Universidade de Brasília

Faculdade de Tecnologia
Departamento de Engenharia Mecânica

RELATÓRIO DE TRABALHO DE GRADUAÇÃO II

NAVEGAÇÃO DO ROBÔ MÓVEL NOMAD XR4000 BASEADA EM MAPEAMENTO DINÂMICO E ESTÁTICO

POR

Márcio Moreira de Sousa

Paulo Roberto Corrêa Dutra

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Mecatrônico

ORIENTADOR:
Prof. Alberto José Álvares

Brasília, Fevereiro de 2003.

DEDICATÓRIA

Dedicamos esse trabalho aos nossos familiares.

AGRADECIMENTOS

Agradecemos a todos os professores e técnicos dos departamentos de Engenharia Mecânica, Engenharia Elétrica e Engenharia Mecatrônica que nos auxiliaram no desenvolvimento desse projeto.

RESUMO

O desenvolvimento de robôs autônomos teleoperados aconteceu devido a existência de ambientes impróprios à ação humana. Estes robôs, por sua vez, necessitam de técnicas de navegação como mapeamento do ambiente para que possam se movimentar livremente em ambientes sujeitos à variações de posicionamento de objetos. Este projeto consiste em construir mapas do ambiente que circunda o robô utilizando uma representação bidimensional. Os mapas serão construídos através da fusão de sensores sonares, infravermelhos e na estimativa de posição do robô. A navegação será realizada utilizando-se os mapas ambientais armazenados. O usuário, através de uma interface gráfica acessada localmente ou via WEB, definirá um objetivo a ser alcançado pelo robô. Através de um planejamento da trajetória, o robô se movimentará até o objetivo.

ABSTRACT

The development of teleoperated autonomous robots occurred due the existence of improper environments to the human action. These robots, in turn, need navigation techniques as mapping of the environment so that they can be put into motion freely in subject environment to the variations of object positioning. This project consists of constructing maps of the environment that surrounds the robot using a bidimensional representation. The maps will be constructed through the fusing of sensory sonars, infra-red ray and in the estimate of position of the robot. The navigation will be carried through stored the ambient maps. The user, through a graphical interface that had a local access or a WEB acces, will define an objective to be reached by the robot. Through a planning of the trajectory, the robot will put into motion itself until the objective.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

Abreviaturas

ex. = exemplo

Siglas

AGV - Automated Guided Vehicles - Veículos Guiados Automatizados
AI - Artificial Intelligence - Inteligência Artificial
API - Application Programming Interface - Interface de programação e aplicação
DSP - Digital Signal Processing - Processamento digital de sinais
GRACO – Grupo de Automação e Controle da Universidade de Brasília
GUI - Graphical User Interface - Interface gráfica para usuário
HTML - HyperText Markup Language - Linguagem Simbológica de Hipertexto
IP - Internet Protocol - Protocolo Internet
IR - Infrared - Infravermelho
JNI – Java Native Interface - Interface Nativa do Java
JVM - Java Virtual Machine - Máquina virtual Java
LED - Light Emissor Diode - Diodo Emissor de Luz
TCP - Transmission Control Protocol - Protocolo de transmissão e controle
TOF - Time of Flight - Tempo de Viagem
UML - Unified Modeling Language - Linguagem Unificada de Modelagem
WWW - World Wide Web - Rede mundial de computadores

1 INTRODUÇÃO

1.1 Motivação

As aplicações de robótica vão, hoje, desde as associadas à substituição do homem em tarefas repetitivas, susceptíveis de erro, degradantes e ate às associadas à operação em ambientes hostis (ex. ambientes radioativos, zonas profundas dos mares, espaço, etc). Os robôs móveis podem ser usados, por exemplo, em ambientes industriais perigosos para transporte de peças entra as estações de fabricação e de armazenagem.

A existência de ambientes onde a ação humana não é viável levou ao desenvolvimento de robôs autônomos teleoperados. Estes robôs, por sua vez, necessitam de técnicas de navegação como mapeamento do ambiente para movimentar-se livremente em ambientes sujeitos a variações posicionais dos objetos.

A capacidade de um robô autônomo de se movimentar no espaço, evitando qualquer tipo de colisão com os objetos que o rodeiam, tem sido um tema de grande discussão no âmbito da robótica móvel sendo a principal motivação deste trabalho de graduação.

1.2 Proposta de Trabalho

O objetivo desse trabalho é o desenvolvimento de um sistema de navegação para o robô NOMAD XR4000. A navegação será realizada por meio do mapeamento do ambiente utilizando o sistema sensorial do robô composto por sensores ultra-sônicos, infravermelhos e de colisão.

O sistema de navegação será baseado na arquitetura cliente-servidor aplicada à tecnologia de teleoperação via Internet. O robô será programado utilizando-se a linguagem de programação C/C++ e o usuário, através de uma interface gráfica desenvolvida em linguagem de programação Java, fará requisições de tarefas a serem realizadas pelo robô. Essa interface gráfica será acessada via WWW.

1.3 Organização deste Documento

O capítulo 2 apresenta uma revisão bibliográfica sobre navegação de robôs móveis, representação de ambientes, a arquitetura e linguagem de programação do robô Nomad e tecnologias de teleoperação. O capítulo 3 apresenta a metodologia adotada para a resolução

do projeto bem como a arquitetura do sistema de navegação. Em seguida, o capitulo 4 apresenta os resultados das simulações realizadas e os resultados obtidos no projeto. O capítulo 5 apresenta as conclusões do trabalho desenvolvido.

2 REVISÃO DE LITERATURA

Nesta seção serão apresentados tópicos sobre navegação de robôs, representação de ambientes, plataforma e programação do robô Nomad XR4000, fusão sensorial, odometria, planejamento de trajetória, sistema de referência, semáforos, *threads* e *Java Native Interface*.

2.1 Navegação de Robôs

Navegação é a ciência (ou arte) de direcionar o curso de um robô móvel à medida que o mesmo atravessa o ambiente [1]. Em todo sistema de navegação deseja-se alcançar o destino sem que o robô se perca ou colida em algum obstáculo. A navegação envolve três tarefas: mapeamento, planejamento e direção. Um processo de alto nível, denominado de planejamento de tarefas, especifica o destino e restrições para o sistema, como o tempo.

De forma simples, o problema da navegação é encontrar um caminho ligando o local atual inicial até uma meta, e atravessá-lo sem colisões.

A primeira parte da navegação, o mapeamento, é o enfoque principal desse projeto. Esta etapa da navegação é realizada através do uso de mapas pré-armazenados ou de mapas "aprendidos" pelo sistema sensorial à medida que o robô atravessa o ambiente. A modelagem do ambiente é realizada pela análise dos dados sensoriais para a construção e modificação dos mapas.

A segunda tarefa da navegação é o planejamento que é feito através da procura de caminhos possíveis no mapa construído. Caso não exista um mapa, o caminho é encontrado entre os objetos sentidos pelo sistema de sensoriamento no momento da construção do mapa. A melhor alternativa é então escolhida de forma a atender às restrições impostas pela tarefa.

Depois de planejado o caminho, a terceira tarefa realizada na navegação é a guiagem do robô através do caminho definido anteriormente. O movimento do robô é então controlado utilizando-se o seu modelo cinemático e dinâmico. Durante a movimentação, o processo de percepção examina continuamente os dados sensoriais a fim de detectar colisões potenciais. Quando uma colisão é possível, o planejamento da trajetória é refeito. O nível final de segurança em todo robô móvel é obtido através da detecção de colisão por sensores de toque ou contato que inibem a continuidade do movimento.

A Figura 1 mostra a hierarquia da Navegação em Robôs.

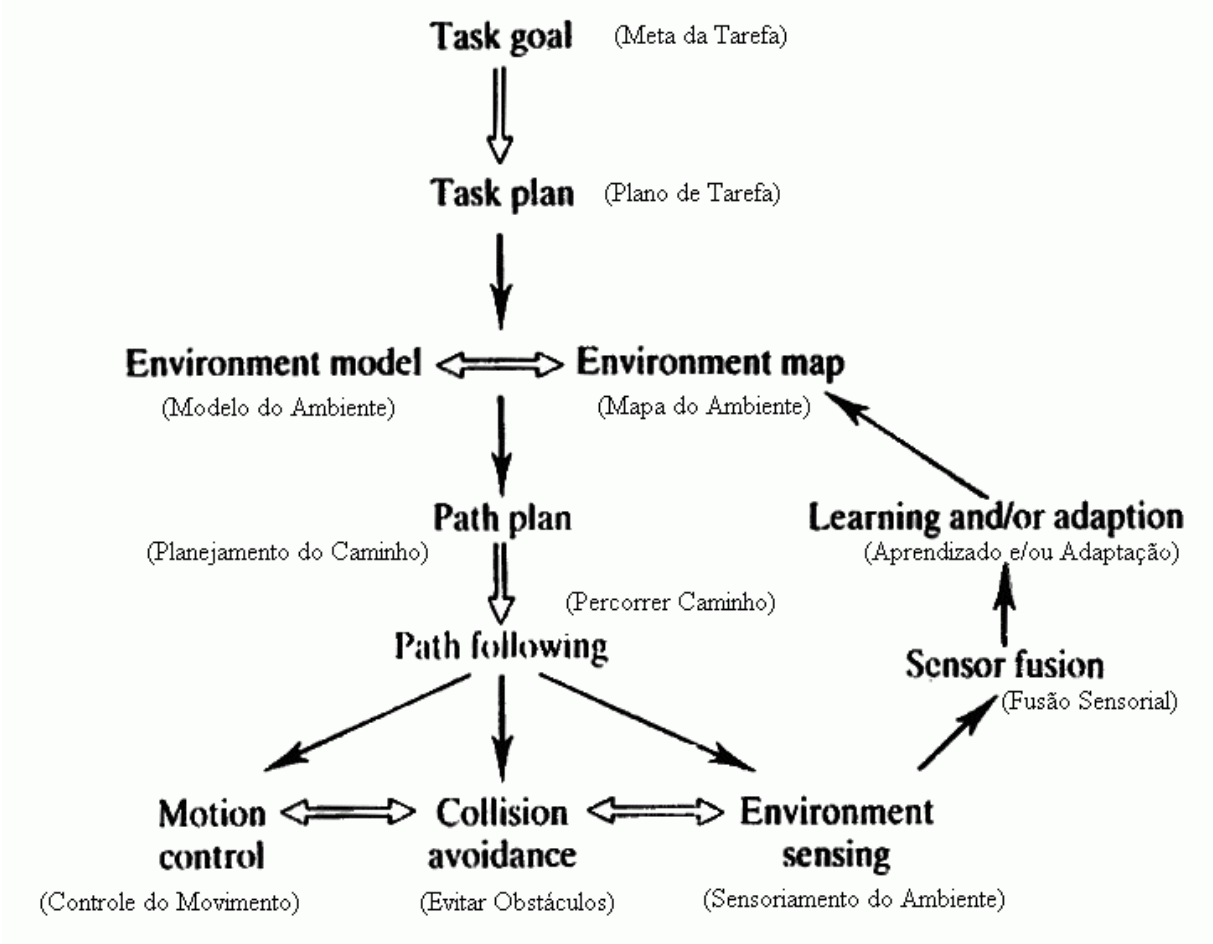


Figura 1 - Hierarquia da Navegação em Robôs (Mckerrow [1], 1991)

2.1.1 Sistemas guiados

Os sistemas guiados utilizam mapas de caminhos pré-determinados. Alguns desses sistemas armazenam a rede de caminhos como um grafo e alguns outros armazenam uma seqüência de movimentos "ensinados". Nas indústrias os veículos guiados automatizados (AGVs - *Automated Guided Vehicles*) seguem guias colocadas no chão da fábrica. Essas guias são detectadas por indução. Alguns AGVs utilizam outros métodos físicos para definir o caminho incluindo trilhas desenhadas, linhas de ímã , linhas invisíveis fosforescente e raio laser.

Esses sistemas são limitados a seguirem um número fixo de caminhos definidos e no caso de colisão o sistema é desligado e pára totalmente. Enquanto esse nível de navegação é suficiente para um chão-de-fábrica que não muda, a instalação e as modificações futuras são caras e totalmente inflexíveis. Para um industria automobilística, por exemplo, esses problemas são aceitáveis, contudo, em vários outros não.

2.1.2 *Dead reckoning*

Um método clássico de navegação é o *dead-reckoning*, cuja precisão depende diretamente da qualidade dos sensores utilizados, sendo inevitável a acumulação de erro de posição, o que requer integração com outros tipos de sensores, de modo a se compensar este erro.

Pesquisas recentes em navegação buscam a integração entre sistemas de odometria ("*dead-reckoning*") e sensores de obstáculos (sonares ou laser), de forma a gerar um mapeamento confiável do ambiente do robô.

Por exemplo, em Fuke e Krotkov [2] integra-se a informação do número de pulsos detectados por encoders instalados nas rodas do veículo (odômetro) com a informação de acelerômetros e girômetros solidários ao veículo. Em Murata e Hirose [3] e Lages etalii [4] são utilizados algoritmos de processamento de imagens que integrados à informação dos encoders estimam posição e orientação. Em Borenstein e Feng [5] é apresentado um método que combina as informações de um girômetro e um odômetro, objetivando melhorar as estimativas da orientação de um veículo móvel. Uma vez que ambos os sensores apresentam erros crescentes no tempo e a interação do veículo com o meio possui características imprevisíveis, faz-se necessária uma cuidadosa modelagem estatística dos erros. Em Barshan e Durrant-Whyte [6] encontra-se um estudo extensivo da utilização de sensores inerciais, girômetros e acelerômetros, em robótica móvel, com particular ênfase na análise do erro tipo bias, pois é o que mais degrada o desempenho de girômetros e acelerômetros.

2.1.3 *Beacons* (Sinais de Orientação para Navegação)

Para reduzir os problemas com o *dead reckonig*, alguns robôs detectam *beacons* ao longo do caminho. O robô calcula sua localização pelo sensoriamento desses *beacons* e tanto corrige sua trajetória ou modifica sua trajetória planejada para compensar esses erros.

Cada *beacon* transmite um trem de pulsos. Quando operando no "modo *beacon*", o sinal enviado contém o número de identificação. No "modo comunicação", o *beacons* tornam-se um transdutor infravermelho passando dados entre o robô e um computador.

Os robôs MELDOG [7] navegam por ruas de uma cidade utilizando um mapa. Esse mapa contém informações sobre as intersecções entre ruas, distância entre as intersecções, orientação a intersecções adjacentes e marcos (*landmarks*). O robô sente o marco para identificar em qual intersecção está e corrigir sua posição e orientação. Esse sistema de navegação consiste de uma máquina de estados no qual se move ao próximo estado quando

um marco é detectado. Esse sistema possui dois problemas. Primeiro, os marcos devem ser instalados no ambiente. Segundo, se o robô perder um marco ou fazer uma curva errada, ele se perde porque o próximo marco não contem informação de localização.

2.2 Representação do ambiente

Para a representação do ambiente é fundamental o projeto das estruturas de dados para armazenar os mapas do ambiente bem como a manipulação dessas estruturas de dados. Desse modo a primeira questão a ser levada em consideração é a forma com que o ambiente será representado internamente pelo robô.

De uma forma em geral, as representações do ambiente são divididas em mapas métricos e topológicos [8]. Os mapas métricos podem ser subdivididos em mapas que realizam decomposição espacial do ambiente (grades de ocupação) e aqueles que utilizam uma representação geométrica (mapas baseados em característica e mapas probabilísticos).

As grades de ocupação [9] podem ser definidas como sendo uma representação do ambiente em forma de uma matriz de células, onde cada célula representa uma região quadrada do ambiente e armazena um valor que indica a probabilidade de ocupação desta área. A facilidade de representação do ambiente e a possibilidade de integração de leituras de diferentes sensores constituem as principais vantagens desses mapas.

