UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Desenvolvimento de Rotas para Monitorar Ambientes Internos

Heitor Luis Polidoro

São Carlos / SP
Junho de 2010

Desenvolvimento de Rotas para Monitorar Ambientes Internos

Heitor Luis Polidoro

Orientador: Prof. Dr. Denis Fernando Wolf

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, para a obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.
Área de Concentração:
Robótica Móvel, Monitoramento, Navegação.

São Carlos / SP
Junho de 2010

Dedicatória

Aos meus pais, Heitor e Sonia, pelo esforço, carinho, educação, dedicação e amor.

Agradecimentos

Agradeço primeiramente à Deus por me permitir realizar este projeto de Mestrado.

Ao professor Denis Fernando Wolf pela orientação, incentivo e paciência.

Ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP - por prover os recursos necessários para minha formação e conclusão desse projeto de Mestrado.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES - pelo suporte financeiro à esse projeto de Mestrado.

E a todos aqueles que, de alguma forma, contribuíram para a conclusão desse projeto de Mestrado.

"*Na mudança de atitude não há mal que não se mude nem doença sem cura.*
Na mudança de postura a gente fica mais seguro.
Na mudança do presente a gente molda o futuro."

Gabriel, O Pensador

Resumo

A robótica móvel é uma área de pesquisa que está obtendo grande atenção da comunidade científica. O desenvolvimento de robôs móveis autônomos, que sejam capazes de interagir com o ambiente, aprender e tomar decisões corretas para que suas tarefas sejam executadas com êxito é o maior desafio em robótica móvel. O desenvolvimento destes sistemas inteligentes e autônomos consiste em uma área de pesquisa multidisciplinar considerada recente e extremamente promissora que envolve; por exemplo, inteligência artificial, aprendizado de máquina, estimação estatística e sistemas embarcados. Dentro desse contexto, esse trabalho aborda o problema de navegação e monitoramento de ambientes utilizando robôs móveis. Dada uma representação do ambiente (mapa topológico) e uma lista com urgências de cada uma das regiões do mapa, o robô deve estimar qual o percurso mais eficiente para monitorar esse ambiente. Uma vez que a urgência de cada região não visitada aumenta com o tempo, o trajeto do robô deve se adaptar a essas alterações. Entre as diversas aplicações práticas desse tipo de algoritmo, destaca-se o desenvolvimento de sistemas de segurança móveis inteligentes.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo

1

Introdução

1.1 Contextualização

A robótica é uma tecnologia utilizada para auxiliar ou substituir o homem em tarefas que o ser humano não é capaz de executar; em tarefas que podem ser automatizadas; em tarefas em ambientes com alto grau de periculosidade, como fundo do mar, incêndios, desarmamento de bombas, áreas com contaminação radioativa ou com gases tóxicos e em tarefas repetitivas como uma linha de produção industrial, por exemplo.

Dentre as muitas definições de robô podemos destacar:

- Dispositivo ou máquina que realiza funções normalmente associadas a seres humanos [**?**];
- Máquinas que, além de serem capazes de reproduzir tarefas e movimentos implícitos em sua construção, complementam a parte mecânica com dispositivos eletrônicos inteligentes [?];
- Órgão mecânico versátil equipado com atuadores e sensores sob o controle de um sistema computacional [**?**] e

- Simplesmente um agente artificial e ativo cujo ambiente é o mundo físico [**?**].

Inicialmente os robôs eram utilizados somente para automação industrial, fixados em posições específicas na linha de montagem, os braços robóticos podem mover-se a uma grande velocidade e precisão para realizar tarefas repetitivas. No entanto esses robôs apresentam uma fundamental desvantagem que é a falta de mobilidade. Um manipulador fixo tem alcance limitado que depende de onde foi colocado enquanto que robôs móveis são capazes de locomoverem-se pela fábrica [?]. Com a evolução tecnológica os robôs passaram a ser utilizados em outras áreas como: medicina de precisão, ambientes perigosos, ambientes insalubres, na área de entretenimento, serviços domésticos, etc. Nesse contexto surgiram as pesquisas para o desenvolvimento de robôs móveis autônomos, que sejam capazes de atuar em ambientes reais e reagir a situações desconhecidas de forma inteligente [**?**].

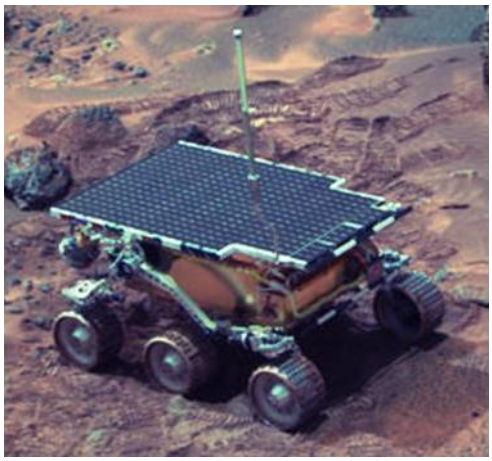
Entre todas essas diversas aplicações da robótica móvel, pode-se citar o robô Sojourner (Figura 1.1(a)) da NASA, que explorou e enviou fotos e outras muitas informações do planeta Marte para a Terra [**?**], e o robô desenvolvido pela universidade Carnegie Mellon, chamado Groundhog, que explora minas abandonadas, que além do risco de desabamento, em muitos casos também contém gases tóxicos [**?**].

A comunidade científica aposta que sistemas robóticos estejam cada vez mais presentes em nossa vida cotidiana, o que torna esta área de pesquisa extremamente promissora e desafiadora. Segundo Bill Gates [**?**], a humanidade está entrando numa nova era da computação, comparando os robôs industriais com os *mainframes* de antigamente, e este autor prevê que existirá um robô em cada casa no futuro.

A robótica consiste em uma área multidisciplinar de pesquisa, envolvendo desde elementos de engenharia mecânica, elétrica, computação até áreas de humanas como psicologia e estudos comportamentais. O que diferencia a robótica móvel de outras áreas de pesquisa em robótica, é a sua ênfase nos problemas relacionados com locomoção em ambientes complexos, que se modificam dinamicamente, compostos tanto por obstáculos fixos quanto por obstáculos móveis. Para operar nesses ambientes o robô deve ser capaz de adquirir e utilizar conhecimento sobre o ambiente, tais como: estimar posições dentro do ambiente (sua posição, de um obstáculo, de um *landmark*, de uma meta), reconhecer obstáculos, e responder em tempo real às situações que podem ocorrer nesses ambientes. As tarefas de perceber o ambiente, localizar-se no ambiente e mover pelo ambiente são problemas fundamentais da robótica móvel [**?**].

A robótica móvel, além de ser uma área de grande potencial científico, tem atraído

a atenção de empresas de tecnologia que cada vez mais investem no desenvolvimento de produtos; como por exemplo os robôs que realizam trabalhos domésticos autonomamente, entre os quais o aspirador de pó Roomba da IRobot [**?**] (Figura 1.1(b)) e o robô cortador de grama Robomow (Figura 1.1(c)) da Friendly Robotics [**?**]. Ambos apresentam sucesso comercial.



(a) Sojourner [?]

(b) Roomba [?]

(c) Robomow [?]

Figura 1.1: Exemplos de robôs.

Robôs móveis podem ser classificados quanto à sua mobilidade como: humanóides, com pernas, com rodas, com esferas, aéreos e aquáticos. Robôs com rodas são mais simples de serem construídos e mais fáceis de controlar, as rodas permitem uma maior praticidade de locomoção e dão um maior suporte estático aos robôs terrestres [?]. A autonomia de robôs móveis é essencial em ambientes remotos, tais como outros planetas, onde, devido à distância, o tempo de comunicação entre o robô e seu operador não permite a execução de ações em tempo real.

Normalmente, a navegação é a principal tarefa a ser executada por um robô móvel. Esta consiste na localização do robô no ambiente, planejamento de um caminho entre a posição inicial e o destino final e a execução do movimento pelo caminho planejado. Esta

tarefa pode ser mais, ou menos complexa, dependendo do ambiente em que o robô se encontra [**?**] e das informações que ele possui.

1.2 Motivação

O desenvolvimento de sistemas para controlar robôs móveis autônomos tem se mostrado um grande desafio para a Inteligência Artificial. Diferentes abordagens de sistema de controle para robôs móveis autônomos vêm sendo utilizadas em diversas áreas de pesquisa. Existem diversas aplicações possíveis para robôs móveis (transporte, vigilância, inspeção, limpeza, exploração, auxílio a deficientes físicos e etc.) [?].

A motivação desta dissertação de mestrado é desenvolver um sistema de monitoramento de ambientes internos. Existem diversas aplicações práticas para esse tipo de aplicação. Dentre elas, pode-se citar o desenvolvimento de um robô vigia que monitora um ambiente, dando ênfase para áreas de maior importância; um robô que monitora a temperatura e umidade de ambientes onde esse fator é crítico; um robô que faz coleta de lixo ou correspondência; um robô para fazer limpeza de ambientes; um robô para fazer entregas de remédios a pacientes em um hospital e etc.

1.3 Objetivos

Esta dissertação tem como objetivo o desenvolvimento de uma estratégia eficiente para determinar uma seqüência de áreas a serem visitadas em ambientes internos com a finalidade de monitoramento destes ambientes utilizando um robô móvel. O problema a ser resolvido consiste na divisão de um ambiente previamente conhecido em áreas de interesse. A cada uma dessas áreas é atribuído um valor (peso) referente à sua importância de monitoramento. A prioridade com que o robô deve visitar determinadas áreas é calculada com base na importância dessas áreas e no tempo decorrido desde a sua última visita. Áreas de maior importância devem ser visitadas mais freqüentemente.

Capítulo

2

Robôs Móveis

Este capítulo apresenta as principais características dos Robôs Móveis Autônomos. Um robô móvel autônomo é dotado de três características importantes: Atuadores, responsáveis, entre outras coisas pela locomoção; Sensores, que fazem o papel da percepção do robô e Sistema de Controle, que aciona os ativadores de acordo com a leitura dos sensores, seguindo o propósito para qual o robô está sendo utilizado.

2.1 Locomoção

Robôs móveis precisam de mecanismos de locomoção que os permitam moverem-se livremente pelo ambiente, mas existe uma grande variedade de modos de se locomover: andar, pular, correr, deslisar, patinar, nadar ou voar, utilizando rodas ou esteiras [?]. A maioria dos mecanismos de locomoção são inspirados na natureza, exceto os que usam rodas, esteiras ou propulsão (aquática ou aérea).

Robôs com rodas são os mais populares por inúmeras razões. Eles são mecanicamente simples de construir. A razão da carga peso/mecanismo é favorável. Os outros mecanismos geralmente precisam de um hardware mais complexo para carregarem a

mesma carga [**?**]. São mais eficientes em gasto de potência por velocidade como mostra a Figura 2.1. Além disso, como os robôs com rodas são projetados para três ou mais rodas estarem em contato com o chão todo o tempo, o equilíbrio não é usualmente um problema pesquisado [?].

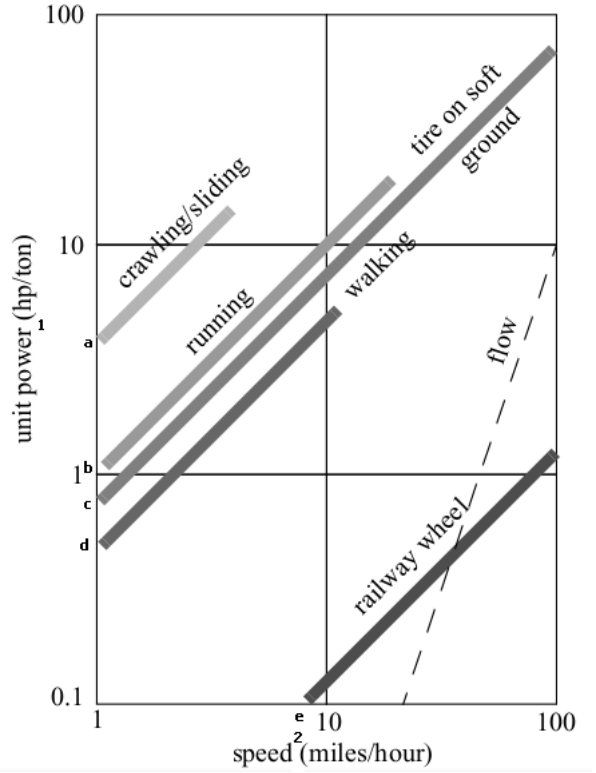


Figura 2.1: Potência por velocidade de vários mecanismos de locomoção: (a) Rastejar e deslizar. (b) Correr. (c) Pneu em chão suave. (d) Andar. (e) Em ferrovia. (1) Unidade de potência (Cavalos/toneladas). (2) Velocidade (milhas/hora) [?].

Ao invés de se preocuparem com o equilíbrio, as pesquisas em robôs com rodas tendem a focar em problemas como tração, estabilidade, manobrabilidade e controle [?].

Existem quatro classes principais de rodas, como é mostrado na Figura 2.2. Elas se diferenciam grandemente na sua cinemática, e portanto, a escolha da classe de roda tem um grande impacto na cinemática do robô móvel. A roda padrão e a roda castor têm um eixo primário de rotação e são portanto altamente direcionais. Para mover em uma direção diferente, a roda precisa ser direcionada primeiro ao longo do eixo vertical. A diferença entre essas duas rodas é que a roda padrão consegue concluir seu direcionamento sem efeitos colaterais; porque o centro de rotação passa pelo contato com o chão, ao passo que a roda castor rotaciona em torno de um eixo deslocado, causando uma força a ser transmitida para o chassi do robô durante o direcionamento [?].

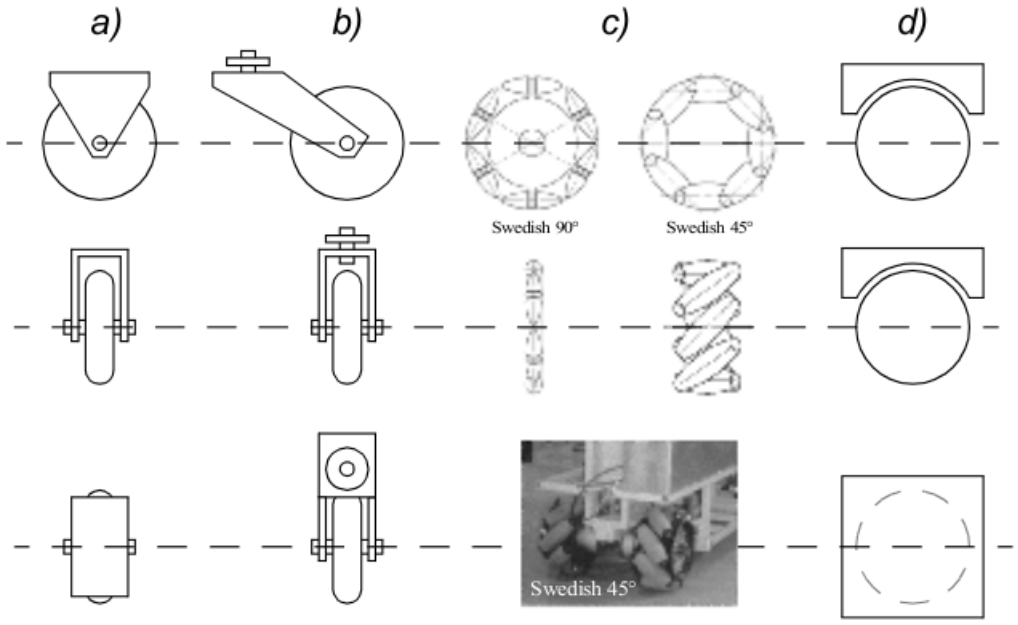


Figura 2.2: Quatro tipos básicos de roda. (a) Roda padrão. (b) Roda castor. (c) Roda sueca. (d) Bola ou roda esférica [?].

As funções da roda sueca são as de uma roda normal, mas provêm baixa resistência em outra direção, algumas vezes perpendicular à direção convencional, como a Sueca 90, e algumas vezes em um ângulo intermediário, como a Sueca 45. Os pequenos roletes ao redor da roda são passivos. A vantagem principal desse modelo é que enquanto a rotação é provida somente através de um eixo principal, a roda pode cinematicamente mover-se com pouco atrito em diferentes possíveis trajetórias, não somente para frente e para trás [?].

A roda esférica é uma roda omnidirecional verdadeira, muitas vezes projetada de modo que ela possa ser alimentada ativamente para girar ao longo de qualquer direção. Um modo para implementar a roda esférica é imitar o *mouse* do computador, provendo ativamente rolamentos alimentados que encostam na superfície superior da esfera e concedem força rotacional [?].

A principal desvantagem das rodas é que elas necessitam de uma rua ou uma superfície relativamente plana [**?**], sendo que em terrenos desiguais, elas podem ter uma performance pobre. Genericamente, se a altura do objeto for aproximadamente o raio da roda a roda não consegue passar sobre esse objeto. Uma solução simples seria utilizar rodas grandes o suficiente, mais que o dobro da altura de quaisquer possíveis obstáculos, mas em muitos casos isso é impraticável [?].

2.2 Sensores

Uma das tarefas mais importantes em um sistema autônomo de qualquer tipo é adquirir conhecimento sobre o ambiente. Isso é feito adquirindo medidas usando vários sensores e extraindo informações significativas dessas medidas [?]. Existe um vasto número de sensores sendo usados em robótica, aplicando diferentes técnicas de medidas e usando diferentes interfaces [?]. Como humanos, nós conseguimos ver uma xícara em cima da mesa e, sem pensar muito, conseguimos pegar essa xícara. De fato, completar a simples tarefa de alcançar e erguer uma xícara requer uma combinação complexa de sensores, interpretação, cognição e coordenação [?].

Alguns sensores são utilizados para medidas simples como a temperatura interna do robô ou a velocidade de rotação dos motores, mas outros mais sofisticados podem ser utilizados para adquirir informação sobre o ambiente ou até para medir diretamente a posição global do robô [?].

Pode-se classificar os sensores em dois importantes eixos funcionais: proprioceptivo/exteroceptivo e passivo/ativo [?]. **Proprioceptivos** são os sensores que medem valores internos ao sistema, como velocidade do motor, voltagem bateria, ângulo dos braços por exemplo. **Exteroceptivos** são os sensores que adquirem informações sobre o ambiente do robô, como medidas de distância, intensidade de luz e amplitude do som, por exemplo. Conseqüentemente, as medidas de sensores exteroceptivos são interpretadas pelo robô para extrair as informações significantes do ambiente. **Passivos** são sensores que medem parâmetros do ambiente que entram no sensor. Por exemplo: sondas de temperatura, microfones e câmeras. **Ativos** são sensores que emitem energia para o ambiente, então medem a reação do ambiente à essa energia. Sensores deste tipo têm uma performance superior, entretanto, existem alguns riscos, como a falta de energia, que pode afetar a característica que o sensor está tentando medir. Sensores ativos também podem sofrer interferências de sinais que estão fora do seu controle. Por exemplo, sinais emitidos por robôs próximos ou sensores similares no mesmo robô, podem influenciar no resultado das medidas. Exemplos de sensores ativos incluem sensores ultra-sônicos, sonares, e LASERs para medir distância [?].

2.2.1 Odômetro

A odometria é obtida através de um encoder, que é um sensor proprioceptivo que mede as rotações das rodas e baseado nessa informação é possível determinar a posição do robô no mapa. O principal problema dos odômetros é com relação à sua precisão, pois muitos fatores podem atrapalhar a leitura, como por exemplo, uma roda patinando ou um pneu murcho poderá fazer com que o odômetro interprete que o robô esteja fazendo um movimento diferente do real. Para corrigir esse tipo de problema existem os algoritmos de localização, que serão explicados em 3.1.

