, de acordo com a seguinte equação (Figura 47):

$$k = \sum_{j=1}^{n} i_j w_j \tag{7.1}$$

A soma ponderada é então comparada com um limiar fixo Θ para produzir um sinal de saída o. Se k exceder Θ, o sinal de saída é 1. Se k estiver abaixo do limiar, o sinal de sáida é 0 ou -1.

McCullhoch e Pitts provaram que, dada a escolha adequada dos pesos, uma estrutura de neurônios é capaz de implementar qualquer função computacional. O problema reside justamente na escolha adequada dos pesos.

A escolha dos pesos pode, em casos simples, ser determinada pelo bom senso. Entretanto, para funções complexas, o bom senso não é suficiente. Passa a ser desejável um

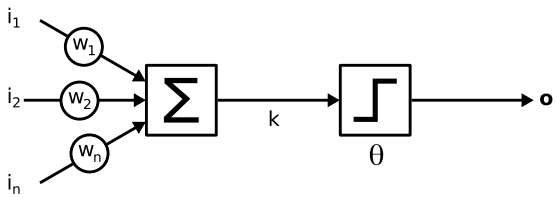

Figura 47: Modelo matemático de neurônio de McCulloch e Pitts.

mecanismo de aprendizado que pode determinar os pesos automaticamente. Um mecanismo de aprendizado também possibilita ao robô aprender.

7.2.2.2 Perceptron

Apoiando-se neste conceito de aprendizado, Rosenblatt[87, 88] propôs uma rede de uma única camada, chamada *perceptron* (Figura 48), de fácil implementação, que requer poucos recursos computacionais, de fácil aprendizado, e demonstrou que ela poderia ser treinada para reconhecer padrões.

O perceptron consiste de uma rede com duas camadas:

Camada de entrada: camada que apenas repassa os sinais de entrada à próxima camada, sem qualquer processamento.

Camada de saída: camada responsável pelo processamento neuronal, é composta por neurônios do modelo de McCulloch e Pitts.

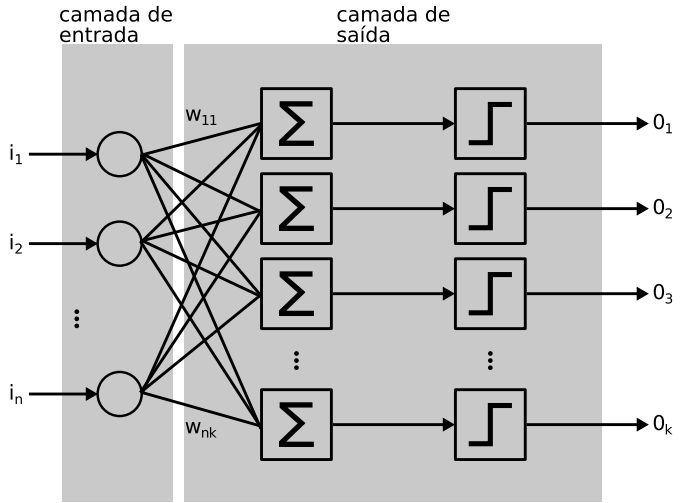

Figura 48: Perceptron.

As unidades de entrada simplesmente repassam os sinais de entrada $\vec{i}$ a todas as unidades de saída, a saída o_j da unidade de saída j é determinada por:

$$o_j = f(\sum_{k=1}^{M} w_{jk} i_k) = f(\vec{w_j} \cdot \vec{i}) \tag{7.2}$$

onde $\vec{w_j}$ é o vetor de pesos da unidade de saída j. M é o número de unidades de saída, e f é a função de transferência. A função de transferência do perceptron é uma função degrau:

$$f(x) = \begin{cases} 1 & \forall\ x > \Theta \\ 0(ou -1) & caso\ contrario \end{cases} \tag{7.3}$$

7.2.2.3 Regra de Aprendizado do Perceptron

Para redes e mapeamentos complexos de entrada-saída é desejável uma regra de aprendizado de forma a proporcionar o aprendizado de robôs autônomos. A regra para determinar os pesos de um perceptron é definida por:

$$\Delta\vec{w_k} = \eta(t)(\tau_k - o_k)\vec{i} \tag{7.4}$$

$$\vec{w_k}(t+1) = \vec{w_k}(t) + \Delta\vec{w_k} \tag{7.5}$$

onde τ_k é o valor alvo para a unidade k, isto é, a saída desejada para a unidade de saída k, e o_k a saída obtida pela pela unidade k. A velocidade de aprendizado é determinada pela taxa $\eta(t)$. Um valor elevado de η, por exemplo 0,8, resulta em uma rede com rápida adaptação a mudanças, mas que também se esqueçe rapidamente, podendo se tornar "neurótica", assimilando qualquer sinal errático. Um baixo valor de η, por exemplo 0,1, vai resultar em uma rede letárgica, que toma um tempo considerável para aprender qualquer nova função. Diferentes valores e variações de η podem ser usadas para se obter variados comportamentos da rede ao longo do tempo.

7.2.2.4 Perceptron Multicamada

Minsky e Papert[89] provaram que os *perceptrons* poderiam apenas particionar linearmente um espaço de decisão. Deste modo, elas não são capazes de separar regiões não-lineares e não-convexas. Isto se deve ao fato de que cada camada de uma rede cons-

tituída de neurônios de McCulloch e Pitts estabelece um hiperplano para separar as duas classes de conhecimento que a rede deve aprender, os 1's e 0's. Se a separação de duas ou mais classes não pode ser realizada por apenas um hiperplano, como é o caso clássico da função lógica XOR, uma rede de apenas uma única camada não é capaz de mapear a função.

Isto motivou a comunidade de pesquisadores de redes neurais no desenvolvimento de redes multicamadas (Figura 49), que poderiam superar as limitações dos *perceptrons*. Entretanto, o treinamento destas redes multicamadas se mostrou problemático. Finalmente, a introdução da *retro-propagação* (*backpropagation* – BP), independentemente desenvolvida por Werbos[90], Parker[91] e LeCun[92], forneceu uma teoria para o treinamento de redes multicamadas, com funções de ativação não-lineares. A BP usa a retropropagação do erro da saída para determinar as mudanças necessárias nos pesos da rede.

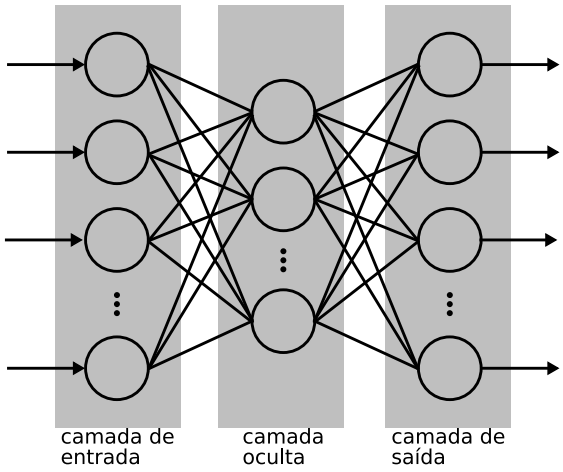

Figura 49: Perceptron multi-camadas.

A rede é iniciada com seus pesos contendo valores aleatórios. Os limiares Θ são substituídos por pesos para uma entrada que é sempre $+1$, o que permite a atualização destes limiares junto com os pesos.

Iniciada a rede, o treinamento é feito pela apresentação de pares de entrada-saída à rede. Estes padrões de treinamento são usados para adaptar os pesos. A saída o_j de cada unidade j é determinada por:

$$o_j = f(\vec{w_j} \cdot \vec{i_j}) \tag{7.6}$$

onde $\vec{w_j}$ é o vetor de pesos da unidade j, e $\vec{i}$ é o vetor de sinais de entrada. A função f é a função de ativação diferenciável, geralmente sigmóide, definida por:

$$f(z) = \frac{1}{1 + e^{-kz}} \tag{7.7}$$

onde k é uma constante positiva que controla o comportamento da função sigmóide. Para $k \to \infty$ a função sigmóide se torna uma função limiar em degrau, a mesma usada para neurônios de McCulloch e Pitts.

Calculada as saídas de todas as camadas, desde as ocultas até a camada de saída, a rede pode ser treinada em função do erro da saída final. Todos os pesos w_{ij} da unidade i até a unidade j são treinados de acordo com:

$$w_{ij}(t+1) = w_{ij} + \eta \delta_{pj} o_{pj} \tag{7.8}$$

onde η é a taxa de aprendizado, δ_{pj} é o sinal de erro para a unidade j e o_{pj} é o sinal de entrada da unidade j vinda da unidade p.

Os sinais de erro são determinados primeiramente para as unidades de saída, posteriormente para as unidades ocultas. Desta forma o treinamento começa pela camada de saída e progride para as camadas intermediárias, ocultas. Para cada unidade j da camada de saída, o sinal de erro δ_{pj}^{saida} é determinado por:

$$\delta_{pj}^{saida} = (t_{pj} - o_{pj}) o_{pj} (1 - o_{pj}) \tag{7.9}$$

onde t_{pj} é o sinal de referência, e o_{pj} é a sáida obtida da unidade sendo atualizada. Uma vez calculados os erros da unidade de saída, resta a atualização da camada oculta, segundo:

$$\delta_{pj}^{oculta} = o_{pj}(1 - o_{pj}) \sum_{k} \delta_{pk} w_{kj} \tag{7.10}$$

onde o_{pj} é o sinal de saída da unidade da camada oculta sendo atualizada, δ_{pk} é o erro da unidade k na camada subseqüente da rede, e w_{kj} o peso entre a unidade oculta j e a unidade subseqüente k, da próxima camada. Este processo de treinamento é repetido até que o erro caia abaixo de um limiar pré-estabelecido ou qualquer outro parâmetro de parada relevante, como tempo ou número de retropropagações.

Hornik *et al.* [93] provou que uma rede de três camadas – a primeira camada de entrada, a segunda oculta e a terceira de saída – constitui um aproximador universal de funções, porém, ao custo de um processo de aprendizado computacionalmente custoso, o que eventualmente pode se tornar um problema para a robótica móvel.

Topologicamente, as redes neurais são divididas em *redes em avanço* (*feedforward*) e *redes recorrentes*. As redes em avanço incluem os *perceptrons* de uma e múltiplas camadas, como também as *redes de base radial*. Montana e Davis apresentam o treinamento de redes *feedforward* por meio de algoritmos genéticos[94]. McInerney e Dhawan fazem o mesmo em [95]. Kitano, em [96], traz estudos empíricos da convergência do treinamento de redes neurais artificiais pelo uso de algoritmos genéticos.

7.2.2.5 Redes de Base Radial

Da mesma forma que o perceptron multicamada, as *redes neurais de base radial* (*radial basis funcions* – RBF) também podem mapear funções não-linearmente separáveis[14]. É uma rede de duas camadas, na qual a camada oculta realiza um mapeamento não-linear baseado em uma função de base radial[97] (Figura 50), enquanto a camada de saída realiza uma soma ponderada linear da saída da camada oculta. Os mecanismos fundamentais da rede de base radial e do perceptron multicamadas são similares.

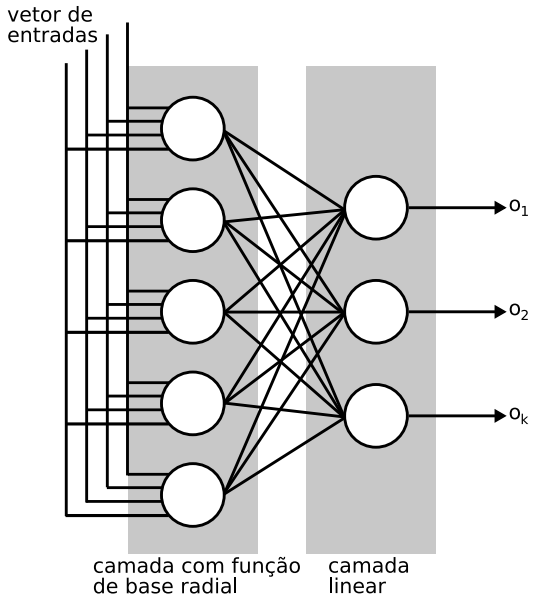

Figura 50: Rede com função de base radial (RBF).

As unidades ocultas possuem uma região Gaussiana[97] de captura que serve para identificar similaridades entre o vetor de entrada e o vetor de pesos das unidades ocultas. A camada oculta realiza um mapeamento não-linear no espaço de entrada. A camada de saída associa a classificação da camada oculta ao sinal de referência através de mapeamento linear, como no perceptron de uma ou múltiplas camadas. A saída $o_{oculta,j}$ da unidade j na camada oculta é determinada por:

$$o_{oculta,j} = \exp\left(-\frac{\|\vec{i} - \vec{w_j}\|}{\sigma}\right) \tag{7.11}$$

onde $\vec{w_j}$ é o vetor de pesos da unidade j, $\vec{i}$ é o vetor dos sinais de entrada, e σ é o parâmetro que controla o comprimento da região de captura da função de base radial.

A saída $\vec{o_k}$ de cada unidade da camada de saída é determinada por:

$$o_k = \vec{o}_{oculta} \cdot \vec{v}_k \tag{7.12}$$

onde $\vec{o}_{oculta}$ é a saída da camada oculta, e $\vec{v}_k$ é o vetor de pesos da unidade de saída k.

O treinamento da camada de saída é realizado aplicando-se as regras de aprendizado do perceptron[14]. A rede de base radial funciona pelo mapeamento do espaço de entrada em um espaço de maior dimensão através de uma função não-linear, como a função Gaussiana, por exemplo. Os pesos da camada oculta têm de ser escolhidos de tal forma que o espaço de entrada possa ser completamente representado. Uma forma é espalhar igualmente os centros ao longo do espaço de entrada. Em alguns casos é suficiente posicionar os centros aleatoriamente ao longo do espaço de entrada. O ideal é concentrar a densidade dos centros das funções radiais onde a densidade do espaço de entrada é alta.

As redes recorrentes cobrem as *redes competitivas*, *mapas auto-organizáveis* (*self-organizing maps* – SOM), redes de Hopfield e modelos de *teoria adaptativa ressonante*. Enquanto redes em avanço são usadas sob aprendizado supervisionado, redes recorrentes são geralmente usadas sob aprendizado não-supervisionado, memória associativa e auto-organização.

7.2.2.6 Mapas Auto-organizáveis

Todos os métodos de aprendizado aprensentados até agora são classificados de *aprendizado supervisionado*, o aprendizado é realizado pelo uso de padrões de referência, fornecidos externamente, por um "supervisor", pares de entrada e saída usados para o aprendizado. Entretanto, existem aplicações onde não há sinal de treinamento, como aplicações de aglomeração (*clustering*) de algum espaço de entrada. Pode ser útil na robótica móvel aglomerar um espaço de entrada de alta dimensão e mapeá-lo automaticamente, de uma forma não-supervisionada, em um espaço de saía de menor dimensão. Esta redução da dimensão de espaços é uma forma de generalização, reduzindo a complexidade do espaço de entrada enquanto retendo todas as caracterísitcas relevantes deste espaço. Os mapas

auto-organizáveis (SOM), ou redes de Kohonen[98], são um mecanismo de mapeamento não-supervisionado[14] (Figura 51).

Os SOM são constituídos de uma matriz bidimensional de unidades. Todas as unidades recebem o mesmo vetor de entrada $\vec{i}$. Inicialmente, os vetores de pesos $\vec{w}_j$ são iniciados com valores aleatórios normalizados.

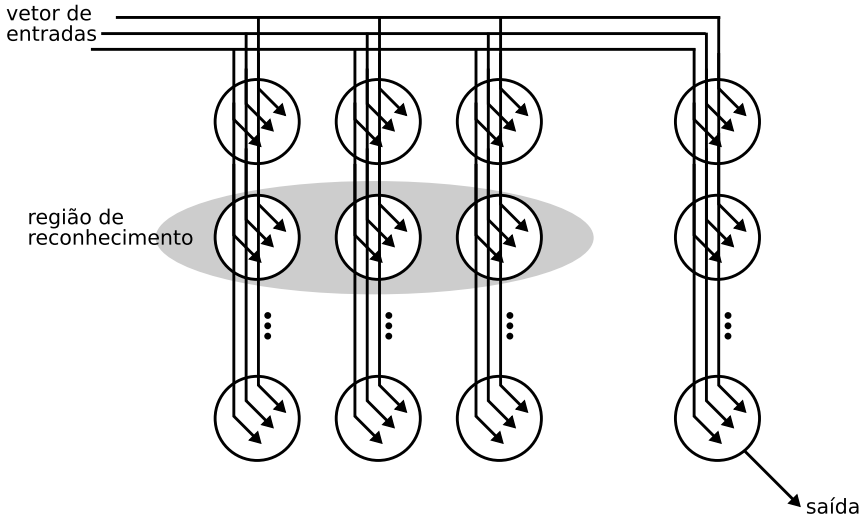

Figura 51: Rede SOM.

A saída o_j de cada unidade j é determinada por:

$$o_j = \vec{w}_j \cdot \vec{i} \tag{7.13}$$

Devido aos valores aleatórios na configuração inicial da rede, as saídas de todas as unidades serão diferentes uma das outras, e cada unidade vai responder com maior força a um vetor de entrada particular. A "unidade vencedora" e suas unidades circundantes serão treinadas para responder com ainda mais força a este vetor em particular, pela aplicação da regra definida pela equação:

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \eta(\vec{i} - \vec{w}_j(t)) \tag{7.14}$$

onde η é a taxa de aprendizado. Depois de atualizados, os vetores de peso são novamente normalizados. As redondezas ao redor da unidade vencedora são escolhidas maiores nos estágios iniciais do processo de treinamento, e tornam-se menores ao longo do tempo.

Como resultado do processo de treinamento, certas áreas da rede se tornam mais sensíveis a certos estímulos de entrada, aglomerando o espaço de entrada em um espaço de

saída bidimensional. Esta aglomeração ocorre sob forma topológica, mapeando entradas similares em regiões da rede.

As redes SOM podem ser utilizadas para aglomerar um espaço de entrada para a obtenção de uma representação abstrata porém significativa do espaço de entrada. Similaridades entre as leituras de sensores (percepções) podem ser abstraídas para a codificação de políticas, a resposta do robô a um determinado conjunto de entradas.

Os trabalhos de Miller *et al.*[99] e Kitano[96] apresentam o projeto de redes neurais usando algoritmos genéticos. Harp *et al.*[100] e Alba *et al.*[101] defendem a síntese genética de redes neurais e o uso de algoritmos genéticos como heurística para a otimização do projeto de ANN, respectivamente. Gruau desenvolve trabalho semelhante, porém, com uma rede neural booleana[102].

O manual elaborado por Nolfi e Parisi[103] é uma referência apropriada para a introdução ao assunto de *redes neurais evolutivas*. Outra boa referência é o trabalho de Yao[104]. Meyer elabora um texto sobre o assunto, porém, focado na aplicação à robótica móvel[105].

Floreano e Mondada[106] apresentam a criação de um agente autônomo, um robô móvel controlado por uma rede neural, sintonizado por meio de técnicas evolutivas. Outros trabalhos, como [69, 107, 108, 109, 110, 111, 112, 113, 114], cobrem temas como controladores neurais, aprendizado de comportamentos, redes neurais dinâmicas, uso de redes neurais para a adaptação a ambientes dinâmicos e até mesmo a competição de futebol entre times de robôs móveis. Mais uma vez, a partir da revisão bibliográfica, fica demonstrada a sinergia entre as diversas técnicas da inteligência computacional.

7.2.3 Computação Evolutiva

Na *Computação Evolutiva* ou *Evolucionária* (do inglês *evolutionary computing* – EC) os algoritmos apresentam um comportamento adaptativo que lhes permite manipular problemas não-lineares, multi-dimensionais, sem a necessidade de equações diferenciais ou conhecimento explícito da estrutura do problema. Como resultado, estes algoritmos são bastante robustos para variações no tempo, mesmo que apresentem baixa velocidade de convergência. A computação evolutiva compreende muitas famílias importantes da computação estocástica, incluindo *estratégias evolutivas* (do inglês *evolutionary strategies* – ES), proposta por Rechenberg[115] e Schwefel[116], *programação evolutiva* (do inglês *evolutionary programming* – EP), introduzido por Fogel[117, 118], *algoritmos genéticos* (do

inglês *genetic algorithms* – GA), baseado no trabalho de Fraser[119], Bremermann[120], Reed *et al.*[121], e Holland[122, 123, 124], e ainda contém um subconjunto de *programação genética* (do inglês *genetic programming* – GP), introduzida por Koza[125].

A história da computação evolutiva pode ser remontada ao trabalho de Friedberg[126], que estudou a evolução de uma máquina capaz de aprender através da computação de uma dada função de entrada-saída; ao trabalho de Fraser[119] e Bremermann[120], que investigaram alguns dos conceitos dos algoritmos genéticos usando codificação binária para o genótipo; remonta também ao trabalho de Barrichelli[127], que realizou algumas simulações numéricas de processos evolutivos, e de Reed *et al.*[121], que explorou conceitos similares em uma simulação simplificada do jogo poker.

A estratégia evolutiva foi originalmente proposta para a otimização de parâmetros no espaço contínuo. Nos seus primórdios, a estratégia evolutiva usou apenas uma representação contínua e um operador de mutação agindo em um único indivíduo para criar um novo. Posteriormente, um operador de recombinação foi adicionado ao de mutação à medida em que a ES foi expandida para evoluir uma população completa de indivíduos. Cada indivíduo tem a mesma probabilidade de gerar seus descendentes. Ainda, cada indivíduo tem seu próprio operador de mutação, que é herdado e pode ser diferente para cada parâmetro.