Essa representação foi desenvolvida para lidar com a incerteza espacial gerada pelos sensores do tipo sonar e tal método é conhecido como *certainty grid* [10]. Essa incerteza está relacionada com a posição dos objetos dentro da grade de ocupação e quanto maior a incerteza maior será o número de células ocupadas pelo objeto.

A representação do ambiente por grades de ocupação possui diversos problemas como exemplo: granularidade, escalabilidade e extensibilidade. Estes problemas estão relacionados ao tamanho fixo do mapa e de suas células (granularidade e escalabilidade) ocasionando no aumento do custo computacional e limitando as dimensões (extensibilidade) e precisão do ambiente representado. Outro problema é a não possibilidade de representar obstáculos estáticos diferentemente de obstáculos dinâmicos e a dificuldade de representar entidades simbólicas como: portas, cadeiras, etc.

Os mapas baseados em características representam o ambiente através de suas características e propriedades geométricas. Essa representação de ambientes internos foi proposta inicialmente por Chatila e Laumond [11]. O ambiente é representado por segmentos de linhas, pontos, arcos de círculo e poliedros a partir da informação originada pelas leituras

dos sensores de proximidade juntamente com sua incerteza de localização. Essa abordagem é comumente utilizada em conjunto com o filtro de Kalman para a minimização dos erros das estimativas de posição e orientação do robô e das leituras dos sensores.

A eficiência na representação do ambiente é uma das principais vantagens dos mapas que utilizam primitivas geométricas. Contudo existem três problemas que devem ser levados em consideração [12]. O primeiro problema é a perda de estabilidade no qual é caracterizada por pequenas variações das leituras dos sensores que prejudicam a representação. O segundo problema é a dificuldade na localização. Devido a esta representação ser baseada em primitivas geométricas é possível que a representação não seja única. E o terceiro problema é perda do poder expressivo, decorrente da dificuldade em representar completamente todas as características do ambiente através de primitivas geométricas.

De acordo com Dam [13], se o ambiente é desconhecido, a construção de um mapa baseado em características é muito mais complicada do que o mapa construído utilizando a representação das grades de ocupação. Outra consideração importante é que um mapa baseado em características é difícil de ser utilizado em ambientes não-estruturados devido a dificuldade de modelar alguns obstáculos usando primitivas geométricas. Entretanto a posição do robô pode ser calculada de forma mais eficiente do que utilizando grades de ocupação. A principal vantagem da representação baseada em grade é a possibilidade de representar tanto o espaço livre quanto o espaço ocupado do ambiente, os quais podem ser facilmente utilizados em algoritmos de navegação.

Os mapas probabilísticos constituem uma representação no qual tanto os obstáculos encontrados quanto o robô são representados como um conjunto de objetos descritos pela sua localização juntamente com uma matriz de covariância que descreve a relação espacial entre eles [14]. Surgiram como uma ferramenta para tratar as incertezas espaciais originadas das medidas sensoriais e dos movimentos do robô na representação espacial.

Essa representação é utilizada normalmente com o filtro de Kalman estendido. O filtro de Kalman pode ser definido como uma ferramenta recursiva para o problema de filtragem linear de dados discretos. Sua principal característica é a capacidade de estimar estados passados, correntes e futuros mesmo quando a dinâmica do sistema é desconhecida. Esta propriedade é utilizada para atualizar e corrigir o estado do sistema e sua incerteza associada. Esta estratégia tem sido muito utilizada em ambientes externos e internos.

Um grande problema nesse tipo de representação é que o custo computacional cresce muito quando o ambiente a ser mapeado é grande. Esse fato decorre da necessidade de se armazenar uma matriz de covariância para convergência do mapa, no qual tal matriz de

covariância cresce numa taxa quadrática com a quantidade de objetos observados. Dessa forma, quando o ambiente a ser mapeado é grande, aplicações em tempo real se tornam inviáveis [15].

Os mapas topológicos correspondem a um grafo composto por vértices e arestas, onde os vértices correspondem a locais distintos, os quais são locais de distâncias máximas entre os obstáculos, e as arestas correspondem a ligações entre diferentes locais contendo informações que permitirão ao robô navegar entre os vértices. Estas informações correspondem ao conjunto de comportamentos que permitirão ao robô ir de um vértice a outro. Também podem corresponder às informações métricas sobre a posição relativa entre os vértices.

A representação topológica apresenta três grandes vantagens sobre a abordagem baseadas em grades [16]: permite rápido planejamento de caminho, faz interface entre o solucionador do problema e o planejador simbólico e permite uma interface natural às instruções humanas.

2.3 Plataforma Nomad XR4000

O Nomad XR4000 é um sistema robótico móvel composto por sistema de controle, rede de comunicação, gerenciamento de energia, sensores e tecnologias de comunicação e software.

Figura 2 - Robô Nomad XR4000

2.3.1 Sistema Motor

O Robô Nomad XR4000 possui um subsistema formado por 8 motores para as 4 rodas. O subsistema motor oferece três graus de liberdade, dois de translação e um de rotação. Cada roda possui um eixo de translação e um de rotação independente fornecido por dois motores. O controle dos oito eixos é realizado por três DSP (*Digital Signal Processing*) e um

microcontrolador dedicado. Esse sistema de controle também realiza o cálculo da posição estimada ("*dead reckoning*")

2.3.2 Sistema Sensorial

O sistema sensorial do robô Nomad XR4000 é formado por três tipos sensores: sensores táteis, sensores por sonar ou ultra-som e os sensores por infravermelho.

Os sensores táteis são os sensores de colisão (Sistema *Sensus 150*). Possuem dois níveis de sensibilidade: um nível para contatos fracos e um nível para contatos fortes.

O sistema de sonar (Sistema *Sensus 250*) possui 48 sonares posicionados em diferentes níveis e ângulos (24 sonares posicionados na parte superior e 24 na parte inferior). A taxa de disparo e a ordem de disparo podem ser configuradas para otimizar o desempenho em qualquer ambiente. Esse sensor é utilizado para fornecer informações de distância entre 15 a 700 cm.

O sistema de sensores por luz infravermelha (Sistema *Sensus* 350) possui 48 transdutores infravermelhos posicionados na parte superior e inferior do Robô Nomad. Devido à sua alta taxa de amostragem, o *Sensus* 350 é utilizado para fornecer informações de distância de até 90 cm. A distância do objeto é determinada pela intensidade da luz refletida do objeto para o sensor. Dessa forma quanto menor o valor retornado pelo sensor, mais distante estará o robô. A refletividade da superfície e o nível da luz ambiental afetam a leitura dos sensores.

2.3.2.1 Sonar

Os sensores de proximidade sonar são comumente utilizados para fornecer a distância de objetos que estão relativamente próximos. A informação da distancia é calculada por meio da multiplicação da velocidade do som pelo tempo de viagem ("Time of flight – TOF") de um pulso ultra-sônico.

A série de sensores ultra-sônicos Polaroid 6500 emite 16 ciclos de onda quadrada (9.4 KHz) através de um transdutor eletrostático. Após um período de estabilização, o transdutor atua como um receptor, alimentando o eco detectado em um amplificador de ganho variável no tempo. O fator de ganho desse dispositivo aumenta com o tempo para compensar a propagação perdida e a atenuação do som no ar. A saída do amplificador passa então por um circuito *thresholding*. Assim que o *threshold* é excedido, o tempo decorrido desde o começo da transmissão do pulso é medido, e convertido em distância através de fator de calibração.

O transdutor não emite energia homogeneamente em todas as direções. A maior parte da energia se concentra no eixo acústico do sensor. A Figura 3 apresenta a característica de atenuação do sinal recebido.

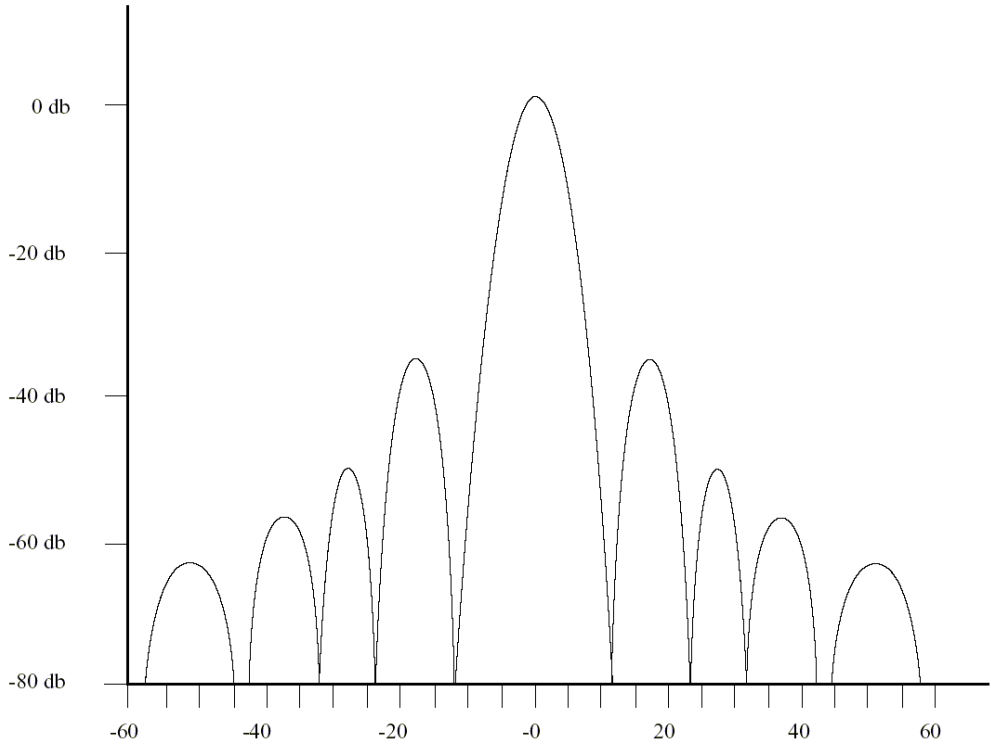


Figura 3 - Característica do Sonar

As características dos alvos de um sensor sonar podem ser divididas em dois grupos:

- Refletores: possuem dimensões maiores que o comprimento de onda (6.95 mm a 20ºC)
- Difratantes: possuem dimensões menores que o comprimento de onda. Apesar de objetos desse tamanho serem raros, superfícies ásperas como concreto comportam-se como objetos difratantes. Superfícies lisas como mesas metálicas, paredes pintadas e portas são refletoras. Uma conseqüência das propriedades refletoras das superfícies é o efeito dos múltiplos ecos.

Um problema na resposta do sonar pode ocorrer na medição de cantos (paredes perpendiculares entre si). Devido a dupla reflexão especular (uma para cada superfície do canto), a onda propagada pode demorar um pouco mais para voltar ao sonar que indicará uma

distância maior que a medida real. A Figura 4 ilustra o problema na detecção de obstáculos na medição de cantos.

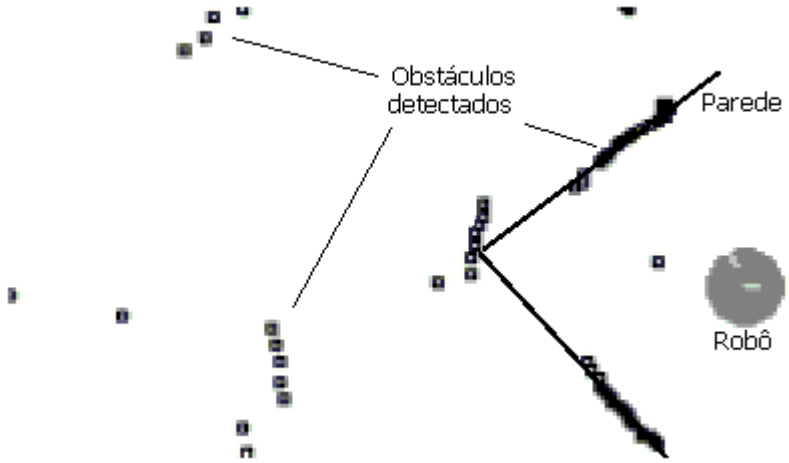


Figura 4 - O problema na detecção de obstáculos em cantos

2.3.2.2 Infravermelho

Os sensores de infravermelho fornecem informação de distância para objetos próximos, tipicamente menores do que 30 a 50 centímetros de distância. A determinação da distância é realizada pela emissão de energia infravermelha utilizando-se LEDs de alto corrente e sentindo a quantidade de energia retornada com fotodiodos infravermelho. A energia retornada é inversamente proporcional a distancia dos objetos. Esses sensores podem ser usados como sensores de distância ou de proximidade.

A energia retornada é também uma função da refletividade dos objetos. Objetos com alta refletividade retornam grandes quantidades de energia infravermelha e objetos com baixa refletividade retornam proporcionalmente baixas quantidades de energia infravermelha. A diferença na refletividade entre objetos pode causar erros nas medições de distância caso não for levado em consideração.

Cada sensor é composto de dois emissores infravermelhos e um fotodiodo. O receptor está disposto entre os dois emissores. Para obter a energia refletida, uma leitura do infravermelho é feita com os emissores desligados (medição da energia infravermelha do ambiente) e outra com os emissores ligados. A diferença entre as duas leituras é proporcional a energia refletida de um objeto próximo e largamente independente da energia infravermelha do ambiente.

Os fatores que influenciam as medições obtidas pelos sensores são as seguintes:

Geometria

Para um objeto de superfície normal ao eixo do emissor, a energia refletida será quase toda recebida pelo o receptor. Ao contrário, para um objeto de superfície quase paralela ao eixo do emissor, a energia que chegará ao receptor será mínima, indicando uma distância maior que a verdadeira já que energia recebida é inversamente proporcional a distancia do objeto.

Iluminação

A presença de uma grande quantidade de energia infravermelha no ambiente, como no caso de locais iluminados por lâmpadas incandescentes, pode saturar o receptor, reduzindo sua sensibilidade. De um modo geral, a utilização de lâmpadas fluorescentes quase não afeta as medidas de sensores infravermelhos.

Cor e Superfície

A refletividade do objeto à luz infravermelha, ou seja, a cor possui grande influência nas leituras obtidas. Além disso, a rugosidade superficial também influi na reflexão do sinal, sendo maior para superfícies lisas que rugosas. A Figura 5 mostra a energia recebida pelo infravermelho para diferentes objetos dispostos em um ângulo de 0 graus.