2.2.2 LASER

O LASER é um sensor Exteroceptivo Ativo, pois ele emite um sinal de luz e calcula a distância do objeto em relação ao sensor. Um emissor envia o sinal e calcula o tempo que esse sinal demora para retornar ao emissor, com isso é possível calcular a distância naquela direção, e através de um espelho rotativo é possível atingir até 360° de alcance. A Figura 2.3 mostra um LASER 180°.

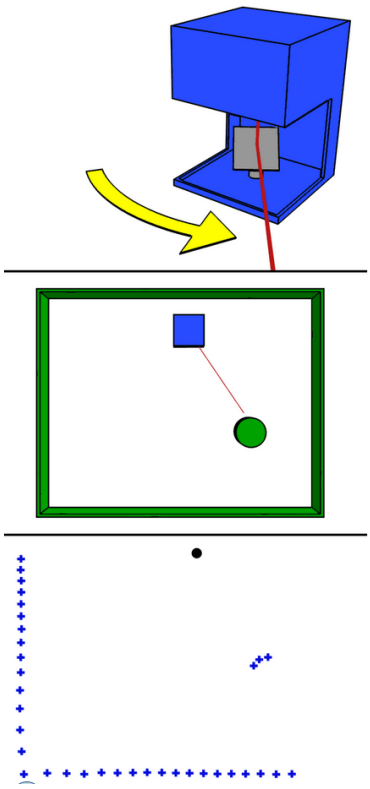


Figura 2.3: Exemplo de Funcionamento do LASER [?].

2.3 Robô Pioneer

O Pioneer 3 DX (Figura 2.4) é o robô disponível para testes experimentais no Laboratório de Robótica Móvel (ICMC - USP), ele é um robô móvel ágil e versátil. Construído em um sistema cliente-servidor, ele oferece um processamento de visão *onboard*, comunicação *ethernet*, LASER, GPS, sonar, e outras funções autônomas. Para programá-lo deve-se utilizar a biblioteca Player [**?**].

O Pioneer 3 DX pode carregar até 23kg, chega até 1.6 m/s, possui rodas de 19cm, oito sonares na parte frontal e um computador interno com algumas funções já implementadas, como a utilizada no projeto que é o desvio de obstáculos utilizando o algoritmo VFH (*Vector Field Histogram* explicado na seção 3.4.1).

Figura 2.4: Robô Pioneer.

2.4 Player

O Player é um servidor de rede para controlar robôs. Executando embarcado no robô, o Player provê uma interface simples e clara dos sensores e atuadores do robô sobre uma rede IP. O programa cliente "conversa" com o Player utilizando sockets TCP, lendo dados dos sensores, escrevendo comandos nos atuadores e configurando dispositivos em tempo de execução. O servidor Player foi desenvolvido para que seus clientes sejam independentes de linguagem e de plataforma. O programa cliente pode ser executado em qualquer máquina que tenha conexão de rede com o robô, e pode ser escrito em qualquer linguagem que suporte sockets TCP. Atualmente existem clientes disponíveis em C++, Tcl, Java e Phyton [**?**]. O Player é indiferente sobre como o programa de controle do

robô é estruturado, ou seja, pode-se escrever desde programas *multi-threads* altamente concorrentes até programas seqüencias simples.

2.4.1 Interface dos Drivers

Usualmente as interfaces entre o programa do usuário e o robô se dá como mostra a Figura 2.5, onde o programa do usuário tem que fazer a aquisição de dados diretamente dos sensores e tratar esses dados de modo que o planejador compreenda. E gerar os comandos para o motor ou outros atuadores. Sendo que para cada modelo de motor, os comandos são diferentes, assim como cada câmera, LASER e outros sensores têm leituras diferentes de acordo com o modelo.

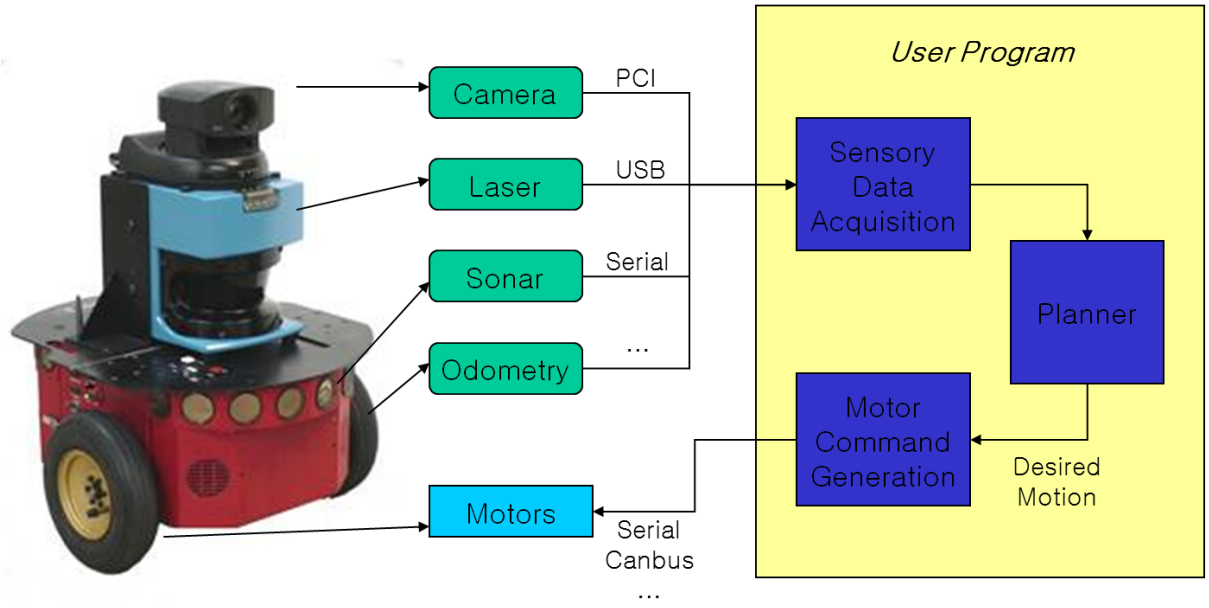


Figura 2.5: Interface usual.

O Player tem como função fazer essa interface entre os sensores e atuadores e o programa do usuário (Figura 2.6), adquirindo os dados de qualquer modelo de um mesmo sensor (que seja aceito pelo Player) e fornecendo ao programa do usuário os dados já tratados, ou seja, para qualquer modelo de um mesmo sensor (que seja aceito pelo Player) o programa do usuário lê da mesma forma, assim como para os atuadores como mostra a Figura 2.7. Por exemplo, para fazer com que o robô ande para frente na velocidade de 1 m/s, basta o programa do usuário informar a direção e velocidade, o comando vai ser o mesmo para diversos modelos de robôs, que possuem tamanhos e números de rodas diferentes, o que realiza os cálculos para determinar qual a rotação da roda para atingir tal velocidade é a interface do Player, assim como para ler do sensor LASER, basta informar de qual ângulo quer ler.

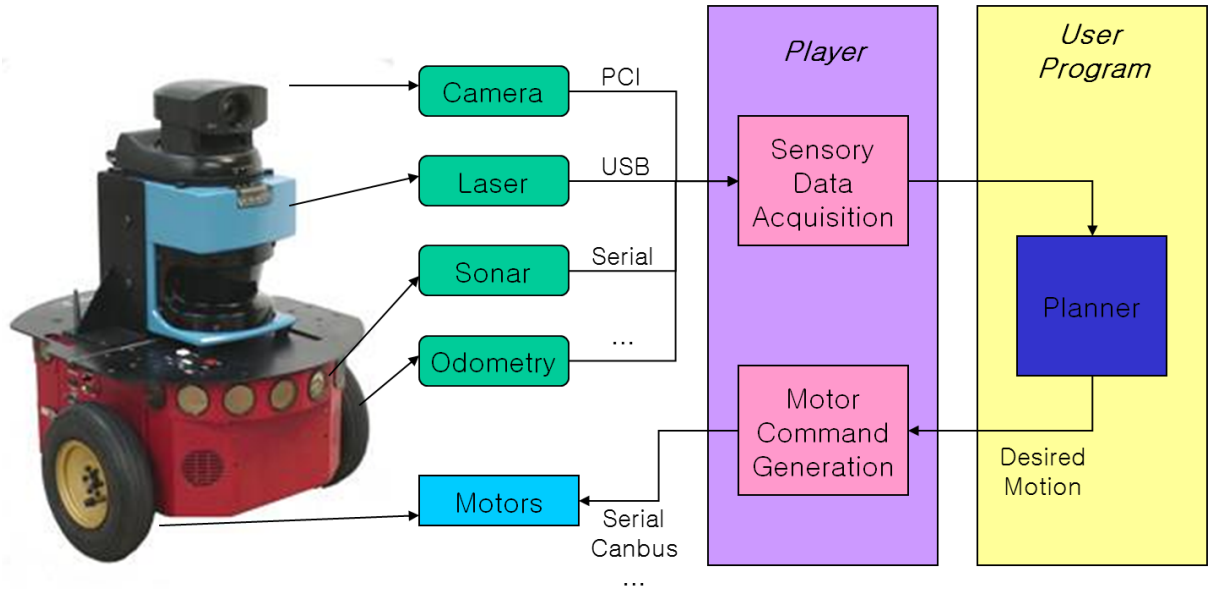


Figura 2.6: Interface do Player

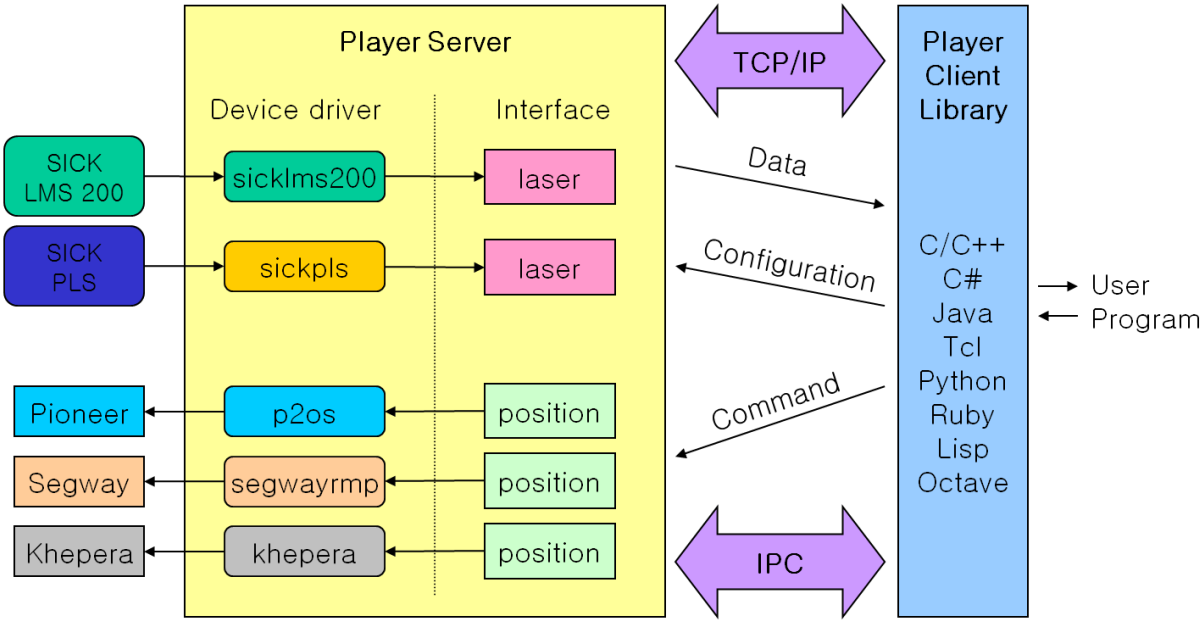


Figura 2.7: Interface e Modelo Cliente-Servidor

A Figura 2.7 também ilustra o funcionamento do modo Cliente-Servidor do Player, como a comunicação é feita por meio de *sockets* o programa cliente pode ser escrito em qualquer lingüagem que permita comunicação através de *sockets*. Esse modo Cliente-Servidor também permite a fácil utilização de diversos simuladores, pois como a comunicação é a mesma, o programa cliente não sabe se está interagindo com o robô real ou com algum simulador (Figura 2.8).

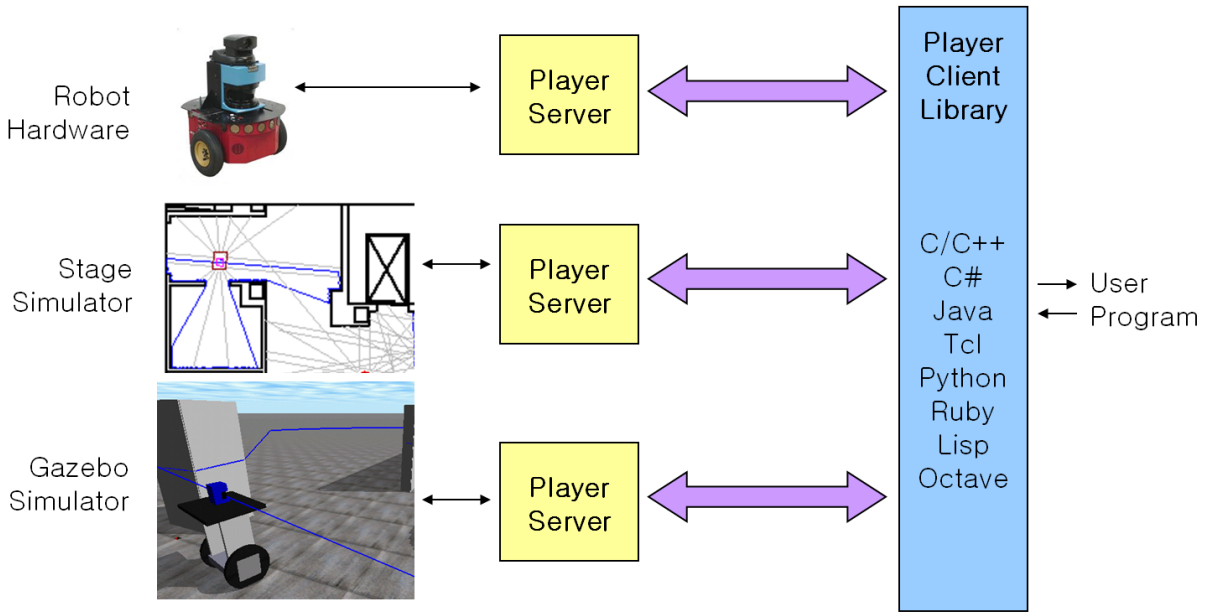


Figura 2.8: Abstração de Hardware

Esse modelo Cliente-Servidor também trás outras vantagens como:

- Clientes podem se conectar a múltiplos servidores;
- Servidores aceitam conexão de múltiplos clientes;
- Diferentes programas/processos/*threads* podem processar dados de diferentes sensores do mesmo servidor.

A Figura 2.9 ilustra esse funcionamento, onde a letra 'P' indica que naquele dispositivo existe um programa Player sendo executado e a letra 'C' indica que naquele dispositivo está sendo executado um programa cliente.

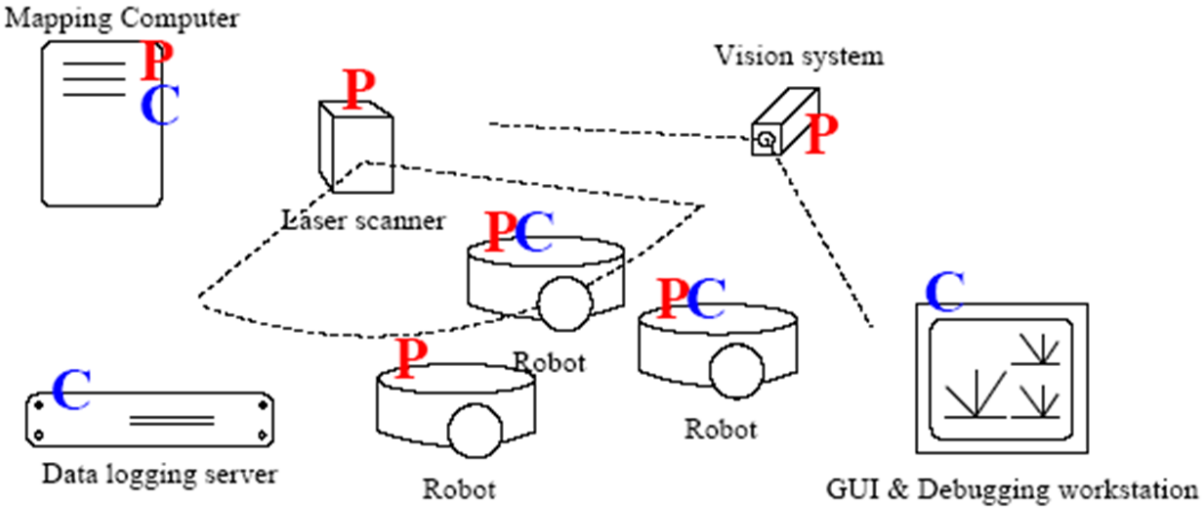


Figura 2.9: Exemplo de Clientes e Servidores

2.4.2 Stage

O Stage é usado normalmente como um *plugin* para o Player, provendo uma série de dispositivos virtuais para os clientes Player. Os usuários escrevem as rotinas e algoritmos normalmente, como clientes para um servidor Player. Não é possível para clientes distinguir a diferença entre os dispositivos reais do robô e os equivalentes simulados pelo Player/Stage. Com isso clientes Player desenvolvidos usando o Stage precisarão de pouca ou nenhuma modificação para trabalhar com o robô real e vice-versa. Em muitos casos, basta somente mudar no cliente o endereço IP de onde está o servidor. O Stage também pode simular uma população de robôs móveis, sensores e objetos num ambiente bidimensional (Figura 2.10) [?]. Nesta dissertação será utilizado somente um robô e o sensor LASER.

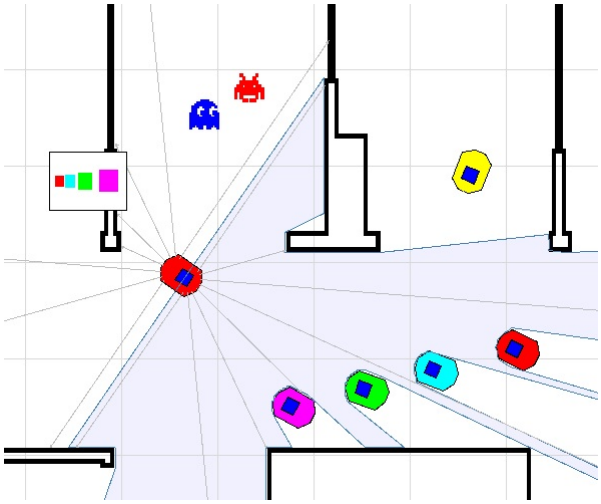


Figura 2.10: Simulação com 5 robôs, 2 objetos, LASER, sonar e *blobfinder* [?].

Capítulo

3

Navegação de Robôs Móveis

Em robótica, navegação é a ciência, arte, prática ou tecnologia de planejar e direcionar o percurso de um robô móvel enquanto percorre o meio ambiente (terra, água, ou ar). Inerente em qualquer esquema de navegação é o desejo de alcançar um destino sem se perder ou colidir com algum obstáculo [?]. Vagar é uma forma de navegação que consiste em o robô andar sempre em frente e desviando dos obstáculos. Em geral, navegação é um processo incremental que, segundo Murphy [**?**], pode ser resolvido respondendo a quatro perguntas:

- **Para onde estou indo?** Geralmente determinado por um humano ou uma missão;
- **Qual o melhor caminho?** Esse é o problema de planejamento de trajetória e é a área da navegação que recebe mais atenção;
- **Por onde passei?** Enquanto o robô explora o ambiente, pode ser parte da missão mapear esse ambiente;
- **Onde estou?** Para seguir uma trajetória ou construir um mapa o robô precisa saber onde ele está.