As primeiras versões da programação evolutiva foram aplicadas para a identificação de seqüências preditivas, controle automático, reconhecimento de padrões e estratégias ótimas em jogos. Para atingir estes objetivos, a programação evolutiva evolui máquinas de estado finitas que tentam maximizar uma função de adequação. Entretanto, versões recentes de programação evolutiva têm se focado na otimização de parâmetros contínuos e treinamento de redes neurais. Como resultado, a programação evolutiva tem se tornado mais similar à estratégia evolutiva. A principal diferença entre a estratégia evolutiva e a programação evolutiva é que a última está focada no comportamento de espécies, enquanto a primeira está focada no comportamento de indivíduos. Além do mais, a programação evolutiva não faz uso do operador de recombinação.

Os algoritmos genéticos realizam uma busca global aleatória em um espaço solução[128], usando representações de genótipo ao invés de fenótipo[58]. Algoritmos genéticos canônicos codificam cada solução candidata para um determinado problema como uma *string* binária, inteira ou real, chamada de cromossomo. Cada elemento da *string* representa uma característica particular da solução. A *string* é avaliada por uma função de adequação para a determinação da qualidade da solução: soluções melhor adaptadas sobrevivem e

produzem uma nova geração, enquanto soluções menos adaptadas são eliminadas da população. Para a evolução das *strings*, as soluções candidatas, cada *string* pode ser modificada por "operadores genéticos", tais como recombinação (*crossover*) e mutação (*mutation*). A recombinação age na tentativa de capturar as melhores características de dois indivíduos e passá-las para a próxima geração, é o componente hereditário da técnica. A mutação é um operador probabilístico que tenta introduzir novas características na população de soluções, constituí o componente de busca aleatória. Os novos indivíduos criados por estes operadores formam a próxima geração de soluções potenciais, que tendem a se mover em direção a soluções ótimas. As limitações do problema são geralmente modeladas por penalidades na função de adequação codificadas diretamente na estrutura das soluções.

A programação genética é uma forma particular de algoritmo genético na qual o cromossomo tem uma estrutura hierárquica ao invés de linear, cujo tamanho não é definido. Os indivíduos na população são programas estruturados em árvore, e os operadores genéticos são aplicados ao seus ramos (sub-árvores) ou a cada nó. Este tipo de cromossomo pode representar o *parser* (interpretador) de uma expressão, uma expressão aritmética em si, etc. A programação genética tem sido usada na predição de séries-temporais, controle e otimização de topologias de redes neurais.

7.2.3.1 Representação Cromossômica

O primeiro passo para a aplicação de GAs a um problema é representar cada possível solução x no espaço de busca como uma seqüência de símbolos s gerados a partir de um alfabeto finito A. No caso mais simples, usa-se um alfabeto binário $A = \{0, 1\}$[128]. Cada seqüência s representa um *cromossomo*, e cada elemento de s é um *gene*[4]. Como qualquer gene pode assumir qualquer valor de A, cada elemento de A é equivalente a um *alelo*, ou seja, um valor possível para qualquer gene. A posição de um gene em um cromossomo, seu índice dentro da seqüência (*string*), é denominado *locus gênico* (Figura 52).

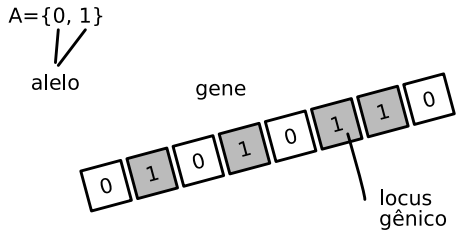

Figura 52: Representação cromossômica.

[4]Em uma analogia direta com a genética biológica.

Além disso, na maior parte dos GAs, assume-se que cada indivíduo seja constituído de um único cromossomo de comprimento constante, e o uso de uma população com número fixo de elementos[129].

7.2.3.2 Fluxo Básico de um Algoritmo Genético

Definida a representação cromossômica para o problema, gera-se um conjunto inicial de possíveis soluções, chamadas de *soluções candidatas*. Um conjunto de soluções codificadas de acordo com a representação cromossômica corresponde a uma *população* de indivíduos POP. A cada iteração de um GA é dado o nome de *geração*. Designando-se cada geração por um índice t, o seguinte pseudo-código ilustra o fluxo de um algoritmo genético.

Algoritmo 4: Pseudo-código do algoritmo genético; com operadores de *seleção*, *recombinação* e *mutação*.

```
t ← 0;
Inicializar a população POP(t);
Avaliar a população POP(t);
repetir
    t ← t + 1;
    Selecionar POP(t) a partir de POP(t − 1);
    Recombinar POP(t);
    Mutar POP(t) ;
    Avaliar população POP(t);
até (condição de término não atingida) ;
```

7.2.3.3 Iniciação da População

Em grande parte das aplicações, a população inicial de N indivíduos é gerada aleatoriamente ou através de um processo heurístico. Os algoritmos genéticos se apoiam fortemente na existência da variedade de indivíduos[5], e variados graus de adaptabilidade ao problema. É importate que a população inicial cubra a maior área possível do espaço de busca, isto é, que possua um bom *espalhamento inicial*.

[5]Valendo-se das idéias essenciais de seleção natural.

7.2.3.4 Avaliação da População

Os GAs fazem uso de uma *função objetivo*, ou *função de adequação*, para avaliar cada membro da população.Em casos simples, pode-se usar justamente a função que define o espaço de busca, a qual se busca maximizar ou minimizar. A função objetivo tem a finalidade de atribuir a cada indivíduo uma medida – *fitness* – de quão bem adaptado ao ambiente ele está, ou seja, quanto maior o valor retornado[6] pela função objetivo, mais adaptado é o indivíduo e, segundo as leis da seleção natural, maiores devem ser suas chances de sobreviver e reproduzir-se, o que favorece a perpetuação de seu material genético para as gerações seguintes.

A função de adequabilidade talvez seja o componente mais importante de um AG[129]. Ela tem a capacidade de afetar severamente o desempenho geral do algoritmo em termos de qualidade de resultado e tempo de execução. O processo evolutivo é inteiramente guiado pelos resultados fornecidos pela função de avaliação.

Uma função de avalição que não represente corretamente as características e requisitos de um problema irá orientar mal a busca pela solução. Em contrapartida, uma função que descreva excessivamente os detalhes do problema, com um grande número de restrições e particularidades poderá proporcionar bons resultados, mas a definição de tal função poderá se mostrar extremamente complexa, às vezes até mesmo inviável ou impossível. Funções de avaliação que apresentam um alto custo computacional produzem um grande impacto no tempo total de execução necessário para avaliar toda a população. Deve-se perseguir o compromisso na redução da complexidade da função de adequação e o aumento da qualidade dos resultados fornecidos por essa função.

7.2.3.5 Seleção de Indivíduos

O mecanismo de seleção em GAs emula os processos de reprodução assexuada[7] e seleção natural. Gera-se uma população temporária de tamanho N, com indivíduos extraídos com probabilidade proporcional à adequabilidade relativa de cada indivíduo na população, que pode ser definido por:

$$p_{sel} = \frac{a(s)}{\sum_{i=1}^{N} a(s_i)} \tag{7.15}$$

[6]Para casos de maximização. Já para casos de minização é feita a busca pelo menor valor de adequação, ou *fitness*.

[7]Reprodução assexuada no sentido de não haver a necessidade direta de outro indivíduo para a seleção.

Onde $a(\bullet)$ é a função de adequabilidade. Neste processo, indivíduos com baixa adequabilidade terão elevada probabilidade de extinção, ao mesmo tempo que indivíduos mais aptos terão maiores chances de sobreviver. Este método de seleção também é conhecido como *método da roleta*.

7.2.3.6 Recombinação

O processo de recombinação é um processo sexuado, envolve mais de um indivíduo, emulando o fenômeno de *crossover*, a troca de fragmentos entre pares de cromossomos (Figura 53). Em sua forma mais simples, é um processo aleatório que ocorre com probabilidade p_{rec}, que escolhe aleatoriamente um locus gênico que servirá de pivô (ponto de cisalhamento) para a troca dos pares de fragmentos genéticos.

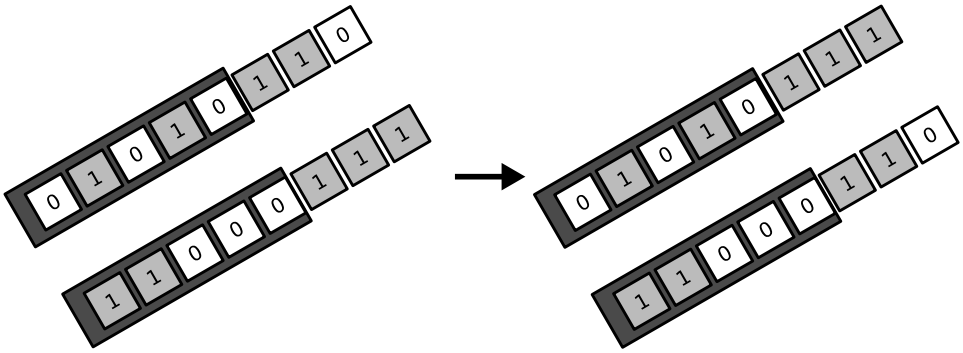

Figura 53: Representação do *crossover*.

O operador de recombinação é uma das características dos GAs, uma vez que as outras técnicas evolutivas se apoiam basicamente sobre operadores de mutação[128].

7.2.3.7 Mutação

O processo de mutação é o componente aleatório do algoritmo genético. Seleciona-se aleatoriamente um locus gênico de um indivíduo, e muda-se o valor do gene correspondente para um outro alelo possível, de forma também aleatória (Figura 54). O processo é controlado por um parâmetro de probabilidade p_{mut}.

7.2.3.8 Condições de Término

Tratando-se de problemas de otimização, o ideal seria que o algoritmo terminasse sua busca assim que o ponto ótimo fosse encontrado. Porém, no caso de funções multimodais, apenas um ponto ótimo pode ser adequado ou todos ou o maior número possível de

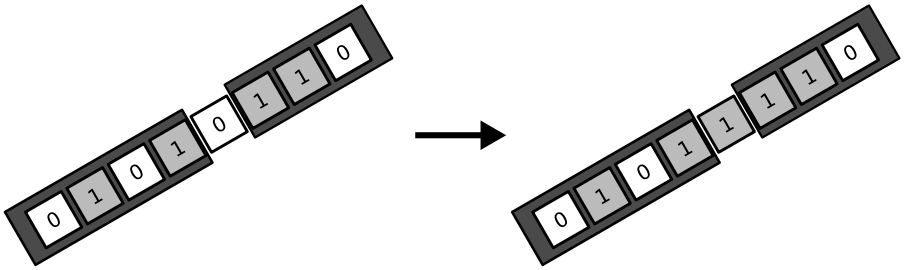

Figura 54: Representação da mutação.

pontos ótimos pode se fazer necessário. Problemas práticos se caracterizam pela incerteza de pontos ótimos representarem pontos ótimos globais. Como conseqüência, usam-se outros critérios para a parada do algoritmo, como o número máximo de gerações ou um tempo limite de processamento. Outro parâmetro muito utilizado é a idéia de estagnação, quando não se observa melhora na adequação média (*fitness*) da população depois de várias gerações consecutivas, é provável que o algoritmo encontrou um ponto de máximo/mínimo.

7.2.4 Síntese de Redes Neurais por Algoritmos Genéticos

Dentre as diversas configurações de redes neurais artificiais existentes, provavelmente, o perceptron de múltiplas camadas é o mais utilizado, isso devido a sua simplicidade, flexibilidade e amplo campo de aplicação. O uso do perceptron de múltiplas camadas envolve etapas altamente experimentais e computacionalmente custosas, como o uso do algoritmo de *backpropagation*.

O primeiro passo para o uso do perceptron de múltipla camada é a especificação de sua estrutura física, sua arquitetura:

- número de camadas;
- número de neurônios para cada camada;
- padrão de conectividade entre os neurônios.

O próximo passo consiste em especificar os parâmetros de aprendizagem, os pesos iniciais e só então prosseguir para o procedimento de treinamento, geralmente usando um método de gradiente, como *backpropagation*. Este último procedimento é lento e computacionalmente custoso, além de ser dependente, em alguma extensão, de procedimentos heurísticos.

Os algorítmos genéticos têm sido propostos para a determinação dos pesos de uma rede neural de arquitetura conhecida, determinação da própria arquitetura, ou a determinação de ambos, os pesos e arquitetura de uma rede neural artificial.

7.2.5 Aprendizado Evolutivo

O treinamento clássico do perceptron de múltiplas camadas é realizado por meio de algum método de gradiente, geralmente uma variante da *regra delta*, usando o algoritmo de *backpropagation* para o cálculo de derivadas parciais. Esse método é particularmente indicado para a busca de mínimos locais no espaço de erros do perceptron de múltiplas camadas.

Com o objetivo de proporcionar uma busca global, especialmente de superfícies complexas e multimodais, GAs podem ser empregados para a determinação dos pesos de uma rede neural de configuração pré-determinada. Neste caso, uma população de redes de mesma arquitetura é usada, onde cada perceptron, com seus respectivos pesos, passam a representar um indivíduo. Os pesos podem ser codificados por alfabetos binários ou de números reais. A avaliação de cada indivíduo é feita com base em um conjunto de padrões de teste, de modo que a adequabilidade seja uma função descrescente do erro quadrático. Em alguns casos, usa-se GAs somente para a busca global no início do treinamento e posteriormente um método de gradiente convencional.

Montana a Davis apresentam em seu trabalho[94] o treinamento de redes *feedforward* pelo uso de algoritmos genéticos. Como citado anteriormente, Kitano[96] apresenta estudos empíricos da convergência do treinamento de redes neurais fazendo uso de algoritmos genéticos. Os trabalhos [95, 101] estendem o tema de aprendizado evolutivo.

7.2.6 Arquiteturas Evolutivas

Outra aplicação plausível de GAs é a determinação de arquiteturas de *perceptrons* de múltiplas camadas adequadas a uma tarefa específica. Cada indivíduo de uma população corresponde a uma arquitetura distinta, com diferentes números de camadas, diferentes números de neurônios por camada, e diferentes padrões de conectividade.

A representação cromossômica se torna rica e complexa, o mesmo pode-se dizer dos operadores genéticos, e o tempo de avaliação de cada indivíduo é relativamente alto. Estes fatores geralmente corroboram contra o uso de GAs para busca de arquiteturas de redes neurais. A avaliação é iniciada com redes possuindo pesos gerados aleatoriamente para

cada indivíduo, estas redes são treinadas por um conjunto de padrões de referência. Depois disso, cada rede é avaliada segundo um outro padrão de testes. Só então a adequabilidade de cada indivíduo é estimada usando-se informações como velocidade de convergência, erro de reconhecimento e outros parâmetros que se façam releventes.

Miller *et al.*[99], Kitano[96], Harp *et al.*[100], Nolfi e Parisi[103] e Yao[104] descrevem em detalhes o projeto e síntese de redes neurais por meio de técnicas evolutivas.

7.3 Algoritmos Culturais

Nas sociedades humanas, a cultura pode ser vista como o veículo para o armazenamento da informação que é globalmente acessível a todos os membros da sociedade, e que pode ser útil para guiar as atividades de soluções de problemas.

O algoritmo cultural é uma classe de modelo computacional derivado a partir da observação do processo evolutivo cultural na natureza humana. É um sistema de herança dupla que caracteriza a evolução da cultura humana tanto ao nível macro-evolutivo, que toma lugar no *espaço de crenças*, e ao nível micro-evolutivo, que ocorre no *espaço populacional*[130]. A principal idéia por trás dos algoritmos culturais é adquirir conhecimento sobre a solução do problema (crença) a partir da população em evolução e aplicar este conhecimento para guiar a busca[131, 132]. O conhecimento produzido no espaço populacional, dentro do nível micro-evolutivo, é seletivamente aceito ou passado ao espaço de crenças e usado para ajustar as estruturas simbólicas lá existentes. Este conhecimento pode então ser usado para influenciar as mudanças feitas na próxima geração da população.

Métodos de computação evolutiva (EC) convencionais têm sido aplicados em larga escala em problemas desde a busca heurística à otimização devido à sua natureza imparcial, o que permite o bom desempenho em situações com pouco ou nenhum conhecimento do domínio do problema[133]. Entretanto, a incorporação de mecanismos baseados em conhecimento pode melhorar consideravelmente o desempenho de um algoritmo evolutivo quando o conhecimento adquirido durante o processo evolutivo, de solução do problema, é usado para guiar o processo de solução do problema[131, 132, 134]. O conhecimento, padrões identificados durante a busca da solução, pode ser usado para guiar a geração de soluções candidatas: promover mais instâncias de candidatos desejados, ou reduzir o número de candidatos indesejados na população, de forma a guiar apropriadamente a otimização.

Entretanto, métodos de computação evolutiva tradicionais têm mecanismos limitados,

ou implícitos, para a representação e armazenamento do conhecimento global de um indivíduo, que possa ser passado de geração a geração. Então, algoritmos culturais, os quais modelam a evolução do componente cultural de um sistema computacional evolutivo ao longo do tempo, apresentam um mecanismo explícito de aquisição, armazenamento e integração da experiência e comportamento na solução do problema de indivíduos e grupos. Um modelo de busca evolutivo, como um algoritmo genético, programação genética, estratégia evolutiva, ou programação evolutiva, pode ser usado na modelagem do componente populacional em algoritmos culturais[135].

O pseudo-código básico de algoritmo cultural é[136]:

Algoritmo 5: Pseudo-código do algoritmo cultural.

$t \leftarrow 0$;
Inicializar a população $POP(t)$;
Inicializar o espaço de crenças $BLF(t)$;
repetir
  Avaliar população $POP(t)$;
  $Atualizar(BLF(t), Aceitar(POP(t)))$;
  $Gerar(POP(t), Influenciar(BLF(t)))$;
  $t \leftarrow t + 1$;
  Selecionar $POP(t)$ de $POP(t-1)$;
até *(condição de término não atingida)* ;

Onde $POP(t)$ representa o componente populacional no instante t, $BLF(t)$ representa o espaço de crença no instante t. O algoritmo começa com a inicialização da população e do espaço de crenças, então entra no laço evolutivo até a condição de término ser satisfeita.

Os indivíduos da primeira geração são avaliados usando uma função de desempenho, $obj()$. Posteriormente, a informação de desempenho é usada como referência para a produção de generalizações sobre a experiência na solução de um problema dos indivíduos selecionados. Uma função de aceitação, $Aceitar()$, determina a quais indivíduos, na geração atual da população, serão permitidos contribuir com o espaço de crenças. As experiências dos indivíduos selecionados serão usadas para ajustar o conhecimento do espaço de crenças pela função $Atualizar()$ (Figura 55).

Crenças são produzidas como resultado da experiência de um indivíduo e então fundidas com aquelas de outros indivíduos, para formar um novo grupo de crenças. Esta atualização do espaço de crenças ocorre durante a *fase hierárquica*. Estas novas crenças serão usadas para guiar e influenciar a evolução da população. Elas podem influenciar a

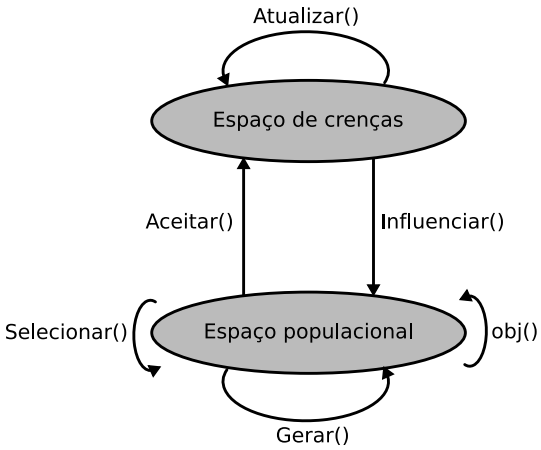

Figura 55: *Framework* de um algoritmo cultural.

seleção de indivíduos para reprodução pela modificação do desempenho dos indivíduos, de forma a refletir o desempenho das crenças que os indivíduos atualmente representam. Elas também podem influenciar como a variação dos operadores é aplicada às novas cópias dos indivíduos selecionados. Os dois caminhos de fluxo de informação, um através da funções $Aceitar()$ e $Influenciar()$, e o outro através da experiência individual de indivíduos e da função $obj()$ criam um *sistema de hierarquia dupla.*

O algoritmo cultural repete este processo para cada geração, até a condição de término pré-especificada ser atingida. Desta forma, o componente populacional e o espaço de crença interagem influenciando um ao outro, de uma forma similar à evolução cultural humana.

7.3.1 Categorias de Conhecimento

Saleem e Reynolds[130] identificam cinco categorias básicas de conhecimento que podem ser úteis na tomada de decisões. Elas são o *conhecimento normativo* (intervalos de comportamentos aceitáveis), *conhecimento situacional* (exemplares de soluções de sucesso ou insucesso), *conhecimento de domínio* (conhecimento do domínio dos objetos, seus relacionamentos, e interações), *conhecimento histórico* (padrões temporais de comportamento), e *conhecimento topográfico* (padrões espaciais de comportamento). Este conjunto de categorias é visto como completo para um dado domínio no sentido de que todo o conhecimento disponível pode ser expressado em termos de uma destas classificações.