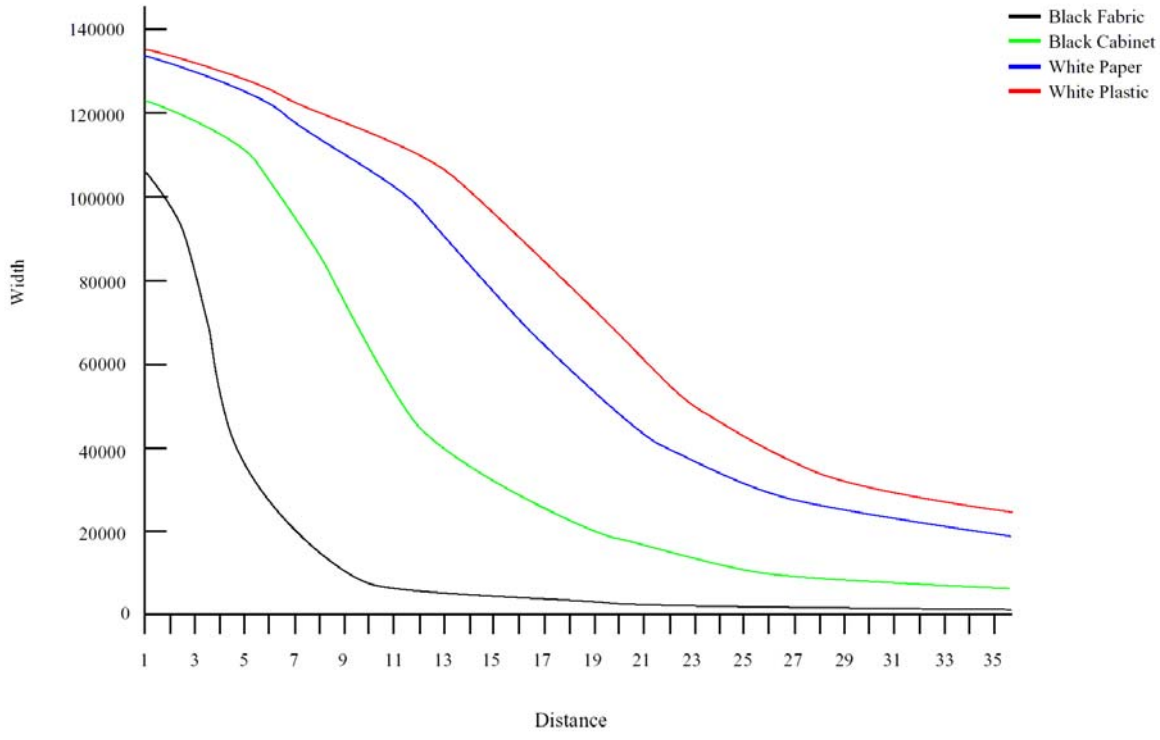


Figura 5 - Energia Recebida versus Distância do Obstáculo

2.3.2.3 Sistema de Comunicação

A comunicação do robô XR4000 com a rede local é realizada através da interface *Proxim RangeLan 2* (freqüência de 2.4 GHz), e um rádio Ethernet. Esse sistema fornece acesso à rede local utilizando o protocolo TCP/IP.

2.3.2.4 Sistema de Controle Multi-processado

O robô Nomad possui um sistema de controle (XR MemNET) multi-processado com memória compartilhada. Têm capacidade para até três computadores Pentium com sistema operacional Linux interconectados utilizando o protocolo TCP/IP. Essa arquitetura fornece ferramentas padrão de redes de comunicação e velocidade de comunicação bastante alta. A interface de memória compartilhada é *full duplex*, significando que não há colisões e o tempo de transferência pode ser precisamente determinado.

2.3.2.5 Arquitetura XRDev

XRDev é uma arquitetura multi-processada. Uma aplicação XRDev é constituída de vários processos: processos do robô, processos do usuário, processos da interface, etc., comunicando-se através da rede. Não há limite para o número de robôs que podem ser controlados na arquitetura XRDev nem para o número de programas do usuário que controla o robô. Também não há restrições no número de robôs que podem ser controlados por um programa do usuário nem no controle compartilhado de um robô por vários programas do usuário. Os programas dos usuários podem ser executados de qualquer lugar na rede.

Existem três tipos de processos:

- Processo Nrobot. È o processo que controla um robô. Podem existir quaisquer números de processos Nrobot na aplicação, mas somente um processo Nrobot sendo executado em cada robô.
- Processo Ngui. È o processo que oferece uma interface gráfica. Através dessa interface, um usuário pode enviar comandos e receber dados de qualquer processo do robô.
- Processo do usuário. O programa do usuário que controla o robô está embutido no processo do usuário

Há algumas configurações possíveis utilizando-se várias combinações desses processos.

A configuração mais simples é o Nrobot sendo executado no robô e a Ngui sendo executada em outra máquina, comunicando-se pelo rádio Ethernet. O usuário controla o robô manualmente pela Ngui. A Figura 6 mostra o diagrama dessa configuração:

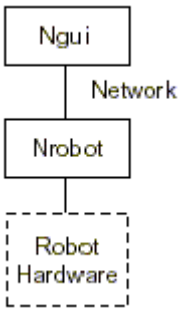


Figura 6 - Configuração Simples

Uma configuração mais elaborada envolve um processo Nrobot e um programa cliente usuário linkado à biblioteca cliente-hóspede (*Host Client Library*). Nessa configuração, o programa do usuário linkado à biblioteca cliente-hóspede pode ser compilado e executado em qualquer máquina que possua uma conexão ao robô. O processo Nrobot é executado no próprio robô. A Figura 7 mostra o diagrama dessa configuração:

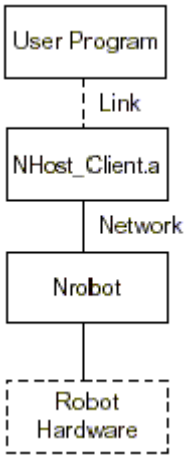


Figura 7 - Um robô e programa do usuário via rede

2.4 Programação do Robô NOMAD XR4000

2.4.1 Introdução

O paradigma de programação da arquitetura XRDev baseia-se na estrutura *N_RobotState* que armazena todos os dados dos sensores e de configuração do robô Nomad XR4000 [17]. Um ponteiro para essa estrutura é recebido pela chamada da função *N_GetRobotState*. A modificação dos parâmetros dessa estrutura indicará o comportamento dos módulos componentes do robô.

2.4.2 Estabelecendo Comunicação

Inicialmente deve-se estabelecer um canal de comunicação entre o programa cliente e o robô. Esse canal de comunicação é aberto através da chamada N_InitializeClient(), no qual é declarada da seguinte forma:

```
int N_InitializeClient (const char *scheduler_hostname,
                        unsigned short scheduler_socket)
```

O parâmetro *scheduler_hostname* é nome de rede da máquina, ou seja, o nome do robô na rede de comunicação. O outro parâmetro, *unsigned short scheduler_socket*, indica a porta TCP/IP, ou seja, o socket de comunicação entre os processos clientes e o robô.

Em seguida, deve-se estabelecer a comunicação com o robô por meio do comando:

```
int N_ConnectRobot (long RobotID)
```

O parâmetro passado é o identificador do robô que normalmente é o número 1. Da mesma forma, a comunicação com o robô deve ser finalizada através do comando:

```
int N_DisconnectRobot (long RobotID)
```

2.4.3 Estrutura de Dados do Robô

A estrutura de dados do robô *N_RobotState* é utilizada em todos os intercâmbios de dados entre o programa cliente e o robô. Os módulos componentes do robô devem ser acessados através dos campos dessa estrutura. Cada campo define uma funcionalidade disponível pelo robô. Essa estrutura é definida da seguinte forma:

```
struct N_RobotState
{
        N_CONST long RobotID;
        N_CONST char RobotType;
        struct N_Integrator Integrator;
        struct N_AxisSet AxisSet;
        struct N_LiftController LiftController;
        struct N_Joystick Joystick;
        struct N_SonarController SonarController;
        struct N_InfraredController InfraredController;
        struct N_BumperController BumperController;
        struct N_Compass Compass;
        struct N_LasetSet LaserSet;
        struct N_S550Set S550Set;
        struct N_BatterySet BatterySet;
        struct N_Timer Timer;
};
```

Um programa cliente pode acessar a estrutura *N_RobotState* com o comando *N_GetRobotState*() no qual é declarado da seguinte forma:

```
struct N_RobotState *N_GetRobotState (long RobotID);
```

Esse comando retorna um ponteiro para a estrutura ativa no robô. Feito isso, o programa cliente estará habilitado para configurar e requisitar os dados dessa estrutura.

2.4.3.1 Localização

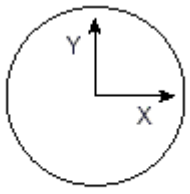
O robô Nomad possui um sistema de localização no qual estima a sua posição cartesiana (x,y,θ) em relação a um referencial fixado no chão no momento que o robô é inicializado. Esse sistema é denominado "*dead reckoning*". A estimativa da posição do robô pode ser acessada através da estrutura *N_Integrator* no qual é definida da seguinte forma:

```
struct N_Integrator
{
        BOOL DataActive;
        BOOL TimeStampActive;
        long x;
        long y;
        long Steering;
        long Rotation;
        unsigned long TimeStamp;
};
```

As coordenadas do robô podem ser obtidas pela leitura dos campos dessa estrutura[17]. Essa leitura é realizada utilizando-se o comando *N_GetIntegratedconfiguration* e a para a configuração das coordenadas do robô utilizando-se o comando *N_SetIntegratedconfiguration*. Esses comandos são declarados da seguinte forma:

```
int N_GetIntegratedConfiguration (long RobotID)
int N_SetIntegratedConfiguration (long RobotID)
```

O sistema de referência utilizado pelo sistema de localização do robô é mostrado na Figura 8:



Nomad XR4000

Figura 8 - Sistema de Referência do Nomad XR4000

2.4.3.2 Movimentação

Os comandos de movimentação do robô podem ser obtidos através do acesso à estrutura *N_AxisSet* que está definida dentro da estrutura *N_RobotState*. Por meio dessa estrutura é

possível definir todos os parâmetros de movimentação do robô como velocidade, aceleração, etc. Essa estrutura é definida da seguinte forma:

```
struct N_AxisSet
{
        BOOL Global;
        unsigned char Status;
        N_CONST unsigned int AxisCount;
        struct N_Axis Axis[N_MAX_AXIS_COUNT];
};
```

O campo BOOL Global da estrutura *N_AxisSet* define o sistema de referencia ser utilizado pelo robô. Este parâmetro quando configurado com o valor TRUE, define que todos os movimentos do robô serão em relação a um referencial global fixado no chão no momento que o robô é ligado. No caso desse valor ser configurado para FALSE, os comandos de movimentação serão interpretados em relação ao referencial fixado no próprio robô, ou seja, em relação à movimentação das juntas do robô.

O campo *struct N_Axis* define a estrutura para cada eixo de movimentação individual do robô. Essa estrutura é definida da seguinte forma:

```
struct N_Axis
{
        BOOL DataActive;
        BOOL TimeStampActive;
        BOOL Update;
        unsigned long TimeStamp;
        char Mode;
        long DesiredPosition;
        long DesiredSpeed;
        long Acceleration;
        long TrajectoryPosition;
        long TrajectoryVelocity;
        long ActualPosition;
        long ActualVelocity;
        BOOL InProgress;
};
```

A movimentação do robô dependerá da configuração dos campos dessa estrutura[17]. O acesso às informações dessa estrutura é realizado utilizando-se o comando *N_GetAxes* e a movimentação utilizando-se o comando *N_SetAxes*. Esses comandos são declarados da seguinte forma:

```
int N_SetAxes (long RobotID)
int N_GetAxes (long RobotID)
```

2.4.3.3 Sensores

O robô móvel nomad XR4000 possui 03 tipos de sensores, cada um com as suas aplicações específicas. O sensor de colisão, também chamado de sensor tátil, é utilizado para

fornecer informações de colisão com objetos no ambiente. Normalmente é extremamente desejável que não ocorra contato físico entre o robô e tais obstáculos, sendo necessário por sua vez, a utilização de sensores de proximidade. No entanto nem sempre os sensores de proximidade podem evitar esse contato físico.

O XR4000 possui 03 portas e 02 conjuntos (01 na parte superior e 01 na parte inferior) por porta totalizando 06 conjuntos. Cada conjunto possui 08 sensores de colisão totalizando 48 sensores dispostos em torno no robô. A Figura 9 mostra a disposição desses sensores.

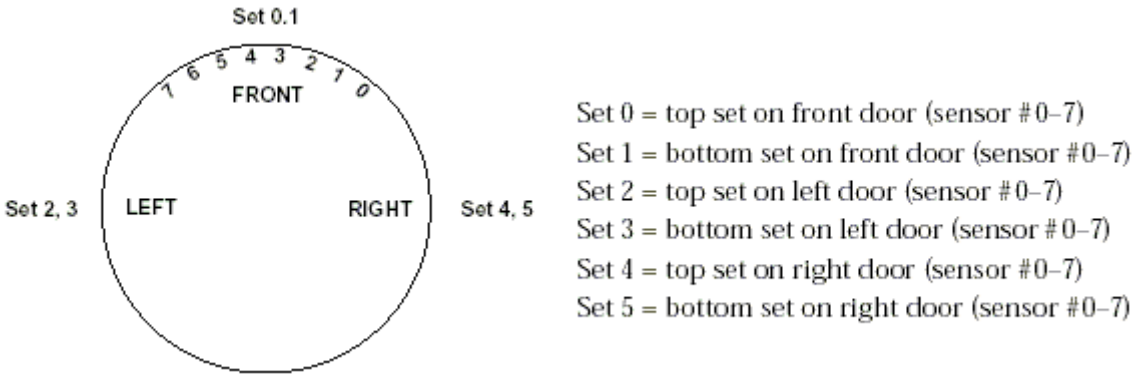


Figura 9 - Disposição dos sensores no Nomad XR4000

As informações dos sensores de contato são armazenadas na estrutura *N_BumperController* definida na estrutura *N_RobotState*. A estrutura *N_BumperController* é definida da seguinte forma:

```
struct N_BumperController
{
        N_CONST unsigned int BumperSetCount;
        struct N_BumperSet BumperSet[N_MAX_BUMPER_SET_COUNT];
};
```

A estrutura *N_BumperSet*, definida na estrutura *N_BumperController*, contém informações de cada conjunto de sensores. A estrutura *N_BumperSet* é definida da seguinte forma:

```
struct N_BumperSet
{
        BOOL DataActive;
        BOOL TimeStampActive;
        N_CONST unsigned int BumperCount;
        struct N_Bumper Bumper[N_MAX_BUMPER_COUNT];
};
```

Cada estrutura *N_BumperSet*, correspondente a um conjunto de sensores, possui a definição da estrutura de cada sensor de contato individual pertencente a tal conjunto. Cada

sensor individual possui a sua identificação armazenada na estrutura *N_Bumper* que é definida da seguinte forma:

```
struct N_Bumper
{
        char Reading;
        unsigned long TimeStamp;
};
```

O campo Reading da estrutura *N_Bumper* poderá armazenar sempre somente 3 constantes definidas no *Nclient.h*. São elas: *N_BUMPER*_NONE (sem colisão), *N_BUMPER*_LOW (fraca colisão) e *N_BUMPER*_ HIGH (forte colisão).

Um outro tipo de sensor utilizado pelo XR4000 é o sensor de proximidade infravermelho. Possui 48 sensores dispostos como pode ser observado na Figura 9. As informações dos sensores infravermelhos são armazenadas na estrutura N_InfraredController no qual é definida na estrutura *N_RobotState*. Essa estrutura é definida da seguinte forma:

```
struct N_InfraredController
{
        BOOL InfraredPaused;
        N_CONST unsigned int InfraredSetCount;
        struct N_InfraredSet InfraredSet[N_MAX_INFRARED_SET_COUNT];
};
```

A estrutura *N_InfraredSet*, definida na estrutura *N_InfraredController*, contém informações de cada conjunto de sensores infravermelhos. A estrutura *N_InfraredSet* é definida da seguinte forma:

```
struct N_InfraredSet
{
        BOOL DataActive;
        BOOL TimeStampActive;
        N_CONST unsigned int InfraredCount;
        struct N_Infrared Infrared[N_MAX_INFRARED_COUNT];
};
```

Cada estrutura *N_InfraredSet*, correspondente a um conjunto de sensores, possui a definição da estrutura de cada sensor infravermelho individual pertencente a tal conjunto. Cada sensor infravermelho individual possui a sua referência armazenada na estrutura *N_Infrared* que é definida da seguinte forma:

```
struct N_Infrared
{
        long Reading;
        unsigned long TimeStamp;
};
```

O campo Reading da estrutura *N_Infrared* armazena um valor de 0 a 255 representando a quantidade de energia infravermelha refletida de um objeto. O valor 0 representa ausência

de energia refletida (indicando que não há objeto nas proximidades do sensor) enquanto um valor de 255 representa a máxima energia refletida.