Essas questões podem ser sumarizadas em quatro passos segundo Goldberg [**?**]:

- Percepção e modelagem do ambiente;
- Localização;
- Planejamento e decisão do movimento;
- Execução do movimento;

A relação entre esses passos pode ser vista na Figura 3.1 [?].

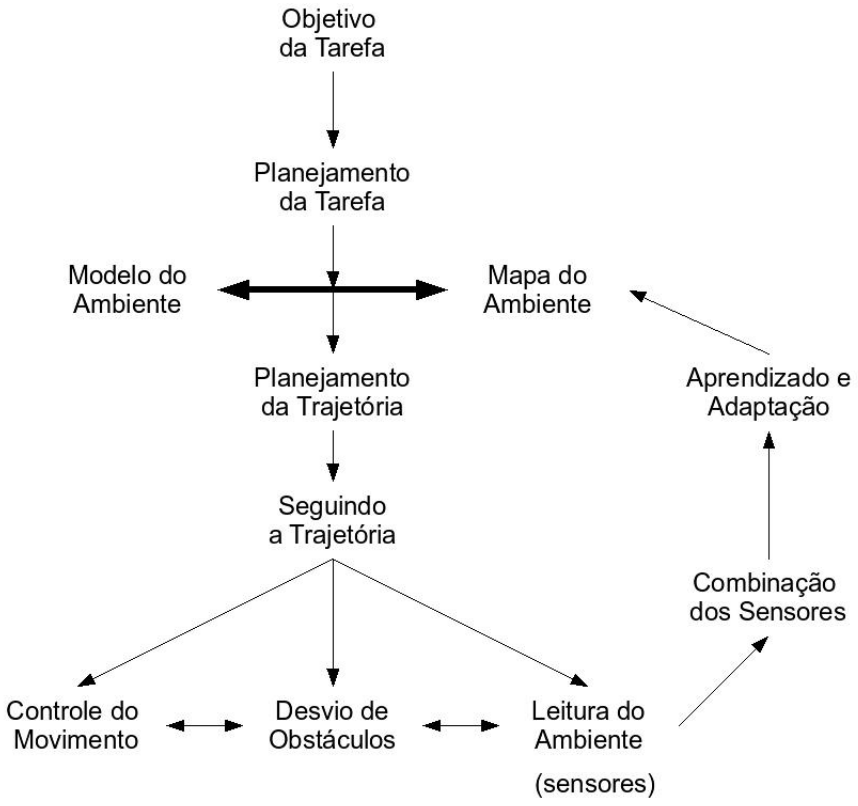


Figura 3.1: Hierarquia do controle de um robô móvel mostrando o fluxo de informação [?].

Navegação é a instância do paradigma geral da robótica "perceber - decidir - agir". A implementação da tarefa de navegação pode ser mais ou menos complexa dependendo do contexto em que a tarefa vai ser executada [?].

- **O ambiente:** pode ser inicialmente conhecido, parcialmente conhecido ou completamente desconhecido, pode ser estático ou com objetos móveis;
- **A meta:** pode ser especificada por *landmarks* ou coordenadas;
- **A navegação em si:** pode ter restrições como tempo e melhor caminho;

- **As habilidades do robô:** poder de computação, sensores e suas incertezas, tamanho do robô e sua cinemática.

A solução para o problema de navegação vai depender de todas essas restrições [?].

A navegação pode ser dividida em duas grandes áreas: planejamento de trajetória e desvio de obstáculos. Para o planejamento da trajetória o robô utiliza uma representação do ambiente (total ou parcial) e planeja uma trajetória que o leve de seu ponto de origem até seu destino, atendendo a requisitos como menor caminho ou menos curvas, por exemplo. O desvio de obstáculos é utilizado principalmente em ambientes dinâmicos, onde possam haver obstáculos móveis; sua função é fazer com que o robô chegue em seu destino de forma segura, ou seja, não colida com obstáculos que podem ser móveis ou não, utilizando sensores geralmente de distância, como LASERs e sonares, ou até mesmo utilizando câmeras.

3.1 Localização

Localização de robôs móveis é o processo de se estimar a posição de um robô dentro do ambiente no qual está inserido [**?**]. Para fazê-lo é necessário que o robô utilize um mapa que modele o mundo ao seu redor e estime através deste modelo sua posição corrente e sua orientação. Sendo assim, a habilidade de estimar sua posição, através de seus sensores, é uma das pré-condições básicas para a autonomia de robôs móveis [**?**, **?**, **?**].

O mapa do ambiente pode ser obtido automaticamente por um robô que seria responsável por explorar previamente o ambiente ou manualmente projetado por um ser humano [**?**]. Para as técnicas de localização serem consideradas robustas, elas devem apresentar os seguintes requisitos [?, ?, ?]:

- Serem capazes de lidar com incertezas, pois os mapas que modelam o ambiente são geralmente imprecisos e os sensores, muitas vezes, não são perfeitos;
- Lidarem com simetria, uma vez que muitos locais possuem mapas que não podem ser diferenciados com apenas uma medida. Por exemplo, corredores em um escritório;
- Permitirem a integração da leitura de vários sensores diferentes, porque a fusão sensorial aumenta a confiabilidade do sistema compensando, assim, os possíveis ruídos.

3.2 Mapeamento

Representar o ambiente onde o robô se encontra é importante pois, decisões baseadas na representação do ambiente podem ter impacto nas escolhas disponíveis para a representação da posição do robô. Muitas vezes a fidelidade da representação da posição é limitada pela fidelidade do mapa [?].

Segundo Siegwart e Nourbakhsh [?], três relações fundamentais têm de ser entendidas quando se escolhe uma representação particular de mapa:

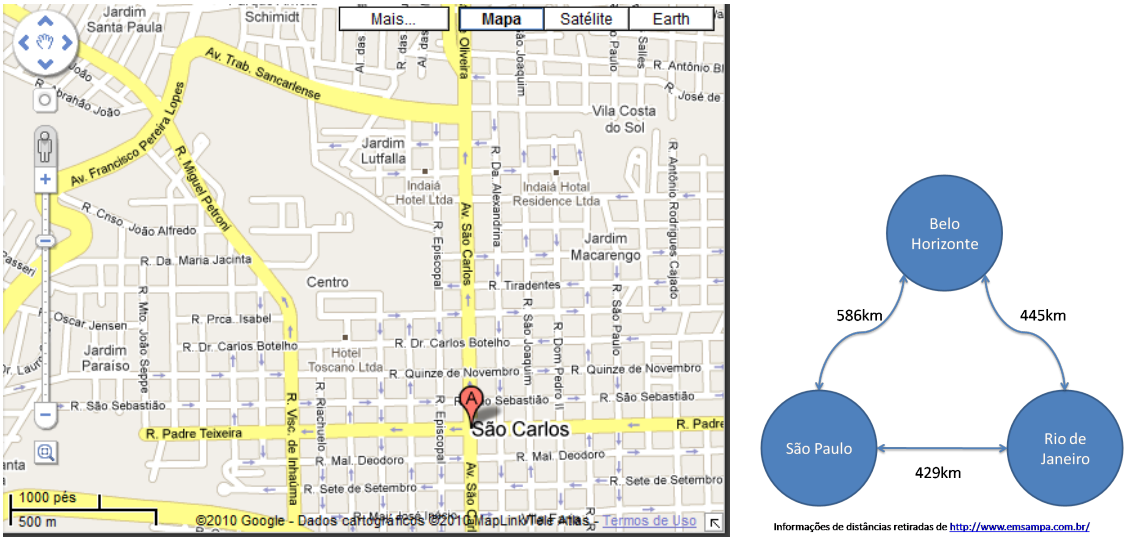
- A precisão do mapa precisa ser compatível com a precisão da necessidade do robô para atingir seus objetivos;
- A precisão do mapa e o tipo de dados dos recursos representados precisam ser compatíveis com a precisão e os tipo de dados retornados pelos sensores do robô;
- A complexidade da representação do mapa tem impacto direto com a complexidade computacional.

Existem várias maneiras de representar um mapa: Decomposição por célula, decomposição fixa, decomposição adaptada (com variação de célula), *occupancy grid* e representação topológica etc. [?].

Os mapas comumente conhecidos, que são vistos em guias ou na *internet*, são os mapas métricos, onde as distâncias entre os pontos desejados podem ser medidas com uma régua e aplicando a escala indicada no mapa, se tem a exata distância entre esses pontos. Os mapas topológicos não representam o mapa como um todo, representam apenas pontos de interesse e a conectividade entre esses pontos. Mapas topológicos são representados em forma de grafos, onde os vértices são os pontos que se desejam representar e as arestas indicam sua conectividade, se é possível ir de um ponto à outro, qual a distância (indicada no peso da aresta) e a direção, ou seja, é possível ir do vértice **a** ao vértice **b** e vice-versa. A Figura 3.2 traz exemplos de mapas métrico e topológico.

3.3 Planejamento de Trajetória

O primeiro passo para planejar uma trajetória é transformar um possível modelo contínuo do ambiente em um mapa discreto compatível com o algoritmo de planejamento de trajetória escolhido. É possível identificar três estratégias gerais de composição [?]:



(a) Mapa Métrico [?] (b) Mapa Topológico

Figura 3.2: Exemplos de mapas.

- *Roadmap*: identificar um conjunto de rotas nos espaços livres;
- Decomposição em células: discriminar entre células livres e ocupadas;
- Campos potenciais: impor uma função matemática sobre o espaço.

3.3.1 *Roadmap*

A técnica de *roadmap* para planejamento de trajetórias consiste em capturar a conectividade do espaço livre do ambiente em uma rede de curvas. Essa rede é vista como um conjunto padrão de caminhos. O planejamento da trajetória então se reduz à conectar os pontos inicial e final do robô no *roadmap* e buscar neste um caminho entre esses dois pontos. Se existir um caminho ele será dado pela junção de três subcaminhos: um subcaminho entre o ponto inicial até algum ponto do *roadmap*, um subcaminho do *roadmap* e um subcaminho do *roadmap* até o ponto final [?].

Vários métodos propostos foram baseados nessa idéia, dentre eles: grafos de visibilidade, diagrama de Voronoi, rede de caminho livre e silhueta. Nesta dissertação será utilizado o diagrama de Voronoi.

Diagrama de Voronoi : É um método completo de mapa de rotas que tende a maximizar a distância entre o robô e os obstáculos no mapa. Para cada ponto livre no mapa

é calculada a distância para o obstáculo mais próximo. O diagrama de Voronoi consiste nos pontos que são equidistantes de um ou mais obstáculos. Um exemplo de diagrama de Voronoi em um mapa pode ser visto na Figura 3.3 [?].

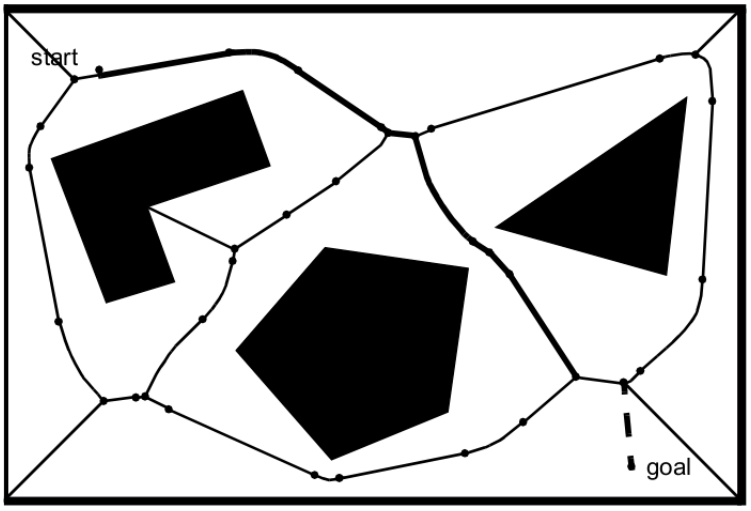


Figura 3.3: Exemplo de um diagrama de Voronoi [?].

O diagrama de Voronoi tem uma deficiência importante, sensores de localização com alcance limitado. Como o algoritmo maximiza a distância entre o robô e um objeto no ambiente, qualquer sensor de curto alcance no robô poderá falhar para perceber o mundo ao seu redor. Se o sensor de curto alcance está sendo usado para localização, então, o caminho designado pelo diagrama de Voronoi será pobre no ponto de vista para localização [?]. Por outro lado, como por definição, o caminho é criado baseado em um ponto equidistante dos obstáculos, isso garante um rota segura do robô pelo mapa.

3.3.2 Decomposição em Células

Este método consiste em dividir o espaço livre do mapa em células, de forma que um caminho entre quaisquer duas células possa ser facilmente obtido. Um grafo, chamado *grafo de conectividade*, representa a relação de adjacência entre as células; onde os vértices representam as células extraídas do espaço livre. Somente existe uma aresta entre dois vértices se, e somente se, as células correspondentes foram adjacentes. O resultado de um caminho é uma seqüência de células denominada *canal*, de onde pode ser computado um caminho contínuo [?]. A Figura 3.4 demonstra um exemplo de mapa utilizando decomposição em células.

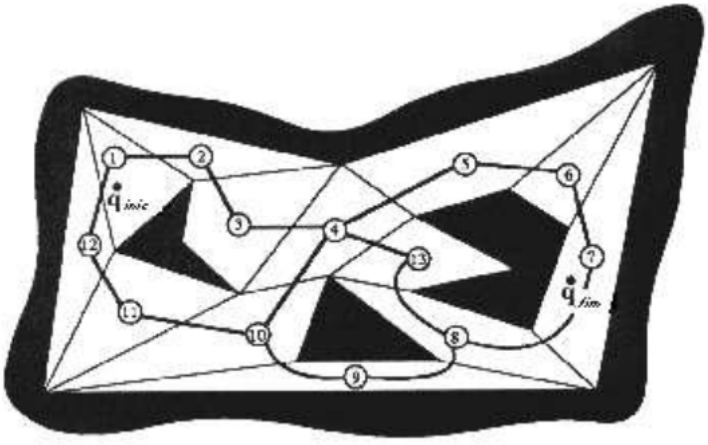


Figura 3.4: Exemplo de um mapa com decomposição em células [?].

3.3.3 Campos Potenciais

Neste método, o espaço livre é discretizado em uma fina grade, a cada posição é associada uma função com a qual pode-se fazer uma analogia a um campo potencial. Como o tamanho da grade é grande, pois ela é fina, são utilizados métodos heurísticos para encontrar um caminho. A analogia que o método sugere é que o robô seja uma partícula movendo-se sob a influência de um campo potencial gerado pelos obstáculos e pelo seu ponto de destino. O ponto de destino gera um campo potencial que atrai o robô, enquanto os obstáculos geram um campo que repele o robô. O caminho final é dado pela força resultante desses campos potenciais.

Um exemplo de campos potenciais é mostrado na Figura 3.5. O campo potencial atrativo (b) é um parabolóide com ponto de mínimo localizado na posição do objetivo. O campo potencial repulsivo (c) é diferente de zero somente a partir de uma determinada distância dos obstáculos. O caminho (e) é construído pela direção oposta a do gradiente do potencial resultante (d). Em (f) tem-se uma matriz de orientações do vetor gradiente, que são as orientações das forças induzidas pelo campo potencial [?].

3.4 Desvio de Obstáculos

Recentemente, muitas pesquisas voltaram sua atenção para o problema de desvio de obstáculos. Entretanto, existem alguns métodos clássicos de desvio de obstáculos que devem ser citados [?]: Detecção de borda, *certainty grid*, campos potenciais, campo de força virtual e *vector field histogram*.

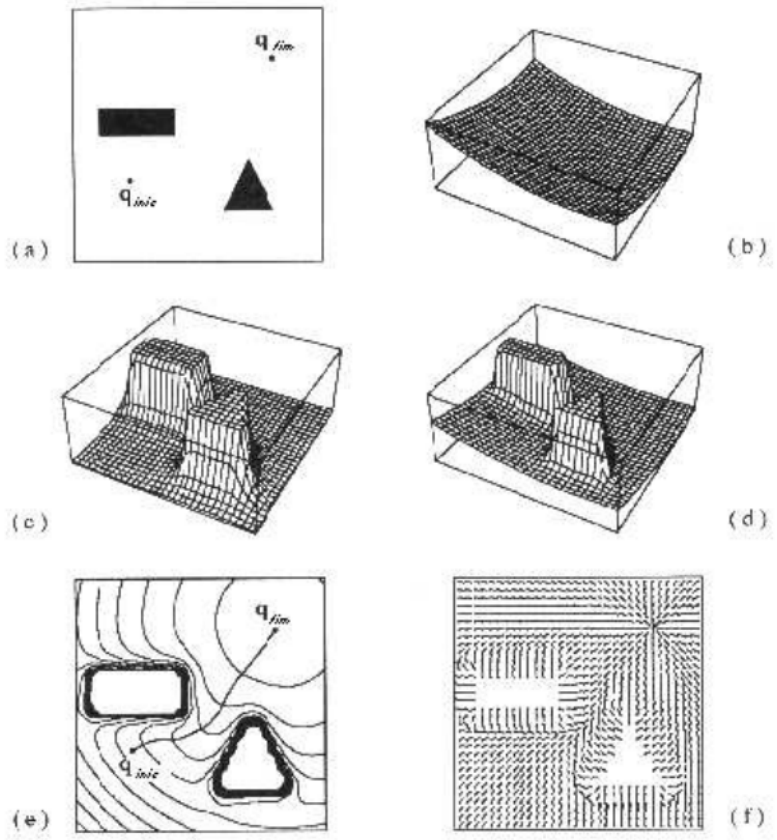


Figura 3.5: Exemplo de um mapa com campos potenciais. [?]

Detecção de borda é um método bem popular que extrai as bordas verticais do objeto e guia o robô ao redor dessas bordas.

Certainty grid é um método de representação probabilístico de obstáculos que modela o mundo em uma grade, onde a área de trabalho do robô é modelada em um arranjo de quadrados em 2D, chamadas de células. Cada célula tem um valor de certeza que indica o grau de confiança de que algum objeto está na área dessa célula.

O método de campos potenciais funciona tanto para planejamento de trajetória quanto para desvio de obstáculos, para isso basta calcular a força potencial resultante em tempo de execução, com isso o robô poderá desviar de obstáculos móveis.

O campo de força virtual (do inglês *Virtual Force Field* - VFF) é um método para veículos que necessitam de uma resposta mais rápida para fazerem curvas. É um método baseado na *certainty grid*, onde uma grade de histograma cartesiano 2D é usado para representar a probabilidade de cada célula conter um obstáculo, depois a idéia de campos potenciais é aplicada ao histograma.

E, por fim, o método de *vector field histogram* - VFH, que cria um mapa de

certainty grid local e, ao invés de utilizar um histograma cartesiano 2D, utiliza um histograma polar $(\alpha - P)$, onde α é o ângulo do sensor e P é a probabilidade de haver um obstáculo nessa direção. O VFH é explicado melhor a seguir.

3.4.1 *Vector Field Histogram*

Analisando melhor o método VFF percebe-se um problema que é a redução drástica excessiva dos dados quando forças de repulsão individuais do histograma da grade de células são somadas para calcular o vetor resultante. Centenas de pontos de dados são reduzidos em um passo para direção e magnitude. Conseqüentemente, informações detalhadas sobre a distribuição local do obstáculo é perdida. Para evitar essa situação, foi desenvolvido um novo método chamado *Vector Field Histogram* - VFH. O Método VFH utiliza dois estágios de redução de dados [?].