7.3.1.1 Conhecimento Normativo

O conhecimento normativo é um conjunto de intervalos de variáveis considerados promissores, que provêm padrões para comportamentos dos indivíduos e diretivas para os ajustes dos indivíduos. O conhecimento normativo instrui os indivíduos para "se mover no intervalo certo" se eles ainda não estiverem lá. Para cada componente do conhecimento normativo, existe um limite superior e inferior, e o valor de desempenho para indivíduos nos limites superiores e inferiores (Figura 56).

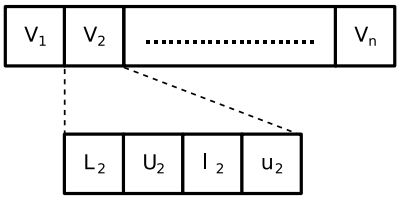

Figura 56: Estrutura do conhecimento normativo.

Para cada variável V_i, a estrutura de dados contém os limites inferior e superior l_i, u_i, e o valor de desempenho para os indivíduos nos limites inferior e superior, L_i e U_i. A atualização do limite inferior para o parâmetro j é determinado por:

$$L_j^{t+1} = \begin{cases} f(x_i) & se\ x_{i,j} \leq l_j^t\ ou\ f(x_i) < L_j^t \\ L_j^t & caso\ contrario \end{cases} \tag{7.16}$$

$$l_j^{t+1} = \begin{cases} x_{i,j}^t & se\ x_{i,j}^t \leq l_j^t\ ou\ f(x_i^t) < L_j^t \\ l_j^t & caso\ contrario \end{cases} \tag{7.17}$$

onde o i-ésimo indivíduo pode afetar o limite inferior para o parâmetro j, e l_j^t representa o limite inferior para o parâmetro j na geração t. L_j^t representa o valor de desempenho. A atualização do limite superior para o parâmetro j é ditada por:

$$u_j^{t+1} = \begin{cases} x_{k,j} & se\ x_{k,j} \geq u_j^t\ ou\ f(x_k^t) < U_j^t \\ u_j^t & caso\ contrario \end{cases} \tag{7.18}$$

$$U_j^{t+1} = \begin{cases} f(x_k) & se\ x_{k,j} \geq u_j^t\ ou\ f(x_k^t) < U_j^t \\ U_j^t & caso\ contrario \end{cases} \tag{7.19}$$

onde o k-ésimo indivíduo afeta o limite superior para o parâmetro j, e u_j^t representa o limite superior para o parâmetro j. U_j^t representa o valor de desempenho.

Usando o mecanismo apresentado acima, o intervalo pode se mover em direção aos limites inferior e superior em tempo real, de forma a acompanhar alterações no ambiente (Figura 57).

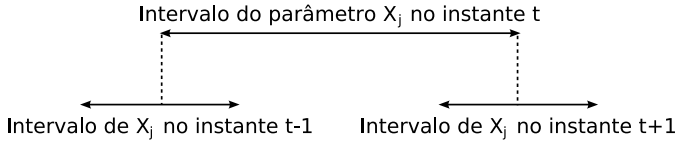

Figura 57: Re-escalonamento do conhecimento normativo.

7.3.1.2 Conhecimento Situacional

O conhecimento situacional provê um conjunto de casos amostrados que são úteis para a interpretação da experiência dos indivíduos. O conhecimento situacional guia os indivíduos para se "mover em direção aos melhores indivíduos", servindo como uma espécie de exemplo a ser seguido. Contém um conjunto de exemplares da população $e_1, e_2, \ldots, e_n$, onde n é o número de exemplares no conhecimento situacional.

A estrutura de dados do conhecimento situacional é representada como uma lista de indivíduos, onde cada exemplar no conhecimento contém um valor para cada parâmetro e seu valor de adequação (Figura 58).

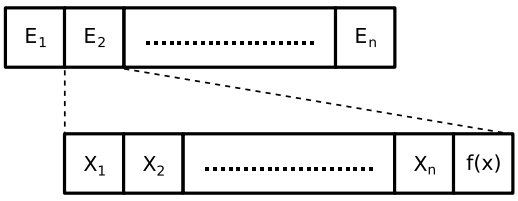

Figura 58: Estrutura do conhecimento situacional.

A atualização do conhecimento situacional adiciona o melhor indivíduo da população ao conjunto de conhecimento se ele superar o melhor conhecido ou ela reinicia a conjunto do conhecimento situacional quando uma alteração do ambiente é detectada, como mostado a seguir:

$$< E_1^{t+1}, E_2^{t+1}, \ldots, E_e^{t+1} >= \begin{cases} < x_{bst}^t, E_1^t, \ldots, E_{e-1}^t > & se\ f(x_{bst}^t) > f(E_i^t) \\ < x_{bst}^t > & se\ alteracao\ detectada \\ < E_1^t, E_2^t, \ldots, E_{e-1}^t > & caso\ contrario \end{cases} \quad (7.20)$$

onde x_{bst}^t é o melhor indivíduo da população no instante t.

7.3.1.3 Conhecimento Topográfico

O conhecimento topográfico foi originalmente proposto para atuar sobre padrões de cenários funcionais baseados em regiões[137]. Todo o cenário é dividido em células, de acordo com características espaciais, e cada célula mantém informações dos melhores indivíduos dentro de suas regiões. O conhecimento topográfico guia os indivíduos para emular o melhor da célula (equivalente a um ótimo local).

É representado por uma matriz multi-dimensional com céluas descritas por $c_1, \ldots, c_j$, onde j é o número de dimensões e c_i é o tamanho da célula para a i-ésima dimensão. O conhecimento estruturado é iniciado pela amostragem de uma solução em cada célula da matriz, e a criação de uma lista das melhores n células na matriz (Figura 59).

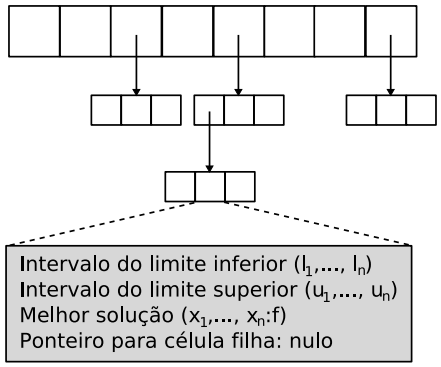

Figura 59: Estrutura do conhecimento topográfico.

A função de atualização divide uma célula em células menores se a adequação (*fitness*) de um indivíduo é melhor do que a adequação da melhor solução encontrada nessa célula. Se uma célula é dividida em células menores, as novas células são amostradas e os resultados usados para atualizar a lista de atuais melhores n células.

Uma célula é particionada se a adequação de um indivíduo selecionado nessa célula é melhor do que a melhor solução encontrada nessa célula até então, ou se a adequação se a adequação da melhor solução aumentou depois da detecção de um evento de mudança.

Quando um evento de mudança no ambiente ocorre, todas as ligações da lista se tornarão nulas e a melhor solução de cada célula são re-avaliadas de modo a proporcionar um novo particionamento das mesmas. Cada célula na estrutura de dados contém um intervalo inferior e superior $([1, u]_1, \ldots, [1, u]_n)$ para as n variáveis, as melhores soluções encontradas até o momento, e um ponteiro para suas subdivisões.

7.3.1.4 Conhecimento do Domínio

O conhecimento do domínio usa conhecimento sobre o domínio do problema para direcionar a busca. Por exemplo, em um cenário funcional composto por cones, o conhecimento acerca da forma do cone e dos parâmetros relacionados é útil no trabalho sobre os mesmos, durante o trabalho de busca. É generalizado para o intervalo de domínios de cada parâmetro e os melhores exemplos do espaço populacional, de forma semelhante ao conhecimento situacional.

O mecanismo de atualização para o conjunto de conhecimento de domínio adiciona a melhor solução encontrada até um determinado momento se ela for melhor que a adequação do melhor indivíduo no conhecimento de domínio. A principal diferença do conhecimento situacional é que o conhecimento de domínio não é reiniciado nas alterações do ambiente. A idéia é usar a variação do valor de adequação para gerar um nível de diversidade e um passo de mutação proporcional à magnitude de mudança na solução ótima antes e depois da ocorrência dessa mudança. A diferença na solução ótima é então mapeada para cada intervalo de parâmetro afim de gerar passos de mutação usando uma função como:

$$S_j = \frac{r_j \cdot \lambda}{melhor} \tag{7.21}$$

onde S_j é o tamanho do passo do parâmetro j, r_j é o intervalo do parâmetro j, e λ é a diferença do valor de adequação (*fitness*) entre o melhor indivíduo e o novo melhor indivíduo encontrado até agora. A função de atualização do conhecimento de domínio é dada por:

$$< D_1^{t+1}, D_2^{t+1}, \ldots, D_d^{t+1} >= \begin{cases} < x_{bst}^t, D_1^t, \ldots, D_{d-1}^t > & se\ f(x_{bst}^t) > f(D_i^t) \\ < D_1^t, D_2^t, \ldots, D_{d-1}^t > & caso\ contrario \end{cases} \tag{7.22}$$

onde D_1^{t+1} e D_2^{t+1} são o melhor e segundo melhor indivíduos do conhecimento de domínio, respectivamente, e x_{bst}^t é o melhor indivíduo do espaço populacional.

7.3.1.5 Conhecimento Histórico

O conhecimento histórico ou temporal monitora o processo de busca e armazena eventos importantes da busca. Estes eventos importantes podem ser um movimento significativo no espaço de busca ou a detecção de uma alteração no cenário. Indivíduos guiados

pelo conhecimento histórico podem consultar estes eventos armazenados para auxiliar a seleção da direção do movimento.

O conhecimento histórico contém uma média da mudança de distância e direção, e uma lista de eventos de mudança armazenados (Figura 60).

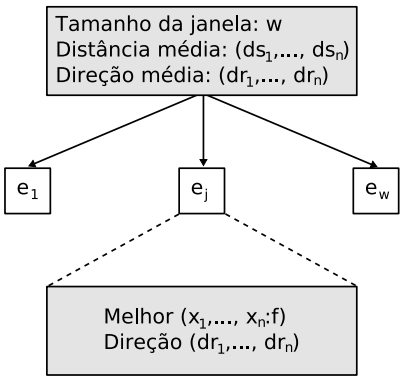

Figura 60: Estrutura do conhecimento histórico.

onde w representa o tamanho da memória para o histórico de eventos de mudança, $(ds_1, \ldots, ds_n)$, e $(dr_1, \ldots, dr_n)$ são a média das mudanças ambientais na distância e direção respectiva a cada n parâmetro. e_l até e_w são eventos de mudança para cada alteração que a melhor solução do ambiente anterior e a direção a qual cada parâmetro move em direção ao melhor indivíduo armazenado na lista de histórico, de tamanho w.

Quando uma alteração ambiental ocorre no instante t, a melhor solução atual $(x_1, \ldots, x_n : f)$ é armazenada junto com sua mudança de direação $(dr_1, \ldots, dr_n)$ entre os parâmetros do melhor indivíduo do momento e o melhor indivíduo antes da alteração ambiental. A direção dr_j pode tomar até três valores: 1, -1, ou 0. Quando $e_k \cdot dr_j$ é igual a 1, indica que o valor do parâmetro j aumenta entre ótimos na atual mudança do ambiente e_k e e_{k-t}; ou $e_k \cdot dr_j$ se iguala a -1 para indicar que o valor do parâmetro j foi diminuído, ou ainda zero, caso contrário. A seguinte função sumariza estas condições e é usada na atualização da direção do parâmetro j do k-ésimo evento:

$$e_k \cdot dr_j = \begin{cases} 1 & se\ e_k \cdot x_j - e_{k-1} \cdot x_j > 0 \\ -1 & se\ e_k \cdot x_j - e_{k-1} \cdot x_j < 0 \\ 0 & caso\ contrario \end{cases} \tag{7.23}$$

Quando um novo evento ocorre, a distância média do movimento é determinada por:

$$ds_j = \frac{\sum_{k=1}^{w-1} |e_k \cdot x_j - e_{k+1} \cdot x_j|}{w-1} \tag{7.24}$$

onde x_j é o valor do parâmetro j da melhor solução no momento que o evento e_k ocorre, w é o número de eventos na lista de histórico. A direção média de movimento para o j-ésimo parâmetro pode ser determinada usando os valores calculados por (7.23), como:

$$dr_j = \begin{cases} 1 & se \sum_{k=1}^{w} e_k \cdot dr_j > 0 \\ -1 & se \sum_{k=1}^{w} e_k \cdot dr_j < 0 \\ 0 & caso\ contrario \end{cases} \tag{7.25}$$

O conhecimento histórico é atualizado a cada evento de mudança pela atualização da lista de histórico e o movimento das médias de cada parâmetro como determinado pelas equações (7.24, 7.25). A janela da lista de histórico é atualizada pela adição de um novo evento de mudança e_i, formado pela melhor solução $(x_1, \ldots, x_n : f)$ e as direções $(dr_1, \ldots, dr_n)$ determinadas pela equação (7.23). Se a lista atingiu seu maior tamanho permitido w, o n-ésimo valor é descartado.

$$< e_1, \ldots, e_k >^{t+1} = \begin{cases} < e_1, \ldots, e_k >^t + e_{k+1} & se\ k < n \\ < e_2, \ldots, e_k >^t + e_{k+1} & caso\ contrario \end{cases} \tag{7.26}$$

Saleem e Reynolds[130] observaram os papéis e a contribuição destes cinco tipos genéricos de classes de conhecimento para otimizar o processo de solução, usando programação evolutiva (o mais comum em algoritmos culturais) como modelo populacional. A vantagem da abordagem da programação evolutiva é seu foco nos fenótipos de toda a população reprodutiva, da mesma forma que características culturais são expressas freqüentemente sob o aspecto de fenótipo[138]. Eles observaram a emergência de certas fases de solução de problemas, em termos de desempenho relativo, de diferentes fontes de conhecimento ao longo do tempo. Eles denominaram estas fases de *imitação grosseira* (*coarse grained*), *imitação fina* (*fine grained*), e fase de *backtracking*. Cada fase é caracterizada pela dominância de um subconjunto das fontes de conhecimento que são mais aptas na geração de novas soluções naquela fase. De fato, o subconjunto dominante das fontes de conhecimento é freqüentemente aplicado em uma seqüência especifica dentro de cada fase, o que faz parecer que uma fonte de conhecimento produz novas soluções que são conseqüentemente exploradas por outra fonte de conhecimento. As transições entre as fases ocorrem quando soluções produzidas por uma fase podem ser melhor exploradas por fontes de co-

nhecimento associadas à próxima fase. Estas fases emergiram em ambientes de problemas estáticos, dinâmicos e decrescentes (*deceptive*).

A fase de imitação grosseira freqüentemente emerge no início do processo de busca, ou quando o cenário da solução do problema muda dinamicamente, e a busca por uma nova solução deve começar. Na fase de imitação grosseira o conhecimento topográfico é dominante, produzindo as melhores soluções ao longo de 50% do tempo. O conhecimento situacional é o segundo mais apto, produzindo as melhores soluções ao longo de 25% do tempo. Na fase de imitação fina, o conhecimento situacional é o mais apto na geração dos melhores indivíduos, enquanto o conhecimento normativo e de domínio estão numa distante segunda posição. Na fase de *backtracking*, todas as fontes de conhecimento são igualmente aptas na geração de novas soluções. Problemas estáticos são a exceção, quando o conhecimento histórico tem pouca influência (o que de certa forma é de se esperar). Semelhantemente, em ambientes não-decrescentes (*non-deceptive*) a fase de *backtracking* ocorre com menos freqüência que as outras duas fases.

7.3.2 Ajustando o Espaço de Crenças

O exemplares do espaço situacional consistem nos melhor indivíduos encontrados até um determinado momento t, isto é, o espaço situacional deve ser atualizado por (7.20). Cada espaço de crenças deve ser atualizado por suas respectivas regras de atualização, isto é, o espaço normativo deve ser atualizado pelas regras definidas pelas equaçãoes (7.16, 7.17, 7.18, 7.19), o espaço de domínio atualizado por (7.22), o espaço histórico é atualizado por (7.23, 7.24, 7.25, 7.26)

7.3.3 Produzindo uma Nova Geração Influenciada Pelo Espaço de Crenças

Nos casos específicos da Programação Evolutiva (EP), o espaço de crenças pode influenciar o operador evolutivo de mutação, de duas maneiras principais: (*i*) determinando o tamanho do passo, o tamanho das mudanças; (*ii*) determinando a direção da mudança, aumentando ou diminuíndo o valor atual de um parâmetro. As funções de influência para Algoritmos Culturais sobre uma população regida pela Programação Evolutiva são geralmente expressadas por esses termos. A função de influência para o espaço normativo pode ser definida como:

$$x'_{i,j} = x_{i,j} + \mathcal{L}(I_i) \cdot N(0,1) \tag{7.27}$$

onde $\mathcal{L}(I_i)$ é o tamanho do intervalo de crença para a variável i, e $N(0,1)$ é um valor aleatória com distribuição normal de média 0 e desvio padrão 1. A função de influência para o espaço situacional pode ser definida por:

$$x'_{i,j} = \begin{cases} x_{i,j} + |\sigma_{i,j} \cdot N(0,1)| & se\ x_{i,j} < s_i \\ x_{i,j} - |\sigma_{i,j} \cdot N(0,1)| & se\ x_{i,j} > s_i \\ x_{i,j} + \sigma_{i,j} \cdot N(0,1) & caso\ contrario \end{cases} \tag{7.28}$$

onde $\sigma_{i,j}$ representa o tamanho do deslocamento para o i-ésimo parâmetro do j-ésimo indivíduo. s_i é o valor do melhor indivíduo para o parâmetro i, no espaço de crença.

Formulação semelhante pode ser feita para os demais espaços de conhecimento.

7.4 Conclusões

A IA possui uma aplicação acentuada em problemas que não se apresentam prontamente de uma forma algorítmica, embora muitas, senão todas as ferramentas da IA sejam algorítmicas. Mas é nesta disparidade que reside uma das características primordiais da IA dita convencional ou clássica. É preciso elaborar uma representação do problema de forma que ele seja aplicável às técnicas da IA clássica. Isto é, muito da inteligência da IA reside nesta representação, e não à uma determinada técnica, que realiza as busca pela solução a partir das informações agregadas a uma representação em particular.

Ao contrário da IA clássica, a *soft computing*, ou inteligência computacional, aborda problemas não algorítmicos a partir de uma direção aparentemente contrária da IA clássica. Muitas vezes a inteligência computacional não demanda qualquer representação do ambiente, pode aceitar os dados diretamente fornecidos pelos sensores de um robô móvel, por exemplo. A inteligência reside então na própria estrutura ou algoritmo usados para a busca de uma solução. Essa é uma diferença fundamental, uma vez que pode tornar desnecessário um passo que muitas vezes torna a solução de um problema demasiadamente complexo, às vezes inviabilizando a busca de uma solução pelos meios disponíveis pela IA clássica.

8 *Implementação do B-Spline:CAEP*

A implementação do algoritmo *B-Spline*:CAEP tomou como base a implementação de Chan Jin Chung para o algoritmo CAEP, *Cultural Algorithm Evolutionary Programming*[139]. Originalmente, o algoritmo de Chung suporta a minimização por meio de algoritmos culturais (CA), usando programação evolutiva (EP) como meio populacional, quanto por meio da programação evolutiva adaptativa padrão (SAEP - *Standard Adaptative Evolutionary Programming*). O espaço de crenças é restrito a intervalos simples para o domínio.

A estrutura SAEP não foi implementada no algoritmo *B-Spline*:CAEP, uma vez que o objetivo único do trabalho é a análise do comportamento do algoritmo cultural na geração de trajetórias definidas por *B-Splines*.

8.1 O Algoritmo CAEP

A implementação de Chung é bastante aderente ao algoritmo básico de Reynolds e Zhu[136], e ainda fornece um laço externo ao laço evolutivo para várias tentativas (*trials*) consecutivas, o que facilita a experimentação. Sucintamente, eis a implementação da função `main` do CAEP de Chung, em pseudo-código[1]:

[1] O código apresentado foi simplificado. Isso para torná-lo facilmente legível, e que todos os relacionamentos sejam visíveis.

Algoritmo 6: Pseudo-código para a função `main` do CAEP de Chung.

```
1  Iniciar a semente de aleatoriedade;
2  (TRIALS, NGENS, nindividuos) ← ConfigurarCAEP;
3  POP ← vetor de individuos POP(2 · nindividuos);
4  DOMAIN ← estrutra de dominio DOMAIN;
5  BS ← espaco de crencas BS;
6  τ1 ← (1.0/√(2.0 · nindividuos));
7  τ2 ← (1.0/√(2.0 · √nindividuos));
   enquanto (t < TRIALS) faça
8      Iniciar(POP, DOMAIN);
9      Avaliar(POP);
10     Iniciar(BS, POP);
       enquanto (g < NGENS) faça
11         GerarDescendentes(POP, τ1, τ2, DOMAIN, BS);
12         Avaliar(POP);
13         ContarVitorias(POP);
14         melhor ← Rank(POP);
15         Atualizar(BS, POP, melhor, g);
16         g ← g + 1
       fim
17     t ← t + 1
   fim
```

Na linha 3, é alocado um vetor com $2 \cdot nindividuos$, de tal forma que o vetor POP mantenha tanto a população vigente, quanto sua próxima geração na mesma estrutrua de dados.