Ao realizar a chamada a função *N_GetInfrared*, as informações dos sensores infravermelhos são atualizadas na estrutura N_InfraredController que é declarada da seguinte forma:

```
int N_GetInfrared (long RobotID)
```

O terceiro tipo de sensor utilizado pelo XR4000 é o sensor de proximidade ultra-sônico. Possui 48 sensores dispostos como pode ser observado na Figura 9. As informações dos sensores ultra-sônicos são armazenadas na estrutura N_SonarController no qual é definida na estrutura *N_RobotState*. Essa estrutura é definida da seguinte forma:

```
struct N_SonarController
{
        N_CONST unsigned int SonarSetCount;
        struct N_SonarSet SonarSet[N_MAX_SONAR_SET_COUNT];
        BOOL SonarPaused;
};
```

A estrutura *N_SonarSet*, definida na estrutura *N_SonarController*, contém informações de cada conjunto de sensores sonares. A estrutura *N_SonardSet* é definida da seguinte forma:

```
struct N_SonarSet
{
        unsigned int FiringOrder[N_MAX_SONAR_COUNT + 1];
        long FiringDelay;
        long BlankingInterval;
        BOOL DataActive;
        BOOL TimeStampActive;
        N_CONST unsigned int SonarCount;
        struct N_Sonar Sonar[N_MAX_SONAR_COUNT];
};
```

O campo *FiringOrder* armazena a ordem de disparo dos sensores ultra-sônicos que compõem o conjunto especificado. A ordem de disparo pode ser importante na tentativa de minimizar a interferência de outros sonares na leitura de um dado sonar.

O campo *FiringDelay* é o atraso em milisegundos entre dois disparos consecutivos. O ajuste desse parâmetro para um valor alto pode prevenir ecos de outros sonares.

Cada estrutura *N_SonarSet*, correspondente a um conjunto de sensores sonares, possui a definição da estrutura de cada sensor ultra-sônico individual pertencente a tal conjunto. Cada sensor sonar individual possui a sua referência armazenada na estrutura *N_Sonar* que é definida da seguinte forma:

```
struct N_Sonar
{
        long Reading;
        unsigned long TimeStamp;
};
```

O campo Reading da estrutura *N_Sonar* armazena a distancia medida pelo sensor em milímetros. Quando um sensor não recebe eco (por exemplo, devido a distâncias excessivas) o valor armazenado será *N_SONAR_TIMEOUT*.

A obtenção dos parâmetros de configuração dos sonares é realizado através da chamada a função *N_GetSonarConfiguration.* A configuração desses parâmetros, como *FiringOrder*, *FiringDelay*, etc.) pode ser realizada através da chamada a função *N_GetSonarConfiguration*. Ao realizar a chamada a função *N_GetSonar*, as informações dos sensores sonares são atualizadas na estrutura N_SonarController. Essas funções são declaradas da seguinte forma:

```
int N_GetSonar (long RobotID)
int N_GetSonarConfiguration (long RobotID)
int N_SetSonarConfiguration (long RobotID)
```

2.4.3.4 Sistema de Alimentação

As informações do sistema de alimentação do robô Nomad XR4000 podem ser monitoradas através da medição das tensões de cada bateria que acompanham o robô. As informações das baterias são armazenadas na estrutura *N_Batteryset* definida na estrutura *N_RobotState*. A estrutura *N_ Batteryset* é definida da seguinte forma:

```
struct N_BatterySet
{
        struct N_Battery Battery[N_MAX_BATTERY_COUNT];
        BOOL DataActive;
};
```

Cada bateria individual possui a sua referencia armazenada na estrutura *N_Battery* que é definida da seguinte forma:

```
struct N_Battery
{
long Voltage;
};
```

O campo *Voltage* armazena a tensão da bateria em milivolts. Ao realizar a chamada a função *N_GetBattery*, as informações dos estados das baterias são atualizadas na estrutura N_GetBattery declarada na estrutura *N_RobotState*. Essa função é declarada da seguinte forma:

```
int N_GetBattery (long RobotID)
```

2.5 Fusão de Sensores

Robôs móveis não podem confiar somente no sistema odométrico (*dead-reckoning*) para determinar sua localização porque os erros odométricos são acumulados a todo tempo. Por essa razão, robôs móveis devem ser equipados com sensores externos que obtém informação do ambiente para ajudar o robô determinar sua localização mais precisamente.

Naira [18] utiliza dois tipos de informações sensoriais: a distância das imagens e a intensidade das imagens obtidas de um sensor laser. Pela medição da distância aos objetos, a distância da imagem fornece informação de posição. Por outro lado, a intensidade da imagem fornecida pelo sensor laser (equivalente a um sistema de visão monocular), pode ser usada para determinar fronteiras verticais, correspondendo a quinas, estruturas, etc., no qual fornece informação angular.

2.6 Odometria

O termo "*dead-reckoning*" refere-se a duas formas de localização relativa: Odometria e Navegação Inercial.

A Odometria é a determinação da localização do robô móvel por meio da observação e integração consecutiva do movimento das rodas do robô.

Os odômetros são instrumentos que medem a distância percorrida por veículos. No mundo da robótica os odômetro mais usados são os *encoders*. O conceito de odometria foi descrito por Vitruvius no século 1 ac. Leonardo da Vinci por volta do ano 1500 construiu um aparato que usava pedras para calcular a distância percorrida. O funcionamento do odômetro de Leonardo da Vinci está descrito na Figura 10:

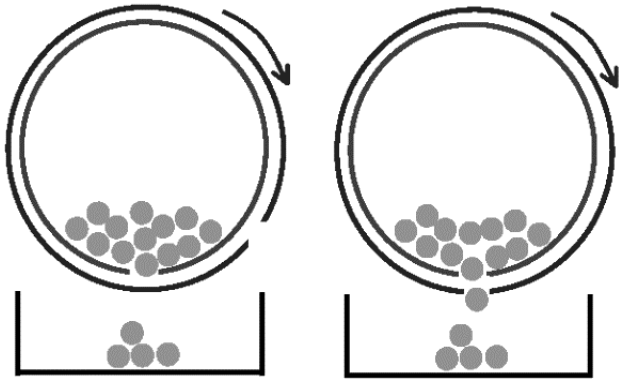


Figura 10 - Modelo do Odômetro de Leonardo da Vinci

O contentor fixo à estrutura do veículo está cheio com pedras e tem um orifício na parte inferior. A rotação de uma das rodas faz rodar um tambor em torno do contentor.O tambor tem um orifício do mesmo tamanho do existente no contentor. Depois de um certo número de rotações da roda, os dois orifícios coincidem e uma pedra cai, sendo recolhida numa caixa. O número de pedras recolhidas na caixa num intervalo de tempo permite estimar a distância percorrida.

2.6.1 Fontes de Erro na Odometria

A integração das informações sobre os movimentos recebidas pelos odômetros é incremental e, por isso, geram erros acumulados.

Os erros sistemáticos podem ser características do robô ou dos sensores. Esses erros contribuem mais fortemente com o erro total em solos regulares por que são fontes constantes de erros. Em outros tipos de solo, os erros não sistemáticos contribuem com maior intensidade.

Alguns exemplos de erros sistemáticos:

- Diferença dos diâmetros das rodas do robô;
- Desalinhamento das rodas do robô;
- Diâmetro das rodas diferente do diâmetro nominal fornecido;
- Incerteza sobre o ponto de contato da roda em solos irregulares;

Os erros não sistemáticos são causados principalmente por irregularidades no solo. Essas irregularidades podem causar os seguintes tipos principais de erros:

- Movimento sobre obstáculos inesperados no solo;
- Escorregamento das rodas;
- Grandes acelerações do veículo;
- Rotações rápidas;

Um exemplo de erro causado por escorregamento é mostrado pela Figura 11, onde o comando dado ao robô tem velocidades iguais nas duas rodas do robô.

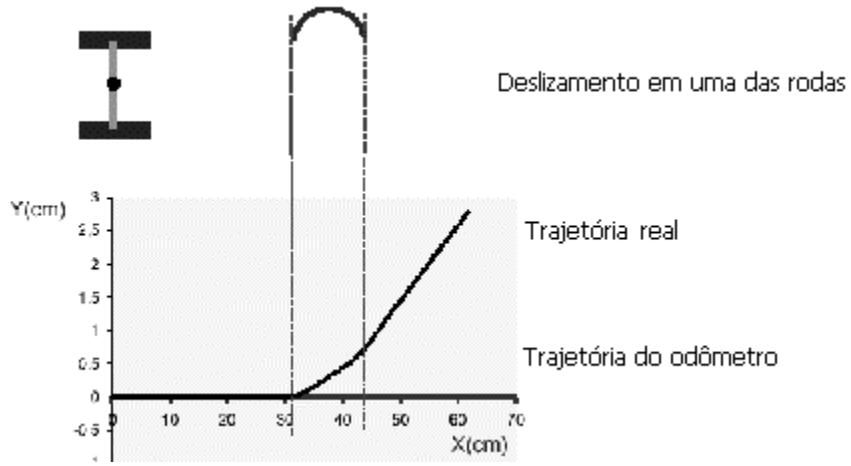


Figura 11 - Exemplo de erro não sistemático

2.7 Planejamento de Trajetória

Os mapas ambientais são utilizados para dois propósitos principais: gravar por onde o robô passou e para planejar caminhos para onde ele pode ir. A maioria dos trabalhos de planejamento de caminhos é derivada de trabalhos realizados no planejamento de trajetória de manipuladores robóticos [1]. No contexto da robótica móvel, o planejamento de trajetória é simplificado pela restrição dos movimentos do robô em três graus de liberdade em um plano.

Os requisitos de um planejador de trajetória para um robô móvel são:

- Encontrar um caminho através de um mapeamento do ambiente de forma que o robô possa segui-lo sem colidir com nenhum objeto;
- Manipular incertezas no modelo ambiental sensoriado e erros na execução da trajetória;
- Encontrar um caminho ótimo.

É importante observar que nem todos esses requisitos são aplicados em todas as situações, e, em algumas situações, são necessários requisitos adicionais.

2.7.1 Planejamento de Trajetória Automático

Para o planejamento de trajetória, o grafo de nodos, no qual representa um grafo de caminhos possíveis, é tratado como uma árvore com a raiz sendo o nodo inicial. Utilizando técnicas de Inteligência Artificial, um programa realiza uma busca na árvore para verificar se uma rota, a partir de um nodo inicial até o nodo de destino, existe. Essa rota é extraída como um sub-grafo.

Se vários caminhos existem, os sub-grafos são conectados para formar um grafo de múltiplos caminhos. O caminho de menor comprimento é encontrado utilizando-se programação dinâmica.

A maior parte dos planejadores de trajetória abstrai o espaço de busca em um grafo de caminhos possíveis. Esse grafo é então pesquisado para encontrar o menor caminho. Essa aproximação pode originar naturalmente no aprendizado do ambiente, onde o robô pode passar por vários caminhos para mapear o ambiente.

2.7.2 Algoritmo de Busca A*

Selecionar o melhor caminho de um grafo com vários caminhos possíveis, é um de vários problemas que utilizam técnicas de inteligência artificial de pesquisa. Algumas técnicas de busca são modificações da busca *branch and bound,* que procura sempre pelo sucessor de um estado cujo caminho possui o mínimo custo total a partir do estado inicial. A busca inicia-se expandindo o nodo raiz, no primeiro passo, para formar uma árvore de profundidade dois. O custo (comprimento) de cada caminho é calculado a partir da raiz até o sub-nó. Esses custos são comparados e o caminho de menor custo é escolhido. O nodo no fim desse caminho é expandido, em um passo, e o custo desse conjunto de novos caminhos é calculado. Os custo de todos os caminhos (incluindo o caminho antigo encontrado no primeiro passo) são comparados e o caminho de menor custo é escolhido. Esse processo se repete até que nodo objetivo seja alcançado.

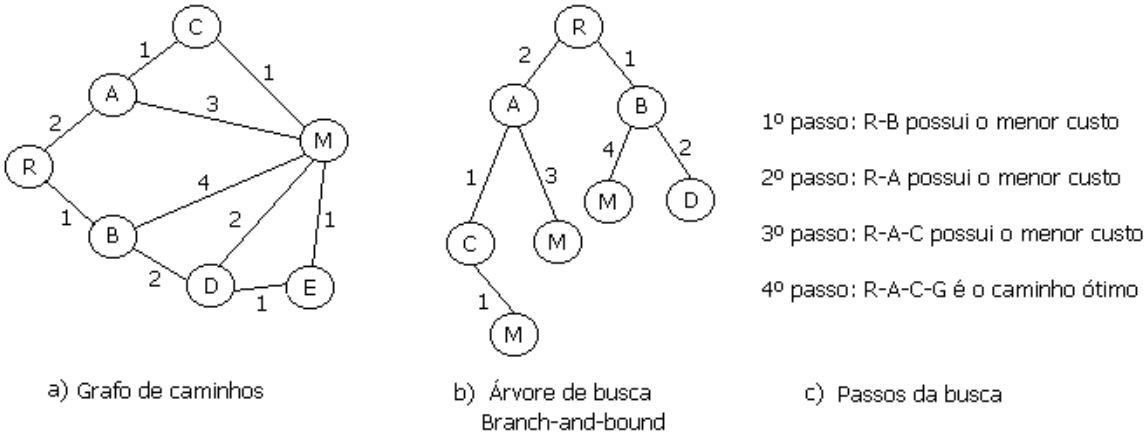


Figura 12 - Árvore de Busca

A busca A* é um refinamento da busca em árvore *branch and bound.* O espaço de busca é reduzido excluindo múltiplos caminhos para um determinado sub-nó, e deixando

somente o caminho de menor custo (Princípio da programação dinâmica). A escolha do caminho de menor custo para expansão em cada estágio é melhorada adicionando-se ao custo atual um custo estimado para os caminhos restantes. Se o custo estimado for sempre menor que o custo atual, a busca A* produzirá uma solução ótima. Uma função típica utilizada na estimativa desse custo no planejamento de trajetória é a distância de uma linha reta entre a posição representada pelo nodo atual e a posição objetivo.

Esse algoritmo é aplicável quando se dispõe de uma função heurística do custo, denotada por f() de acordo com a equação (1) [19]. Em muitas aplicações, o custo incorrido ao percorrer, a partir do nó inicial, de um certo caminho até o nó N, pode ser calculado através de uma função g(). Se for disponível uma função heurística h() que permite estimar o custo para percorrer a arvore de N até o nó objetivo então a função heurística de custo é:

$$f(n) = g(n) + h(n) \qquad (1)$$

No caso da navegação de um robô móvel, a distancia direta (sem considerar obstáculos) poderia ser considerada uma função heurística h() para avaliar a conveniência ou não de se tomar uma decisão sobre a direção a ser seguida.

O algoritmo de busca A* [20] está apresentado abaixo.

1. Utilize uma fila para armazenar um conjunto de caminhos candidatos.
2. Adicione na fila um caminho de custo zero do nó meta para nenhum lugar.
3. **Repita**

    Examine o primeiro caminho na fila

    Se esse caminho alcança o nó objetivo então sucesso

    Senão (continua a busca)

        Remova o primeiro caminho da fila

        Expanda o último nó desse caminho em um passo

        Calcule o custo desses novos caminhos

        Adicione esses novos caminhos na fila

        Ordene a fila na ordem crescente dos custos

        Se mais de um caminho leva a um sub-nó

            Então remova todos esses caminhos menos o de menor custo

    Até que a meta seja encontrada ou a fila se esvazie
4. Se a meta foi encontrada retorna sucesso e o caminho senão retorna meta não alcançada.

2.8 Transformação de Coordenadas

A posição de um objeto no espaço é descrita por coordenadas Cartesianas. No entanto, para descrever o movimento desse objeto, é necessário descrever um sistema de coordenadas em relação a outro sistema de coordenadas, normalmente fixado nesse objeto. Um novo sistema de coordenadas pode ser descrito com uma matriz de vetores 4x4 (3D): um vetor representando a translação da origem e os outros vetores representando as direções dos novos eixos.