O VFH mantém um mapa métrico probabilístico simplificado baseado em malhas, semelhante ao *Certainty grid*, porém utiliza valores inteiros entre zero e cinco, no lugar de valores reais entre zero e um, como no *Certainty grid* [?].

Outra diferença entre o VFH e o *Certainty grid* é em relação à maneira como o mapa é gerado, pois para conseguir operar com rapidez para desviar de obstáculos em velocidades elevadas, o custo para a manutenção do mapa tem que ser baixo, ou seja, ocupar pouco tempo do processador. Para isso o mapa é obtido através de amostragem rápida, onde cada célula é iniciada em zero e é incrementada em um cada vez que um sensor detectar um obstáculo naquela posição, até que alcance cinco (valor máximo) e não mudará mais [?].

A cada iteração do algoritmo é realizada uma amostragem rápida, criando um histograma polar. Esse histograma é criado de tal forma que as regiões ao redor do robô são divididas em k setores, contendo cada um sua densidade de ocupação. Esse histograma não é feito no mapa todo, e sim somente em uma região ativa, cujo centro é o robô [?].

Existem três níveis de representação de dados [?]:

- O nível mais alto contém uma descrição detalhada do ambiente do robô;
- No nível intermediário é armazenado um histograma polar unidimensional sobre a localização momentânea do robô;

- O nível mais baixo é a saída do algoritmo VFH, valores de referência para o controle de movimento do robô.

No primeiro estágio da redução de dados é mapeada a grade de histograma do ambiente (Figura 3.6) em um histograma polar (Figura 3.7). E no segundo estágio é calculado o ângulo Ω entre o robô e a meta; para isso, escolhe-se o setor k, referente ao ângulo mais próximo de Ω, com a densidade de ocupação menor que uma constante pré-definida. Sendo assim, o ângulo θ, referente a esse setor, determina para onde o robô deve virar. E a velocidade linear pode ser calculada com heurísticas que utilizam as densidades de ocupação próximas ao ângulo θ [?].

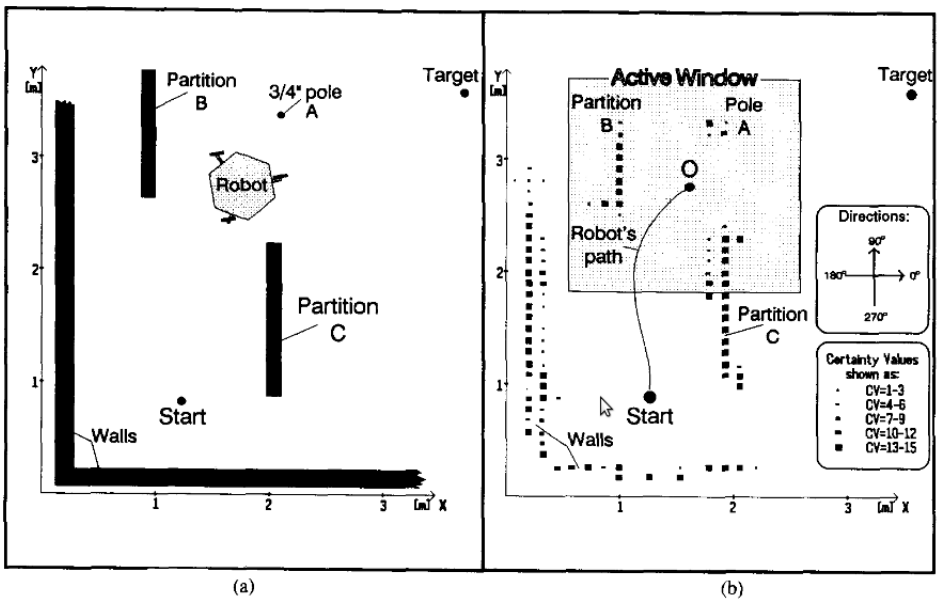


Figura 3.6: (a) Exemplo de um mapa. (b) Grade de histograma correspondente. [?]

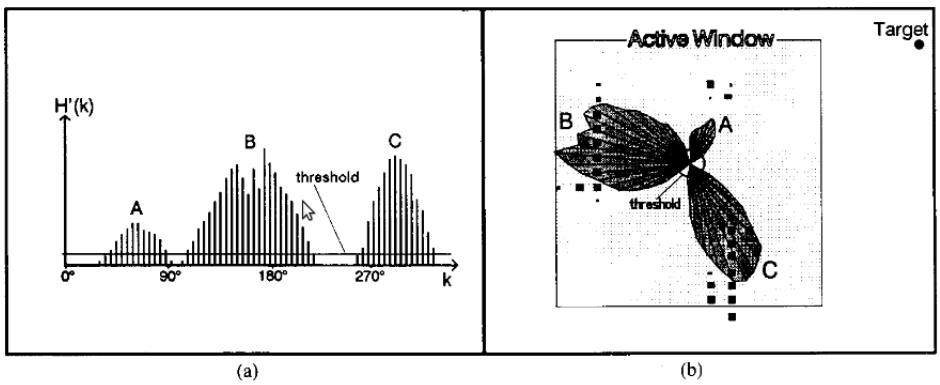


Figura 3.7: (a) Densidade polar de obstáculo em um histograma polar relativo à posição do robô em O (Figura 3.6(b)). (b) O mesmo histograma polar de (a) mostrado em forma polar e por cima da grade de histograma da Figura 3.6(b). [?]

3.5 Considerações

Neste projeto para o robô monitorar os ambientes ele deve ser capaz de navegar com segurança nos mesmos. Para isso são necessários os conceitos apresentados nessa seção. Saber onde o robô está no mapa, conhecer o mapa, planejar a melhor trajetória para chegar ao destino e desviar dos obstáculos no caminho.

Capítulo

4

Grafos e Aplicações

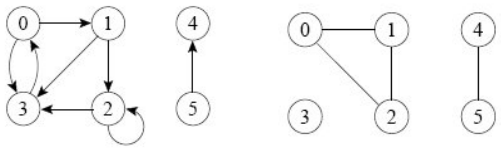
Grafos e mapas topológicos, são muito utilizados na literatura para navegação de robôs móveis, como pode ser constatado em [?, ?, ?, ?, ?, ?]. Como o problema proposto (seção 5.4) diz respeito à busca de trajetória em um mapa topológico (grafo), uma das soluções se torna uma variação do problema do caixeiro viajante que trata, basicamente, de encontrar o menor ciclo hamiltoniano em um grafo.

As seções seguintes contém uma explicação sobre grafos, mapas topológicos, ciclo hamiltoniano e o problema do caixeiro viajante.

4.1 Grafos

Muitas aplicações em computação necessitam considerar um conjunto de conexões entre objetos. Os relacionamentos dessas conexões podem ser utilizados para responder questões como: Existe caminho de um objeto a outro? Quantos objetos podem ser alcançados a partir de um determinado objeto? Qual a menor distância entre dois objetos? Existe um tipo abstrato de dados chamado grafo que é usado para modelar essas situações [?].

Um grafo é constituído de um conjunto de vértices e um conjunto de arestas que conectam pares de vértices. Um vértice é um objeto que pode conter nomes e outros atributos. Os grafos podem ser direcionados ou não direcionados. Um grafo direcionado G é um par (V, A), em que V é um conjunto finito de vértices e A é um conjunto de arestas com uma relação binária. A Figura 4.1(a) mostra um grafo direcionado com o conjunto de vértices $V = 0, 1, 2, 3, 4, 5$ e de arestas $A = (0, 1), (0, 3), (1, 2), (1, 3), (2, 2), (2, 3), (3, 0), (5, 4)$. Em um grafo não direcionado as arestas (u, v) e (v, u) são consideradas as mesmas. A Figura 4.1(b) mostra um grafo não direcionado com o conjuntos de vértices $V = 0, 1, 2, 3, 4, 5$ e de arestas $A = (0, 1), (0, 2), (1, 2), (4, 5)$. Em grafos direcionados podem existir arestas de um vértice para ele mesmo, chamadas de *self-loops*, como a aresta $(2, 2)$ no grafo direcionado da Figura 4.1(a) [?].



(a) Grafo direcionado

(b) Grafo não direcionado

Figura 4.1: Exemplo de grafos [?]

Em um grafo direcionado, a aresta (u, v) sai do vértice u e entra no vértice v. Se (u, v) é uma aresta do grafo $G = (V, A)$, então o vértice v é adjacente ao vértice u. Quando o grafo não é direcionado, a relação de adjacência é simétrica [?].

Em um grafo, um caminho $(v_0, v_1, \cdots, v_k)$ forma um ciclo se $v_0 = v_k$ e o caminho contém pelo menos uma aresta. O ciclo é simples se os vértices $v_1, v_2, \cdots, v_k$ são distintos. O *self-loop* é um ciclo de tamanho 1. Na Figura 4.1(a), o caminho $(0, 1, 2, 3, 0)$ forma um ciclo. Dois caminhos $(v_0, v_1, \cdots, v_k)$ e $(v'_0, v'_1, \cdots, v'_k)$ formam o mesmo ciclo se existir um inteiro j tal que $v'_i = v_{(j+i) mod k}$ para $i = 0, 1, \cdots, k - 1$. Na Figura 4.1(a), o caminho $(0, 1, 3, 0)$ forma o mesmo ciclo que os caminhos $(1, 3, 0, 1)$ e $(3, 0, 1, 3)$. Um grafo sem ciclos é um grafo acíclico [?].

Um grafo G é definido como *Hamiltoniano* se possui um ciclo contendo todos os vértices de G. Esse nome foi dado, porque, em 1856, Willian Rowan Hamilton inventou um jogo matemático que consistia em um dodecaedro no qual cada um dos vinte vértices recebeu o nome de uma cidade. O objetivo do jogo era viajar pelas arestas do dodecaedro, visitando cada cidade exatamente uma vez e retornando para o ponto inicial [?]. A Figura 4.2 mostra uma solução para o jogo.

Figura 4.2: Dodecaedro de Hamilton. [?]

Definições segundo Gondran, Minoux e Vajda [**?**]:

- Um caminho passando somente uma vez em cada vértice de G é chamado Caminho Hamiltoniano e tem comprimento de $N - 1$
- Um ciclo Hamiltoniano é um ciclo que passa somente uma vez em cada vértice de G e tem comprimento N.

Um grafo não direcionado é conectado se cada par de vértices está conectado por um caminho. Um grafo direcionado é fortemente conectado se cada dois vértices quaisquer são alcançáveis a partir um do outro. Um grafo ponderado possui pesos associados às suas arestas. Esses pesos podem representar, por exemplo, custos ou distâncias. Um grafo completo é um grafo no qual todos os pares de vértices são adjacentes [?].

4.2 Busca do Menor Caminho

4.2.1 Algoritmo de Dijkstra

Existem diversos algoritmos para buscas em grafo na literatura [**?**]. O algoritmo de Dijkstra utiliza a técnica do relaxamento, que nada mais é que verificar se é possível melhorar o caminho obtido até o momento passando por um vértice diferente. O algoritmo de Dijkstra apresenta uma solução $O([m + n] \times log(n))$ para a determinação do menor caminho [**?**], e é composto por três passos:

- Passo 1: Iniciar os valores:

  para cada $v \in V[G]$ **faça**
  $d[v] \leftarrow \infty$
  $\pi[v] \leftarrow nulo$

fim-para
$d[s] \leftarrow 0$

$V[G]$ é o conjunto de vértices v que forma o grafo G.

$d[v]$ é o vetor de distâncias do vértice s até cada vértice v.

$\pi[v]$ identifica o vértice de onde se origina uma conexão até v de maneira a formar um caminho mínimo;

- Passo 2: Tem-se que usar dois conjuntos: S, que representa todos os vértices v onde $d[v]$ já contém o custo do menor caminho e Q que contém os vértices restantes;

- Passo 3: Realiza-se uma série de relaxamentos das arestas:

enquanto $Q \neq \emptyset$ **faça**
$u \leftarrow$ `extraia-mín(`Q`)`
$S \leftarrow S \cup u$
para cada v adjacente a u **faça**
se $d[v] > d[u] + w(u, v)$ então
$d[v] \leftarrow d[u] + w(u, v)$
$\pi[v] \leftarrow u$
fim-para
fim-enquanto

$w(u, v)$ é o peso da aresta que vai de u a v. u e v são vértices quaisquer e s é o vértice inicial. `extraia-mín(`Q`)`, retorna o menor elemento.

4.3 Caixeiro-Viajante

O problema do caixeiro-viajante envolve um conjunto de cidades e é da classe de problemas de roteamento de nós, onde um caixeiro sai de uma cidade base, visita todas as cidades somente uma vez, e retorna à cidade base, otimizando um ou mais objetivos. Problemas de caixeiro-viajante são definidos em grafos orientados ou não orientados [?].

A definição do problema do caixeiro-viajante é: Considera-se um grafo não orientado $G = (N, E)$, em que o conjunto N consistem em n cidades e E representa o conjunto de arestas entre essas cidades. Supondo que G é um grafo completo, isto é, para qualquer par de cidades $i, j \in \mathbb{N}, i \neq j$, existe uma aresta (i, j). A distância entre as cidades i, e j é c_{ij}, e quando $c_{ij} = c_{ji}$, o problema é dito simétrico. Um caixeiro deve visitar n cidades,

passando por cada cidade somente uma vez, e retornar à cidade de partida. Esse percurso é denominado ciclo Hamiltoniano do grafo G, e o problema consiste em determinar o ciclo Hamiltoniano, ou rota, de distância mínima. Devido à sua aplicação em diversas áreas, este é um dos problemas combinatórios mais pesquisados [?].

Define-se as variáveis

$$x_{ij} = \begin{cases} 1 \text{ se o caixeiro vai diretamente da cidade } i \text{ à cidade } j, i \neq j \\ 0 \text{ se o caixeiro não vai da cidade } i \text{ à cidade } j, i \neq j \end{cases}$$

E considera-se o seguinte modelo:

$$\min \sum_{i=1}^{n} \sum_{j>i} c_{ij} x_{ij} \tag{4.1}$$

$$\sum_{j<i} x_{ji} + \sum_{j>i} x_{ij} = 2, i = 1, \cdots, n \tag{4.2}$$

$$x \in B^{n(n-1)/2} \tag{4.3}$$

A função objetivo (4.1) expressa a minimização da distância total da rota, e a restrição (4.2) impõe que cada cidade tenha somente uma cidade sucessora imediata e uma cidade predecessora imediata, ou seja, é visitada uma única vez. Uma solução para o modelo anterior pode gerar sub-rotas desconexas (Figura 4.3) [?].

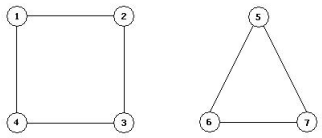


Figura 4.3: Exemplo de possíveis sub-rotas. [?]

Seja S uma sub-rota, por exemplo, $S = 1.2.3.4$ na Figura 4.3. A eliminação de sub-rotas pode ser obtida através da restrição:

$$\sum_{i \in S} \sum_{i \; in S j > i} \leq |S| - 1, S \subset \mathbb{N}, 3 \leq |S| \leq \left\lfloor \frac{n}{2} \right\rfloor \tag{4.4}$$

Que garante que, para cada conjunto S, existem no mínimo duas arestas que entram e/ou saem de S, ou seja, existem no mínimo duas arestas entre cidades de S e cidades fora de

S. A cardinalidade de S é no mínimo 3 (pois ciclo em um grafo não orientado tem pelo menos 3 vértices) e no máximo $\lfloor \frac{n}{2} \rfloor$, pois ao se eliminar ciclos com k vértices, elimina-se ciclos com $n - k$ vértices. A sub-rota $S = 1, 2, 3, 4$ é eliminada.

$$x_{15} + x_{16} + x_{17} + x_{25} + x_{26} + x_{27} + x_{35} + x_{36} + x_{37} + x_{45} + x_{46} + x_{47} \geq 2$$

Como o número de subconjuntos distintos de um conjunto de cardinalidade k é 2^k, a restrição 4.4 têm cardinalidade da ordem de 2^k, $k \geq 6$, ou seja, o crescimento é exponencial em função do número de cidades. Para $k \leq 5$, a restrição 4.2 elimina sub-rotas [?].

4.4 Considerações

Como o mapa que é utilizado no projeto é um mapa baseado em grafo (mapa topológico), é possível utilizar algumas teorias de grafos presentes na literatura como base para se achar uma solução para o problema. Assim como, foi utilizado o algoritmo de Dijkstra nas buscas de menor caminho para o robô fazer a menor trajetória de um vértice a outro, e a fundamentação matemática da teoria do caixeiro-viajante que foi utilizada como base para se achar uma solução *offline*.

Capítulo

5

Desenvolvimento e Resultados

Esta dissertação tem como objetivo o desenvolvimento de uma estratégia eficiente para determinar uma seqüência de áreas a serem visitadas em ambientes internos com a finalidade de monitoramento destes ambientes utilizando um robô móvel. O problema a ser resolvido consiste na divisão de um ambiente previamente conhecido em áreas de interesse. A cada uma dessas áreas é atribuído um valor (peso) referente à sua importância de monitoramento. A prioridade com que o robô deve visitar determinadas áreas é calculada com base na importância dessas áreas e no tempo decorrido desde a sua última visita. Áreas de maior importância devem ser visitadas mais freqüentemente.

Para determinar a trajetória do robô foram criadas duas soluções, uma *Offline* (seção 5.3.1) e outra em Tempo Real seção (5.3.2), essas soluções serão confrontadas em dois critérios de avaliação (seção 5.1), um baseado no Grau de Urgência Total e o outro baseado na freqüência relativa de cada sala.

5.1 Critérios de Avaliação

Os algoritmos e estratégias serão então testados e avaliados. As avaliações consistem na comparação dos algoritmos e estratégias. Serão utilizados dois critérios de

comparação, um critério comparando a freqüência relativa de cada sala com sua prioridade relativa, onde o melhor resultado é aquele em que a freqüência relativa se aproximar mais da prioridade relativa, pois se uma sala (a) possui uma prioridade com valor duas vezes maior que uma sala (b), a sala (a) deve ser visitada com o dobro de freqüência do que a sala (b). Esse cálculo é feito somando as diferenças quadráticas entre a prioridade relativa e a freqüência relativa de cada sala ($\sum_{i=0}^{n}(P_i/P_t - F_i/F_t)^2$). Isto indica que o algoritmo ou estratégia se manteve fiel à definição do problema: *Áreas de maior importância devem ser visitadas mais frequentemente.*

O segundo critério é um gráfico mostrando a progressão da somatória dos graus de urgência de todas as salas (Grau de Urgência Total, $\sum_{i=0}^{n} U_i$) , no qual o melhor resultado consiste em manter o menor valor da somatória dos graus de urgência; mostrando que o algoritmo ou estratégia levou o robô às salas com maior eficiência. O grau de urgência U é calculado multiplicando-se a prioridade relatica da sala P/P_t pelo tempo decorrido desde a última visita t ($U_i = P_i/P_t \times t_i$). A casa visita o t é zerado.

5.2 Metodologia

Para a solução do problema descrito, o robô tem uma descrição completa do ambiente em que atua (mapa). Foram utilizados somente dois sensores: Odômetro para localizar o robô no mapa e o LASER para o desvio de obstáculos. O controle do robô foi desenvolvido utilizando a biblioteca Player/Stage. Para desvio de obstáculos utilizou-se a técnica VFH que já vem implementada no Player e se mostrou suficiente para o projeto.

Cada ambiente tem um mapa topológico. O robô deve utilizar esse mapa para se locomover de uma sala para outra no ambiente. Para os algoritmos e estratégias determinarem a seqüência de salas a serem visitadas, é considerado um grafo completo com todas as salas, pois para determinar o melhor caminho entre uma sala e outra será utilizado o algoritmo de Dijkstra no mapa topológico.

Como um dos critérios de avaliação está relacionado ao tempo que o robô fica sem visitar as salas é possível deduzir que a solução seja um ciclo.