Na linha 8, o vetor POP é iniciado sob a influência do domínio do problema. Cada indivíduo (os primeiros $nindivudos$[2]) da população têm seus parâmetros iniciados dentro dos limites mínimos e máximos de seu domínio, distribuídos da seguinte forma:

$$individuo_i = \mathcal{R}(DOMAIN_{inf}, DOMAIN_{sup}) \tag{8.1}$$

onde $\mathcal{R}(\bullet)$ é uma função probabilistica de distribuição normal, com centro em 0 e limite inferior em $DOMAIN_{inf}$ e limite superior em $DOMAIN_{sup}$.

[2]Lembre-se, POP mantém tanto a população vigente quanto seus descendentes.

Na linha 9 do algoritmo, POP é avaliada segundo uma função de adequação (*fitness*). Cada indivíduo da população tem seus parâmetros avaliados segundo uma função `ObjFunction()`, que retorna a resposta a ser minimizada.

Na linha 10, o espaço de crenças – BS – é iniciado, sob influência da população POP. No CAEP de Chung, o espaço de crenças contém apenas o conhecimento de domínio de cada parâmetro. A função `Iniciar(BS, POP)` busca o melhor indivíduo na população recém iniciada, copia os parâmetros desse melhor indivíduo para dentro do espaço de crenças, além de iniciar uma estrutura de referência (*beacons*) contendo os limites máximos e mínimos para cada parâmetro, obtidos a partir do domínio ($DOMAIN$).

A partir da linha 11, o algoritmo já se encontra em seu ciclo evolutivo. Nesta mesma linha, uma geração de descendentes é criada. A segunda metade do vetor POP é preenchida com os descendentes da população vigente. A geração de descendentes é uma das operações mais sensíveis deste algoritmo, pois deve assegurar que o espaço de crenças contenha valores válidos, isto é, se o espaço de crenças é *estável*, de modo que esse influencie a nova geração apropriadamente.

Como o espaço de crenças contém apenas as informações de domínio de cada parâmetro, este espaço precisa apenas ser corroborado contra estas informações. É verificado se o espaço de crenças tem armazenado valores válidos de máximo e mínimo, para cada parâmetro j, de acordo com o domínio, isto é, basicamente ele verifica se a seguinte condição é satisfeita:

$$|BS^j_{min} - BS^j_{max}| < |DOMAIN^j_{min} - DOMAIN^j_{min}| \tag{8.2}$$

Se a condição regida por (8.2) for verdadeira, o parâmetro j, do novo indivíduo $i + nindividuo$ é determinado por[3]:

$$\sigma^j_{i+nindividuos} = \sigma^j_i \cdot \exp\{\tau_1 \cdot N(0,1) + \tau_2 \cdot N(0,1)\} \tag{8.3}$$

$$ind^j_{i+nindividuos} = \begin{cases} ind^j_i + |\sigma^j_{i+nindividuos} \cdot N(0,1)| & se \quad ind^j_i < BS^j_{melhor} \\ ind^j_i - |\sigma^j_{i+nindividuos} \cdot N(0,1)| & se \quad ind^j_i > BS^j_{melhor} \\ ind^j_i + 0.5 \cdot \sigma^j_{i+nindividuos} \cdot N(0,1) & se \quad caso\ contrario \end{cases} \tag{8.4}$$

[3]Lembre-se, POP mantém tanto a população vigente quanto seus descendentes. Então $i + nindividuo$ começa o preenchimento da segunda metade do vetor POP, justamente a seção reservada para os descendentes.

onde $ind^j_{i+nindividuos}$ é o j-ésimo parâmetro do i+$nindividuos$-ésimo indivíduo descendente. ind^j_i é o j-ésimo parâmetro do i-ésimo indivíduo genitor. σ^j_i é um parâmetro de controle de mutação (8.3), exclusivo para cada j-ésimo parâmetro de um indivíduo. Sempre que um novo indivíduo é criado, este parâmetro é iniciado com 0.5. $N(\bullet)$ é uma função de distribuição gaussiana, com média 0 e desvio padrão 1. BS^j_{melhor} é o j-ésimo parâmetro do melhor indivíduo, desde o inicio do espaço de crenças[4].

Se a condição determinada por (8.2) for falsa, o parâmetro j, do novo indivíduo $i + nindividuo$ é determinado por:

$$ind^j_{i+nindividuos} = ind^j_i + \sigma^j_{i+nindividuos} \cdot N(0,1) \tag{8.5}$$

Caso os valores dos parâmetros gerados pelas equações (8.4, 8.5) extrapolem o domínio destes parâmetros em qualquer direção, eles são truncados para os valores de máximo ou mínimo.

Criada uma nova geração, de acordo com um espaço de crenças estável (8.2), e cujos parâmetros pertençam ao seus respectivos domínios, é momento de avaliar a recém criada geração de descendentes, e isso é feito na linha 12, exatamente da mesma forma que na linha 9, aqui, porém, os valores calculados para a segunda metade do vetor POP passam a se tornar relevantes, uma vez que essa seção do vetor passa contém, de agora em diante, a geração descendente.

Na linha 13, é realizado o confronto de cada indivíduo da população, tanto a população genitora, quanto a população descendente. Nesta função, `ContarVitorias(POP)`, cada elemento da população é confrontado com outros 50 elementos, escolhidos aleatoriamente[5]. Se este indivíduo confrontado, possuir um valor de adequação (*fitness*) menor[6] que cada um destes 50 elementos, é computada mais uma "vitória" para o indivíduo. Logo em seguida, na linha 14, o vetor POP é ordenado de forma descrescente em relação ao número de vitórias de cada indivíduo, isto é, o primeiro elemento do vetor POP possuí o indivíduo com o maior número de vitórias. A função `Rank(POP)` ainda retorna o indíce do indivíduo com o menor *fitness* dentro do vetor recém ordenado. As linhas 13 e 14 definem o comportamento pela busca de minimização do valor de adequação.

A linha 15 encerra na prática o ciclo evolutivo do CAEP. Nela o espaço de crenças é atualizado sob a influência de POP e do indivíduo com a menor adequação. Na função

[4] Relembre a linha 10.
[5] Os confrontos podem inclusive se repetir.
[6] Minimização do *fitness*.

`Atualizar(BS, POP, melhor,` g`)` é selecionado um "novo melhor indivíduo" para o espaço de crenças se o indivíduo apontado pelo retorno da função `Rank()` for melhor, isto é, tiver um valor de adequação menor do que aquele já existente no espaço de crenças. Os parâmetros do espaço de crenças, que antes foram iniciados com a função `Iniciar(BS, POP)`, agora são atualizados. A um número de indivíduos, determinado por:

$$selecionados = nindividuos \cdot 0.2 + \frac{10}{(g+1)} \tag{8.6}$$

são permitidos atualizar o espaço de crenças no que tange aos parâmetros lá armazenados. Onde g é número da geração. A estes primeiros indivíduos *selecionados* é permitido atualizar o espaço de crenças.

O critério de seleção é implícito e automático. Na linha 14, via função `Rank()`, a população é ordenada de acordo com o número de vitórias de cada indivíduo, em ordem descrescente. A função `GerarDescendentes()`, na linha 11, lida apenas com a primeira metade do vetor POP como geração genitora, preenchendo a outra metade com os descentes dessa geração.

Os passos evolutivos são repetidos enquanto um número máximo de gerações ($NGENS$) não for atingido. O laço evolutivo é repetido enquanto um número máximo de tentativas ($NTRIALS$) não for atingido. Não existem quaisquer outros critérios de parada.

8.2 A Implementação do *B-Spline*:CAEP

O algoritmo CAEP, que foi implementado na linguagem C por Chung, foi portado para a linguagem Python[7]. A linguagem Python é uma linguagem de *script*, orientada a objetos, o que proporciona a prototipagem rápida de aplicações (RAD[8]). Por ser uma linguagem de *script*, ela também possui gerenciamento automático de memória, inclusive *garbage collection*, o que abstrai detalhes deste tipo de gerenciamento da implementação do *B-Spline*:CAEP. Ela possui ainda um conjunto completo de funções matemáticas, da mesma forma que a biblioteca matemática padrão da linguagem C. Outro ponto forte, é a possibilidade do desenvolvimento de interfaces gráficas a partir do Gtk+/Glade[9] e da biblioteca *libglade*, o que de fato foi feito para a interação por parte do usuário.

O algoritmo *B-Spline*:CAEP foi implementado e testado sobre um sistema operacional

[7] http://www.python.org
[8] *Rapid Application Development.*
[9] http://www.gtk.org e http://glade.gnome.org

FreeBSD (versão 5.4). O sistema operacional, por sua vez, está instalado em um *laptop* ECS, modelo A532-2, com um processador Athlon M, de 1.2GHz, e 256 MB de memória RAM. Qualquer comparação em termos de desempenho, deve levar estas variáveis em consideração.

8.2.1 Alterações ao Algoritmo CAEP Original

Além da implementação em outra linguagem, o uso do paradigma de orientação a objetos melhorou a organização e legibilidade do código fonte. Foram criadas classes para as seguintes estruturas:

classe Domain: classe contendo dois vetores, o primeiro (`max`) contendo os valores máximos para cada parâmetro. O segundo (`min`) contendo os valores mínimos. O vetor `max` é iniciado com {800,600}, o vetor `min` é iniciado com {0,0}, isto devido ao fato do algoritmo utilizar apenas coordenadas do monitor de vídeo;

classe Individual: contém um vetor de parâmetros (`genoma`) de cada indivíduo, os valores de σ (`sigma`) para cada parâmetro, o valor de adequação (`fitness`), o número de vitórias (`wins`). O vetor de parâmetros tem tamanho estático durante a execução do algoritmo, mas pode ser determinado por meio da interface gráfica. Além deste número de parâmetros são considerados ainda os pontos de início e término da *spline*. Cada elemento deste vetor, cada *gene*, é formado por um par de coordenadas (x, y), e corresponde a um ponto de controle da curva *B-Spline*. Cada indivíduo representa uma solução de trajetória desde o ponto de início até o ponto de término;

classe Population: basicamente encerra um vetor de indivíduos (`individuals`) da população, as soluções sob iteração, serve de *interface* para a implmentação da classe `PopulationEP`;

classe PopulationEP: classe que estende a classe `Population` de modo a conter métodos específicos para dar suporte ao algoritmo *B-Spline*:CAEP, tais como a função `PopulationEP::ObjFunction`, `PopulationEP::Evaluate`, específicas ao problema de geração de trajetórias;

classe Beacon: classe que serve de guia para cada parâmetro do espaço de crenças e a geração de novos indivíduos. Contém os membros `lowX`, `lowY`, `low_score`, `highX`, `highY` e `high_score`, que armazenam os mínimos e máximos valores para cada

coordenada x e y, além do *fitness* mínimo e máximo do indivíduo que alterou os limites dos parâmetros;

classe BeliefSpace: classe que contém um vetor de `Beacons`, cada elemento relacionado a um parâmetro, a um par de coordenadas (x, y);

classe Arena: classe que contém os métodos responsáveis pelo confronto da população genitora e descendente, além do método de ordenação destas populações em função do número de vitórias;

classe CAEP: classe para a configuração do algoritmo CAEP e que contém como método principal o laço evolutivo. Dentro deste laço evolutivo existem ganchos (*hooks*) para a interface gráfica;

classe Point: define um ponto com as coordenadas x, y, z, dá suporte à classe `Spline`;

classe Spline: classe que define os pontos para o desenho de uma curva *B-Spline*. Pede como únicos parâmetros o vetor de pontos de controle da curva *B-Spline*, bem como a resolução, o número de pontos usados para definir a curva *B-Spline* final;

classe SplineEx : estende a classe `Spline`, de tal forma que as coordenadas (x, y) dos pontos que definem a curva *B-Spline* são convertidos para as coordenadas de monitor, onde o ponto de origem das coordenadas se encontra no canto superior esquerdo, e cresce para a direita e para baixo;

classe SplineGUI: estende a classe `SplineEx`, é capaz de desenhar a si própria na interface gráfica;

classe Obstacle: classe que, a partir de um conjunto de vértices que definem o contorno de um obstáculo, é capaz de desenhá-lo na interface gráfica. Um vetor de objetos `Obstacle` define um ambiente, um mapa. Para os experimentos foram criados três tipos de mapas: `low_complexity`, `medium_complexity`, `high_complexity`, que supostamente aumentam de complexidade, tanto em termos do número de obstáculos quanto sua forma;

classe CAEPThread: encapsula uma *thread*, de tal forma que o algoritmo *B-Spline*:CAEP seja executado de uma forma semi-dissociada da interface gráfica, o que permite que a interface gráfica permaneça funcional durante a execução do *B-Spline*:CAEP;

classe GUI: responsável por criar, gerenciar, e manipular os eventos da interface gráfica.

O algoritmo CAEP permaneceu praticamente intacto em seu funcionamento. As principais alterações ficaram por conta da classe `Domain`, que é automaticamente iniciada com os limites físicos da janela gráfica, o "ambiente". Valores para x em 799 e 0 para seu máximo e mínimo respectivamente. E valores para y em 599 e 0 para seu máximo e mínimo respectivamente. O que perfaz uma área útil de 800×600 pixels.

Um indivíduo possui um vetor de parâmetros (de fato, cada parâmetro é um *ponto de controle* da curva *B-Spline*). O comprimento deste vetor é estático, mas pode ser ajustado pelo usuário. Cada elemento é formado pelo par de coordenadas (x, y). Além deste número de *pontos de controle*, devem ser considerados, e adicionados, os pontos de início e término da curva, que na verdade nada mais são do que os pontos de início e objetivo da trajetória. Pode-se considerar uma associação implícita entre as classes `Point` e cada elemento do vetor `Individual::genoma`. A coordenada z é ignorada neste trabalho. As demais classes mantém seu funcionamento original, foram alterados apenas os pontos onde o contexto se faz relevante.

Foram necessárias alterações no algoritmo que mantém o espaço de crenças, de tal forma que ele possa lidar apropriadamente com as duas coordenadas de cada parâmetro de uma solução candidata. O método `PopulationEP::GenerateOffspring()` também teve de sofrer este tipo de adequação.

A função de adequação é totalmente dependente do problema tratado. Nesta implementação ela leva em consideração dois aspectos fundamentais. Ela, primeiramente, busca penalizar as soluções cujos os *pontos de controle* sejam responsáveis pela colisão de qualquer segmento que define a trajetória. Depois, usa o comprimento da trajetória como valor adicional ao *fitness* de cada solução[140, 141, 142, 143, 144, 145]. No caso de não haverem colisões, o algoritmo passa apenas a buscar o caminho mais curto. Por se tratar de um algoritmo de minimização, o *B-Spline*:CAEP busca soluções que minimizem o número de colisões e o comprimento de cada trajetória. O pseudo-código, a seguir, ilustra o algoritmo implementado.

Algoritmo 7: Pseudo-código para a função de avaliação `ObjFunction`.

```
1  ctrl_points ← [ ];
   para cada gene do genoma de um individuo faça
2      pt ← Point(gene[X], gene[Y]);
3      Adicionar ao vetor de pontos de controle (ctrl_points) o ponto pt;
   fim
4  spline ← SplineEx(ctrl_points, resolution);
5  spline.SplineCurve();
6  collided_pts ← [ ];
   para cada i no intervalo [0:(resolution - 1)] faça
7      pt ← (spline.outp[i][0], spline.outp[i][1]);
       se pt ∈ obstaculo então
8          Adicionar ao vetor de pontos colididos (collided_pts) o ponto pt;
       fim
   fim
9  penalty ← 0.0 ;
   para cada cpt de ctrl_points faça
10     p ← (cpt.x, cpt.y) ;
       para cada j de collided_pts faça
11         dist ← EuclidianDistance(p, j) ;
12         penalty ← penalty + 100.0 · (dist)/((α· (dist²)) + (β · dist)) ;
       fim
   fim
13 length ← 0.0;
   para cada i no intervalo [0:len(spline.outp)-1] faça
14     pt1 ← spline.outp[i];
15     pt2 ← spline.outp[i + 1];
16     length = length + EuclidianDistance(pt1, pt2)
   fim
```

A linha 3 do algoritmo adiciona as coordenadas contidas em cada indivíduo da população ao vetor de *pontos de controle*. Na linha 4, a partir destes *pontos de controle*, e de um parâmetro estático de resolução `resolution`, é criado um objeto `SplineEx`[10].

[10] Como esta *B-Spline* não será desenhada na tela, ela não precisa conter este tipo de especialização.

A linha 5 requisita ao objeto `spline` o cálculo dos pontos que formam a curva *B-Spline*. Este método preenche um vetor (`spline.outp`) com um número de `resolution` pontos. A linha 8 determina quais destes pontos recém calculados estão contidos dentro dos limites de um obstáculo. Na prática verifica-se se o pixel determinado pelas coordenadas de `pt` possui uma cor característica para os obstáculos, que mesmo definidos por meio de polígonos, são desenhados preenchidos. Os pontos da *B-Spline*, que colidem com um obstáculo, são adicionados ao vetor `collided_pts`.

Na linha 12 é definida a grandeza da penalidade para cada ponto da *spline* colidido com um obstáculo. Esta grandeza é definida levando em consideração a influência de cada *ponto de controle* em relação a ponto sobre o obstáculo. Os parâmetros α e β servem para ajuste da função de penalidade.

Idealmente, cada ponto da *B-Spline* é influenciado por uma tríade de *pontos de controle*. Poder-se-ia determinar a distância euclidiana de cada ponto colidido em relação a cada *ponto de controle*, acumulá-las em um vetor, ordenar este vetor em ordem crescente, e selecionar apenas as três primeiras distâncias para os pontos de maior influência. E, provavelmente, usar estas distâncias como parâmetro de penalização. Porém, a experimentação demonstrou que os pontos que definem a *B-Spline* apresentam um comportamento atrativo não-linear. Foi observado que os *pontos de controle* praticamente exercem uma "força de atração" sobre os pontos da *B-Spline*, de tal forma que a concentração dos pontos da *B-Spline* seja consideravelmente maior nos pontos de inflexão da curva. Aparentemente, quando mais abrupto o ponto de inflexão, maior a concentração de pontos nesta região de inflexão.

De posse desta observação, foi proposta uma função de penalização, que acompanhasse de forma aproximada este comportamento, que fizesse o uso deste comportamento para penalizar os indivíduos em função dos *pontos de controle*. Como cada *ponto de controle* exerce uma atração que é visivelmente inversamente proporcional à distância aos pontos da *B-Spline*, optou-se pelo uso de uma função que apresente comportamento semelhante, neste caso, foi usada como referência inicial a função de gravitação de Newton.

$$F = G \cdot \frac{m_1 \cdot m_2}{d^2} \tag{8.7}$$

onde G é a constante gravitacional, originalmente igual a $6.67 \cdot 10^{-11}$. m_1 e m_2 são as massas dos dois corpos em análise, e d é a distância entre esses dois corpos. Assim, a força gravitacional é inversamente proporcional ao quadrado da distância entre os dois corpos de massa m_1 e m_2.

A função de gravitação de Newton é indicada para distâncias inter-planetárias, e massas planetárias também. O parâmetro G poderia ser usado para ajustar estas proporções astronômicas. Porém, a razão do inverso do quadrado da distância se mostrou inapropriada para o ajuste fino da função de penalização. Após alguma experimentação, a função de penalização, como definida na linha 12 do algoritmo anterior, assumiu a seguinte forma:

$$penalty = 100.0 \cdot \frac{d}{\alpha \cdot d^2 + \beta \cdot d} \tag{8.8}$$

onde d é a distância euclidiana entre um *ponto de controle* e um ponto da *B-Spline* em colisão com um obstáculo. Os parâmetros α e β servem para qualquer ajuste fino que se faça necessário.

Complementando o método `PopulationEP::ObjFunction`, ilustrado pelo algoritmo anterior, o *fitness* de uma solução também leva em consideração seu comprimento.

8.3 Interação com o *B-Spline*:CAEP

A interface gráfica do *B-Spline*:CAEP tenta comportar todas as informações diretamente necessárias aos experimentos(Figura 61). Conta com a configuração dos parâmetros evolutivos do algoritmo, como número de tentativas, tamanho da população, número de gerações, e número de parâmetros por solução, sem considerar os pontos de início e objetivo (Figura 62). Permite as configurações de visualização, resolução da curva *B-Spline* (Figura 63), como um múltiplo do número de parâmetros, a seleção entre os três tipos de mapas, de baixa, média e alta complexidade, e a seleção interativa dos pontos de início e término da trajetória (Figura 64). A cada três gerações a interface apresenta o estado da população.

A interface também é capaz de plotar algumas estatísticas da população (Figura 65), como o melhor *fitness* ao longo das gerações, a média deste valor para cada geração, bem como sua variância. Para estas plotagens faz uso da ferramenta Gnuplot. Caso seja necessário, a ferramenta gera um diretório contendo dados para cada indivíduo da população, além das estatísticas da população. É possível também ajustar e pré-visualizar os parâmetros do método `PopulationEP::ObjFunction` (Figura 66).