Além do mais, precisamos descrever uma seqüência de rotações e translações necessárias para localizar novamente um sistema de coordenadas em relação ao outro. Essa tarefa pode ser feita utilizando-se uma matriz de transformação. Essa matriz é utilizada para representar translações e rotações porque uma seqüência de translações e rotações pode ser combinada para produzir mais facilmente uma re-localização dos sistemas de coordenadas com multiplicação de matrizes do que com adição de vetores.

Em um sistema bidimensional (2D), a localização de um ponto é descrita com vetor p[x,y], onde x e y são as coordenadas do ponto em relação ao sistema de referência global. Se esse ponto for multiplicado por uma matriz de transformação, será produzido um vetor p[x1,y1], onde x1 e y1 serão as novas coordenadas do ponto, conhecidas como coordenadas homogêneas.

A equação de transformação será:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\phi & -sin\phi & m \\ sin\phi & \cos\phi & n \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (2)$$

Onde m e n são as constantes de translações na direção x e y respectivamente e equações de rotações são descritas pelos senos e co-senos.

2.9 Sistema de referência

O sistema de referencia a ser utilizado pelo sistema de navegação do robô se constitui em uma definição bastante importante. Em primeiro lugar, deve-se fixar um referencial global, e a partir dessa referência, localizar o robô e os obstáculos presentes no ambiente.

No entanto, o robô possui o seu próprio sistema de referência, fixado no centro geométrico do robô, e os comandos enviados, como, por exemplo, os de movimentação, devem obedecer ao sistema de referencia do robô. Igualmente no caso das leituras dos

sensores ultra-sônicos e dos sensores infravermelhos, a distância ao obstáculo sentido é referente ao sistema de referência do robô.

Dessa forma se faz necessário uma transformação de coordenadas para que os obstáculos encontrados pelo sistema sensorial do robô sejam mapeados corretamente no ambiente. Isso pode ser feito descrevendo um sistema de coordenadas em relação a outro sistema de coordenas.

Além do mais, precisamos descrever uma seqüência de rotações e translações necessárias para localizar novamente um sistema de coordenadas em relação ao outro. Essa tarefa pode ser feita utilizando-se uma matriz de transformação.

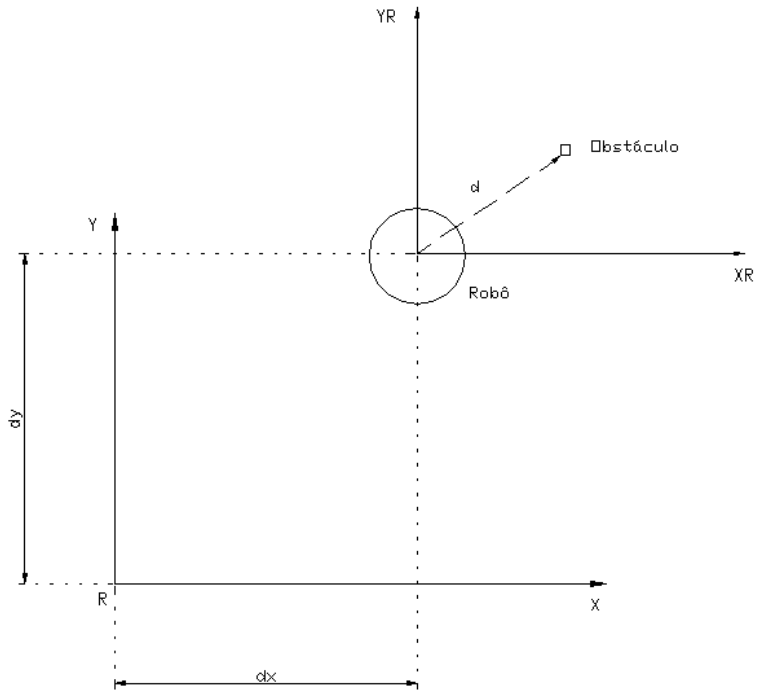


Figura 13 - Sistema de referência

Como pode ser visto na Figura 13 o robô possui seu próprio sistema de referência e o sistema de referência global foi fixado no ambiente, onde:

-R é o sistema de referência global;

-dx é a translação do eixo cartesiano X_R do robô em relação ao referencial global;

-dy é a translação do eixo cartesiano Y_R do robô em relação ao referencial global;

-d é a distância do obstáculo em relação ao referencial do robô;

-θ é a rotação do robô em relação sistema de referencia global.

Na Figura 13, θ é igual a zero, pois os eixos coordenados do sistema de referencia do robô estão paralelos aos eixos coordenados do sistema de referencia global.

Aplicando essas variáveis na matriz de transformação, as coordenadas do obstáculo em relação ao robô (x_d,y_d) podem ser localizadas em relação ao sistema de referencia global da seguinte forma:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -sin\theta & dx \\ sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} \quad (3)$$

Onde:

x e y são as coordenadas do obstáculo em relação ao sistema de referencia global do ambiente.

2.10 Semáforos

Semáforos são mecanismos IPC (*Interprocessing Comunnication*) utilizados para sincronizar o acesso a recursos compartilhados. Os semáforos são criados em *arrays*, onde cada elemento pode ser usado para controlar a execução do processo que realiza operações nesse *array* de elementos [23].

São utilizados para delimitar uma seção crítica, onde somente um processo executará de cada vez e também podem ser utilizados como uma maneira de estabelecer uma certa ordem na execução dos processos. Os semáforos são estruturas de dados sobre as quais são executadas operações indivisíveis.

Os passos para a utilização de semáforos são os seguintes:

- Criação de um conjunto de semáforos;
- Obtenção do identificador do semáforo;
- Execução de operações sobre os semáforos normalmente definidas pelo programador;
- Remoção do identificador do semáforo.

A primitiva de criação e obtenção de semáforos é a semget que possui a seguinte sintaxe:

int semget(key_t key, int nsems, int IPC_CREAT | flag)

O processo que chama esta primitiva cria um conjunto de semáforos definido pelo parâmetro nsems com a chave key e permissões de acesso em flag. O parâmetro IPC_CREAT

determina a criação do semáforo. Essa primitiva retorna o identificador do semáforo. Para a obtenção do semáforo a primitiva é exatamente a mesma com a diferença que o parâmetro IPC_CREAT deve ser omitido.

A operações sobre os semáforos são realizadas através da primitiva semop que possui a seguinte sintaxe:

int semop(int id, struct sembuf *sops, int nsops)

O parâmetro id é o identificador do semáforo e nsops são os números de operações executadas sobre os semáforos. Estas operações estão descritas na estrutura sembuf que possui campos como número do semáforo, tipo de operação e flags do semáforo. Dependendo do valor desses campos o processo ficará bloqueado ou não até que alguma operação no semáforo seja realizada.

Ao final da execução do processo que criou o semáforo, deve-se chamar a primitiva de remoção de semáforos semctl.

2.11 *Threads*

Na maioria dos sistemas operacionais cada processo (programa em execução) tem um espaço de endereçamento e uma única linha de controle(*Thread*). No entanto, existem situações nas quais é desejável a presença de varias linhas de controle compartilhando um único espaço de endereçamento, rodando de maneira quase paralela (paralelismo independente), como se elas representassem a ativação de dois processos separados, exceto pelo fato de compartilharem o mesmo espaço de endereçamento [24].

Um exemplo é o um servidor de arquivos que fica bloqueado ocasionalmente, aguardando pela conclusão de tarefas solicitadas ao disco. Caso o servidor possua mais de uma linha de controle, uma segunda linha pode ser autorizada a executar outra tarefa aumentando desse modo, a performance do sistema.

As linhas de controle, também chamadas de linhas de execução, foram inventadas para permitir a combinação do conceito de paralelismo com os conceitos de execução seqüencial e de bloqueio durante execução de chamadas de sistema.

Uma *thread* seria um fluxo de controle seqüencial isolado dentro de um programa. Permitem que um programa simples possa executar várias tarefas diferentes ao mesmo tempo, independentemente umas das outras.

2.12 *Java Native Interface*

A *Java Native Interface* (JNI) é a interface nativa de programação para a linguagem Java™. Essa interface permite que um programa executando dentro de uma Maquina Virtual Java trabalhe junto com bibliotecas e aplicações escritas em outras linguagens de programação como C, C++, assembly e outras. Dentro da interface JNI existe uma API que permite a invocação de uma Maquina Virtual a partir de um programa nativo. (*The Invocation API)*

A interface JNI pode ser necessária nos casos listados abaixo:

- Quando as bibliotecas padrões do Java não suportarem *features* utilizadas na aplicação que são dependentes de plataforma especifica;
- Quando já existe uma aplicação ou biblioteca escrita em outra linguagem e deseja-se fazer acesso a elas a partir de uma aplicação Java;
- Quando existem partes de tempo crítico em uma aplicação, e deseja-se otimizar essas partes utilizando uma linguagem de baixo nível.

A JNI permite que o código nativo utilize os objetos do Java da mesma forma como são utilizados nos códigos Java. Também permite que os códigos nativos executem chamadas para métodos escritos em Java.

A Figura 14 ilustra chamadas de métodos nativos originados de uma aplicação Java:

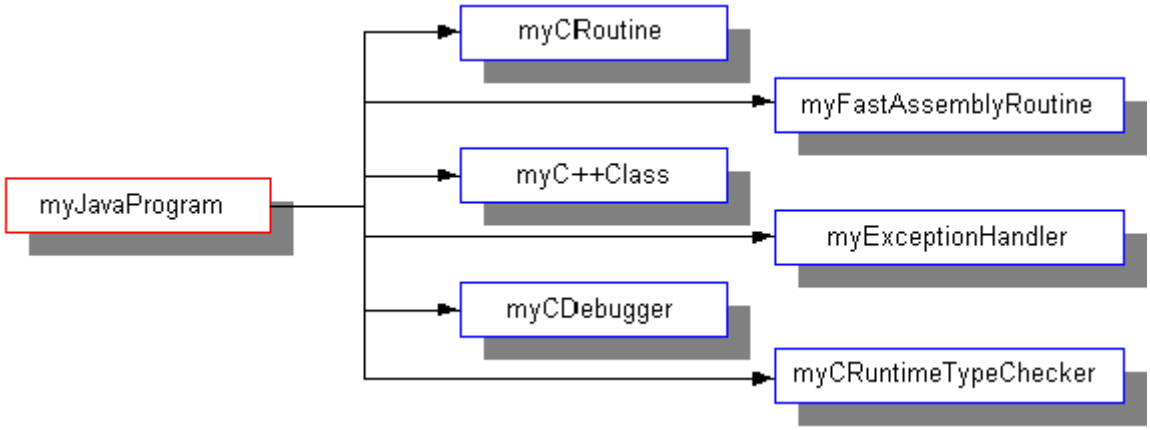


Figura 14 - Ilustração de JNI

A JNI representa o elo de ligação entre a linguagem Java e aplicações nativas.O diagrama abaixo mostra como a JNI liga a parte “C” de uma aplicação com a parte Java:

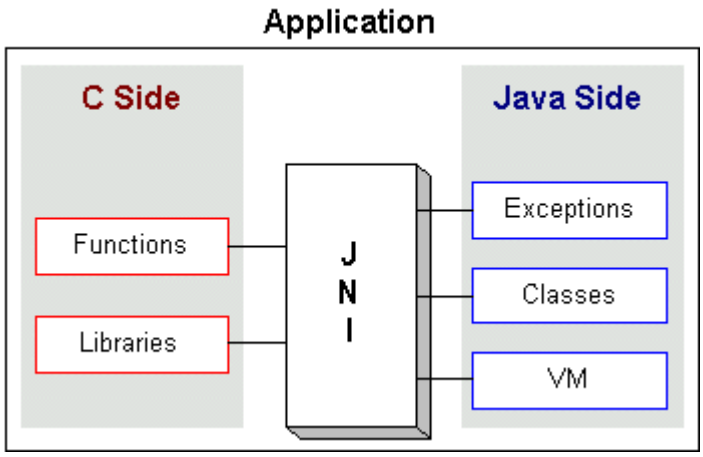


Figura 15 - Diagrama JNI

Na implementação do sistema desenvolvido nesse trabalho foram utilizados os seguintes métodos da interface JNI:

- NewIntArray() - Cria um array de inteiros na linguagem nativa;
- SetIntArrayRegion() – Grava um array de inteiros da linguagem nativa em um array de inteiros da linguagem Java.
- NewFloatArray() – Cria um array de float na linguagem nativa;
- SetFloatArrayRegion() – Grava um array de float da linguagem nativa em um array de float na linguagem java;
- GetStringUTFChars() – obtem uma String da linguagem Java e converte para o tipo string da linguagem nativa (const char* no caso de C/C++).

3 METODOLOGIA

De acordo com os objetivos propostos, nessa seção será apresentada a metodologia de projeto. A partir da definição dos requisitos e características do projeto, serão apresentados os métodos definidos de forma a guiar o desenvolvimento do trabalho.

3.1 Requisitos do Projeto

De uma forma geral, o projeto constitui-se na concepção e implementação de um sistema de navegação para o robô móvel NOMAD XR4000. O módulo de navegação deverá mapear o ambiente a partir das distâncias dos obstáculos. As distâncias dos obstáculos deverão ser obtidas por meio de sensores ultra-sônicos e infravermelhos instalados no robô.

Uma das primeiras decisões a serem tomadas é como o ambiente será representado. Essa representação do ambiente deve satisfazer as necessidades do módulo de planejamento de trajetória a ser implementado. Escolhida a representação do ambiente, deve-se definir a estrutura lógica de armazenamento dos dados e as dimensões do espaço a ser mapeado.

Os mapas do ambiente coletados e armazenados pelo sistema de navegação poderão ser utilizados para tarefas de movimentação do robô. Essa movimentação será comandada pelo usuário por meio de uma interface gráfica.

O usuário escolherá o destino do robô e o sistema de navegação será capaz de calcular o caminho a ser seguido pelo robô evitando obstáculos. Caso seja possível o robô alcançar o objetivo escolhido pelo usuário dessa interface gráfica, o sistema de navegação guiará o robô até o seu destino.

A movimentação do robô deverá ser feita paralelamente ao sensoriamento do ambiente. Dessa forma, o sistema de navegação será capaz de detectar obstáculos não previstos no mapa e poderá tomar alguma decisão evasiva ou calcular uma nova trajetória de acordo com os obstáculos não previstos.

A comunicação entre as partes integrantes desse sistema de navegação deve ser feita de forma rápida e segura. Deve-se estabelecer também um canal de comunicação entre o sistema de navegação e a interface com o usuário. Dessa forma, a definição da arquitetura do sistema se faz bastante importante.

3.2 Mapeamento

A representação espacial utilizada para fornecer o mapeamento do ambiente é baseada em grades de ocupação. Essa representação é utilizada principalmente por fornecer uma

maneira simplificada de fundir diferentes sensores e representar o espaço livre. O ambiente é dividido em células onde cada célula armazena o estado de ocupação da região do espaço correspondente.

O mapa do ambiente é formado pelo conjunto dessas células. Cada célula representa uma região quadrada do mundo real que armazena um valor inteiro que depende do estado de ocupação da célula. Os valores utilizados e armazenados nas células e os seus respectivos significados são:

- (-1): Região não-explorada do espaço. No inicio do mapeamento todas as células do mapa são inicializadas com o valor –1, o que indica que o mapa não possui nenhuma informação de ocupação.
- (0): Região livre do espaço. Significa que não há obstáculos. O conjunto de células livres poderá constituir uma possível trajetória do robô.
- (8): Tolerância: Essa informação é utilizada para evitar colisão do robô com obstáculos devido o diâmetro do robô e a dimensão da célula.
- (9): Região ocupada do espaço: Identifica os obstáculos presentes no ambiente.