Considera-se:

- Um ambiente com S salas;

- Um ciclo hamiltoniano C qualquer;

- C_i é a i-ésima sala visitada no ciclo;

- Uma velocidade constante do robô (tanto linear quanto angular);

- Δt_i o tempo para sair da sala i e chegar na sala $i + 1$.

Como o tempo de viagem entre as mesmas salas é constante todos os Δt_i são constantes, portanto o tempo total do ciclo T é constante ($\sum_{i=1}^{S} \Delta t_i = cte$). Se o tempo do ciclo é constante, o tempo que o robô demora para revisitar cada sala é constante igual a T. Mas isso não é interessante para solução do problema pois o grau de urgência de cada sala é diferente, então se uma sala tem uma prioridade muito alta, seu grau de urgência vai ser muito alto até o robô revisitá-la.

Para resolver esse problema, basta fazer com que o robô revisite essa(s) sala(s) mais de uma vez no ciclo. Então supondo um ciclo de tamanho n sendo $n \geq S$ o tempo total do ciclo continua constante ($\sum_{i=1}^{n} \Delta t_i = cte$). Portanto a solução do problema consiste em encontrar esse ciclo.

5.3 Soluções

Foram definidos dois tipos de soluções: uma solução ***offline***, ou seja, a solução é calculada em um computador e depois informada ao robô qual seqüência de sala ele deve seguir, essa seqüência não é alterada. E a outra é uma solução **tempo real**, ou seja, o robô define para qual sala deve ir durante a execução do algoritmo, baseando suas decisões no que está acontecendo no momento. A solução ***offline*** é dividida em duas partes: **Gerador**, que cria possíveis seqüências de salas ótimas, e o **Avaliador** que analisa e computa uma nota para as seqüências de salas criadas pelo **Gerador**

5.3.1 *Offline*

Esse método consiste em explorar as diversas possíveis combinações de seqüências de salas para encontrar a seqüência ótima antes de informar ao robô qual seqüência de salas deve seguir. Um programa chamado `gerador`, seguindo uma determinada heurística, gera as possíveis seqüências de salas, essas seqüências são fornecidas ao programa `avaliador` que, baseado no critério de maior Grau de Urgência Total, analisa a seqüência e retorna uma avaliação ao `gerador` que tomará a decisão de: descartar a seqüência, preservar a seqüência para gerar futuras seqüências ou guardar a seqüência como possível seqüência ótima.

5.3.1.1 Gerador

O programa chamado `gerador` é utilizado para gerar os candidatos à seqüência de salas ótima do mapa analisado. Esses candidatos são avaliados pelo programa `avaliador` e de acordo com a avaliação o `gerador` irá guardar a seqüência como possível ótima, descartar ou continuar utilizando essa seqüência para gerar novas seqüências.

O programa `gerador` utiliza uma classe (tipo de variável) chamada *Agente* contendo os seguintes atributos:

- `vertice`: Vértice no qual o agente se encontra no momento;
- `caminho`: Vetor de salas que guarda a seqüência de salas que o agente percorreu até chegar no vértice atual;
- `avaliacao`: Avaliação da seqüência de salas do agente;
- `tempo`: Tempo em segundos que o robô levou para percorrer a seqüência de salas até o vértice atual.

O `gerador` recebe apenas um parâmetro como entrada: o nome do mapa que quer achar a seqüência ótima. O programa inicia criando uma seqüência percorrendo as salas na ordem numérica (Sala 1, sala 2..) e classificada como possível ótima. A seguir o programa cria um *Agente* em cada sala do mapa para explorar as possibilidades do robô começar em cada sala, esses *agentes* são organizados em uma fila de candidatos.

O laço principal do programa consiste em retirar um *agente* da fila e para cada sala i é criado um novo *agente*, simulando que o *agente* retirado da fila navegou até a sala i. Em cada novo *agente* é atualizada a seqüência de salas visitadas e o agente é reavaliado. Se a avaliação for um valor negativo o *agente* é inserido na fila de *agentes*. Se o valor retornado for 0 (zero) o *agente* é descartado. Caso o avaliador retorne um valor positivo e esse valor é menor que a avaliação da atual seqüência ótima, o novo *agente* contendo a seqüência melhor é guardado como possível seqüência ótima. Como o algoritmo somente insere na fila os *agentes* que podem gerar seqüências ótimas, o critério de parada é quando não existe mais *agentes* na fila, o programa então retorna o *agente* com a seqüência ótima.

A Figura 5.1 mostra o fluxograma do programa `gerador` e o algoritmo 5.3.1 contém seu pseudocódigo.

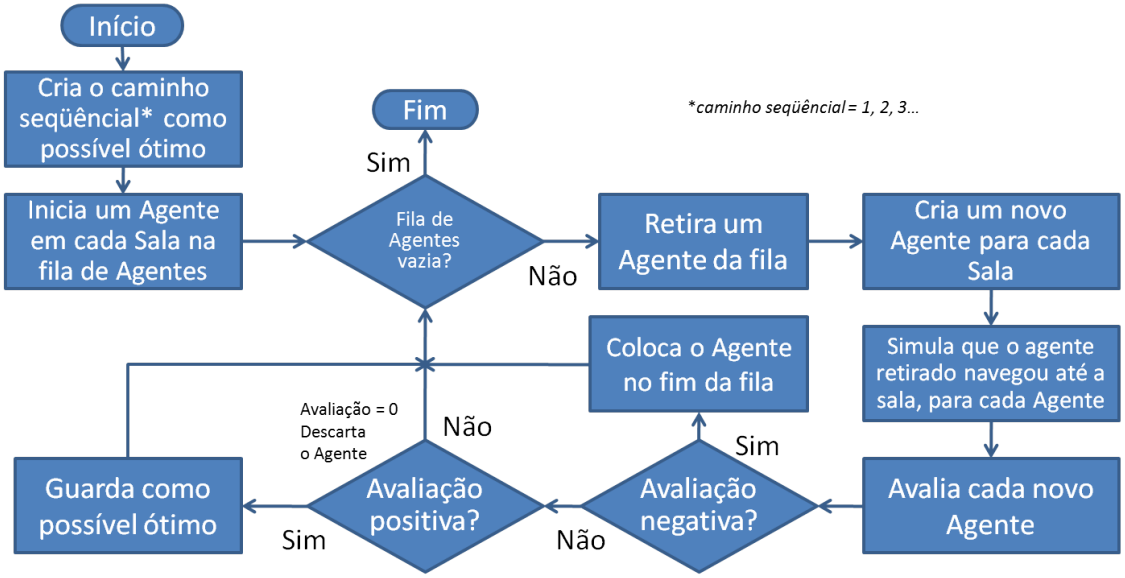


Figura 5.1: Fluxograma do `gerador`

Algoritmo 5.3.1 `gerador`

Entrada: mapa
Saída: *melhor_agente*

1: *agente_otimo.vertice* ← 1
2: *agente_otimo.caminho* ← 1, 2, 3...
3: *agente_otimo.avaliacao* ← *avaliar*(*agente_otimo*)
4: **para cada** sala *s* do mapa **faça**
5: Inicia *agente* na sala *s*
6: Adiciona *agente* na *fila*
7: **fim-para**
8: **repita**
9: Retira o *agente* do topo da fila
10: **para cada** sala *s* do mapa **faça**
11: *novo_agente* ← *agente*
12: Atualiza o vértice do *agente* com a sala *s*
13: Adiciona a sala *s* ao caminho do *novo_agente*
14: Avalia o *novo_agente* como o caminho atual
15: **se** A avaliação for negativa **então**
16: {*novo_agente não visitou todas as salas ou não é um loop*}
17: Adiciona o *novo_agente* no fim da *fila*
18: **fim-se**
19: **se** A avaliação for positiva **então**
20: {*novo_agente é melhor que o ótimo atual*}
21: *agente_otimo* ← *novo_agente* {*Atualiza o agente_otimo com o novo_agente*}
22: **fim-se**
23: **fim-para**
24: **até que** Não exista *agente* na fila
25: **retornar** *agente_otimo*

A seguir, uma explicação mais detalhada de como o `gerador` cria as possíveis seqüências de salas para serem avaliadas. Será usado como exemplo um ambiente com quatro salas. Independente do mapa topológico, o gerador trabalha utilizando um grafo completo contendo somente as salas (Figura 5.2).

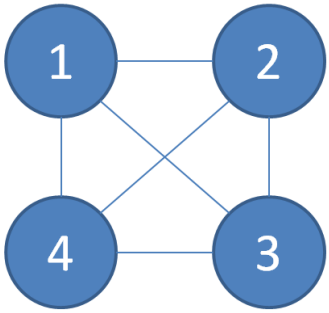


Figura 5.2: Grafo completo das salas.

O `gerador` então cria um *agente* em cada sala como mostra a Figura 5.3.

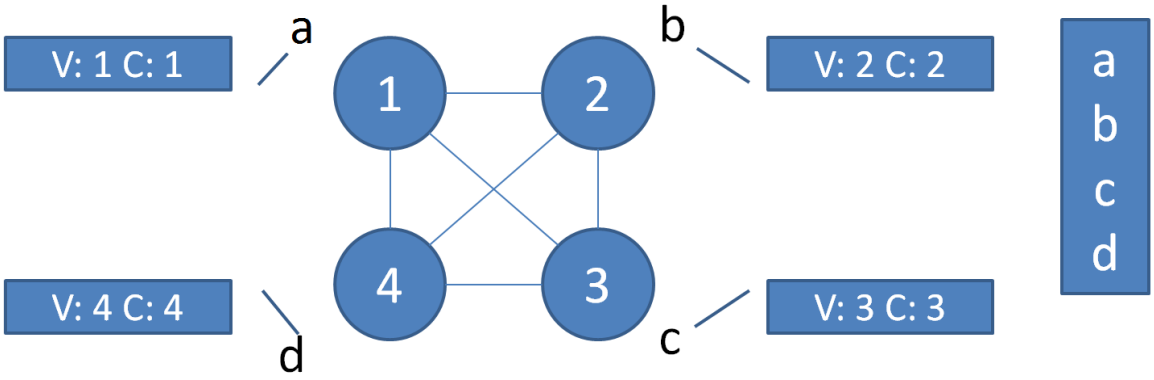


Figura 5.3: Um agente em cada sala, e a fila de *agentes*.

Seguindo o laço principal, o *gerador* remove o primeiro *agente* da fila, no caso é o *agente* **a** (Figura 5.4).

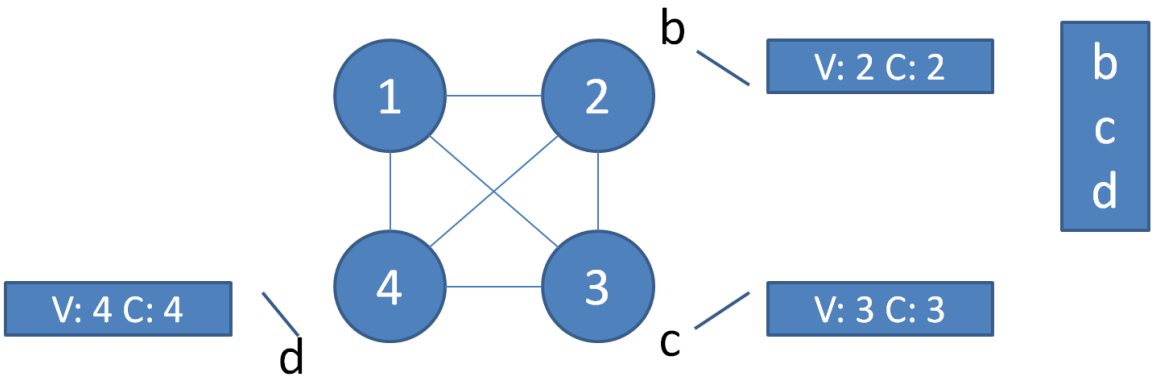


Figura 5.4: Removendo o *agente* **a**.

Em seguida o `gerador` cria uma cópia do *agente* **a** para cada sala, e simula que esses *agentes* navegaram até sua sala correspondente (Figura 5.5). Atualizando o caminho que esse agente percorreu para chegar a essa sala e a avaliação desse caminho.

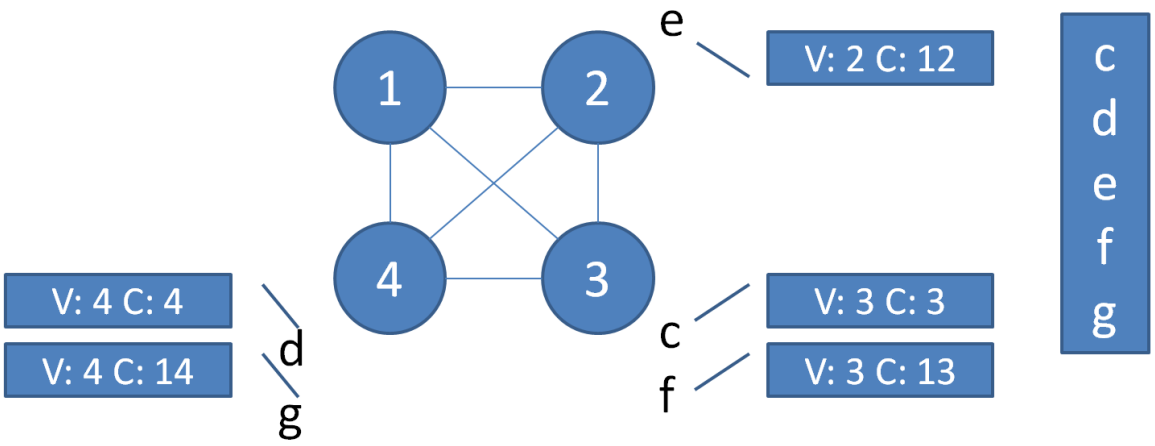


Figura 5.5: Inserindo os *agentes* **e**, **f** e **g**.

Após feito isso o `gerador` recomeça o ciclo, removendo o primeiro *agente* da fila (no caso *agente* **b**), criando outros *agentes* e inserindo-os na fila de acordo com a avaliação adquirida (*agentes* **h**, **i** e **j**) como mostra a Figura 5.6.

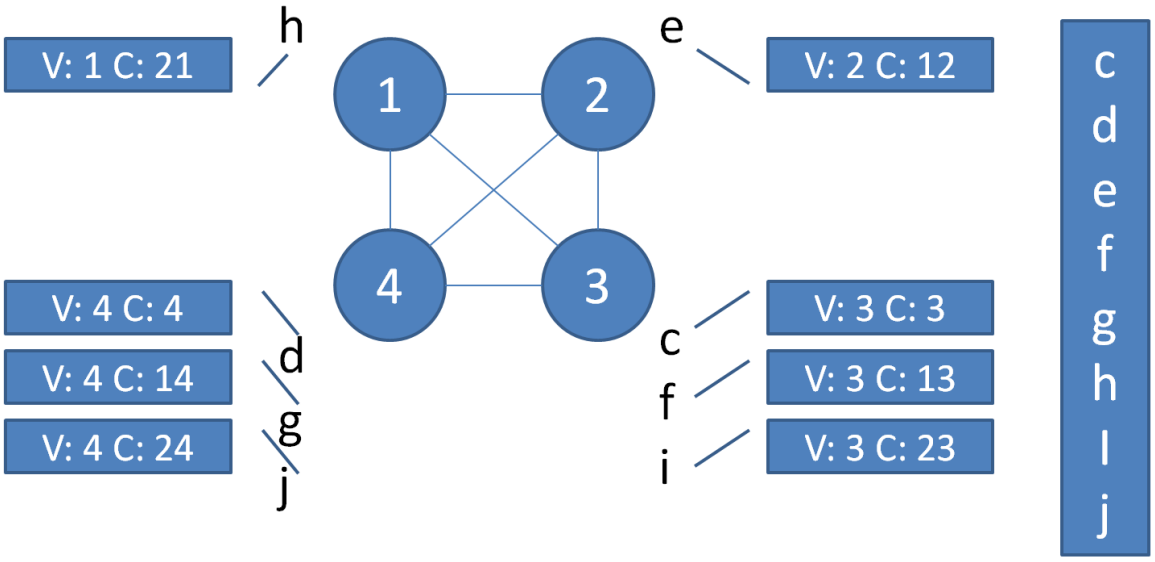


Figura 5.6: Removendo o *agente* **b** e inserindo os *agentes* **h**, **i** e **j**.

O `gerador` permanece nesse ciclo até que não exista mais *agente* na fila.

5.3.1.2 Avaliador

O programa chamado `avaliador` é utilizado para avaliar os candidatos a seqüência de salas ótima do mapa analisado gerado pelo programa `gerador` (Seção 5.3.1.1) e

retorna para o `gerador` um valor numérico. Baseado nesse valor o `gerador` irá guardar a seqüência como possível ótima, descartar ou continuar utilizando essa seqüência para gerar novas seqüência.

O `avaliador` recebe três parâmetros como entrada:

- O nome do mapa onde o caminho será avaliado;
- Um vetor de vértices, representando a seqüência de salas a ser avaliada;
- Um valor opcional de limite para a avaliação.

Com esses parâmetros o `avaliador` inicia a simulação do robô navegando pelo mapa. A velocidade linear, que determina a velocidade com que o robô anda para frente ou para trás, é definida pela constante `SIMULACAO_VEL` como 1 m/s. A velocidade angular, que determina a velocidade com que o robô vira, é definida pela constante `SIMULACAO_ROT` como 0,5 rad/s. O tempo que o robô demora para visitar uma sala é definido pela constante `VISITAR_SALA` como 5 s. Todos esses valores foram definidos arbritrariamente.

Durante a simulação, a cada visita de sala o `avaliador` mede o Grau de Urgência Total e salva o maior valor, esse valor vai ser a avaliação do caminho neste mapa. Após a simulação o `avaliador` verifica se a avaliação do caminho for maior que o limite fornecido o valor 0 (zero) é retornado, se nenhum limite for fornecido o `avaliador` utiliza o valor do maior inteiro do compilador. Se o caminho não visita todas as salas o `avaliador` retorna a avaliação com valor negativo, se todas as salas foram visitadas o `avaliador` verifica se o caminho é um *loop*, ou seja, se o caminho começa e termina no mesmo vértice, caso contrário ele também retorna a avaliação em valor negativo. Se o caminho for um *loop*, visita todas as salas e o valor da avaliação for menor que o limite fornecido (ou menor que o maior inteiro do compilador), o valor da avaliação, maior Grau de Urgência Total em todo o percurso, é retornado pelo `avaliador`.

Para critério de parada o `avaliador` compara o estado das salas, ou seja, se para cada sala, o Grau de Urgência e o número de visitas é o mesmo ao término do caminho.

A Figura 5.7 mostra o fluxograma do programa `avaliador` e o algoritmo 5.3.2 contém seu pseudocódigo.