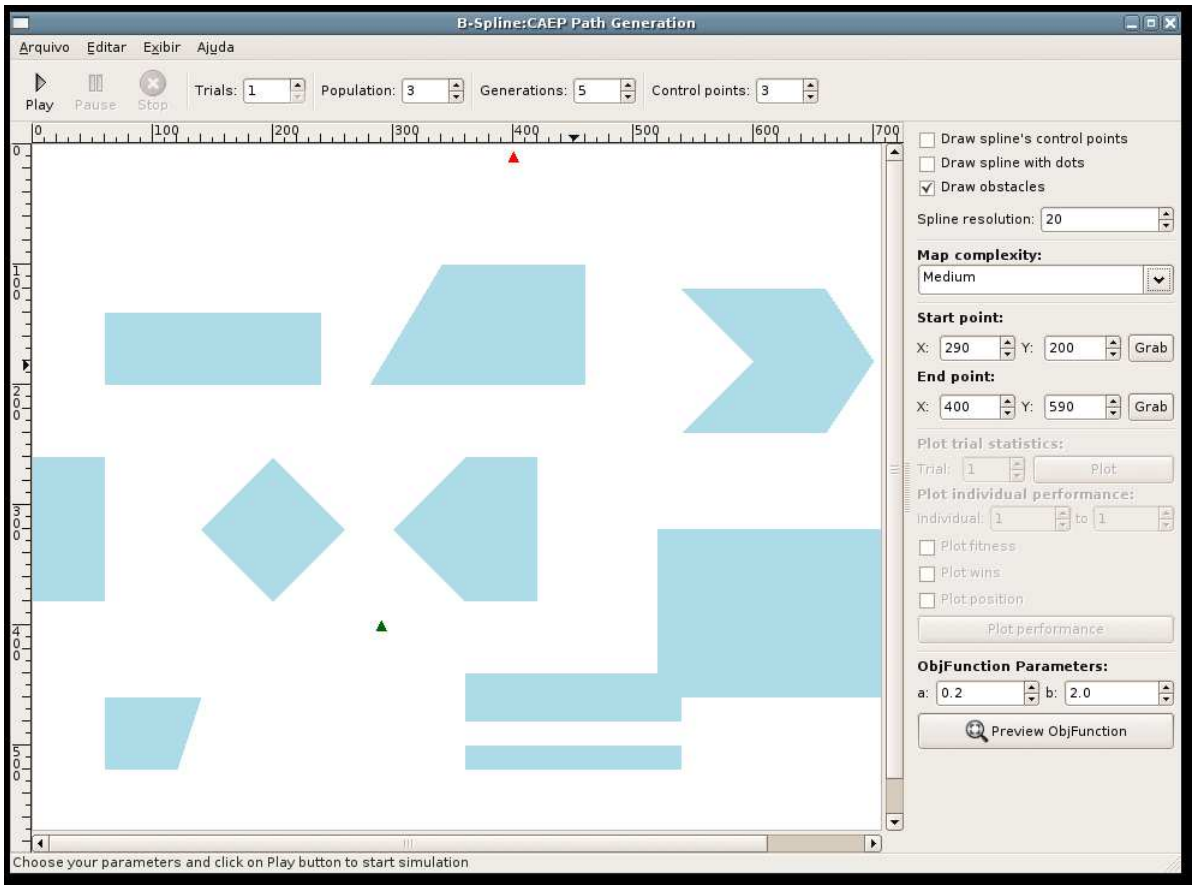

Figura 61: Interface gráfica do *B-Spline*:CAEP.

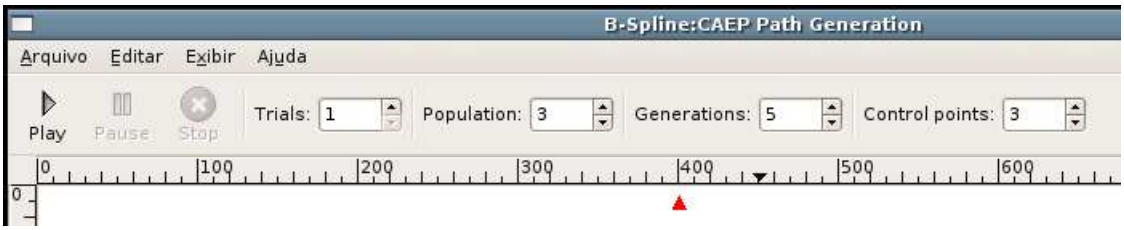

Figura 62: Barra de ferramentas para a configuração dos parâmetros evolutivos.

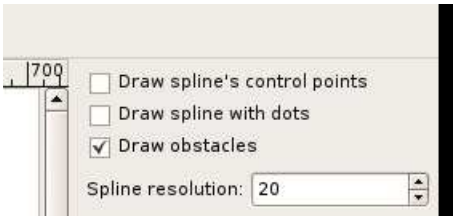

Figura 63: Configurações disponíveis para a visualização e a resolução da *B-Spline*.

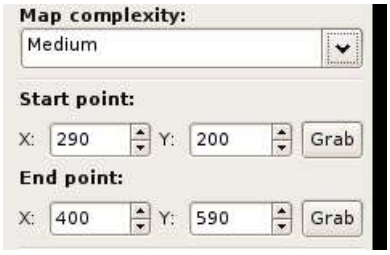

Figura 64: Seleção da complexidade do mapa e os pontos de início e objetivo da trajetória.

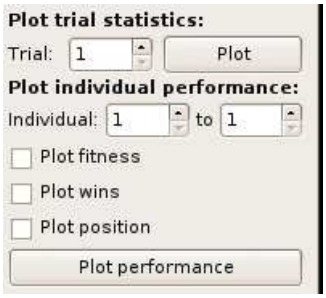

Figura 65: Opções para a plotagem das estatísticas da população.

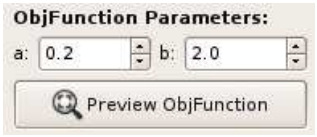

Figura 66: Configuração dos parâmetros α e β do método `PopulationEP::ObjFunction`.

9 *Simulação do B-Spline*:CAEP e Conclusões

As simulações *B-Spline*:CAEP, aqui apresentadas, foram realizadas nos três mapas disponíveis *a priori* à ferramenta. Cada mapa foi submetido a 30 execuções do algoritmo, as soluções e eventos mais relevantes são comentados a seguir.

Ao longo das simulações, pequenas falhas na ferramenta foram constatadas. A mais relevante delas diz respeito à detecção de colisões pelos segmentos constituíntes da curva *B-Spline*. Em algumas situações, a colisão simplesmente não é detectada, enquanto em outras, mais numerosas, o algoritmo consegue detectar as colisões apropriadamente.

Basicamente, o comportamento errôneo advém do fato que, a pesquisa pelos *pixels* marcados como ocupados por um obstáculo nem sempre retorna o resultado correto. Mesmo após a reimplementação do método diversas vezes, o problema parece persistir, aparentemente no *toolkit* usado para a interface gráfica. Ou ainda, a pesquisa pelos *pixels* pode estar sendo realizada enquanto a interface está sendo atualizada, isto é, enquanto o ambiente está sendo redesenhado. Soluções possíveis englobam o uso de semáforos, o que de certa forma degradaria o desempenho da ferramenta, embora esta não seja uma perda tão relevante, ou de técnica de *double buffering*, na qual existem duas regiões para desenho do ambiente, uma para exibição na tela, e a segunda para o trabalho do algoritmo, quando estas duas regiões são alternadas.

9.1 Simulações no Ambiente de Baixa Complexidade

O ambiente de *baixa complexidade* tem como principal finalidade verificar a habilidade do *B-Spline:CAEP* em encontrar caminhos livres de colisão, e em seguida, minimizar o comprimento destes caminhos. O algoritmo, neste ambiente, foi configurado com uma população com 10 soluções, cada uma delas contendo 5 pontos de controle, por 30 gerações.

Dadas estas configurações, as figuras (67 a 73) a seguir ilustram o comportamento

médio *B-Spline*:CAEP.

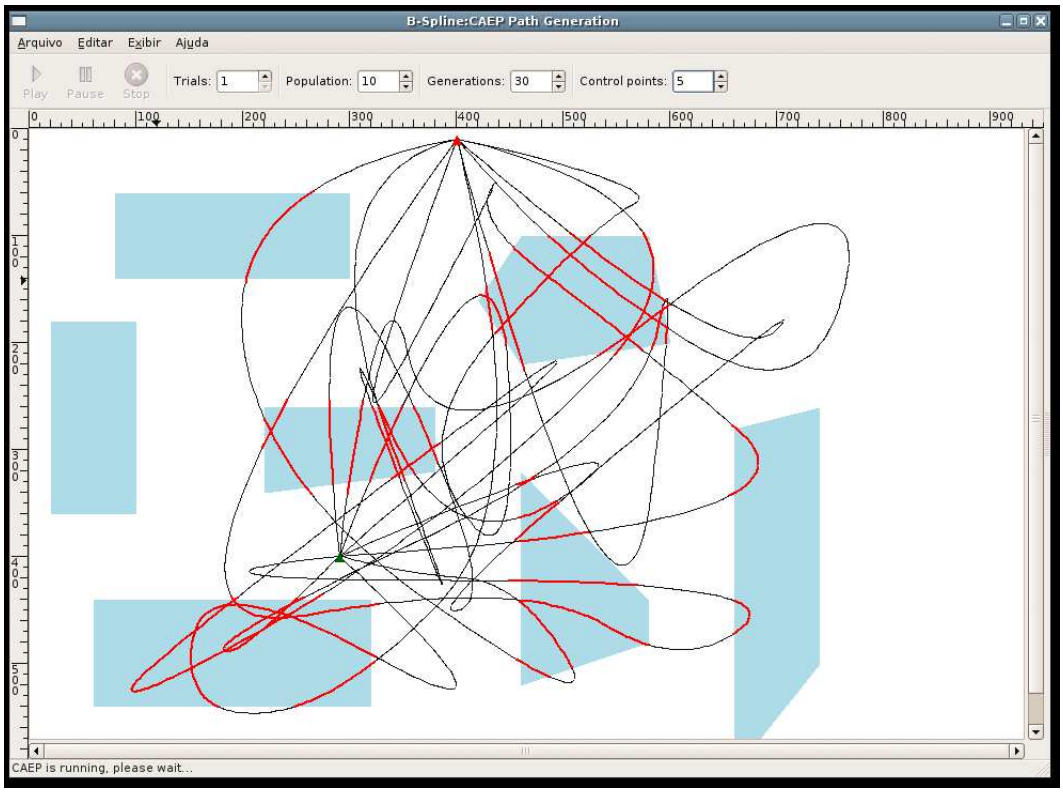

Figura 67: Soluções iniciais.

Para estas configurações o algoritmo não foi capaz de encontrar, em nenhum teste, um caminho pelo menos livre de colisão. Em vista destes resultados, o teste foi repetido, porém agora com uma população com 10 soluções, cada uma delas contendo 5 pontos de controle, por 200 gerações.

As figuras (74) a (80) a seguir ilustram o comportamento médio *B-Spline*:CAEP, para o teste com 200 gerações.

A observação dos resultados, dos testes com 200 gerações, indicam que o *B-Spline*:CAEP é capaz de encontrar caminhos livres de colisão, e até mesmo minimzar seu comprimento. Porém, pode-se notar, em alguns experimentos, um fenômeno ocasionado pelo *conhecimento de domínio*, existente no espaço de crenças. Porções da curva deixam de sofrer mutações significativas, por este conhecimento de domínio "afunilar" as variações de alguns parâmetros (*pontos de controle*), enquanto outras regiões da curva continuam a sofrer variações.

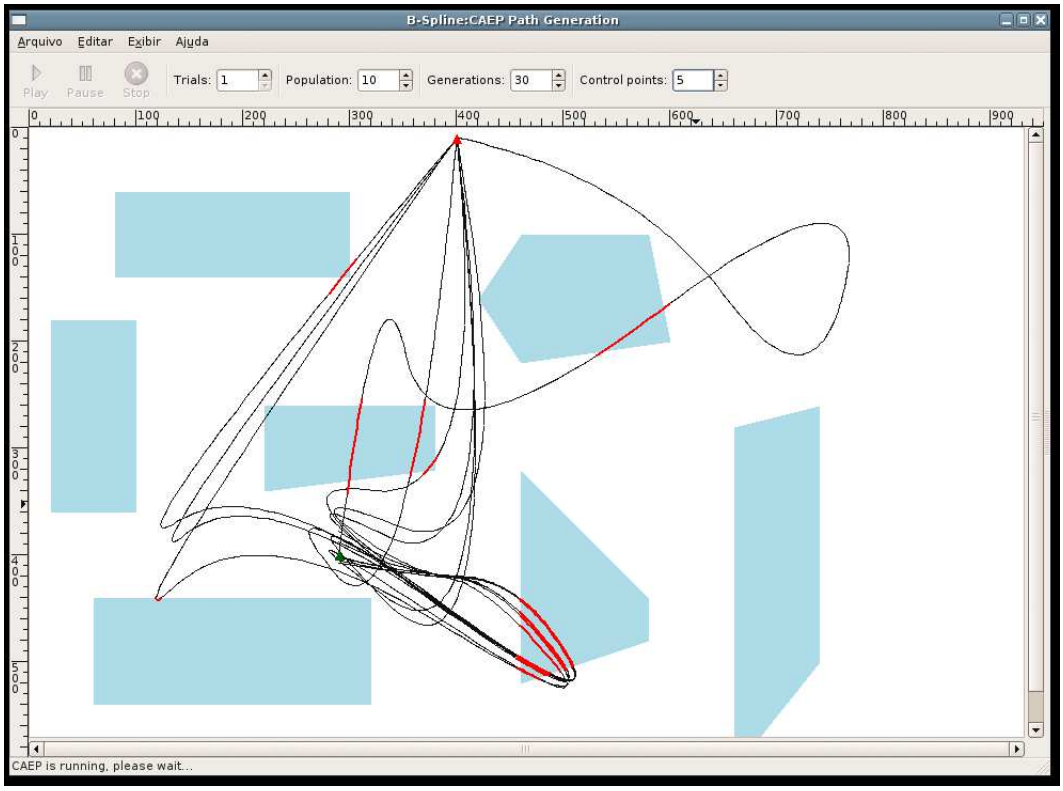

Figura 68: Soluções após 10 gerações.

9.2 Simulações no Ambiente de Média Complexidade

O ambiente de *média complexidade* tem como finalidade encontrar caminhos livres de colisão, minimizar o comprimento destes caminhos em um ambiente relativamente mais complexo do que o anterior, onde existem passagens estreitas e obstáculos côncavos. O algoritmo, neste ambiente, foi configurado com uma população com 10 soluções, por 200 gerações, com 3 e 5 pontos de controle. As figuras (81) a (82) ilustram alguns resultados.

Os testes demonstram que é preciso encontrar um compromisso para o número de pontos de controle da trajetória. Para um ambiente repleto de obstáculos, ou no qual os obstáculos possuem formas complexas, é necessário um maior número de pontos de controle. A situação contrária exige um número menor de pontos de controle; caso contrário, o caminho acaba contendo segmentos desnecessários, que poderiam ser seguramente descartados, e que no final acabam desperdiçando movimentos.

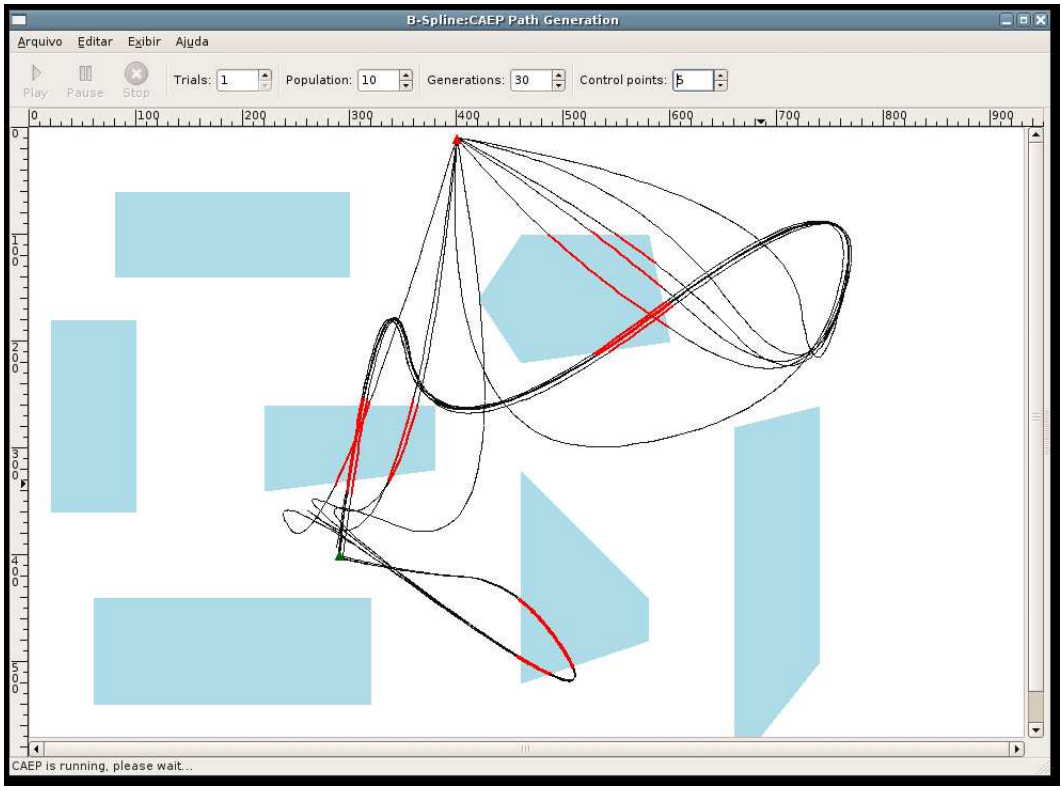

Figura 69: Soluções após 20 gerações.

9.3 Simulações no Ambiente de Alta Complexidade

O ambiente de *alta complexidade* tem como finalidade avaliar o comportamento do algoritmo para obstáculos com forma de "U", encontrar caminhos livres de colisão, e minimizar o comprimento destes caminhos. O algoritmo, neste ambiente, foi configurado com uma população com 10 soluções, por 200 gerações, e 5 pontos de controle. As figuras (83) a (84) ilustram alguns resultados.

Os testes no ambiente de alta complexidade apresentaram um sucesso de praticamente 60%, o que não é um desempenho excepcional.

9.4 Conclusões Sobre as Simulações Realizadas

A análise das simulações trazem à tona algumas constatações importantes. Por exemplo, ficou evidente, após os testes, que a estrutura populacional EP pode não ser a mais indicada para o planejamento de trajetórias, ainda mais associada a um algoritmo cultural (CA). Ela se mostra simplista, uma vez que altera a população apenas por operadores de mutação (busca aleatória). Outras abordagens, sem o componente cultural, mas com

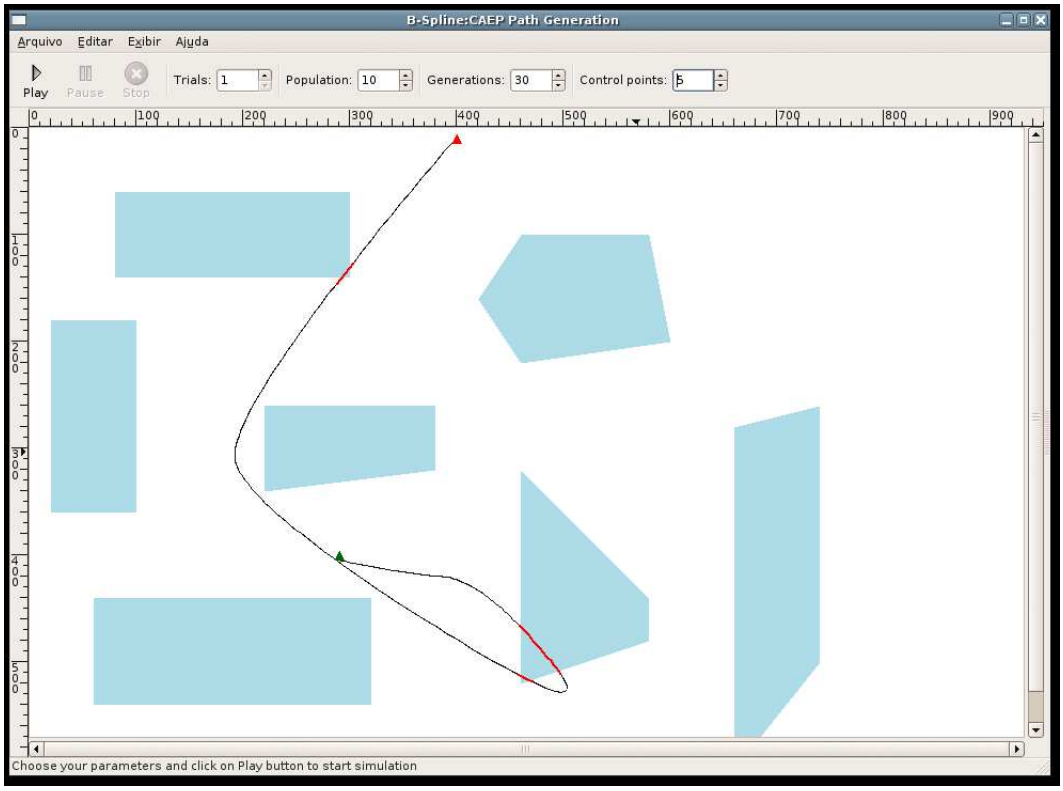

Figura 70: Solução encontrada após as 30 gerações.

um componente populacional de GAs[144, 142], se mostram mais flexíveis no sentido de fazerem uso de outros operadores e de heurísticas, sendo assim mais eficientes.

O próprio comportamento das curvas B-Spline, mesmo que previsível polinomialmente, se mostra complexo quando interage com o ambiente em questão; a forma final da curva é uma combinação do efeito de duas ou mais *funções de mistura*, ou de três ou mais *pontos de controle*. A função de adequação deve possuir harmonia suficiente de forma a fazer uso eficiente destas características e seu relacionamento com o ambiente, as colisões com os obstáculos, neste caso.