O tamanho da célula da grade de ocupação deve ser escolhido levando-se em conta aspectos como custo computacional, precisão e limitação do mapeamento do ambiente real. Nesse trabalho a dimensão definida para cada célula é de 100 x 100 mm. A justificativa para o valor escolhido se resume em termos de precisão na representação do mundo real.

A estrutura de armazenamento dos mapas é um vetor unidimensional com 90000 posições equivalente a uma matriz quadrada de ordem 300. Desse modo um mapa pode representar uma região com dimensões máximas de 30 x 30 m ou 900m^2.

Os mapas são adquiridos a partir do processamento sensorial e armazenados em memória para que possam ser compartilhados pelos diferentes processos que fazem uso desses mapas tais como o módulo responsável pelo planejamento de trajetória e pelo módulo que utiliza essa representação para fornecer uma interface gráfica com o usuário.

3.3 Modelo do Sensor

Neste trabalho utilizamos um modelo sensorial baseado nas medidas fornecidas pelos sensores ultra-sônicos e infravermelhos. Os sensores ultra-sônicos, devido às suas faixas de operação, são utilizados neste trabalho para realizar medições de objetos com distância entre 150 e 3000 mm do robô. Os sensores infravermelhos, também devido à sua faixa de operação, são utilizados para detectar objetos distantes até 150 mm do robô.

Devido à disposição geométrica desses sensores no Nomad XR4000 (Figura 9) as medidas dos sonares e infravermelhos dispostos na mesma direção são comparadas. Inicialmente compara-se a medida entre cada par de sonares, superiores e inferiores. A medida do sonar com o maior valor é descartada. Em seguida, compara-se a medida entre cada par de infravermelhos. O mesmo procedimento feito para o sonar é realizado, com a diferença que a medida de energia do infravermelho é aproximada para um valor de distância em milímetros.

Em seguida, a medidas do sonar e a medida do infravermelho resultante são comparadas entre si e a que possuir o maior valor é descartada. Essa filtragem é necessária, pois um obstáculo pode ter dimensões pequenas e apenas os sensores inferiores o detectarão.

Uma questão bastante importante é como representar esse objeto já que o sonar nos retorna somente medidas pontuais da distância dos objetos.

Normalmente o objeto detectado pelo sonar é representado dentro de um arco de incerteza. O disparo de um sonar varre uma região representada por um cone centrado no eixo acústico do sonar. Foram observados em outros trabalhos diferentes ângulos para o cone de varredura do sonar. Pagac e Nebot [21] utilizam 24º para o ângulo do cone. Bilgç e Burhan [22] faziam esse ângulo variar entre 1,8º e 18º de acordo com a distancia retornada pelo sonar.

O ângulo definido em nosso trabalho foi de 15º. Dessa forma, de acordo com a disposição geométrica (Figura 9) desses sensores no robô, conseguimos varrer toda a região que circunda o robô (360º). A Figura 16 mostra essa representação da varredura do sonar.

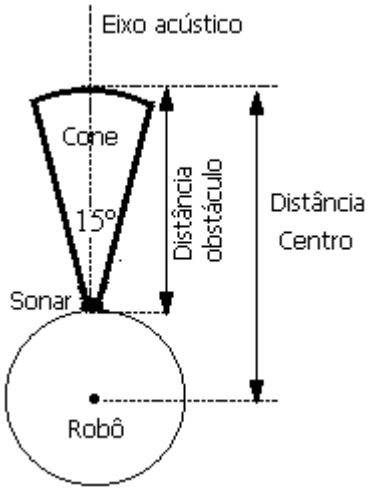


Figura 16 - Varredura do Sonar

O alcance dos sensores sonar foi limitado a 3000 mm, ou seja, as medidas que excederem esse valor são descartadas. No caso dos sensores infravermelhos utilizou-se uma aproximação dos valores de energia para valores em distância em milímetros. A máxima

energia recebida por esse tipo sensor (utilizado pelo Nomad) é 255, indicando um obstáculo muito próximo. À medida que o objeto se afasta do sensor infravermelho a energia recebida diminui. O alcance dos sensores infravermelhos foi limitado a 300 mm e aproximação utilizada para converter valores de energia em distância foi a da equação 4:

$$D = 300 - 1.22 * (energia - 10) \quad (4)$$

Deve-se observar que para valores de energia inferiores a 10 a medida do infravermelho é descartada. A varredura utilizada para o infravermelho é a mesma feita pelos sensores ultra-sônicos (Figura 16).

3.4 Atualização do Mapa

A construção e atualização dos mapas ambientais são realizadas através da fusão dos sensores de proximidade e dos sensores de localização. A informação de posição do robô no espaço é fornecida pelos odômetros. Após o processamento dessas informações sensoriais, realiza-se um processo de atualização das células do mapa. Essa atualização dependerá dos valores informados pelos sensores de proximidade (sonar e infravermelho) e pela informação de posição do robô fornecida pelos odômetros.

Através do modelo sensorial apresentado na seção 3.3 e da posição corrente do robô, modificam-se os valores das células para os estados descritos anteriormente na seção 3.2. As células ocupadas têm os seus valores atualizados para 09 enquanto as células desocupadas têm o seu valor atualizado para 0. Algumas células são atualizadas para conter o valor 08 indicando uma transição entre as células ocupadas e desocupadas. Essa tolerância será determinante na navegação do robô já que o diâmetro do mesmo é superior ao tamanho de cada célula.

3.5 Planejamento da Trajetória

O planejamento de trajetória é realizado utilizando-se os mapas construídos a partir de dados sensoriais provenientes do robô. O algoritmo de planejamento de trajetória é o A* descrito anteriormente na seção 2.7.2 da revisão bibliográfica.

É possível utilizar o algoritmo A* para recalcular um novo caminho à medida que novas informações se tornam disponíveis. Os principais motivos para a utilização do algoritmo A* neste trabalho são a necessidade de se calcular uma nova trajetória no caso de

um obstáculo não previsto, por exemplo, obstáculos dinâmicos e o planejamento de uma trajetória que resulte em um menor caminho a ser percorrido pelo robô resultando em uma menor demanda de energia.

Esse algoritmo calcula o custo de uma decisão sobre a direção a ser seguida e traça o caminho ótimo de um ponto inicial até uma meta. O custo de cada caminho é calculado através dos valores armazenados nas células do mapa e de uma função heurística. Essa heurística é a do menor caminho, ou seja, o caminho ótimo é o que possuir o menor valor.

O custo de se percorrer cada célula individualmente é o próprio valor armazenado na célula. Células ocupadas e não exploradas possuem o maior custo definido como 09. Células vazias possuem custo zero indicando que se pode seguir por essa célula.

Os movimentos são possíveis em todas as direções como mostrado na Figura 17.

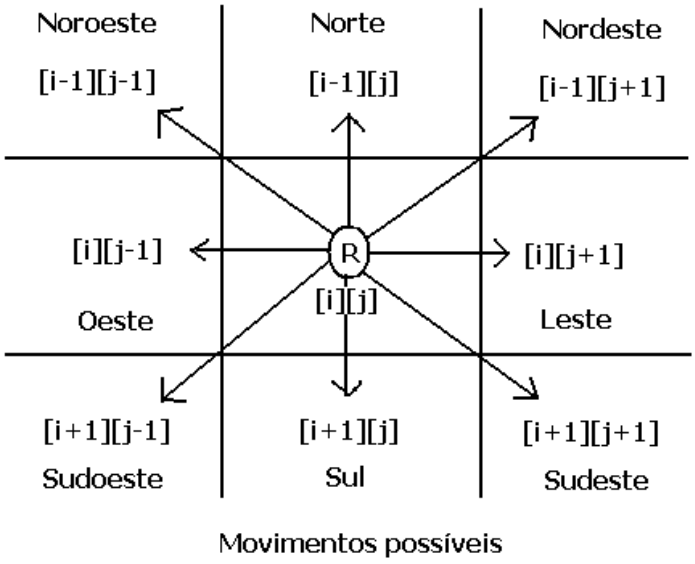


Figura 17 - Movimentos possíveis no planejamento de trajetória

3.6 Teleoperação via Internet

A teleoperação do sistema de navegação do robô poderá também ser realizada via WWW através de aplicativos applets desenvolvidos em linguagem Java. A comunicação entre o cliente e servidor será realizada através de canais de comunicação – sockets – utilizando o protocolo TCP. Essa conexão é realizada especificando o endereço IP da máquina remota servidora e a porta de comunicação a ser utilizada. Após essa requisição de conexão, a máquina servidora recebe o endereço da maquina cliente é está aberto um canal de comunicação baseado em fluxo de dados entre o servidor e o cliente.

3.7 Arquitetura da Aplicação

Nesse trabalho foi desenvolvida a aplicação NavMap. Essa aplicação é apresentada em uma arquitetura cliente-servidor descrita pela Figura 18. Existem duas bibliotecas compartilhas escritas em linguagem C++ e, uma interface amigável escrita em linguagem Java. Existe ainda um servidor TCP/IP para comunicação da interface com a aplicação, quando esta for executada via Web como Applet. Os detalhes de implementação serão descritos na seção 3.8.

A biblioteca **libNavigator** é o cérebro da aplicação NavMap. É nessa biblioteca que existe o tratamento dos dados coletados do robô para fazer o mapeamento do ambiente e o calculo da trajetória. A libNavigator tem acesso aos dados do Nomad por meio da biblioteca **libXR4000**.

A libXR4000 foi desenvolvida com o intuito de facilitar o entendimento dos comandos enviados e recebidos do robô utilizando a biblioteca fornecida pelo fabricante.

Abaixo segue a visão geral da arquitetura da aplicação NavMap:

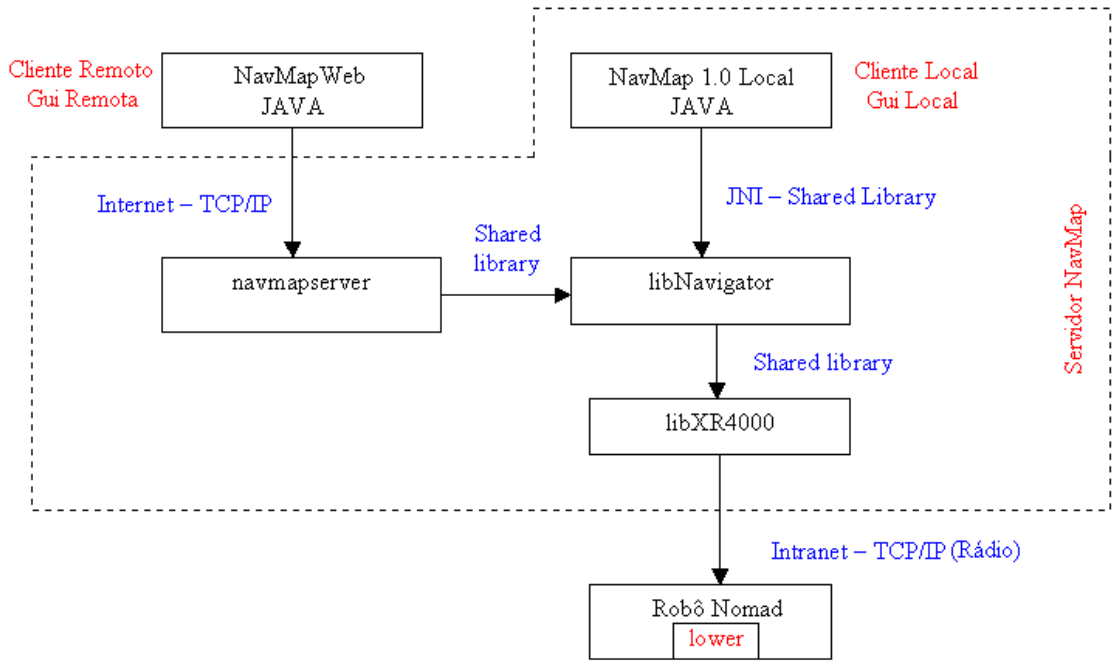


Figura 18 - Arquitetura do Sistema de Navegação

3.8 Implementação

Nesta seção serão descritos com mais detalhes cada componente da arquitetura. Todos os códigos fontes estarão disponibilizados no CDROM anexado ao projeto.

Inicialmente a interface de programação do robô (ver seção 2.4) foi modelada para que funções de comunicação, movimentação, utilização dos sensores e outras pudessem ser utilizadas com uma maior facilidade. A biblioteca dessas funções, a libXR4000, foi escrita utilizando-se a linguagem C++.

O controle do sistema de navegação é realizado pela biblioteca libNavigator descrita na seção 3.8.2. O controle dos sensores e atuadores, aquisição de dados dos sensores, mapeamento do ambiente, detecção de obstáculos dinâmicos e estáticos e cálculo da trajetória é realizado por essa biblioteca implementada. A libNavigator tem acesso aos dados do Nomad por meio da biblioteca **libXR4000**.

Começaremos com a descrição da biblioteca libXR4000, pois é ela que faz acesso direto aos métodos disponibilizados pela biblioteca do fabricante.

3.8.1 libXR4000

A biblioteca XR4000 faz um mapeamento das funções, variáveis e estruturas disponibilizadas na biblioteca do fabricante (Nclient_host) para um modelo orientado a objetos mais simples de ser manipulado dentro de outras aplicações. Essa biblioteca contém as seguintes classes: Nomad, Cinematic, SenBumper, SenIr, SenSonar como poder ser visto na Figura 19 abaixo:

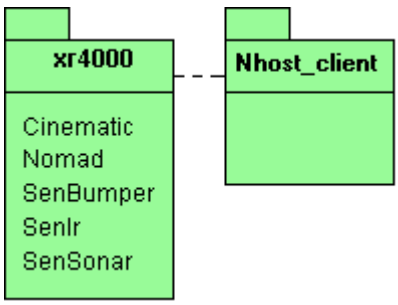


Figura 19 - Arquitetura libXR4000 e conexão com Nhost_client

3.8.1.1 Cinematic

A classe Cinematic é a classe responsável por acessar as variáveis e os métodos referentes ao posicionamento do robô. Abaixo está o diagrama UML da classe Cinematic:

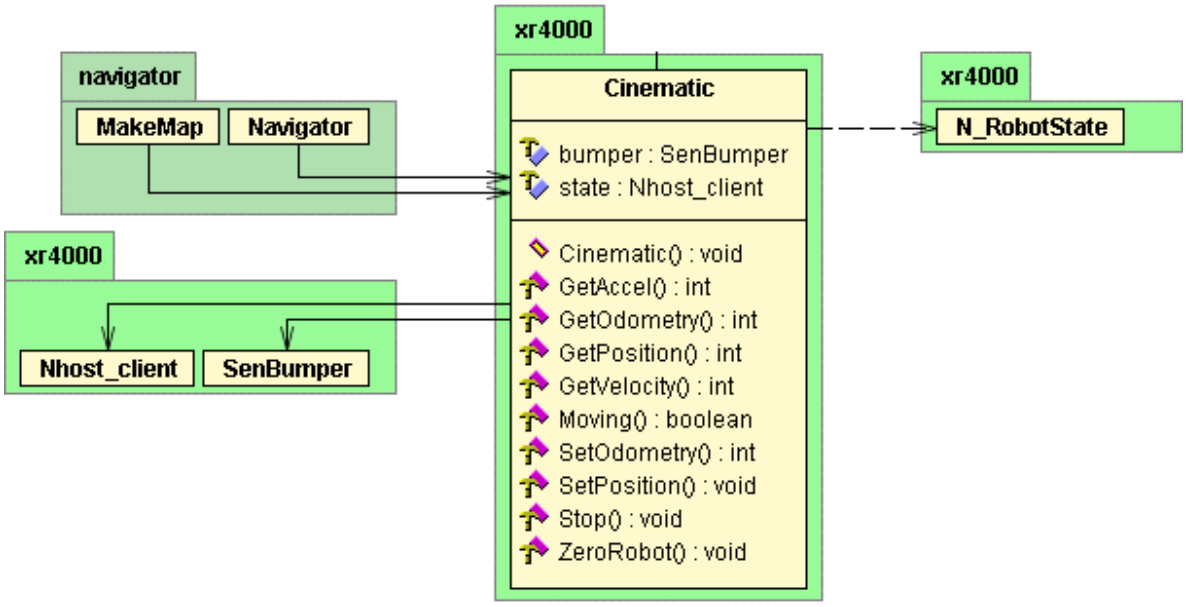


Figura 20 - Diagrama da classe Cinematic

- GetAccel() é usado para obter o valor da aceleração do robô em um determinado instante.
- GetOdometry é usado para obter o valor da odometria do robô;
- GetPosition é usado para obter a posição absoluta do robô a partir do ponto onde o nomad fora ligado;
- GetVelocity é usado para obter a velocidade do nomad durante a execução de um movimento;
- Moving() é usado para saber se o nomad está executando um movimento em um dado instante;
- SetOdometry() é usado para corrigir o valor da odometria;
- SetPosition() é usado para mandar o robô para uma posição especifica. A chamada a essa função faz com que o método Moving() retorne verdadeiro até que o nomad chegue na posição desejada.
- Stop() é usada para parar o nomad;
- ZeroRobot() é usada para zerar as variáveis de posição do nomad;