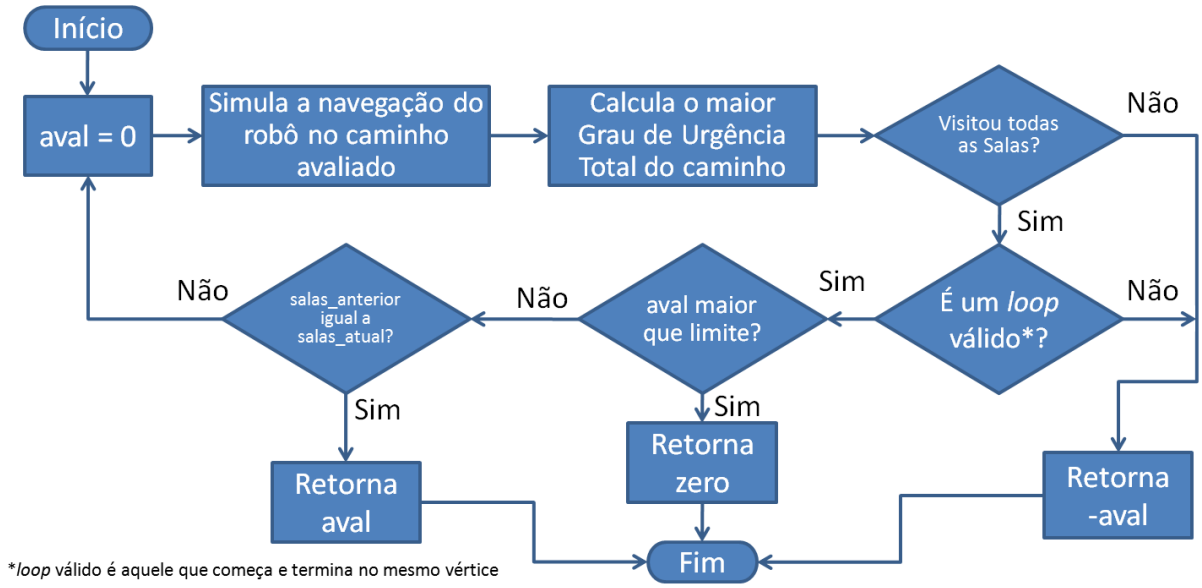


Figura 5.7: Fluxograma do `avaliador`

Algoritmo 5.3.2 `avaliador`

Entrada: mapa, caminho, limite = `INT_MAX`
Saída: A avaliação do *caminho* no *mapa*

```
1: carregar (mapa)
2: aval ← 0
3: repita
4:    salas_anterior ← salas_atual
5:    para cada vértice v em caminho faça
6:       Calcula o tempo necessário para o robô chegar ao vértice v
7:       Soma a esse tempo o tempo para visitar a sala do vértice v
8:       Atualiza os Graus de Urgência das salas com o tempo calculado
9:       se Grau de Urgência atual > maior então
10:         maior ← Grau de Urgência atual
11:      fim-se
12:      Visita a sala do vértice v
13:   fim-para
14:   se aval > limite então
15:      retornar 0
16:   fim-se
17:   se Caminho não visita todas as salas do mapa então
18:      retornar −aval
19:   fim-se
20:   se Primeiro vértice do caminho diferente do último vértice do caminho então
21:      {Ou seja, não é considerado um loop}
22:      retornar −aval
23:   fim-se
24: até que salas_anterior = salas_atual
25: retornar aval
```

5.3.2 Tempo Real

A solução em tempo real inicia as prioridades de todas as salas em zero, cada sala recebe um valor correspondente à chance dessa sala gerar uma emergência. Emergência é quando a sala solicita que o robô vá visitá-la, como uma lixeira cheia ou a sala não ser visitada por um tempo mínimo determinado por exemplo. A cada emergência gerada a prioridade da sala é acrescida de uma unidade. O robô segue o paradigma de ir à sala de maior grau de urgência, visita a sala e procura a próxima sala de maior grau de urgência.

A chance de emergência que cada sala recebe é proporcional às prioridades, para fazer o cálculo basta dividir a prioridade da sala pela soma das prioridades do mapa.

$$p_s = \frac{P_s}{\sum_{i=1}^{TotalSalas} P_i} \tag{5.1}$$

Isso faz com que a soma das chances seja igual a um. Por exemplo, se em um mapa de quatro salas as prioridades forem (Tabela 5.1):

Tabela 5.1: Exemplo de prioridades

Sala	Prioridade
1	5
2	1
3	4
4	5

Suas chances de gerar uma emergência ficaram da seguinte forma (Tabela 5.2):

Tabela 5.2: Exemplo de chances de emergência

Sala	Chance
1	0,333
2	0,067
3	0,267
4	0,333

5.4 Resultados

Para a solução em Tempo Real, por se tratar de uma solução baseada em números aleatórios, foram executados 100 repetições em cada mapa.

Para os testes foram criados sete mapas, cada qual com uma peculiaridade diferente como mostra a tabela 5.3

Tabela 5.3: Mapas para testes

Designação	N# de Salas	Prioridades
A	4	Iguais
B	4	Diferentes
C	5	Diferentes
D	5	Diferentes
E	6	Iguais
F	6	Diferentes
G	8	Diferentes

Na seqüência são mostrados os resultados em cada mapa:

- Mapa A: Com quatro salas de prioridades iguais (Tabela 5.4) e ligação entre todas as salas (Figura 5.8):

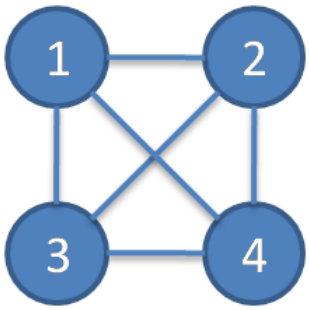


Figura 5.8: Grafo do Mapa A e B.

Tabela 5.4: Prioridades do Mapa A.

Sala	Prioridade
1	1
2	1
3	1
4	1

A seqüência de salas a serem visitadas gerada pelo `gerador` foi: 1 2 4 3.

As prioridades relativas finais da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa, como mostra a Tabela 5.5.

A Tabela 5.6 mostra que as freqüências relativas tanto da solução *Offline* quanto da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa.

Tabela 5.5: Prioridades relativas do Mapa A.

Sala	*Offline*	Tempo Real
1	0,25	0,24989
2	0,25	0,24860
3	0,25	0,25224
4	0,25	0,24927

Tabela 5.6: Freqüências Relativas do Mapa A.

		Offline		Tempo Real	
Sala	Prioridade Rel.	Freq.	Freq. Rel.	Freq.	Freq. Rel.
1	0,25	901	0,25021	771,31	0,25114
2	0,25	900	0,24993	775,88	0,25263
3	0,25	900	0,24993	764,04	0,24877
4	0,25	900	0,24993	760,02	0,24746
Diferença Quadrática		0,00000		0,00002	

A tabela mostra também que a solução *Offline* foi melhor que a solução em Tempo Real através da diferença quadrática.

O gráfico (Figura 5.9) mostra que a solução em Tempo Real depois de estabilizar teve um desempenho inferior à solução *Offline*.

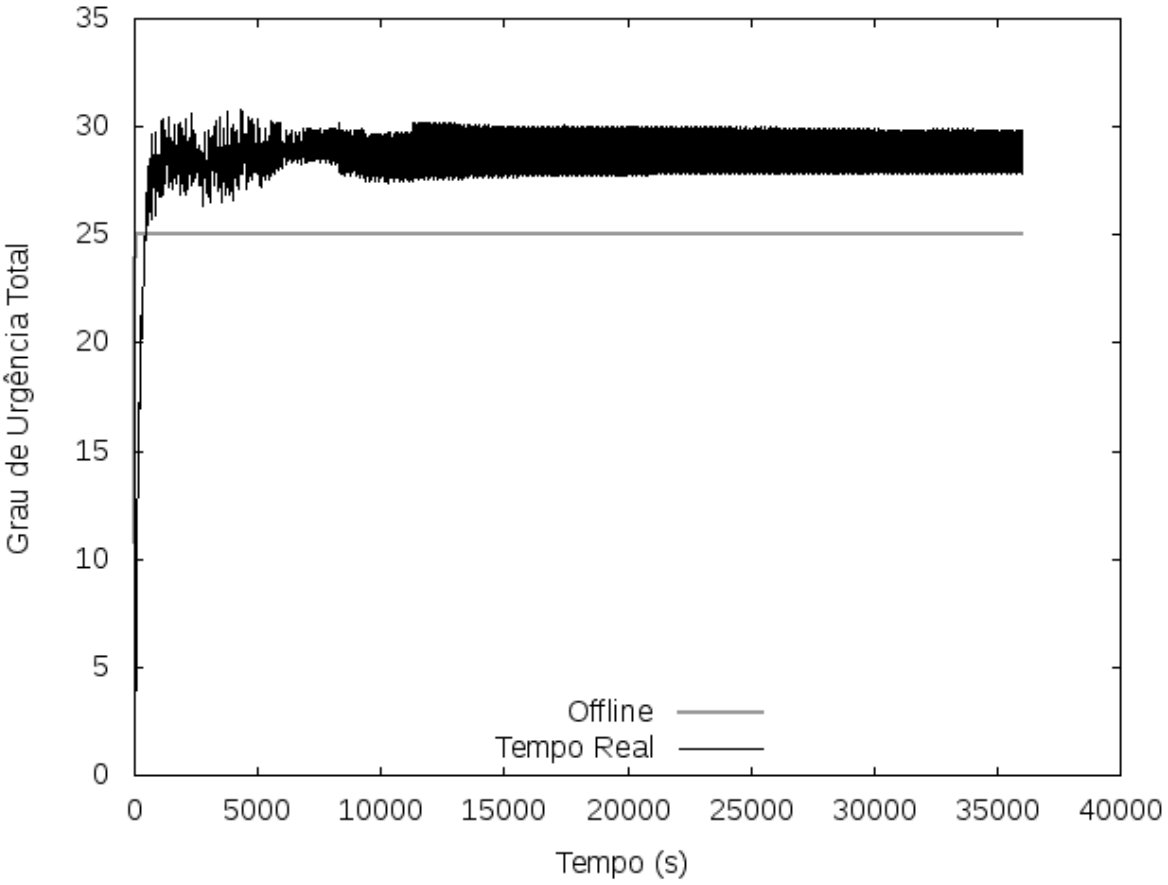


Figura 5.9: Gráfico do Mapa A.

Como o Mapa A é relativamente pequeno (dois metros quadrados) e a seqüência gerada pela solução *Offline* é visitar as salas no sentido anti-horário, o robô leva exatamente o mesmo tempo para ir de uma sala à outra. Como as prioridades iniciais do mapa são todas iguais e para gerar o gráfico foram tiradas medidas (Grau de Urgência Total) de dez em dez segundos, coincidiu de o robô a cada dez segundos estar no mesmo lugar no mapa, por isso o gráfico do Mapa A para a solução *Offline* é uma linha.

- Mapa B: Com quatro salas com prioridades diferentes (Tabela 5.7) e ligação entre todas as salas (Figura 5.8):

Tabela 5.7: Prioridades do Mapa B.

Sala	Prioridade
1	1
2	1
3	5
4	5

A seqüência de salas a serem visitadas gerada pelo `gerador` foi: 1 3 4 2 4 3.

As prioridades relativas finais da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa, como mostra a Tabela 5.8.

Tabela 5.8: Prioridades relativas do Mapa B.

Sala	*Offline*	Tempo Real
1	0,08333	0,08470
2	0,08333	0,08331
3	0,41667	0,41141
4	0,41667	0,42058

A Tabela 5.9 mostra as freqüências relativas das soluções *Offline* e em Tempo Real. Nota-se que as freqüências relativas das salas 1 e 2 da solução *Offline* são aproximadamente o dobro das suas respectivas prioridades relativas iniciais do mapa, porém, em números absolutos, é uma diferença de aproximadamente 0,08. E as freqüências relativas da solução em Tempo Real são próximas às suas respectivas prioridades relativas iniciais do mapa. A tabela mostra também que a solução em Tempo Real foi melhor que a solução *Offline* através da diferença quadrática.

O gráfico do Mapa B (Figura 5.10), diferentemente do gráfico do Mapa A (Figura 5.9), ilustra que a solução *Offline* manteve o Grau de Urgência Total dentro de uma faixa, entre dez e trinta aproximadamente, mas seu desempenho também foi superior à solução em Tempo Real.

Tabela 5.9: Freqüências Relativas do Mapa B.

		Offline		Tempo Real	
Sala	Prioridade Rel.	Freq.	Freq. Rel.	Freq.	Freq. Rel.
1	0,08333	546	0,16677	300,37	0,10130
2	0,08333	546	0,16677	296,35	0,09994
3	0,41667	1091	0,33323	1192,15	0,40206
4	0,41667	1091	0,33323	1176,27	0,39670
Diferença Quadrática		0,02785		0,00121	

- Mapa C: Com cinco salas que formam um X (Prioridades na Tabela 5.10) com ligações entre as salas das pontas (Figura 5.11):

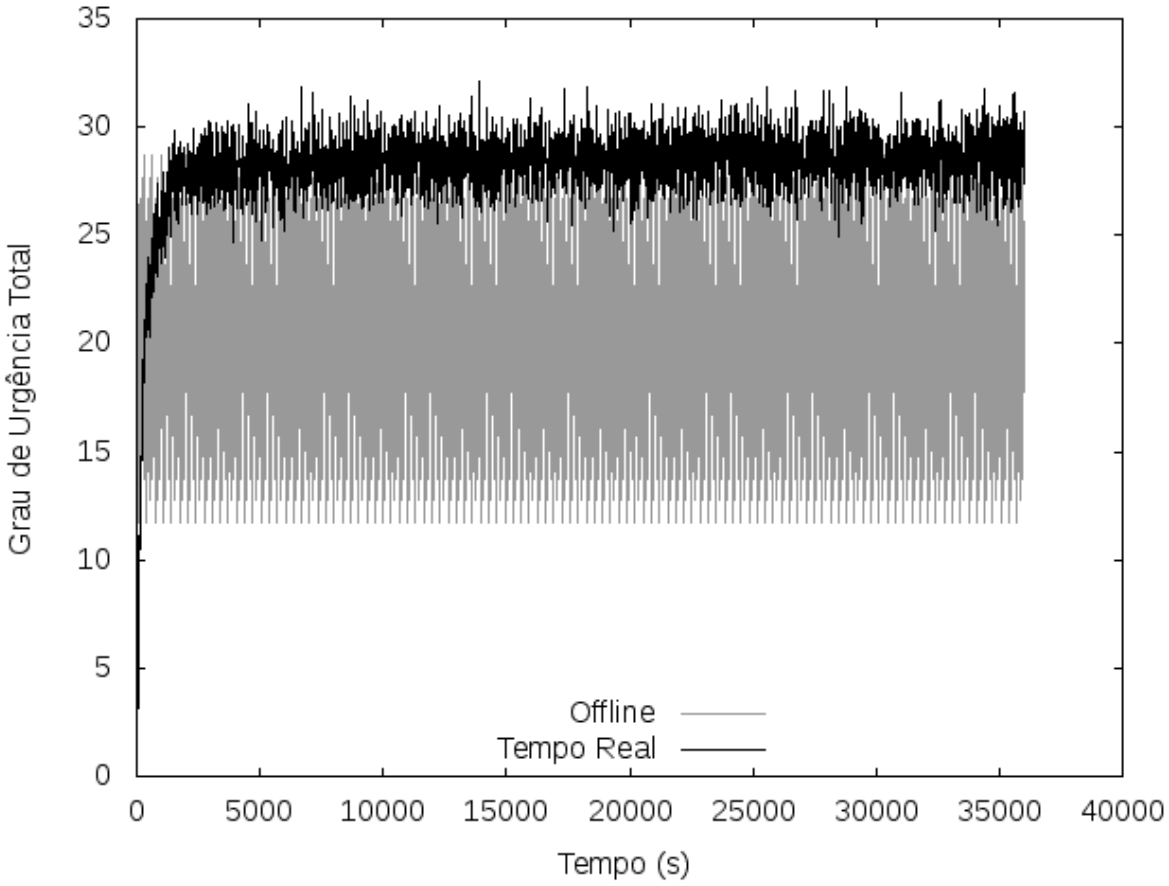


Figura 5.10: Gráfico do Mapa B.

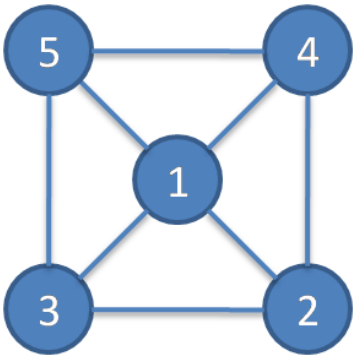


Figura 5.11: Grafo do Mapa C.

Tabela 5.10: Prioridades do Mapa C e D.

Sala	Prioridade
1	1
2	5
3	5
4	5
5	5

A seqüência de salas a serem visitadas gerada pelo `gerador` foi: 1 2 4 5 3 2 4 5 3.

As prioridades relativas finais da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa, como mostra a Tabela 5.11.

Tabela 5.11: Prioridades relativas do Mapa C.

Sala	*Offline*	Tempo Real
1	0,04762	0,04983
2	0,23810	0,24044
3	0,23810	0,23870
4	0,23810	0,23501
5	0,23810	0,23602

A Tabela 5.12 mostra que freqüência relativa da sala 1 na solução *Offline* foi maior que o dobro da sua prioridade relativa inicial do mapa, entretanto, a diferença numérica absoluta é aproximadamente 0,06. As freqüências relativas das salas restantes da solução *Offline* e as freqüências relativas de todas as salas da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa. A tabela 5.12 mostra também que a solução Tempo Real foi melhor que a solução *Offline* através da diferença quadrática.

Tabela 5.12: Freqüências Relativas do Mapa C.

		Offline		Tempo Real	
Sala	Prioridade Rel.	Freq.	Freq. Rel.	Freq.	Freq. Rel.
1	0,04762	328	0,11134	151,35	0,05670
2	0,23810	655	0,22234	633,44	0,23729
3	0,23810	654	0,22200	635,52	0,23807
4	0,23810	655	0,22234	619,58	0,23210
5	0,23810	654	0,22200	629,54	0,23583
Diferença Quadrática		0,00508		0,00012	

O gráfico (Figura 5.12) mostra que a solução em Tempo Real depois de estabilizar teve um desempenho inferior à solução *Offline*.

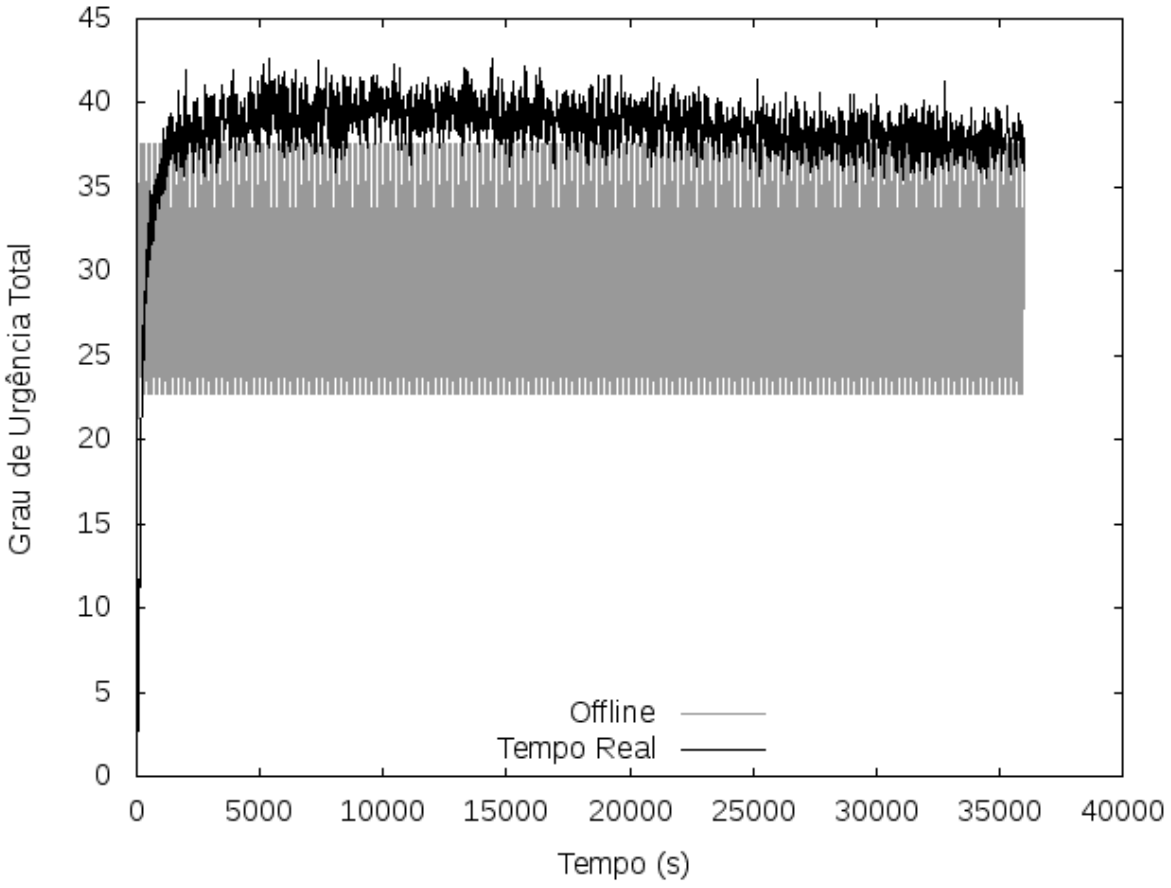


Figura 5.12: Gráfico do Mapa C.