Outra característica que seria desejável, e que não foi implementada no *B-Spline*:CAEP, é o tamanho dinâmico dos genomas, o número variável de *pontos de controle* em cada solução. A complexidade do ambiente, como o número de obstáculos entre o ponto de partida e objetivo, e a própria complexidade geométrica dos obstáculos são fatores relevantes para a determinação do número de parâmetros da trajetória. A ausência do comportamento adaptativo a este fator se mostrou fundamental. Poderiam também ser implementados operadores tais como os sugeridos por [144]:

Reparo: insere um ponto de controle entre outros dois pontos de controle. O objetivo

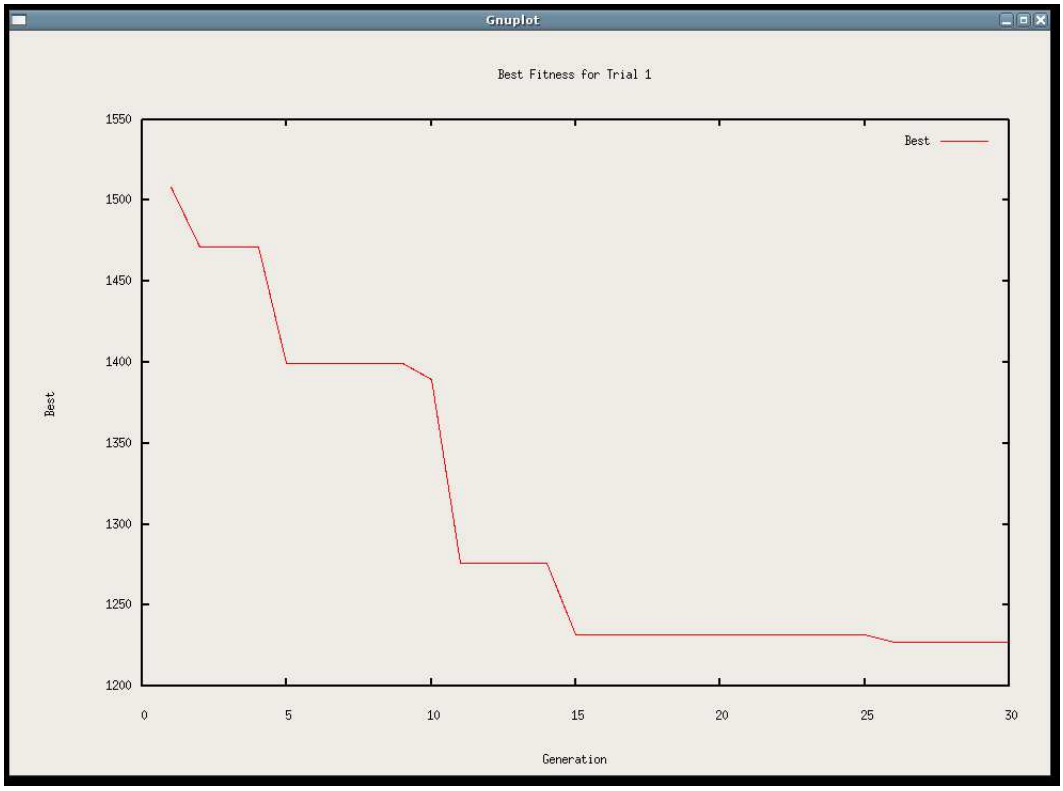

Figura 71: Variação do melhor *fitness*.

deste operador é desviar um segmento que passa sobre uma região ocupada por um obstáculo;

Remoção: escolhe aleatoriamente um ponto de controle para ser removido, apenas se essa remoção não gerar uma colisão. Tem o objetivo de simplificar e reduzir o comprimento do caminho;

Melhoria: escolhe aleatoriamente um ponto de controle e tenta movê-lo para uma região de melhor localização.

Além dos operadores de *recombinação* e *mutação* adaptados para o problema de planejamento de trajetórias.

O próprio espaço de crenças, e seu conhecimento de domínio, se mostrou danoso ao planejamento de trajetórias, em alguns casos. Nestes casos particulares, ficou aparente o "engessamento" de trechos da trajetória, restringindo a diversidade e a busca por novas soluções. O conhecimento de domínio acabou causando uma convergência local prematura dentro do próprio conjunto de parâmetros.

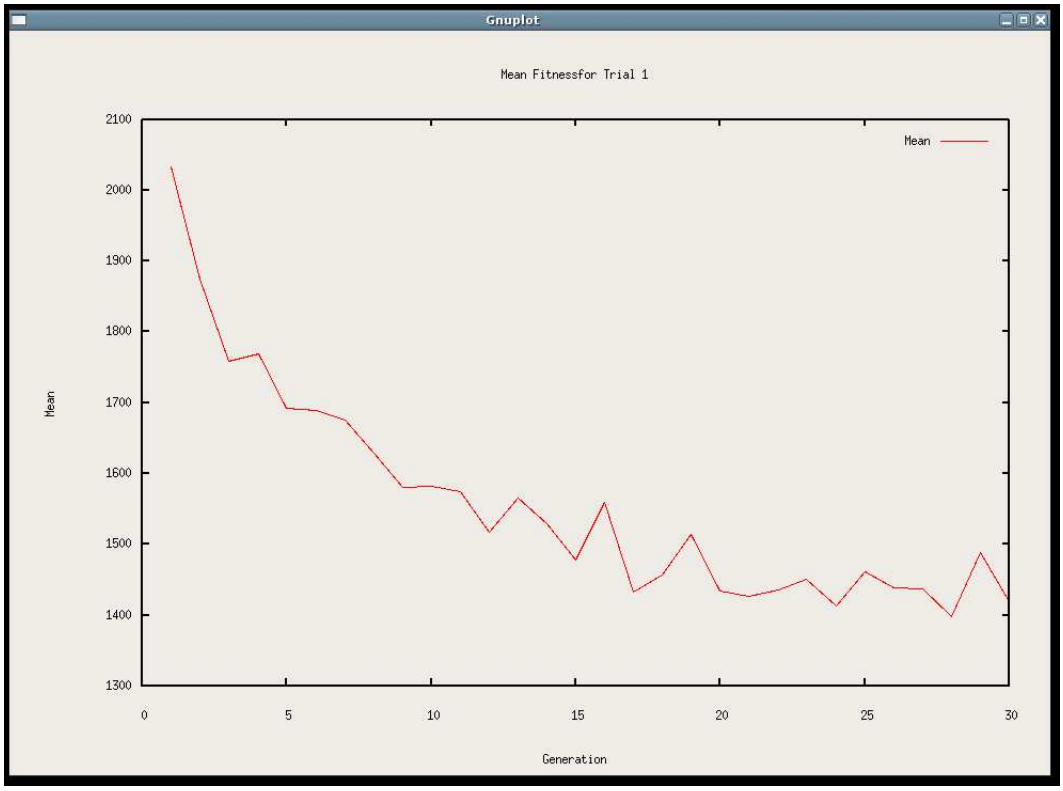

Figura 72: Variação da média do *fitness*.

9.5 Trabalhos Futuros

O algoritmo *B-Spline*:CAEP, a partir da implmentação atual, apresentou um comportamento mínimo para o planejamento de trajetórias. Ele é capaz de traçar trajetórias livres de obstáculos. É até mesmo capaz de minimizar o comprimento deste caminho livre de colisões. Porém, não se mostrou totalmente confiável, uma vez que nem todas as suas soluções são livres de colisão, por exemplo. As sugestões para trabalhos futuros converge, principalmente, na solução destas restrições:

Substituição do componente populacional: visando suplantar as limitações do componente EP. Como um componente GA, como citado anteriormente, e a comparação das principais diferenças;

Criação de novos operadores genéticos: evidente pela restrição na variedade de operadores genéticos do componente populacional EP. Potencializando a busca pelo uso apropriado de heurísticas do problema;

Uso de outros tipos de conhecimento: enriquecer o espaço de crenças com outros tipos de conhecimento. Por exemplo, o *conhecimento histórico* poderia armazenar

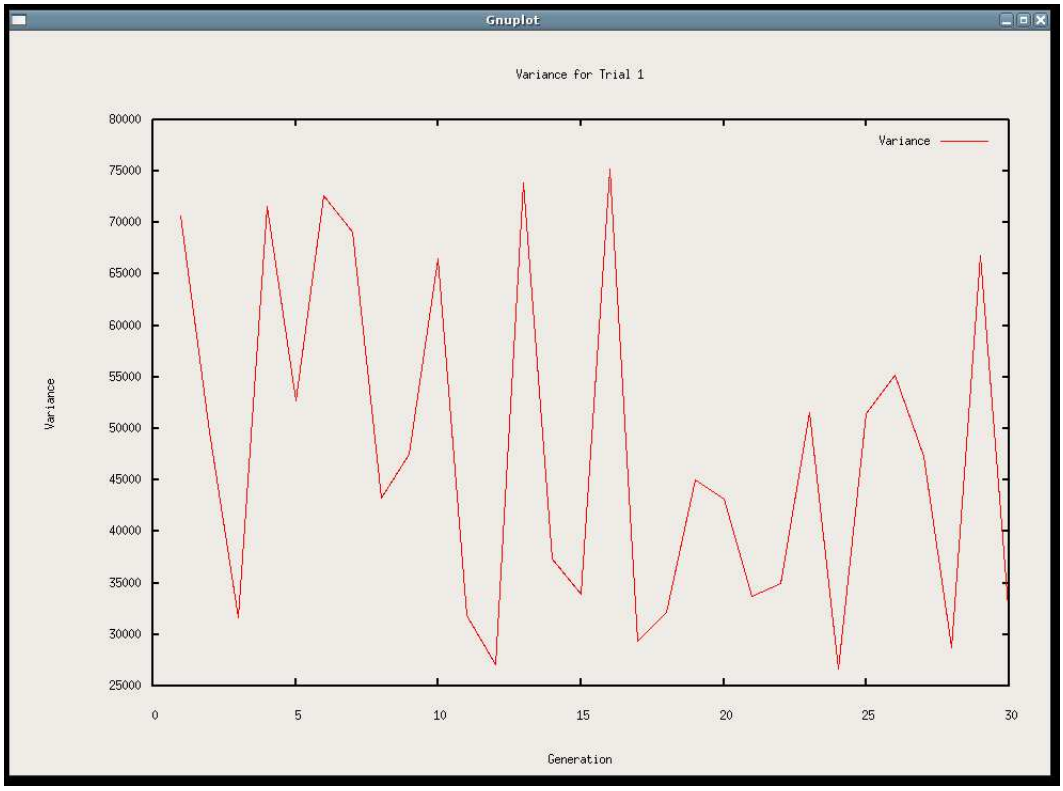

Figura 73: Variância do *fitness*.

o passo que levou à criação de uma trajetória excepcionalmente adaptada, para reproduzí-lo posteriormente. O *conhecimento topográfico* poderia ser usado para classificar diversos grupos de soluções e acirrar a competição intra-grupos;

Comprimento variável dos genomas: esta implementação tornaria as soluções adaptáveis às complexidades do ambiente. Evitaria também a criação e a posterior manutenção de segmentos desnecessários;

Aprimoramentos na geração de estatísticas: gerar "fotos" da população em intervalos regulares, para fins de análise e documentação.

9.6 Conclusões

O trabalho desta dissertação representa os fundamentos do planejamento de trajetórias por meio de um par evolutivo, composto por um componente populacional e outro cultural. Até onde se pôde apurar pela revisão bibliográfica, esta abordagem ainda é inédita para este tipo de problema. Porém, o mais importante, e objetivo primordial, foi a análise que as simulações proporcionaram, e as direções que elas apontam para o

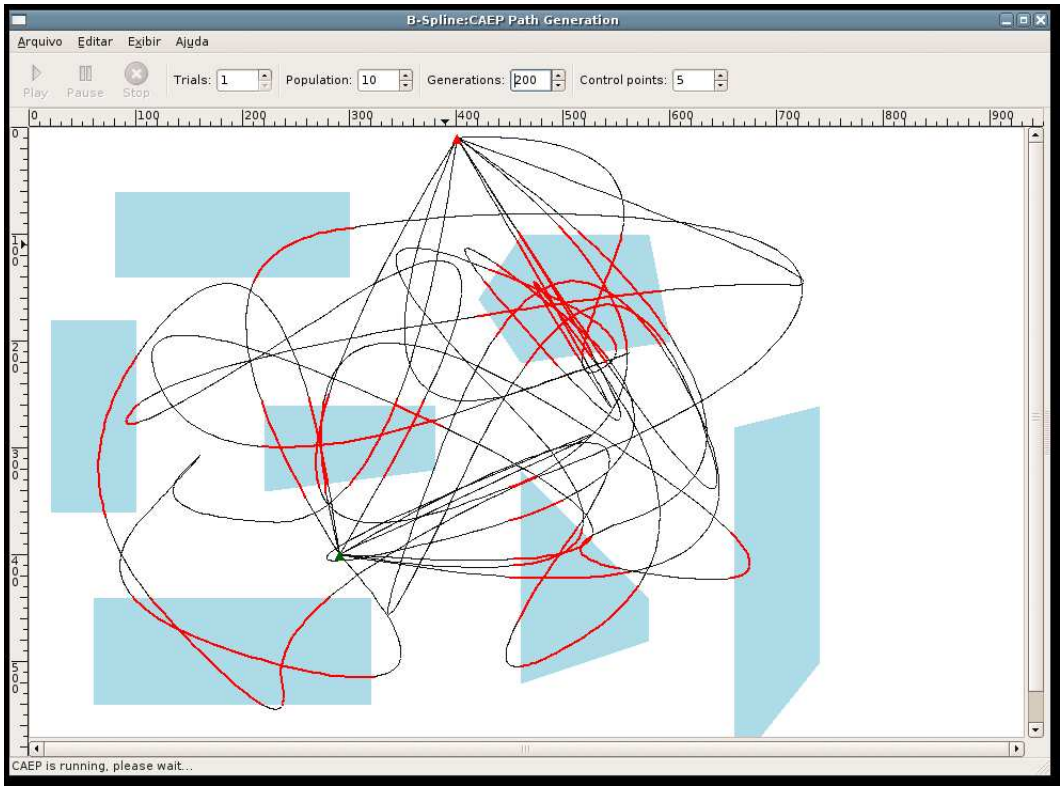

Figura 74: Soluções iniciais.

aprimoramento do trabalho implementado e da técnica proposta.

Esta técnica pode inclusive ser aproveitada por outros campos de pesquisa, como aerodinâmica, projeto do produto, resistência dos materiais, e qualquer outro campo que demande a busca por uma *curva suave*.

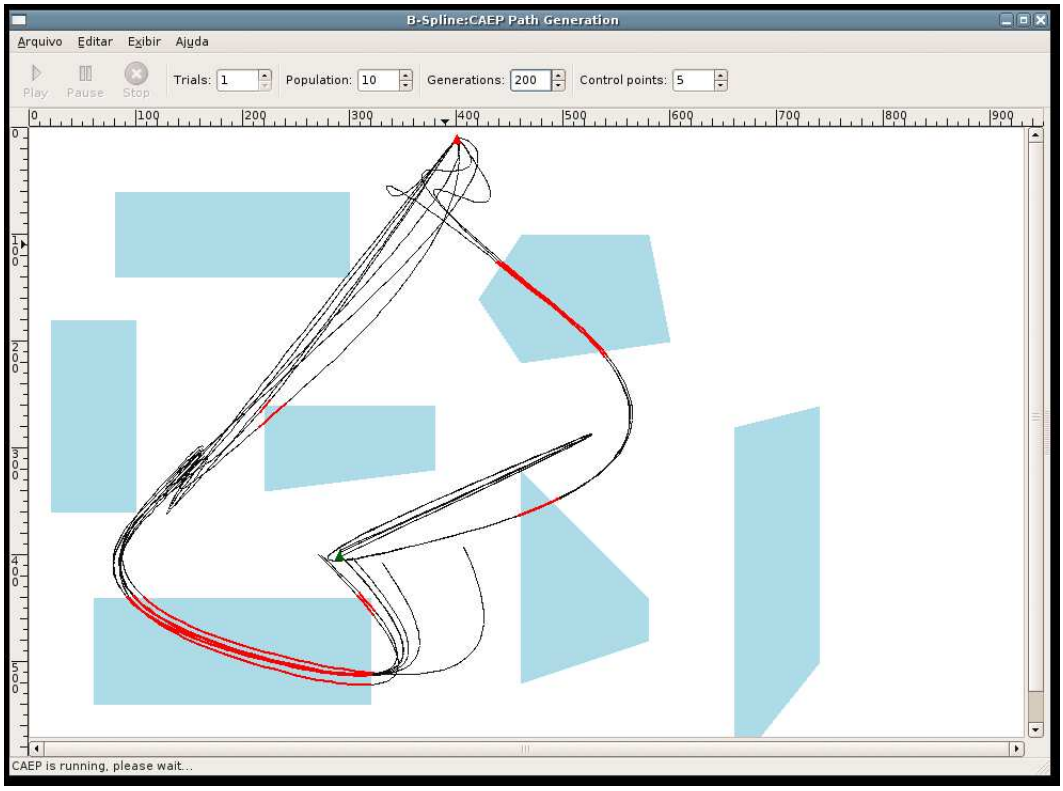

Figura 75: Soluções após 10 gerações.

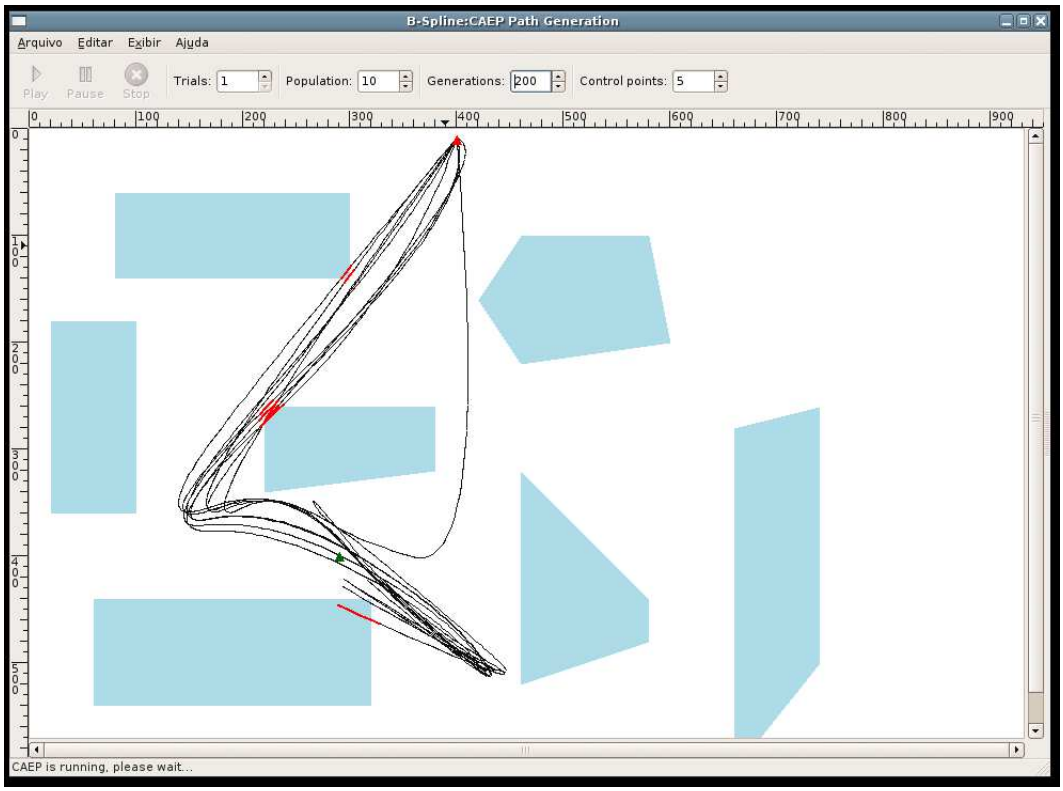

Figura 76: Soluções após 20 gerações.

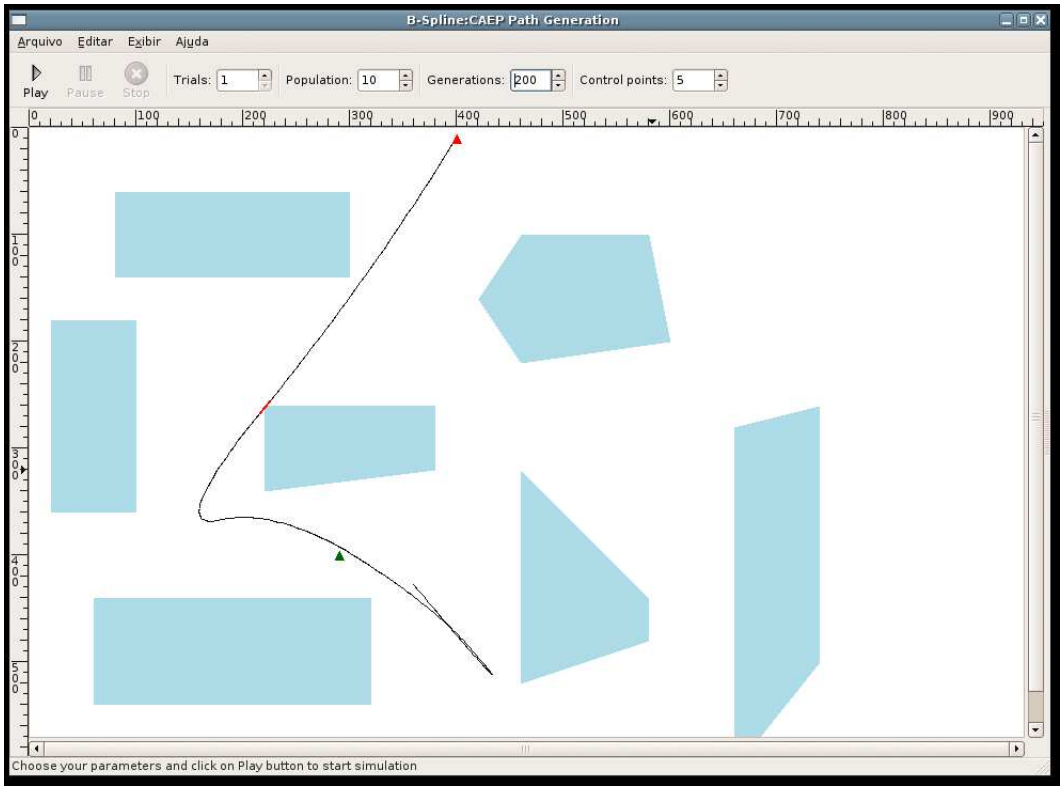

Figura 77: Solução encontrada após as 30 gerações.