3.8.1.2 Nomad

A Classe Nomad é a classe que acessa os métodos referentes às utilidades do Nomad, mas que são essenciais ao uso do Nomad. Abaixo segue o diagrama UML dessa classe:

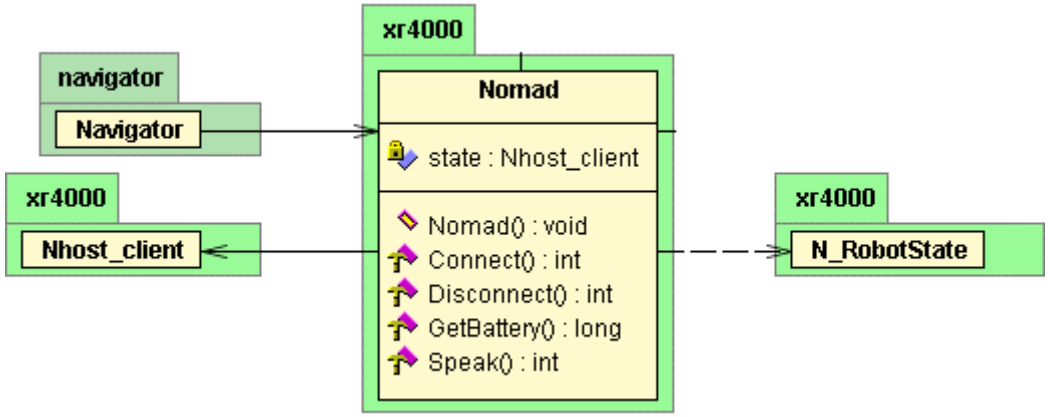


Figura 21 - Diagrama da classe Nomad

- Connect() – Método usado para abrir a conexão com o Nomad;
- Disconnect() – Método usado para fechar a conexão com o Nomad;
- GetBattery() – Método usado para obter as tensões das baterias do Nomad;
- Speak() – Método usado para enviar um texto ao sintetizador do voz do Nomad.

3.8.1.3 SenBumper

A classe SenBumper foi modelada para dar acesso aos dados dos sensores de colisão do robô Nomad. O método Read() dessa classe retorna o sensor onde houve uma colisão quando ocorre uma. Quando não ocorre nenhuma colisão o método retorna -1. Abaixo segue o diagrama UML dessa classe:

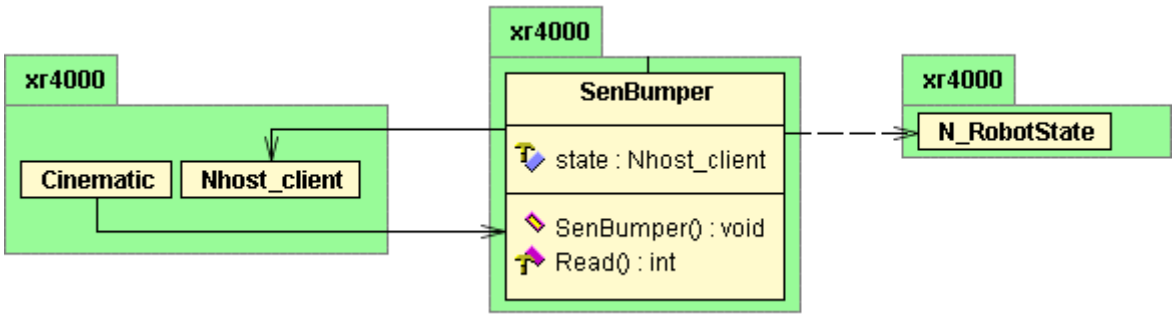


Figura 22 - Diagrama da classe SenBumper

3.8.1.4 SenIr

A classe SenIr é a classe que configura e lê os dados dos sensores infra-vermelhos do robô Nomad. Abaixo segue o diagrama UML dessa classe:

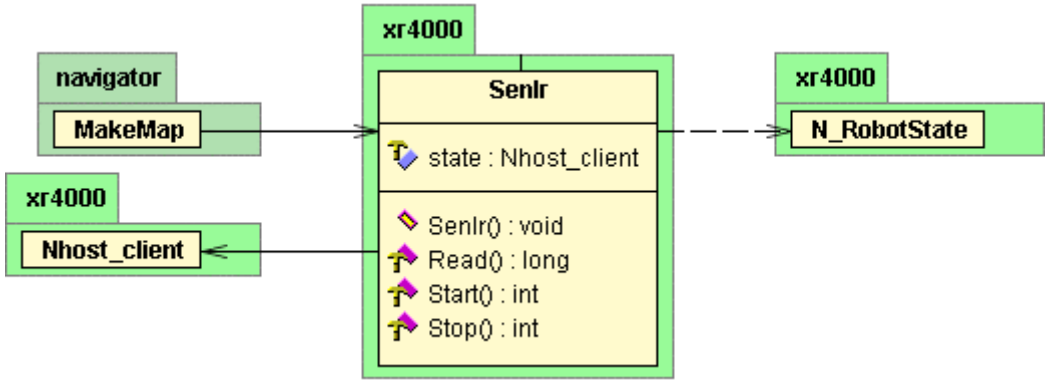


Figura 23 - Diagrama da classe SenIr

- Read() – Método utilizado para obter a leitura dos sensores infra-vermelhos. Retorna a distancia do obstáculo detectado em milímetros;
- Start() – Método utilizado para configurar e iniciar as medidas nos sensores infra-vermelhos;
- Stop() – Método utilizado para desligar os sensores infra-vermelho.

3.8.1.5 SenSonar

A classe SenSonar é a classe responsável pela configuração e leitura dos sensores sonares do Nomad. Abaixo segue o diagrama da classe SenSonar.

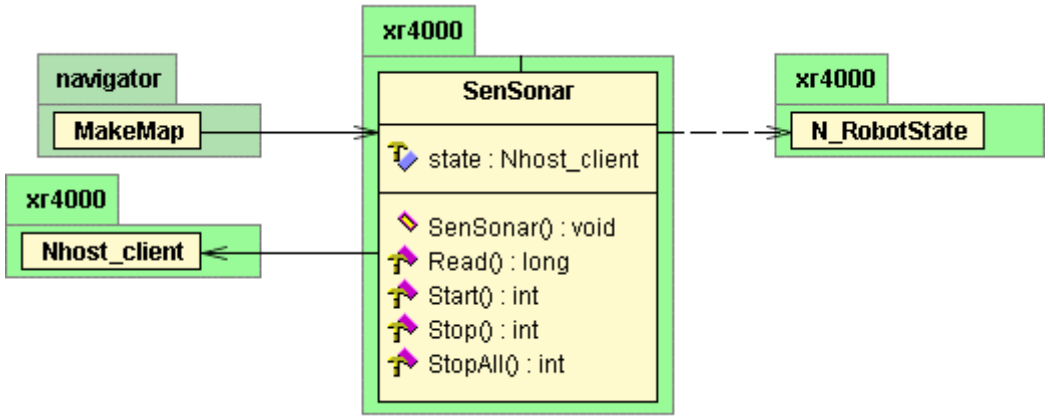


Figura 24 - Diagrama da classe SenSonar

- Read() – Método utilizado para obter as leituras dos sensores. Retorna a distancia do obstáculo detectado em milímetros;
- Start() – Método utilizado para fazer as configurações de disparo para os sensores sonares e iniciar coleta de dados;
- Stop() – Método utilizado para parar determinado sensor sonar;

- StopAll() – Método utilizado para parar todos os sensores sonares;

3.8.2 libNavigator

A biblioteca libNavigator é a biblioteca que contêm a implementação das funcionalidades especificadas no início do projeto: navegação por mapeamento. Ela faz uso da biblioteca libXR4000 para acessar e controlar o robô Nomad. Abaixo pode ser observado o diagrama de todas as classes da biblioteca libNavigator e a ligação com a biblioteca libXR4000.

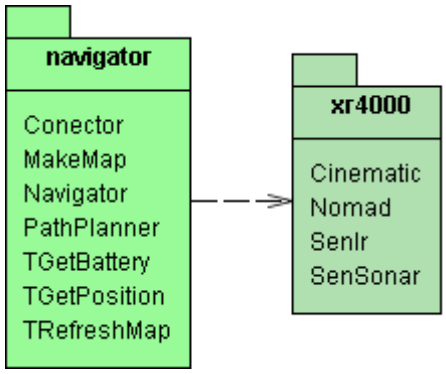


Figura 25 - Arquitetura da libNavigator e link para libXR4000

3.8.2.1 Navigator

A classe Navigator é responsável por integralizar os módulos do sistema. Todas as chamadas feitas pela interface através da classe Conector chegam ao objeto Navigator. Esse, por sua vez, executa o pedido feito.

Para que o objeto Navigator fosse capaz de controlar ações simultâneas foram inseridos nessa classe alguns objetos *Threads,* como a TgetPosition, TgetBattery, TrefreshMap que fazem a atualização da posição, status das baterias e mapa, respectivamente.

Devido ao problema da biblioteca do fabricante não dar suporte as *Threads*, foi encontrado uma dificuldade ao fazer o controle dessas ações. Para resolver esse problema foi utilizado o conceito de semáforos para habilitar apenas uma ação de acesso à biblioteca do fabricante de cada vez.

Abaixo segue o diagrama UML dessa classe:

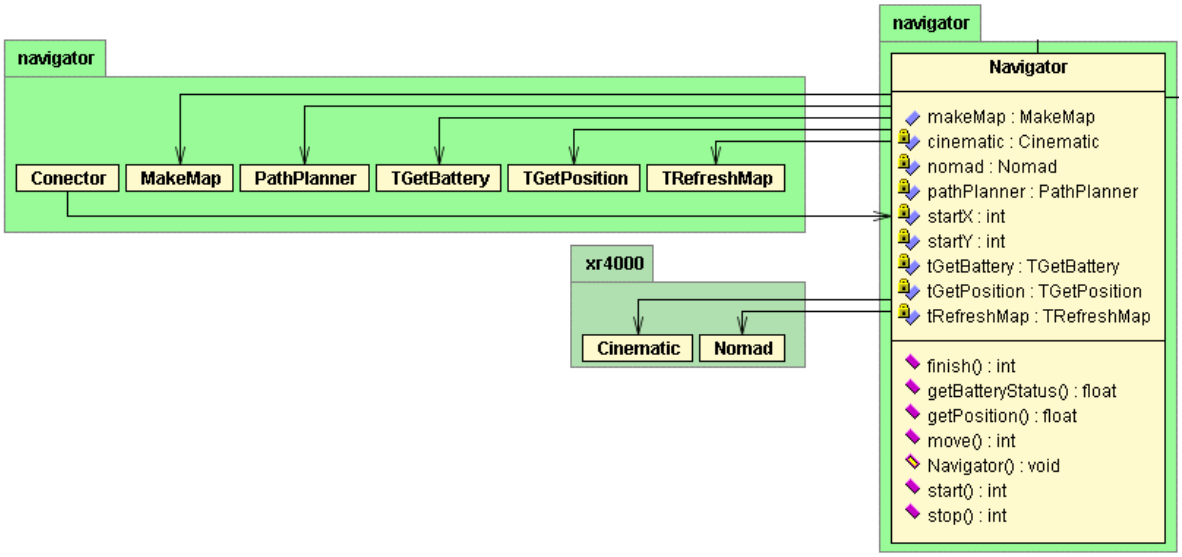


Figura 26 - Diagrama da classe Navigator

- finish() – método usado para finalizar as operações com o Nomad. Esse método finaliza todas as *threads* que estão sendo executadas e libera toda memória desnecessária alocada previamente;
- getBatteryStatus – método utilizado para obter os status das baterias do Nomad. Os status das baterias são atualizados na *thread* TgetBattery que a cada 5 segundos atualiza a estrutura;
- getPosition() – método utilizado para obter a posição atual do robô. A posição atual é atualiza pela *thread* TgetPosition que a cada 500 mili-segundos atualiza as posições na memória da aplicação;
- move() – método utilizado para fazer a movimentação do Nomad. Esse método recebe as coordenadas de origem e destino e as envia para o objeto PathPlanner que calculará a trajetória ótima que deverá ser seguida. Após o calculo da trajetória, o método envia os pontos calculados um a um para o Nomad executar o movimento. Durante o movimento é verificado se existem obstáculos dinâmicos nos arredores do robô, e de acordo com as tolerâncias definidas, ele decide se pode continuar na trajetória, se deve parar ou se deve recalcular outra trajetória até o ponto de destino;
- start() – método responsável por abrir a conexão e iniciar todas as *threads* de atualização de dados a serem coletados;
- stop() – método responsável por fechar a conexão e finalizar todas as *threads* de atualização
- speak() – envia um texto ao sintetizador de voz do robô.

3.8.2.2 MakeMap

O objeto da classe MakeMap é responsável por realizar todas as manipulações necessárias nos mapas, tais como:

- Criar novo mapa;
- Salvar mapa coletado;
- Interpretar os dados recebidos dos sensores e preencher o mapa com esses dados.