- Mapa D: Com cinco salas em formato de X (Prioridades iguais ao mapa anterior indicados na Tabela 5.10) com somente a sala do meio ligando as outras (Figura 5.13):

  A seqüência de salas a serem visitadas gerada pelo `gerador` foi: 1 2 3 4 5.

  As prioridades relativas finais da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa, como mostra a Tabela 5.13.

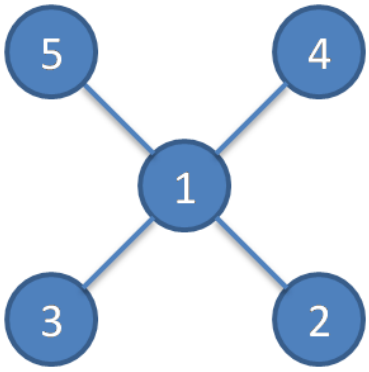


Figura 5.13: Grafo do Mapa D.

Tabela 5.13: Prioridades relativas do Mapa D.

Sala	*Offline*	Tempo Real
1	0,04762	0,04598
2	0,23810	0,23794
3	0,23810	0,24530
4	0,23810	0,23984
5	0,23810	0,23094

A Tabela 5.14 mostra que as freqüências relativas da solução *Offline* foram aproximadamente todas iguais, isso se deve ao fato de que a seqüência de salas a serem visitadas gerada pelo `gerador` manda o robô visitar cada sala apenas uma única vez dentro do *loop*. Na mesma tabela pode-se ver também que as freqüências relativas da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa. A tabela mostra também que a solução Tempo Real foi melhor que a solução *Offline* através da diferença quadrática.

Tabela 5.14: Freqüências Relativas do Mapa D.

		Offline		Tempo Real	
Sala	Prioridade Rel.	Freq.	Freq. Rel.	Freq.	Freq. Rel.
1	0,04762	456	0,20009	107,18	0,05481
2	0,23810	456	0,20009	467,55	0,23908
3	0,23810	456	0,20009	462,17	0,23633
4	0,23810	456	0,20009	455,16	0,23275
5	0,23810	455	0,19965	463,53	0,23703
Diferença Quadrática		0,02906		0,00009	

O gráfico (Figura 5.14) mostra que a solução em Tempo Real depois de estabilizar teve um desempenho inferior à solução *Offline*.

- Mapa E: Com seis salas de prioridades iguais (Tabela 5.15) em formato de espinha de peixe (Figura 5.15):

Tabela 5.15: Prioridades do Mapa E.

Sala	Prioridade
1	2
2	2
3	2
5	2
6	2
7	2

A seqüência de salas a serem visitadas gerada pelo `gerador` foi: 1 5 2 6 3 7.

As prioridades relativas finais da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa, como mostra a Tabela 5.16.

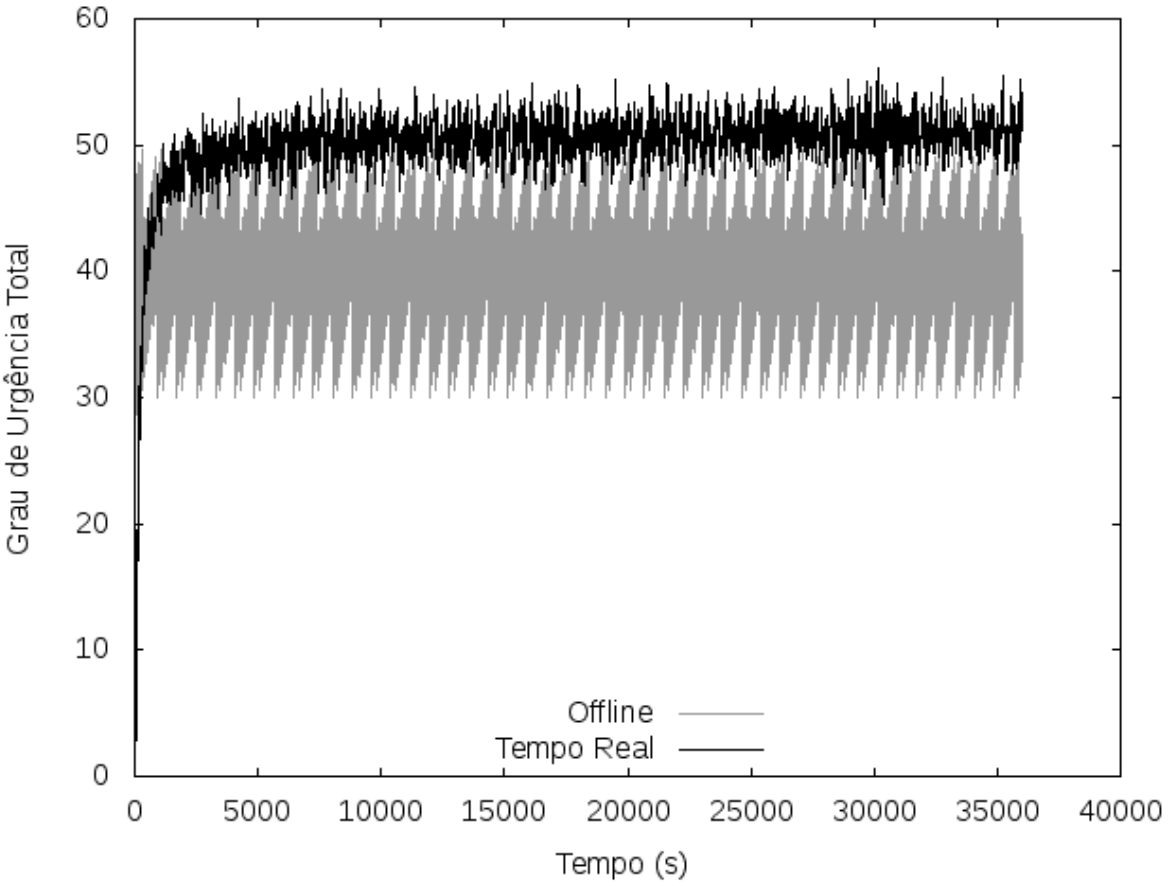


Figura 5.14: Gráfico do Mapa D.

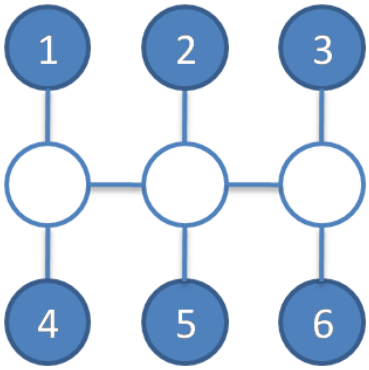


Figura 5.15: Grafo do Mapa E e F.

Tabela 5.16: Prioridades relativas do Mapa E.

Sala	*Offline*	Tempo Real
1	0,16667	0,16990
2	0,16667	0,16613
3	0,16667	0,16571
5	0,16667	0,16339
6	0,16667	0,16720
7	0,16667	0,16768

A Tabela 5.17 mostra que as freqüências relativas tanto da solução *Offline* quanto a em Tempo Real foram próximas às prioridades relativas iniciais do mapa. A Tabela 5.17 mostra também que a solução *Offline* foi melhor que a solução Tempo Real através da diferença quadrática.

Tabela 5.17: Freqüências Relativas do Mapa E.

		Offline		Tempo Real	
Sala	Prioridade Rel.	Freq.	Freq. Rel.	Freq.	Freq. Rel.
1	0,16667	311	0,16694	274,01	0,16869
2	0,16667	311	0,16694	267,74	0,16483
3	0,16667	310	0,16640	271,14	0,16692
5	0,16667	311	0,16694	273,42	0,16832
6	0,16667	310	0,16640	262,80	0,16179
7	0,16667	310	0,16640	275,25	0,16945
Diferença Quadrática		0		0,00004	

O gráfico (Figura 5.16) mostra que a solução em Tempo Real depois de estabilizar teve um desempenho inferior à solução *Offline*.

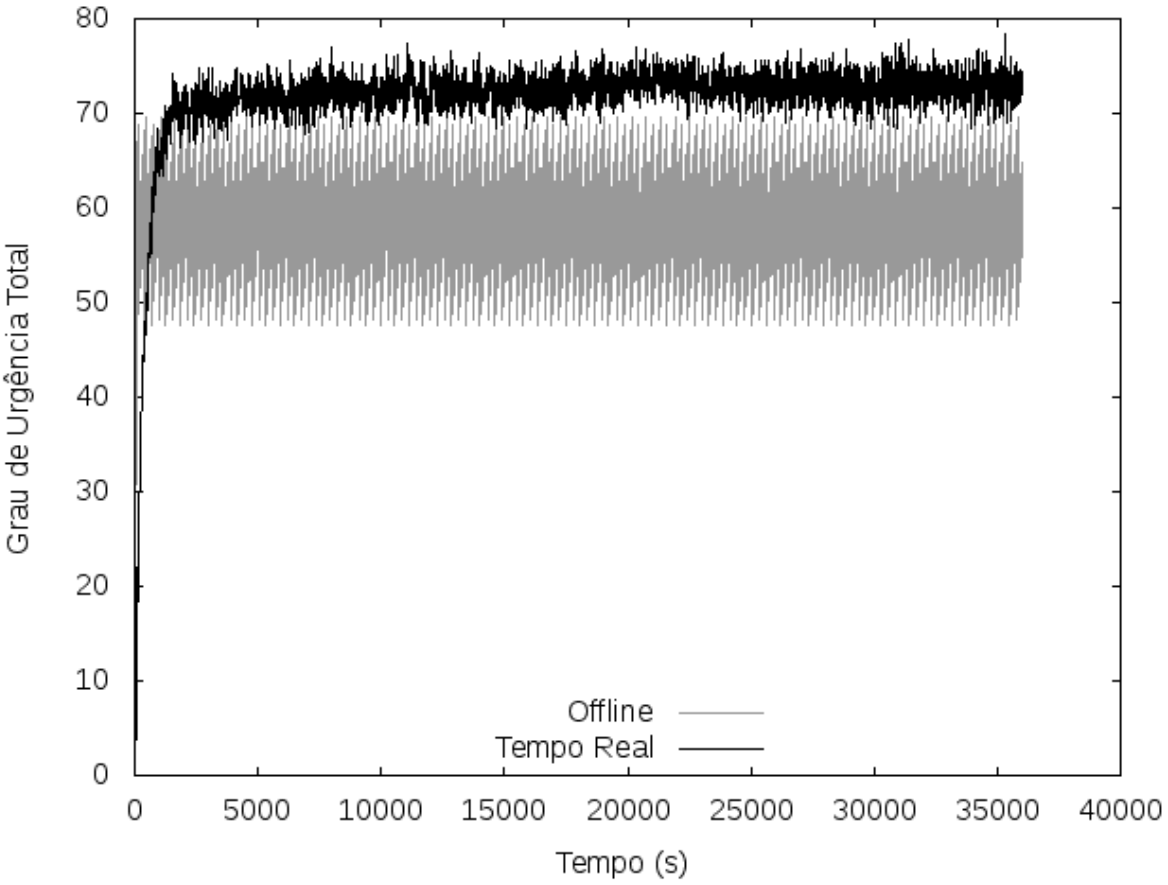


Figura 5.16: Gráfico do Mapa E.

- Mapa F: Com seis salas de prioridades diferentes (Tabela 5.18) em formato de espinha de peixe (Figura 5.15):

Tabela 5.18: Prioridades do Mapa F.

Sala	Prioridade
1	5
2	1
3	2
5	2
6	1
7	1

A seqüência de salas a serem visitadas gerada pelo `gerador` foi: 1 2 6 5 1 7 3 5.

As prioridades relativas finais da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa, como mostra a Tabela 5.19.

Tabela 5.19: Prioridades relativas do Mapa F.

Sala	*Offline*	Tempo Real
1	0,41667	0,41297
2	0,08333	0,08386
3	0,16667	0,16624
5	0,16667	0,17418
6	0,08333	0,07849
7	0,08333	0,08427

A Tabela 5.20 mostra que as freqüências relativas da solução *Offline* foram diferentes das prioridades relativas iniciais do mapa, visitando as salas 1 e 5 o dobro de vezes que as demais salas, pois na seqüência determinada pelo `gerador` as salas 1 e 5 aparecem duas vezes cada uma. Todavia as freqüências relativas da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa. A Tabela 5.20 mostra também que a solução Tempo Real foi melhor que a solução *Offline* através da diferença quadrática.

Tabela 5.20: Freqüências Relativas do Mapa F.

		Offline		Tempo Real	
Sala	Prioridade Rel.	Freq.	Freq. Rel.	Freq.	Freq. Rel.
1	0,41667	462	0,25000	585,01	0,35364
2	0,08333	231	0,12500	167,40	0,10119
3	0,16667	231	0,12500	283,61	0,17144
5	0,16667	462	0,25000	294,42	0,17798
6	0,08333	231	0,12500	156,95	0,09488
7	0,08333	231	0,12500	166,87	0,10087
Diferença Quadrática		0,04167		0,00488	

O gráfico (Figura 5.17) mostra que a solução em Tempo Real depois de estabilizar teve um desempenho inferior à solução *Offline*.

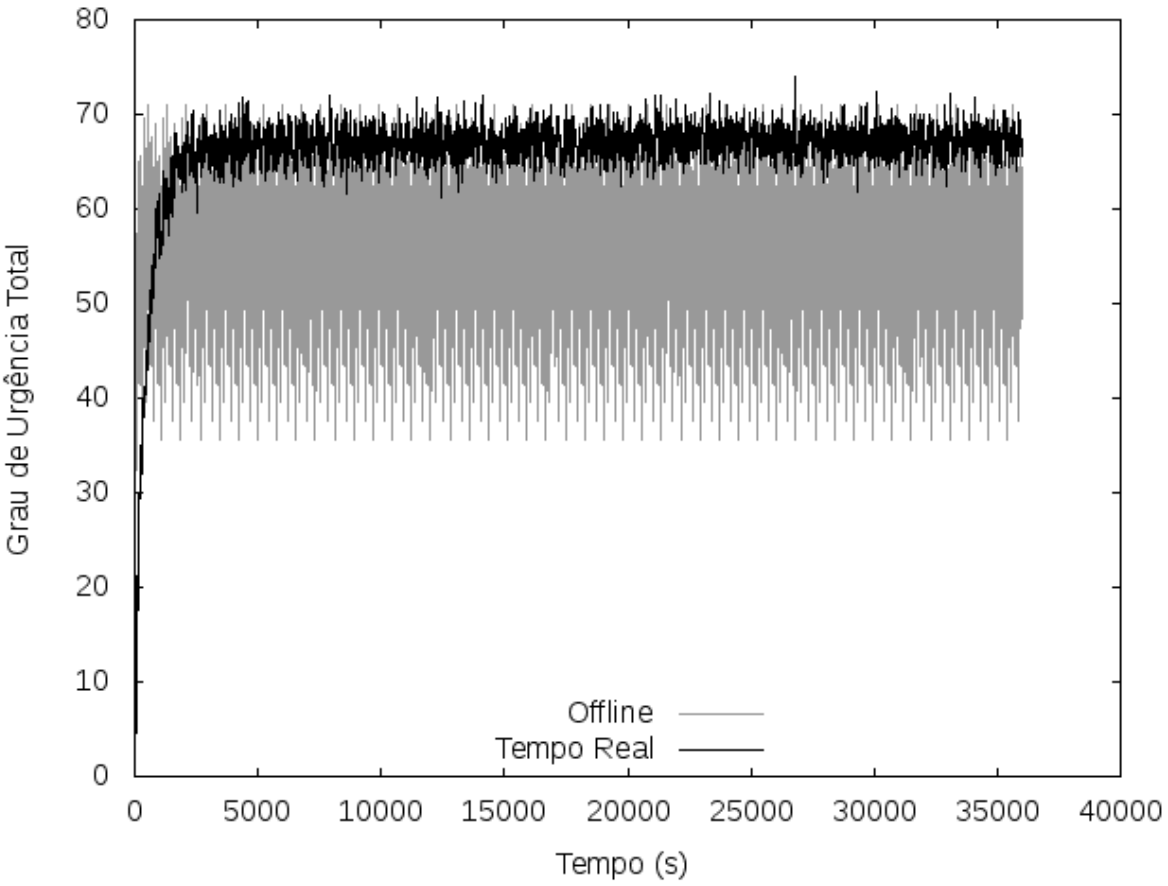


Figura 5.17: Gráfico do Mapa F.

- Mapa G: Com oito salas de prioridades diferentes (Tabela 5.21) simulando um apartamento real (Figura 5.18):

Tabela 5.21: Prioridades do Mapa G.

Sala	Prioridade
1	5
3	5
4	3
5	3
7	5
9	4
11	4
12	4

A seqüência de salas a serem visitadas gerada pelo `gerador` foi: 1 12 9 11 5 7 3 4.

As prioridades relativas finais da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa, como mostra a Tabela 5.22.

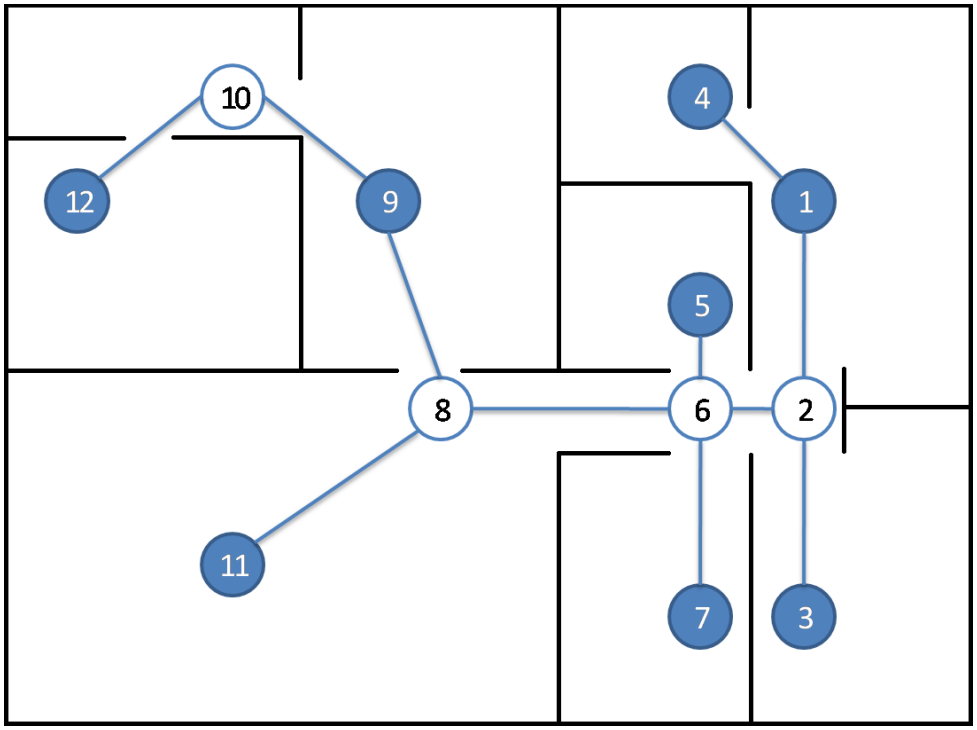


Figura 5.18: Grafo do Mapa G.