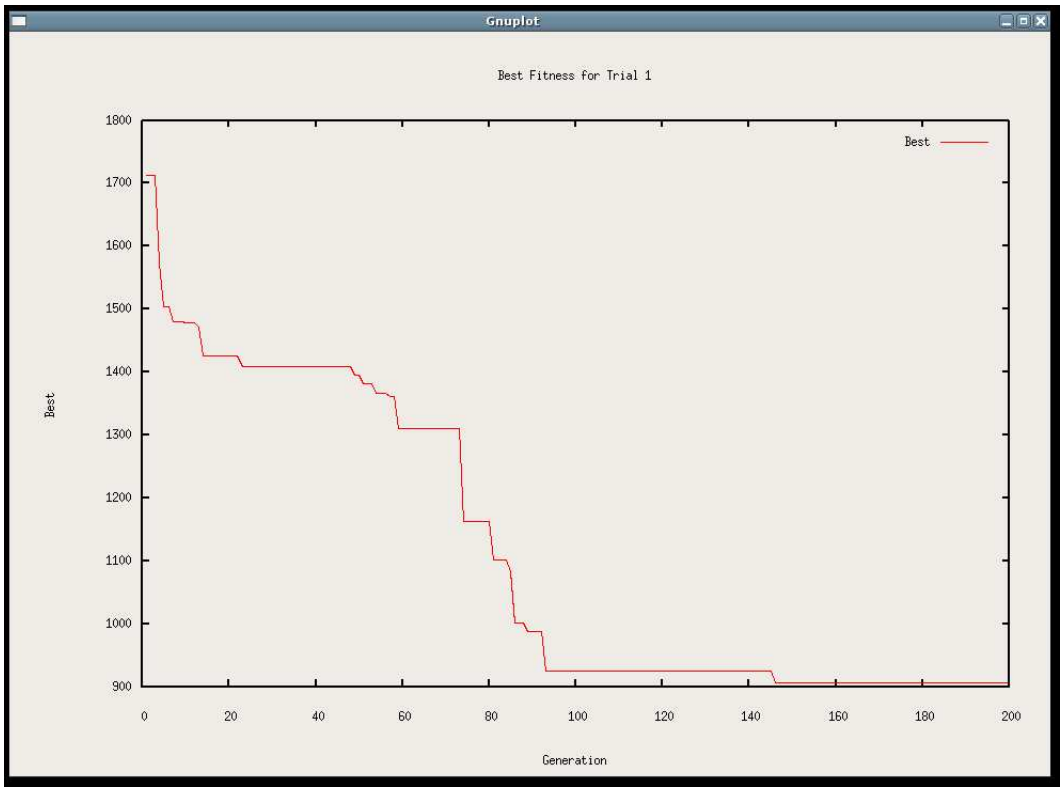

Figura 78: Variação do melhor *fitness*.

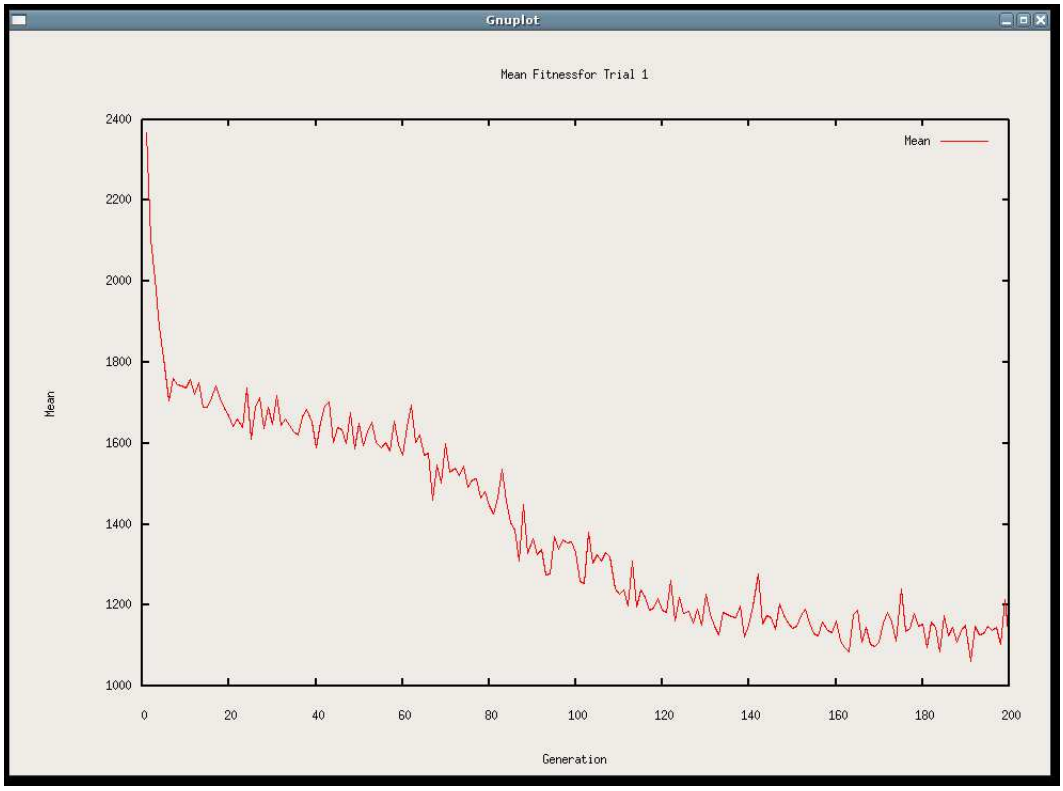

Figura 79: Variação da média do *fitness*.

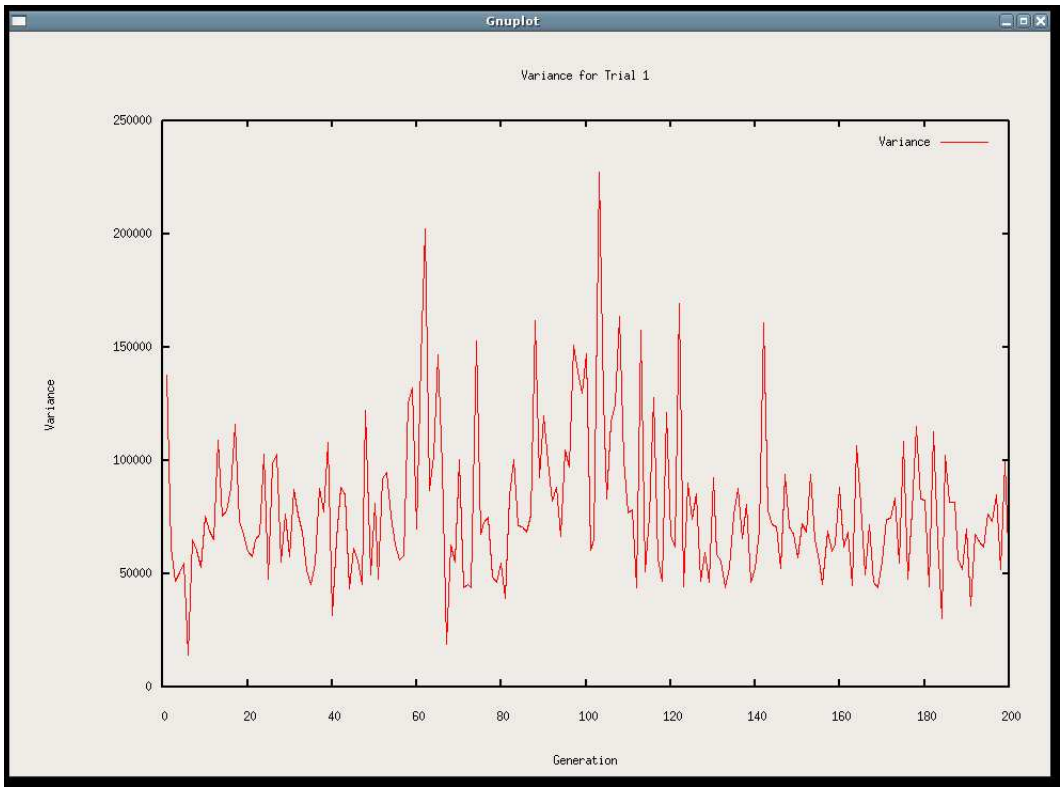

Figura 80: Variância do *fitness*.

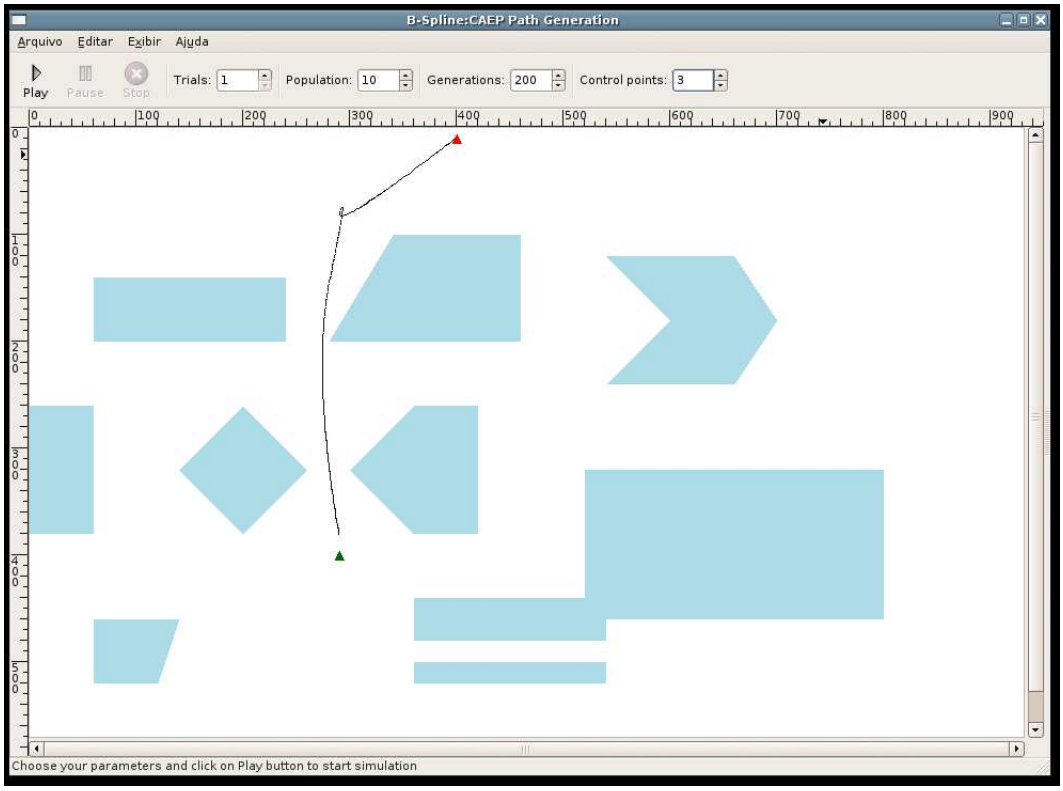

Figura 81: Ambiente de média complexidade, e uma solução com 3 pontos de controle.

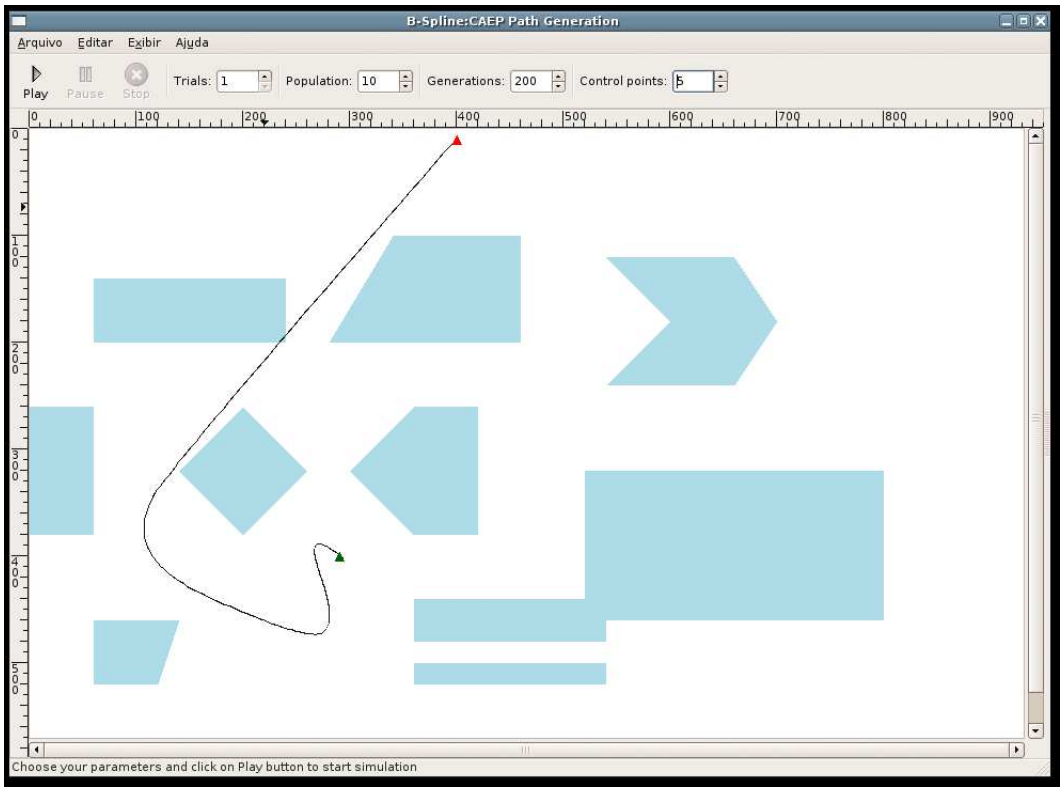

Figura 82: Ambiente de média complexidade, e uma solução com 5 pontos de controle.

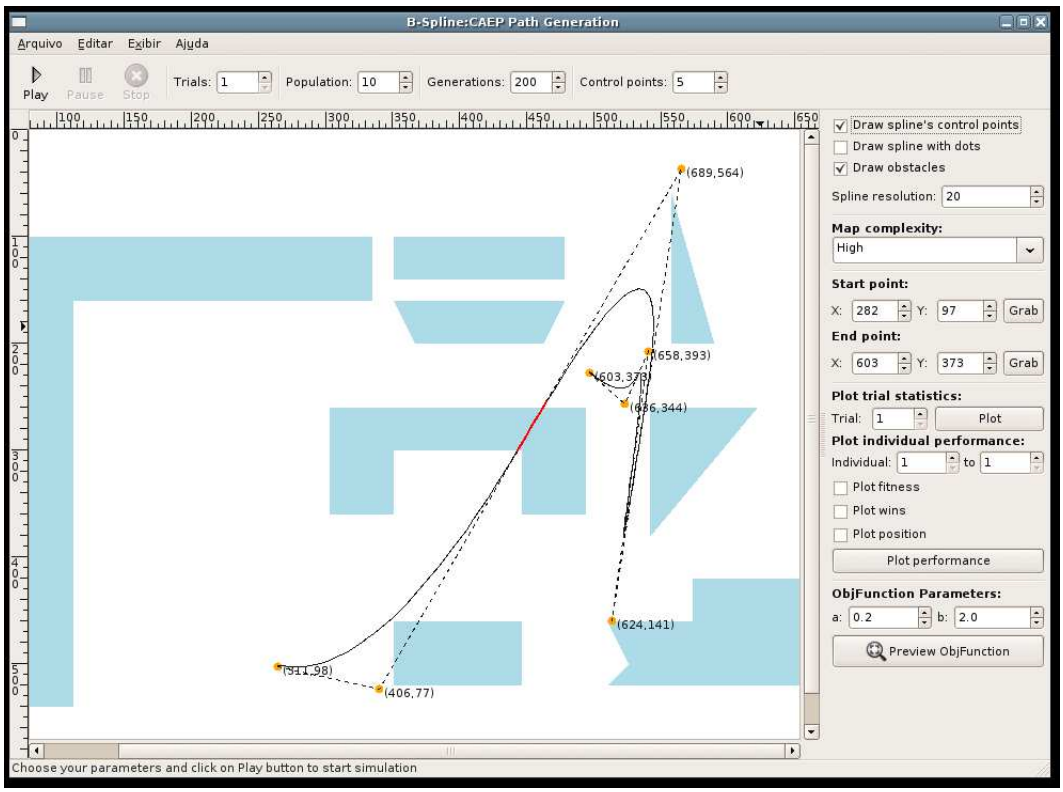

Figura 83: Ambiente de alta complexidade, mostra o caminho, inválido, que tenta contornar um obstáculo em forma de "U".

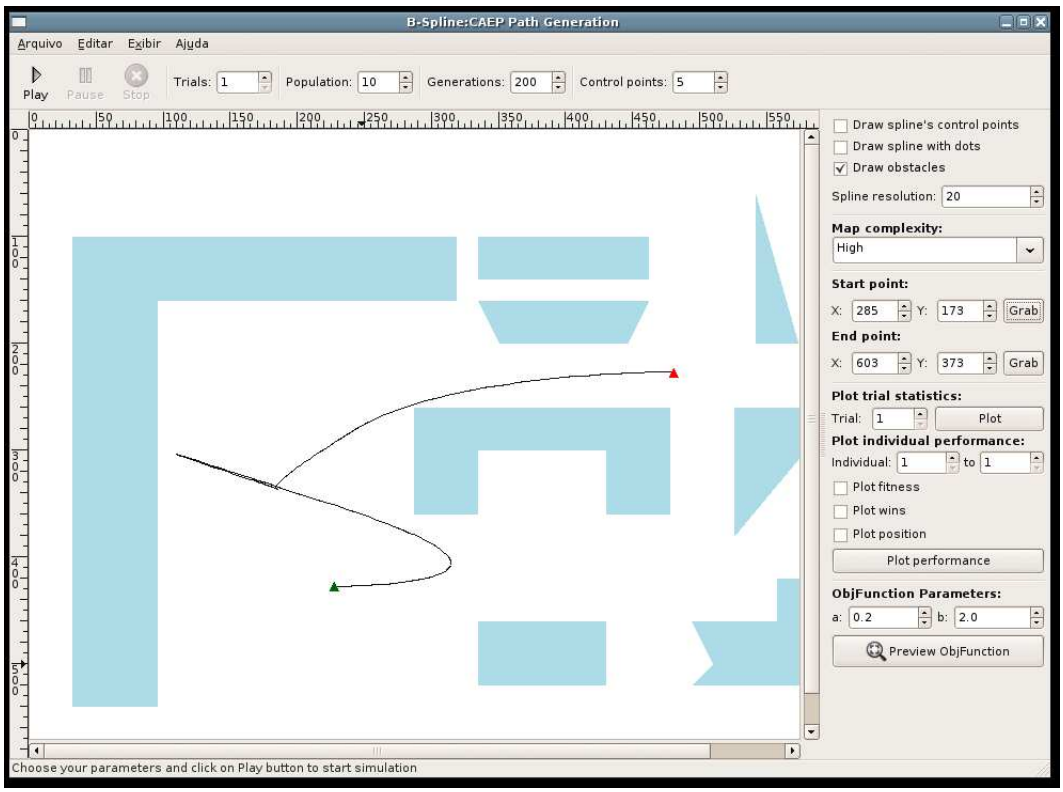

Figura 84: Ambiente de alta complexidade, mostra o caminho, que contorna um obstáculo em forma de "U".

Referências

[1] R. JOHANSSON. Intelligent motion planning for a multi-robot system, 2000.

[2] P.J. ALSINA, L.M.G. GONÇALVES, F.C. VIEIRA, D.P.F. PEDROSA, and A.A.D. MEDEIROS. Minicurso: Navegação e controle de robôs móveis, 2002.

[3] J. MARCHI. Navegação de robôs móveis autônomos: Estudo e implementação de abordagens, 2001.

[4] K. SCHREINER. Operation: Microrobot. *IEEE Inteligent Systems and Their Applications*, 14(1):5–7, January/February 1999.

[5] J.S. BAY. Desing of the "army-ant"cooperative lifting robot. *IEEE Robotics and Automation Magazine*, pages 36–43, March 1995.

[6] R.R. MURPHY. Marsupial and shape-lifting robots for urband search and rescue. *IEEE Inteligent Systems and Their Application*, 15(2):14–19, March/April 2000.