Tabela 5.22: Prioridades relativas do Mapa G.

Sala	*Offline*	Tempo Real
1	0,15152	0,15426
3	0,15152	0,15233
4	0,09091	0,08996
5	0,09091	0,08924
7	0,15152	0,14934
9	0,12121	0,12566
11	0,12121	0,12133
12	0,12121	0,11789

A Tabela 5.23 mostra que as freqüências relativas da solução *Offline* foram aproximadamente todas iguais, isso se deve ao fato de que a seqüência de salas a serem visitadas gerada pelo `gerador` manda o robô visitar cada sala apenas uma única vez dentro do *loop*, porém as maiores diferenças numéricas são as das sala 4 e 5 que são aproximadamente 0,03. Na mesma tabela pode-se ver também que as freqüencias relativas da solução em Tempo Real foram próximas às prioridades relativas iniciais do mapa. A Tabela 5.23 mostra também que a solução em Tempo Real foi melhor que a solução *Offline* através da diferença quadrática.

O gráfico (Figura 5.19) mostra que a solução em Tempo Real depois de estabilizar teve um desempenho inferior à solução *Offline*.

Tabela 5.23: Freqüências Relativas do Mapa G.

		Offline		Tempo Real	
Sala	Prioridade Rel.	Freq.	Freq. Rel.	Freq.	Freq. Rel.
1	0,15152	247	0,12525	234,33	0,14970
3	0,15152	246	0,12475	237,48	0,15172
4	0,09091	246	0,12475	152,10	0,09717
5	0,09091	246	0,12475	147,87	0,09447
7	0,15152	246	0,12475	222,23	0,14197
9	0,12121	247	0,12525	194,75	0,12442
11	0,12121	247	0,12525	187,90	0,12004
12	0,12121	247	0,12525	188,62	0,12050
Diferença Quadrática		0,00446		0,00016	

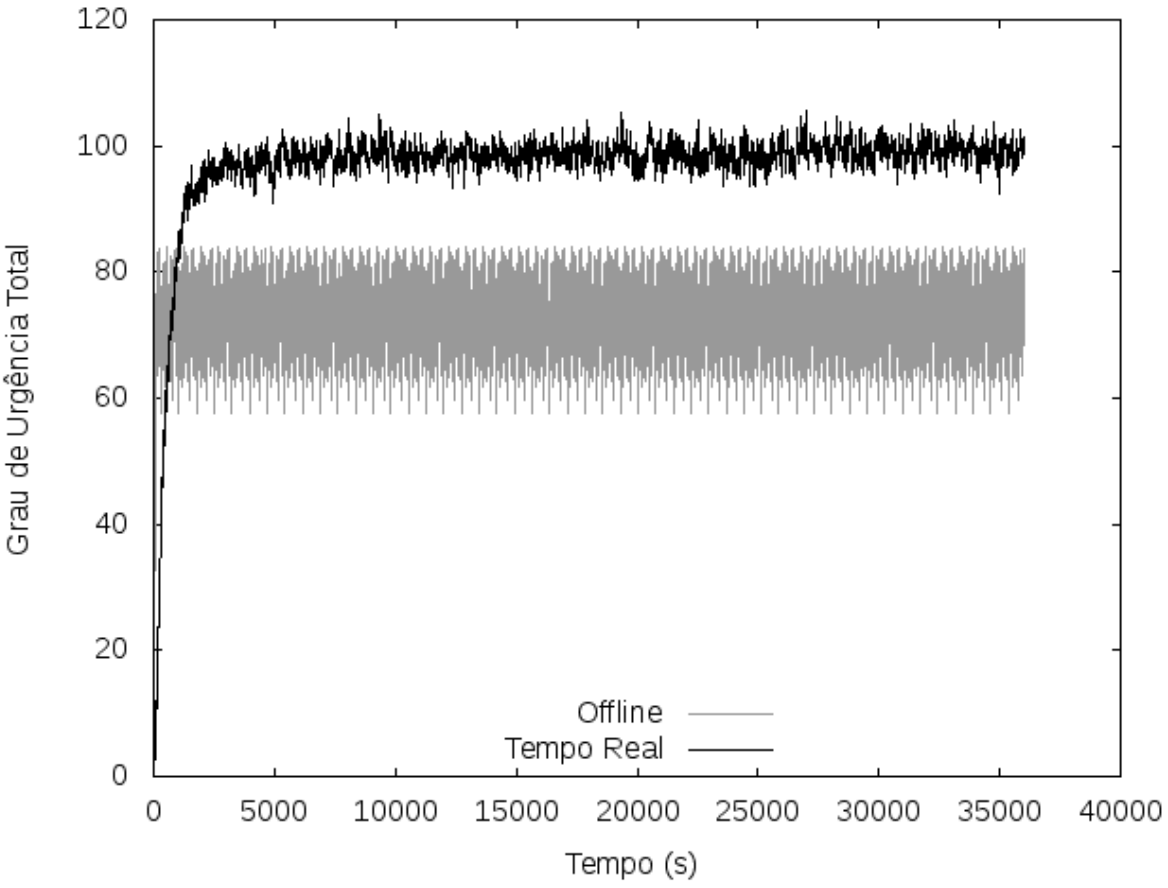


Figura 5.19: Gráfico do Mapa G.

5.4.1 Análise Comparativa

Foi calculada uma média do Grau de Urgência Total de cada solução em cada mapa com os últimos dez mil segundos (Tabela 5.24); analisando essa média nota-se que existe uma tendência de que quanto maior o número de salas mais eficiente será a solução *Offline* em relação à solução em Tempo Real, como mostra o gráfico na

Figura 5.20.

Tabela 5.24: Média dos Últimos Dez Mil Segundos.

Mapa (N# de Salas)	*Offline*	Tempo Real
A (4)	25,000	28,150
B (4)	20,980	28,346
C (5)	28,905	38,753
D (5)	39,719	50,326
E (6)	58,527	72.622
F (6)	55,557	66,711
G (8)	73,318	100,217

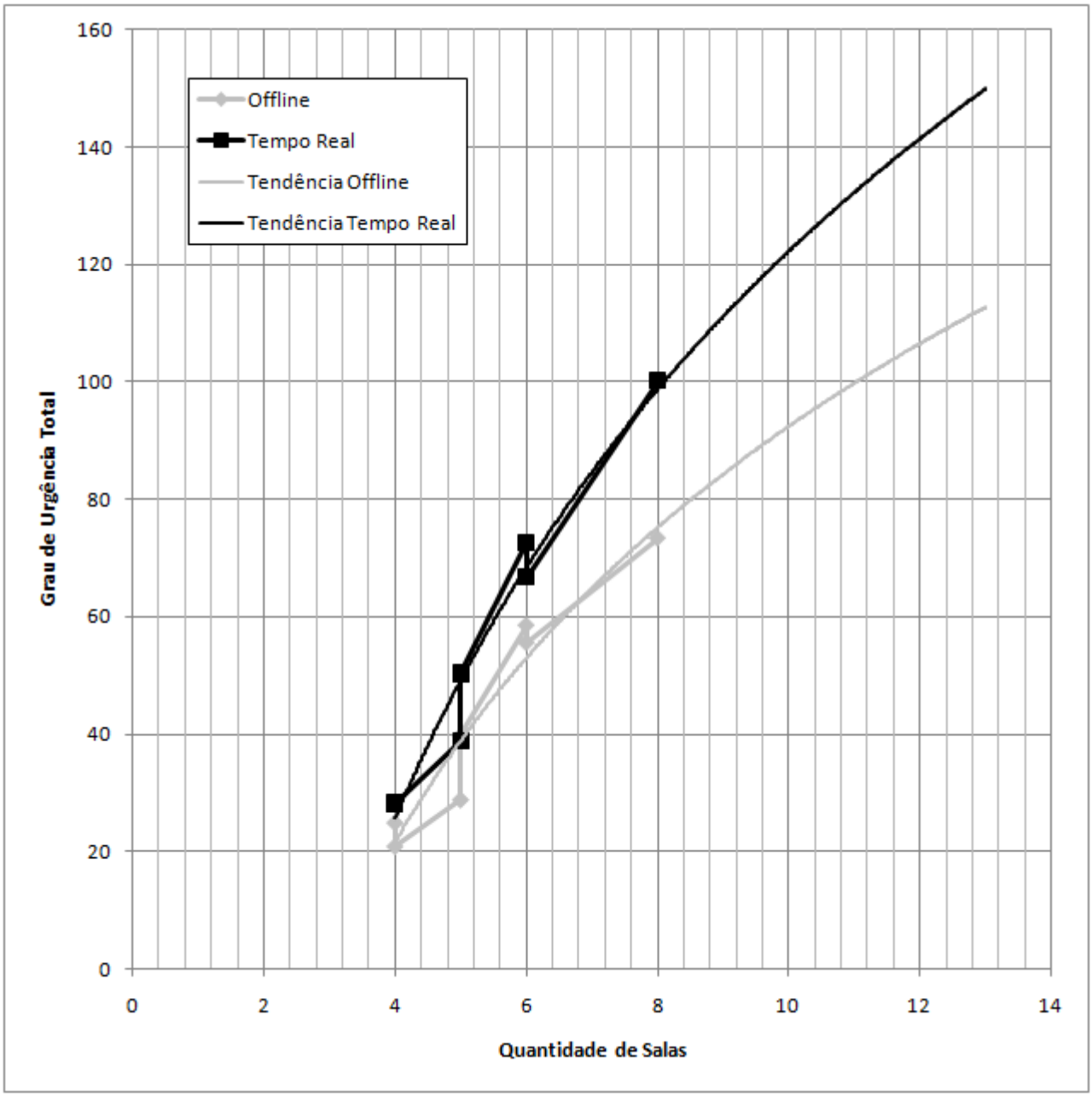


Figura 5.20: Tendência de Desempenho

5.5 Considerações

Analisando os testes pode-se constatar que:

- Segundo o critério de avaliação de freqüência relativa, a solução em Tempo Real foi superior à solução *Offline*, pois foi mais eficiente em cinco dos sete testes e na soma total das diferenças quadráticas (0,10812 da solução *Offline* contra 0,00652 da solução em Tempo Real);
- Segundo o critério de avaliação de Grau de Urgência Total a solução *Offline* foi melhor que a solução em Tempo Real em todos os testes realizados e o gráfico da Figura 5.20 mostrou que existe um tendência de que quanto maior o número de salas maior será a eficiência da solução *Offline* em relação à solução em Tempo Real.

Era esperado que a solução *Offline* tivesse um desempenho pior que a solução em Tempo Real no critério de avaliação de freqüência relativa, pois a avaliação das seqüências de salas à visitar é baseada somente no Grau de Urgência Total; e o desempenho das soluções segundo o critério de avaliação de Grau de Urgência Total mostra que o robô, seguindo uma heurística simples, tem um desempenho inferior do que uma solução que determina a seqüência de salas a serem visitadas, baseada no Grau de Urgência Total, antes de colocar o robô no ambiente.

Por fim, o que determina qual solução deve ser utilizada depende da aplicação. Se para a aplicação a freqüência relativa for mais importante do que manter o Grau de Urgência baixo, deve-se escolher a solução em Tempo Real, porém se para a aplicação for mais importante manter o Grau de Urgência baixo deve-se escolher a solução *Offline*, que foi baseada no Grau de Urgência Total.

Capítulo

6

Conclusão e Trabalhos Futuros

A proposta dessa dissertação é o desenvolvimento de estratégias de monitoramento de ambientes internos. Essas estratégias podem ser utilizadas em diversas aplicações. Dentre elas, um robô vigia que monitora um ambiente; um robô que monitora a temperatura e umidade de ambientes onde esse fator é crítico; um robô que faz coleta de lixo ou correspondência; um robô para fazer limpeza de ambientes; um robô para fazer entregas de remédios a pacientes em um hospital e etc., sempre dando ênfase para áreas de maior importância.

Para solucionar esse problema o ambiente à ser monitorado é dividido em áreas de interesse, relacionando uma prioridade (peso) à cada área referente à sua importância no monitoramento. A urgência com que o robô deve visitar cada área é calculada com base na prioridade da áreas e no tempo decorrido desde a sua última visita do robô. Áreas de maior importância devem ser visitadas mais freqüentemente.

Para determinar a trajetória do robô foram criadas duas soluções, uma *Offline* e outra em Tempo Real. A solução *Offline* consiste em fazer uma busca nas diversas possíveis combinações de seqüências de salas para encontrar a seqüência ótima antes de informar ao robô qual seqüência de salas deve seguir. O programa chamado `gerador` gera as possíveis seqüências de salas, essas seqüências avaliadas pelo programa `avaliador`

que, baseado no critério de maior Grau de Urgência Total, analisa a seqüência e retorna a avaliação ao `gerador` que tomará a decisão de: descartar a seqüência, preservar a seqüência para gerar futuras seqüências ou guardar a seqüência como possível seqüência ótima. A solução em tempo real inicia as prioridades de todas as salas em zero e cada sala recebe uma chance de gerar uma emergência. A cada emergência gerada a prioridade da sala é acrescida de uma unidade. O robô segue o paradigma de ir à sala de maior grau de urgência, visita a sala e procura a próxima sala de maior grau de urgência.

As soluções são comparadas em dois critérios de avaliação, um critério comparando a freqüência relativa de cada sala com sua prioridade relativa, onde o melhor resultado é aquele em que a freqüência relativa se aproximar mais da prioridade relativa. Isto indica que a solução se manteve fiel à definição do problema: *Áreas de maior importância devem ser visitadas mais frequentemente*. O segundo critério é um gráfico mostrando a progressão do Grau de Urgência Total, no qual o melhor resultado consiste em manter o menor Grau de Urgência Total, mostrando que a solução levou o robô às salas com maior eficiência.

Os testes mostraram que é possível utilizar tanto a solução *Offline* quanto a solução em Tempo Real dependendo da aplicação, pois cada uma foi eficiente em um critério específico. A solução *Offline* foi melhor no critério de Grau de Urgência total e a solução em Tempo Real foi melhor no critério de freqüência relativa.

Para dar continuidade ao projeto o próximo é explorar a utilização de mais de um robô para realizar a tarefa, cada robô monitorar um conjunto de salas divididas por prioridade ou proximidade, ou todos os robôs podem monitorar todas as salas etc.

Referências Bibliográficas

[1] B. V. Adôrno, C. S. R. Aguiar, and G. A. Borges. Planejamento de trajetória para o robô omni utilizando o algoritmo mapa de rotas probabilístico. 2005.

[2] H. Andreasson, A. Treptow, and Duckettm T. Self-localization in non-stationary environments using omni-directional vision. *Robotics and Autonomous System*, 55:541–551, 2007.

[3] M. N. Arenales, V. Armentano, R. Morabito, and H. Yanasse. *Pesquisa operacional*. Elsevier Campus, 2007.

[4] A. Baguinski. Laser measurement system object for max. *V2*.

[5] Lowell W. Beineke and Robin J. Wilson. *Selected Topis in Graph Theory*. Academic Press, 1978.

[6] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7(3):278–288, 1991.

[7] T. Braunl. *Embedded robotics: mobile robot design and applications with embedded systems*. Springer-Verlag New York Inc, 2008.

[8] W. Burgard, D. Fox, D. Henning, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. 1996.

[9] T. H. Cormen. *Introduction to Algorithms*. MIT Press, 2001.

[10] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, 2000.

[11] FriendlyRobotics. Robomow® lawn mower - automatic lawnmower. http://www.friendlyrobotics.com/, 2010.

[12] B. Gates. A robot in every home. *Scientific American Magazine*, 296(1):58–65, 2007.

[13] K. Goldberg. *Algorithmic foundations of robotics*. AK Peters, 1995.

[14] M. Gondran, M. Minoux, and S. Vajda. *Graphs and algorithms*. John Wiley \& Sons, Inc., 1984.

[15] Google. Google maps. http://maps.google.com/, 2010.

[16] F. J. Heinen and F. Osório. Sistema de controle híbrido para robôs móveis autônomos. *WTDIA/SBIA - Workshop De Teses E Dissertações D Inteligência Artificial, 2002*, 2002.

[17] IRobot. iRobot corporation. http://store.irobot.com/corp/index.jsp, 2010.

[18] J. L. Jones, A. M. Flynn, and B. A. Seiger. *Mobile Robots: Inspiration to Implementation*. AK Peters, Ltd., 1999.

[19] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[20] P. Mckerrow. *Introduction to Robotics*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1991.

[21] A. A. D. Medeiros. Introdução à robótica. pages 56–65, Natal, RN, Brasil, July 1998.

[22] MobileRobots. Pioneer. http://www.activrobots.com/ROBOTS/p2dx.html, 2010.

[23] R. Murphy. *Introduction to AI Robotics*. MIT Press, 2000.

[24] NASA. Mars pathfinder. http://mpfwww.jpl.nasa.gov/default.html, 2010.

[25] G. Neto and H. Costelha. Navegação topológica para futebol robótico. *Universidade TÃ©cnica de Lisboa*, 2003.

[26] J. R. Oliveira. Um sistema integrado para navegação autônoma de robôs móveis. Master's thesis, ICMC-USP, 2010.

[27] C. F. Olson. Probabilistic self-localization for mobile robot. *IEEE Transactions On Robotics And Automation*, 16(1):55–66, 2000.

[28] G. L. Ottoni. Planejamento de trajetórias para robôs móveis. *Projeto de Graduação em Engenharia de Computação–FURG, Rio Grande*, 2000.

[29] D. P. F. Pedrosa, A. A. D. Medeiros, and P. J. Alsina. Sistema de navegação para robôs móveis autônomos. *Latin American Knowledge Harvester*, October 2001.

[30] Pennsylvania State University Robotics. Player/stage overview. *PSU Robotics*, 2010.

[31] Player. Player project. http://playerstage.sourceforge.net/, 2010.

[32] S. J. Russell, P. Norvig, and J. F. Canny. *Artificial Intelligence*. Prentice Hall, 2003.

[33] J. M. Scatena and E. Marques. Implementação de mapas topológicos para navegação de robôs móveis baseada em computação reconfigurável. *Simpósio Latino Americano em Aplicações de Lógica Programável e Processadores Digitais de Sinais em Processamento de Vídeo, Visão Computacional e Robótica (SLALP 2004)*, 1:1–10, 2004.

[34] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.

[35] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[36] S. Thrun. Probabilistic robotics. *Commun. ACM*, 45(3):52–57, 2002.

[37] S. Thrun and A. Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceeding of the National Conference on Artificial Intelligence*, pages 944–951, 1996.

[38] S. Thrun, J. S. Gutmann, D. Fox, W. Burgard, and B. J. Kuipers. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 989–996. JOHN WILEY & SONS LTD, 1998.

[39] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hahnel, D. Montemerlo, A. Morris, Z. Omohundro, C. Reverte, and Whittaker W. Autonomous exploration and mapping of abandoned mines. *Robotics & Automation Magazine, IEEE*, 11(4):79–91, 2004.

[40] J. Wolf, W. Burgard, and H. Burkhardt. Robust vision-based localization by combining an image-retrieval system with monte carlo localization. *IEEE Transactions on Robotics*, 21(2):208–216, 2005.

[41] N. Ziviani and R. Terada. *Projeto de algoritmos com implementações em Pascal e C*. Thomson, 2004.