[7] A.C. SCHULTZ. The 2000 aaai mobile robot competition and exhibition. *American Association for Artificial Intelligence*, 22(1):67–72, Spring 2001.

[8] J. CASPER and R.R. MURPHY. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *IEEE Transactions on Systems, Man, and Cybernetics - Part B:Cybernetics, Vol. 33, N. 3*, 2003.

[9] HONDA. Honda worldwide | products and tecnology | p3, Setembro 2004. Dinsponível em http://world.honda.com/ASIMO/P3/.

[10] J.J. CRAIG. *Introduction to Robotics and Control.* Boston: Addison-Wesley, 1989.

[11] V.F. ROMANO. *Robótica Industrial Aplicação na Indústria de Manufatura e de Processos.* Editora Edgard Blücher Ltda., 2002.

[12] G. DUDEK and M. JENKIN. *Computational Principles of Mobile Robotics.* Cambridge University Press, 1998.

[13] J.L. JONES, B.A. SEIGER, and A.M. FLYNN. *Mobile Robots, Inspiration to Implementation.* A K Peters, Natick, Massachusetts, 1999.

[14] U. NEHMZOW. *Mobile Robotics: A Practical Introduction.* Springer, 2000.

[15] G. MCCOMB. *Robot Builder's Bonanza, 99 Inexpensive Robotics Projects.* TAB Books, 1987.

[16] G. MCCOMB. *The Robot Builder's Bonanza.* McGraw-Hill, 2000.

[17] J. IOVINE. *Robots, Androids and Animatrons.* Mcgraw-Hill, 1997.

[18] H.R. EVERETT. *Sensors for Mobile Robots, Theory and Application.* A K Peters, Natick, Massachusetts, 1995.

[19] L. FENG and J. BORENSTEIN. Measurement and correction of systematic odometry erros in mobile robots. *IEEE Transactions onrobotics and Automation*, 12(5), 1996.

[20] 1995 International Conference on Intelligent Robots and Systems (IROS'95). Pitsburgh, Pennsylvania. *Correction of Systematic Odometry Errors in Mobile Robots*, 1995.

[21] 1995 SPIE Conference on Mobile Robots. Philadelphia. *UMBmark: A Benchmark Test for Measuring Odometry Erros in Mobile Robots*, 1995.

[22] 1994 IEEE International Conference on robotics and Automation. San Diego, CA. *The CLAPPER: A Dual-drive Mobile Robot with Internal Correction of Dead-reckoning Errors*, May 1994.

[23] JEAN-PAUL LAUMOND and R.M. PAUL E.JACOBS, MICHEL T. MURRAY. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, October 1994.

[24] K. WILLIAMS. *Insectronics, Build You Own Walking Robot.* Mcgraw-Hill, 2003.

[25] J. BORENSTEIN and Y. KOREN. Obstacle avoidance with ultrasonic sensors. *IEEE Journal of Robotics and Automation*, 4(2), 1998.

[26] J. BORENSTEIN and Y. KOREN. Error eliminating rapid ultrasonic firing for mobile robot obstacle avoidance. *IEEE Transactions on robotics and Automation*, 11(1), 1995.

[27] 1992 IEEE International Conference on Robotics and Automation. Nion, France. *Noise Rejection for Ultrasonic Sensors in Mobile Robot Applications*, May 1992.

[28] ANS 6th Topical Meeting on Robotics and Remote Systems, Monterey, CA. *Mobile Robot Navigation in Narrow Aisles With Ultrasonic Sensors*, 1995.

[29] IEEE/RSJ International Conference on Intelligent Robots and Systems 95. *Position Estimation for Mobile Robot Using Sensor Fusion*, 1995.

[30] IECON'01: The 27th Annual Conference of the IEEE Industrial Electronics Society. *Sensor-based Path Planning and Intelligent Steering Control of Nonholonomic Mobile Robots*, 2001.

[31] L. FENG, J. BORENSTEIN, D. WEHE, and H.R. EVERETT. Mobile robot positioning - sensors and techniques. *Journal of Robotic Systems, Special Issue on Mobile Robots*, 14(4), 1995.

[32] J.M.A. MORENO. Curso de robots móvilles. Technical report, Universidad Carlos III de Madrid, Novembro 1999.

[33] T. LOZANO-PEREZ. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2), February 1983.

[34] H. HU and M. BRADY. A parallel processing architecture for sensor-based control of intelligent mobile robots. *Robotics and Automations Systems*, (17), 1996.

[35] S. NOLFI and D. FLOREANO. *Evolutionary Robotics: The Biologiy, Intelligence, and Technology of Self-Organizing Machines.* A Bradford Book. The MIT Press, 2000.

[36] F.S. HILL. *Computer Graphics Using OpenGL.* Prentice-Hall, 2000.

[37] F.S. HILL. *Computer Graphics.* Macmillan Publishing Company, NY. USA, 1990.

[38] B.A. BARSKY and T.D. DEROSE. Parametric curves tutorial, geometric continuity of parametric curves: Three equivalent characterizations. *IEEE Computer Graphics and Applications*, 1989.

[39] B.A. BARSKY and T.D. DEROSE. Parametric curves tutorial part two, geometric continuity of parametric curves: Constructions of geometrically continuous splines. *IEEE Computer Graphics and Applications*, 1990.

[40] V.B. ANAND. *Computer Graphics and Geometric Modeling for Beginners.* John Wiley and Sons, Inc. USA, 1993.

[41] G. BITTENCOURT. *Inteligência Computacional.* Grupo de Inteligência Artificial, UFSC. Disponível em: < http://www.das.ufsc.br/gia/softcomp/ >. Acesso em: Set. 2004.

[42] E. RICH. *Inteligência Artificial.* McGraw-Hill. São Paulo, 1988.

[43] T. YONEYAMA and C. L. Jr. NASCIMENTO. *Inteligência Artificial Em Controle e Automação.* Editora Edgard Blücher Ltda. São Paulo, 2000.

[44] E. CHARNIAK and D. McDERMOTT. *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Company, Reading, MA, 1985.

[45] P.H. WINSTON. *Artificial Intelligence.* Addison-Wesley, 1977.

[46] N.J. NILSSON. *Principles of Artificial Intelligence.* Morgan Kaufmann, 1986.

[47] A. BARR and E.A. FEIGENBAUM. *The Handbook of Artifical Intelligence*, volume I-II. William Kaufmann Inc., Los Altos, California, 1981.

[48] P.H. WINSTON. *Artificial Intelligence.* Addison-Wesley Publishing Company, Reading, MA, 2nd edition, 1984.

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

[50] J.T. SCHWARTZ and M. SHARIR. A survey of motion planning and related geometric algorithms. *Artificial Intelligence Journal*, 37:157–169, 1988.

[51] T. LOZANO-PEREZ and M.A. WESLEY. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[52] Sixth Conference on Uncertainty in AI. *Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception*, July 1990.

[53] J. BORENSTEIN and Y. KOREN. Histogramic in-motion mapping for mobile robot obstacle avoidance. *IEEE Transactions on Robotics and Automation*, 7(4), August 1991.

[54] LUGER and STUBBLEFIELD. *Artificial Intelligence and the Design of Expert Systems*. Benjamin/Cummings, Redwood City, California, 1989.

[55] Y. DOTE and S. J. OVASKA. Industrial applications of soft computing: A review. *Proceedings of the IEEE*, 89(9), September 2001.

[56] IIZUKA'94 3rd. Int. conf. Fuzzy Logic, Neural Nets and Soft Computing. Iizuka, Japan. *Fuzzt logic and soft computing: Issues, contentions and perspectives*, 1994.

[57] L.A. ZADEH and J. BEZDEK. What is computational intelligence in j.m. zurada, computational intelligence: Imitating life. *IEEE Press*, 1994.

[58] I. P. BONISSONE, Y. CHEN, K. GOEBEL, and P. S. KHEDKAR. Hybrid soft computing systems: Industrial and commercial applications. *Proceedings of the IEEE*, 87(9), September 1999.

[59] H. TAKAGI. Introductioni to fuzzy systems, neural networks, and genetic algorithms. *Inteligent Systems: Fuzzy Logic, Neural Networks, and Genetic Algorithms*, 1997.

[60] L.A. ZADEH. Fuzzy sets. *Information and Control*, (8), 1965.

[61] Proceedings of the 1998 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems. Victoria, B.C., Canada. *Adaptative Fuzzy Controller Design for Trajectory Tracking of a 2 D.O.F. Wheeled Mobile Robot using Genetic Algorithm*, 1998.

[62] CEC 99. Proceedings of the 1999 Congress on Evolutionary Computation. *Control of Autonomous Robots Using Fuzzy Logic Controllers Tuned by Genetic Algorithms*, 1999.

[63] The Ninth IEEE International Conference on Fuzzy Systems. *Motion Planning of an Autonomous Mobile Robot by Integrating GAs and Fuzzy Control*, 2000.

[64] IEEE International Conference on Robotics and Automation. *Coordinative Behaviour by Genetic Algorithm and Fuzzy in Evolutionary Multi-Agent System*, 1993.

[65] H. JUIDETTE and H. YOULAL. Fuzzy dynamic path planning using genetic algorithms. *ELECTRONIC LETTERS*, 36(4), 2000.

[66] 6th International Workshop on Advanced Motion Control. *Acquisition of Fuzzy Control Rules for a Mobile Robot Using Genetic Algorithm*, 2000.

[67] J. QIU and M. WALTERS. A ga-based learning algorithm for the learning of fuzzy behaviour of a mobile robot reacitve control system. *Genetic Algorithms in Engineering Systems: Innovations and Applications*, (466), 1997.

[68] CEC 99. Proceedings of the 1999 Congress on Evolutionary Computation. *Perception-Based Genetic Algorithm for a Mobile Robot with Fuzzy Controllers*, 1997.

[69] 1998 IEEE International Conference on Systems, Man, and Cybernetics. *Real-time Self-reaction of Mobile Robot with Genetic Fuzzy Neural Network in Unknown Environment*, 1998.

[70] Proceeding of the 1999 IEEE Intl. Conference on Robotics Automation. Detroit, Michigan. *A Fuzzy-Genetic Based Embedded-Agend Approach to Learning and Control in Agricultural Autonoumous Vehicles*, May 1999.

[71] CEC 99. Proceedings of the 1999 Congress on Evolutionary Computation. *Fuzzy-Genetic Algorithms and Mobile Robot Navigation Among Static Obstacles*, 1999.

[72] 'Intelligent Systems for the 21st Century'., IEEE Systems, Man and Cybernetics. *Fuzzy Trajectory Control and GA-Based Obstacle Avoidance of a Truck with Five Trailers*, 1995.

[73] D.K. PRATIHAR, K. DEB, and A. GHOSH. A genetic-fuzzy approach for mobile robot navigation among moving obstacles. *Internation Journal of Approximate Reasoning 20*, (20):145–172, 1999.

[74] Y. LEE and S.H. ZAK. Genetic fuzzy tracking controllers for autonomous ground vehicles. *AACC*.

[75] International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium., Proceedings of 1995 IEEE International Conference on Fuzzy Systems. *A Wall Following Robot With A Fuzzy Logic Controller Optimized By A Genetic Algorithm*, 1995.

[76] Second International Conference on Knowledge-Based Intelligent Electronic Systems. *Fuzzy Behaviour-Based Control for a Miniature Mobile Robot*, 1998.

[77] *Queen Bee genetic optimization of an heuristic based fuzzy control scheme for a mobile robot*, 2003.

[78] 26th Annual Confjerence of the IEEE Industrial Electronics Society. *Design and Implementation of Fuzzy Cooperative Catching Controller for Multiple Mobile Robots*, 2000.

[79] IEEE International Conference on Fuzzy Systems. *Fuzzy Logic Based Nonlinear Kalman Filter Applied to Mobile Robots Modelling*, 2004.

[80] IEEE International Symposium on Intelligent Control. *Hybrid Fuzzy Control Schemes for Robotic Systems*, 1995.

[81] 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. Human Robot Interaction and Cooperative Robots. *Intelligent Fuzzy Motion Control of Mobile Robot for Service Use*, 1995.

[82] 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation, 1999. CIRA '99. *Learning Fuzzy Rules by Evolution for Mobile Agent Control*, 1999.

[83] Proceedings of the 2000 IEEE International Conference on Robotics and Automation. San Francisco, CA. *Online Learning of the Sensors Fuzzy Membership Functions in Autonomous Mobile Robots*, 2000.

[84] H. MARTÍNEZ-ALFARO and S. GÓMEZ-GARCÍA. Mobile robot path planning and tracking using simulated annealing and fuzzy logic control. *Expert Systems with Applications*, (15):421–429, 1998.

[85] M.R. AKBAZADEH, K. KUMBLA, E. TUNSTEL, and M. JAMSHIDI. Soft computing paradigms for learning fuzzy controllers with applications to robotics. 1996.

[86] W.S. MCCULOCH and W. PITTS. A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophys*, 5, 1943.

[87] F. ROSENBLATT. The perceptron, a percieving and recognizing automaton. *Cornell Aeronautical Lab Report.*, (85-640-1), 1957.

[88] F. ROSENBLATT. *Principle of Neurodynamics: Perceptron and the Theory of Brain Mechanisms*. Spartan. Washingont DC, 1962.

[89] M. MINSKY and S. PAPERT. *Perceptrons*. MIT Press. Boston, MA., 1969.

[90] P. WERBROS. *Beyound regression: New tools for predictions and analysis in the behavioral science*. PhD thesis, Harvard University. Cambridge, MA., 1974.

[91] D. PARKER. Learning logic. *Center for Computational Res. Econ. Manage. Sci, Massachussets Inst. Technol.*, (TR-47), 1985.

[92] Y. LECUN. Une procedure d'apprentissage pour reseau a seuil symetrique. *Cognitiva*, 85, 1985.

[93] K. HORNIK, M. STINCHCOMBE, and H. WHITE. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 1989.

[94] 11th. International Joint Conference in Artifical Intelligence (IJCAI-89). *Training feedforward neural networks using genetic algorithms*, volume 1, 1989.

[95] 1993 IEEE Int. Conf. Neural Networks. *Use of genetic algorithms with backpropagation in training feedforward neural networks*, volume 5, 1994.

[96] 8th. National Conference in Artificial Intelligence (AAAI-90). *Empirical studies in the speed of convergence of neural network training using genetic algorithms*, volume 2, 1990.

[97] A.P. BRAGA, A.P.L.F. CARVALHO, and T.B. LUDERMIR. *Redes Neurais Artificiais: Teoria e Aplicações*. LTC Editora, 2000.

[98] T. KOHONEN. *Selg-Organizing and Associative Memory.* Springer-Verlag, 3 edition, 1989.

[99] 3rd. International Conference on Genetic Algorithms, San Mateo, CA. *Designing neural networks using genetic algorithms*, 1989.

[100] 3rd. Int. Conf. Genetic Algorithms. *Toward the genetic synthesis of neural networks*, 1989.

[101] Int. Conf. Artificial Neural Nets and Genetic Algorithms (ANNGA'93), Innsbruck, Austria. *Genetic algorithms as heuristics for optimizing ANN design*, 1993.

[102] 2nd. Int. Conf. Genetic Algorithms (COGAN'92), Baltimore, MD. *Genetic synthesis of boolean neural networks with a cell rewriting developmental process*, 1992.

[103] S. NOLFI and D. PARISI. Evolution of artifical neural networks. Technical report, Institute of Psychology. National Research Council. Rome, 1994.

[104] Proceedings of the IEEE. *Evolving Artificial Neural Networks*, volume 87, 1999.

[105] IEEE Int. Conf. on Systems, Man and Cybernetics. *Evolutionary Approaches to Neural Control in Mobile Robots*, 1998.

[106] Third International Conference on Simulation of Adaptative Behaviour: From Animals to Animats 3. *Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robots*, 1994.

[107] L.A. MEEDEN. An incremental approach to developing intelligent neural network controllers for robots. *IEEE Transactions on Systems, Man, and Cybernetics*, Part B: Cybernetics, 1996.

[108] Third International Conference on simulation of Adaptative Behaviour: From Animals to Animats 3. *Integrating Reactive, Sequential, and Learning Behaviour using Dynamical Neural Networks*, 1994.

[109] B. YAMAUCHI. Dynamical neural networks for mobile robot control. Technical report, Naval Research Laboratory Memorandum Report AIC-033-93, 1993.

[110] S. BALUJA. Evolution of an artificial neural network based autonomous land vehicle controller. *IEEE Transactions on Systems, Man, and Cybernetics*, Part B: Cybernetics. 26 3, 1996.

[111] S. NOLFI and D. PARISI. Learning to adapt to changing environments in evolving neural networks. *Adptative Behaviour*, 5, 1, 1997.

[112] M. LEE. Evolution of behaviours in autonomous robots using artificial neural network and genetic algorithm. *Departament of Multimedia, School of Multimedia, Yosu National University, San 96-1, Dunduckdong, Yosu*, 2003.

[113] ETFA '99. 1999 7th IEEE International Conference on Emerging Technologies and Factory Automation. *Control of Autonomous Robot using Genetic Algorithms and Neural Networks*, 1999.

[114] A.L. NELSON, E. GRANT, and T.C. HENDERSON. Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous System*, (46):135–150, 2004.

[115] I. RECHENBERG. *Cybernetic solution path of an experimental problem.* Royal Aircraft Establishment, Library Translation, 1965.

[116] H.P. SCHWEFEL. Kybernetische evolution als strategie der experimentellen forschung in der stromungstechnik, 1965.

[117] L.J. FOGEL. *Autonomous automata.* Ind. Res., 1962.

[118] L.J. FOGEL, A.J. OWEN, and M.J. WALSH. *Artificial Intelligence trough Simulated Evolution.* Wiley, New York, 1966.

[119] I: Introduction. Australian J. Bio. Sci. *Simulation of genetic systems by automatic digital computers*, volume 10, 1957.

[120] H BREMERMANN. The evolution of intelligence. the nervous system as a model of its environment. Technical report, Dep. Math. Univ. Washington, Seattle, Tech. Rep. 1, 1958.

[121] J. REED, R. TOOMS, and N. BARICELLI. Simulation of biological evolution and machine learning. *J. Theo. Bio.*, 17, 1967.

[122] J.H. HOLLAND. Outline of a logical theory of adaptative systems. *J. ACM.*, 9, 1962.

[123] J.H. HOLLAND. Nonlinear environments permitting efficient adaptation. *Computer and Information Science IIs. New York: Academic*, 1967.

[124] J.H. HOLLAND. *Adaptation in Natural and Artificial Systems.* Cambridge, MA: MIT Press, 1975.

[125] J. KOZA. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press, 1992.

[126] R.M. FRIEDBERG. A learning machine: Part i. *IBM J. Res. Develop.*, 3, 1958.

[127] N.A. BARRICELLI. Esempi munerici di processi de evoluzione. *Methods*, 1954.

[128] III Simpósio Brasileiro de Redes Neurais. *Algoritmos Genéticos: Tutorial*, 1996.

[129] II Congresso Brasileiro de Redes Neurais; III Escola de Redes Neurais. *Motivação, Fundamentos e Aplicações de Algoritmos Genéticos*, 1995.

[130] S. SALEEM and R.G. REYNOLDS. *The Impact of Environmental Dynamics on Cultural Emergence. Festschrift, in Honor of John Holland.* Oxford University Press, 2000.

[131] R.G. REYNOLDS and J. MALETIC. The use of version space controlled genetic algorithms to solve the boole problem. *International Journal on Artificial Intelligence Tools*, 2(2), 1993.

[132] IEEE International Conference on Evolutionary Computation, Nagoya, Japan. *A Self-Adaptative Approach to Representation Shifts in Cultural Algorithms*, 1996.

[133] D.B. FOGEL. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[134] 1996 Adaptative Distributed Parallel Computing Symposium, Dayton, Ohio. *The Use of Cultural Algorithms Support Self-adaptation in EP*, 1996.

[135] IEEE Swarm Intelligence Simposium. *Cultural Swarms: Modeling the Impact of Culture on Social Interaction and Problem Solving*, 2003.

[136] R.G. REYNOLDS and S. ZHU. Knowledge-based function optimizing using fuzzy cultural algorithms with evolutionary programming. *IEEE Transactions on Systems, Man and Cybernetics, Part B 31 (1)*, 2001.

[137] 1999 Congress on Evolutionary Computation. *Using Knowledge-Based Evolutionary Computation to solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach*, 1999.

[138] Congress on Evolutionary Computation. CEC2004, Wayne State University, USA. *Cultural Algorithms: Knowledge Learning in Dynamic Environments*, 2004.

[139] G.J. GHUNG and R.G. REYNOLDS. Caep: An evolution-based tool for real-valued function optimization using cultural algorithms. *International Journal on Artificial Intelligence Tools*, 7(3), 1998.

[140] European Control Conference (ECC'99). *Global Path Planning of Mobile Robots as an Evolutionary Control Problem*, August/September 1999.

[141] J. SOLANO and D.I. JONES. Generation of collision-free paths, a genetic approach. *Genetic Algorithms for Control Systems Engineering*, 1993.

[142] IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA '03. *Genetic algorithm based path planning for a mobile robot*, 2003.

[143] 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on Systems, Man, and Cybernetics. *Genetic algorithm based path planning and dynamic obstacle avoidance of mobile robots*, 1997.

[144] 16th IFAC World Congress, Prague. *Robot path planning in unstructured environments using a knowledge-based genetic algorithm*, 2005.

[145] IEEE International Conference on Industrial Technology, 2002. IEEE ICIT '02. *Genetic algorithm in robot path planning problem in crisp and fuzzified environments*, 2002.