Abaixo segue o diagrama UML dessa classe:

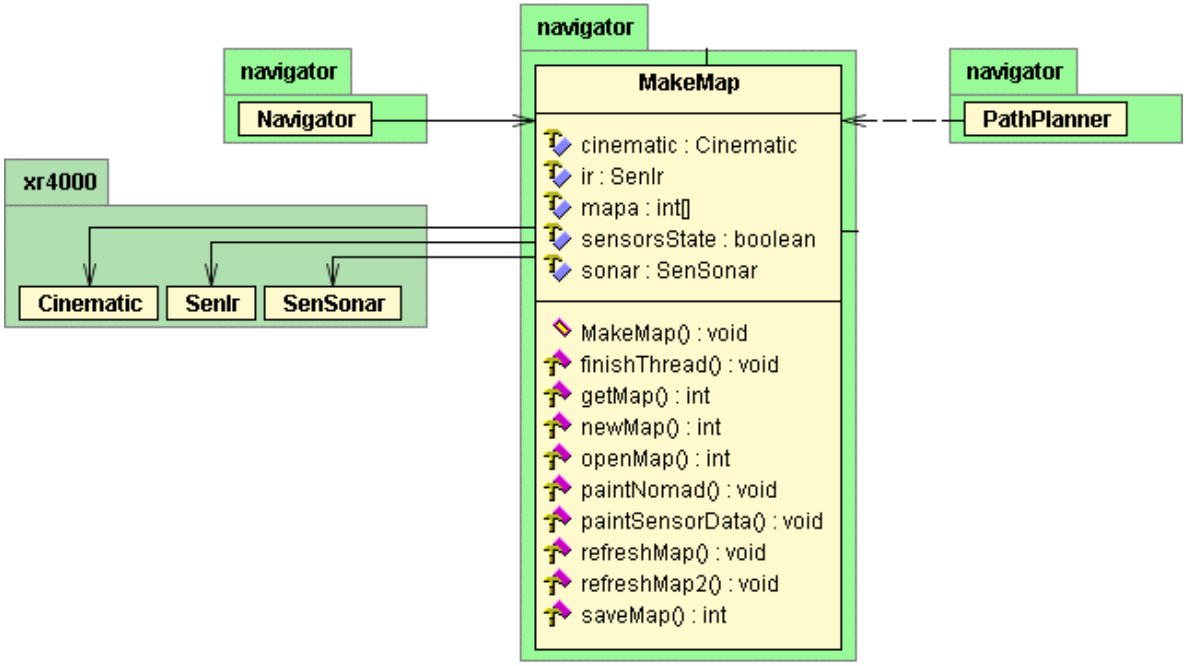


Figura 27 - Diagrama da Classe MakeMap

- finishThread() – método que é chamado pela *thread* de atualização de mapa. Esse método é utilizado para liberar os recursos consumidos pelo objeto da classe MakeMap durante seu tempo de vida;
- getMap() – método que retorna a matriz que representa o grid de ocupação do ambiente mapeado;
- newMap() – método que inicializa uma matriz do grid de ocupação com o valor de "Célula Desconhecida" em todas as posições dessa matriz;
- openMap() – método que atualiza os valores da matriz com os valores lidos através de um arquivo;

- paintNomad() – método que era utilizado anteriormente para representar o robô no mapa. Esse método não está mais sendo utilizado nesse projeto.
- paintSensorData() – método que interpreta os dados coletados dos sensores e os representa na matriz do grid de ocupação;
- refreshMap e refreshMap2 – métodos que atualizam os dados coletados dos sensores. Esses métodos são chamados pela *thread* TrefreshMap e pelo método move() do objeto Navigator;
- saveMap() – método que salva os dados da matriz em um arquivo especificado pelo usuário;

3.8.2.3 PathPlanner

O objeto dessa classe é responsável pelo cálculo das trajetórias que o robô Nomad deverá seguir de acordo com as posições de partida e de destino escolhidas pelo usuário na interface e pelo mapa do momento em que foi requisitado o calculo da trajetória. Essa classe utiliza o algoritmo de inteligência artificial A* para determinar a trajetória ótima.

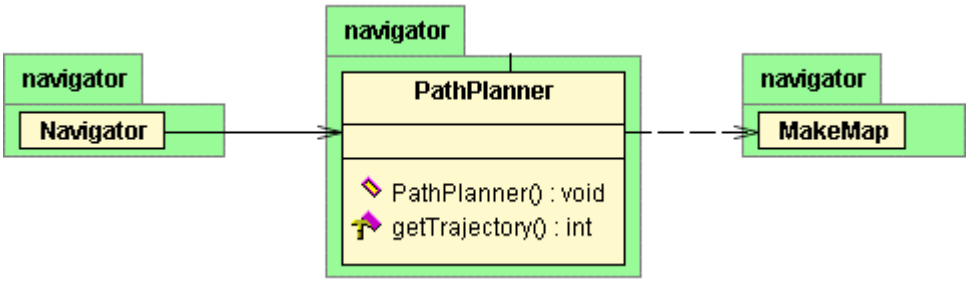


Figura 28 - Diagrama da classe PathPlanner

- getTrajetoria() – Após o calculo da trajetória é feito uma normalização dos pontos para que a trajetória seja o mais suave possível. Depois disso esse método retorna uma lista com todos os pontos normalizados para que o método move() execute o movimento;

3.8.2.4 Conector

O objeto conector é o responsável por fazer a comunicação entre a Interface do Usuário e a biblioteca libNavigator. Esse objeto utiliza a tecnologia JNI que permite que aplicações em Java façam chamadas para bibliotecas nativas. Os métodos dessa classe apenas mapeiam as chamadas da interface para chamadas da biblioteca libNavigator. Abaixo segue o diagrama UML dessa classe.

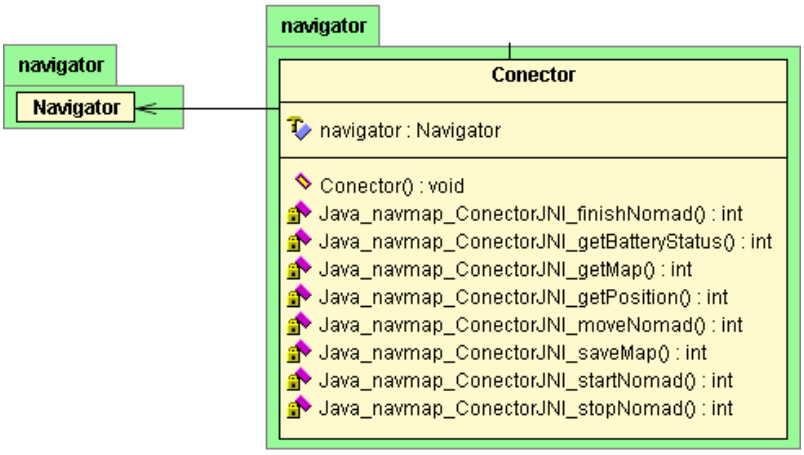


Figura 29 - Diagrama da classe Conector

3.8.3 Servidor para conexão WEB

Um servidor para conexão WEB foi implementado para que a aplicação NavMap fosse portada para ambiente WEB. Esse servidor é um programa que faz um mapeamento, da mesma forma que o Conector, entre os pedidos que os usuários fazem por meio da interface e os métodos da libNavigator. A principal diferença entre o Conector e o servidor para conexão WEB, denominada navmapserver, é que este último recebe os dados por uma conexão TCP/IP antes de fazer o pedido a biblioteca libNavigator e o Conector faz o pedido direto via JNI.

O servidor navmapserver está configurado no serviço INETD. O INETD é um processo responsável pelas conexões realizadas através de canais de comunicação nos sistemas UNIX. Ao receber uma conexão de rede para um determinado canal de comunicação, o INETD direciona a comunicação para o programa servidor configurado para receber conexões desse canal de comunicação.

A configuração do INETD para o servidor navmapserver é a seguinte:

- Serviço: navmapserver (listado no arquivo /etc/services)
- Canal: stream
- Protocolo: TCP
- Espera: nowait
- Usuário: nomad
- Localizacao: /home/nomad/NavMap/server/navmapserver
- Argumentos: navmapserver

Para implementar as configurações acima, o arquivo "/etc/inetd.conf" deve ser acrescido da linha abaixo:

navmapserver stream tcp nowait nomad /home/nomad/NavMap/server/navmapserver navmapserver

O servidor navmapserver está listado no arquivo "/etc/services", onde é definido a porta (8089) e o protocolo que o servidor atende. A configuração do arquivo "/etc/services" é a seguinte:

navmapserver 8089/TCP

O processo navmapserver é disparado quando uma aplicação NavMapWeb cliente é iniciada de algum lugar via internet. Logo que é disparado, o servidor abre um novo canal de comunicação por meio das portas 8091, 8092 e 8093. É por essas portas que toda a comunicação será feita.

Também foi implementado um outro servidor para conexão WEB, para o applet de emergência que é responsável pela parada imediata do robô. Quando o applet de emergência (AppletEmergencia) abre uma conexão com o servidor, este último dispara o comando de parada imediata para o robô Nomad.

As configurações do INETD e do SERVICES da máquina servidora para o servidor de emergência são semelhantes às configurações do servidor navmapserver e são descritas abaixo:

INETD:

emergencyserver stream tcp nowait nomad /home/nomad/NavMap/server/navmapserver emergencyserver

SERVICES:

emergencyserver 8090/TCP

3.8.4 Interface com Usuário

A interface com o usuário foi desenvolvida em linguagem Java. Toda a documentação referente à arquitetura da aplicação encontra em mídia magnética, CDROM anexado, na forma de HTML formatado com os padrões JAVA de documentação.

Os diagramas de classe podem ser encontrados no apêndice B no final desse documento.

4 RESULTADOS ALCANÇADOS

Neste capítulo serão apresentados os resultados encontrados durante a execução deste trabalho. Também serão apresentadas as análises dos resultados, as dificuldades encontradas e as soluções implementadas.

4.1 Simulação do Planejamento de Trajetória

Pode-se observar o comportamento simulado do algoritmo A* através da Figura 30. Essa simulação foi realizada utilizando-se um mapa imaginário. As células não ocupadas são as células em branco. Os obstáculos estão representados nas células em preto. A letra R indica a posição inicial do robô e M a meta a ser alcançada. O caminho encontrado pelo algoritmo A* pode ser observado pelas células marcadas com a cor cinza.

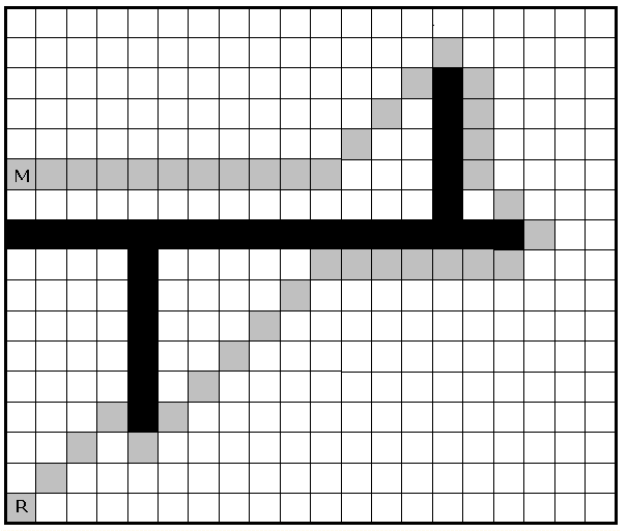


Figura 30 - Resultado da simulação do algoritmo A*

Nesse caso simulado a dimensão de cada célula seria o exatamente o diâmetro do robô. Entretanto, na realidade o robô possui um diâmetro (de aproximadamente 600 mm) 06 vezes maior que o comprimento de cada célula. Dessa forma o robô inevitavelmente sofreria colisão com os obstáculos.

Uma solução para esse problema foi a inserção de uma tolerância ao redor dos obstáculos. As células vizinhas das células ocupadas tiveram os seus valores atualizados para 08 funcionando como uma tolerância e impedindo que o robô chegasse muito próximo aos obstáculos.

Os resultados dessa simulação podem ser visto na Figura 31.

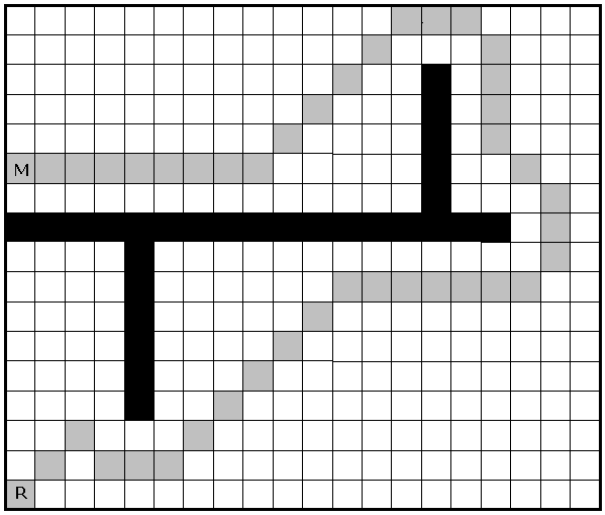


Figura 31 - Resultado da simulação do algoritmo A* com a tolerância

Como pode ser observado o caminho percorrido pelo robô não passa tão próximo dos obstáculos evitando o problema de colisão. Neste trabalho utilizou-se uma tolerância de 500 mm. Este valor é equivalente ao raio do robô (300 mm) mais uma distância de segurança igual a 200 mm.

4.2 Sistema de Navegação

Foi implementado um sistema de navegação seguindo a arquitetura proposta na Figura 18. Nesta seção serão apresentados os principais resultados alcançados.

4.2.1 Análise da Implementação

Um dos grandes problemas encontrados na implementação da arquitetura proposta foi a sincronização entre os processos. Devido à falta de informações sobre a biblioteca *Nclient_host* do fabricante do robô, encontrou-se bastante dificuldade para conseguir um bom funcionamento da aplicação sem paradas indesejáveis.

O problema acontecia quando uma determinada tarefa (*thread*) começava a comunicação com o robô e devido ao escalonamento de processos realizado pelo sistema operacional, uma segunda tarefa (*thread*) iniciava a comunicação com o robô sem que a primeira tarefa terminasse a sua execução. Nesse momento, a execução da primeira tarefa era travada o que originava um erro geral de toda aplicação.

Uma solução encontrada para esse problema foi a utilização de semáforos (ver seção 2.10). Antes de sua utilização, todas as regiões críticas foram identificadas. A região crítica é um trecho de código no qual, o processo começando a sua execução dentro desse trecho de código, nenhum outro processo pode interromper a sua execução.

Com o semáforo, uma vez um processo executando dentro de sua região crítica, nenhum outro processo poderá interrompe-lo. Caso tente interromper, esse processo ficará bloqueado esperando a liberação do semáforo.

Após essa sincronização esses problemas foram solucionados. Entretanto, outros problemas apareceram. A aplicação funcionava perfeitamente por um determinado período de tempo e sem causas aparentes parava de funcionar. O tempo de funcionamento era diferente para cada vez que a aplicação era colocada em execução.

Após algumas tentativas de solucionar o problema, observou-se que a aplicação estava recebendo um sinal (interrupção de software) que impedia uma determinada tarefa de pegar o semáforo. Essa tarefa então entrava na região critica, o que ocasionava o erro de sincronização descrito anteriormente.

Esse problema foi solucionado impedindo que a tarefa entrasse na região crítica caso não conseguisse pegar o semáforo. Somente após a obtenção do semáforo a tarefa poderia continuar a sua execução. A partir desse ponto, a aplicação passou a funcionar muito bem se tornando bastante robusta.

Outro problema encontrado no desenvolvimento da aplicação foi a comunicação TCP/IP entre o servidor e o cliente quando a aplicação era executada pela WEB. Uma solução encontrada foi criar novos canais de comunicações TCP/IP. O servidor navmapserver, após ser disparado através da porta 8089, abre novas conexões com o cliente por outras portas TCP (8091) e UDP (8092). O cliente abre uma outra porta UDP (8093). Com esses novos canais de comunicação foi possível que o servidor recebesse comandos do cliente e enviasse as repostas quando necessárias sem perdas de informações. Com essa configuração, o servidor utiliza a porta 8089 apenas para ser disparado através da WEB e utiliza as outras portas (8091 e 8092) para enviar e receber as informações do cliente. Mesmo com essa solução, devido a quantidade de informações que fluem pelos canais de comunicação ser muito grande, foram limitados os tempos de atualizações dessas informações no cliente. A aplicação cliente, quando executada via WEB, sofre quatro vezes menos atualizações que quando executada de forma local.

4.2.2 Interface Gráfica

Existem duas aplicações diferentes, porém que utilizam as mesmas bibliotecas descritas na seção 3.7. A primeira é a aplicação NavMap que roda em um servidor local. A segunda aplicação é a NavMapWeb que pode ser acessada via WEB na forma de applet (ver seção 3.9). A principal diferença entres as duas aplicações citadas acima é que a aplicação local (NavMap) mostra uma janela própria, enquanto que a aplicação NavMapWeb é mostrada dentro de uma janela de um browser que suporte applets Java.

A interface gráfica implementada pode ser vista na Figura 32.

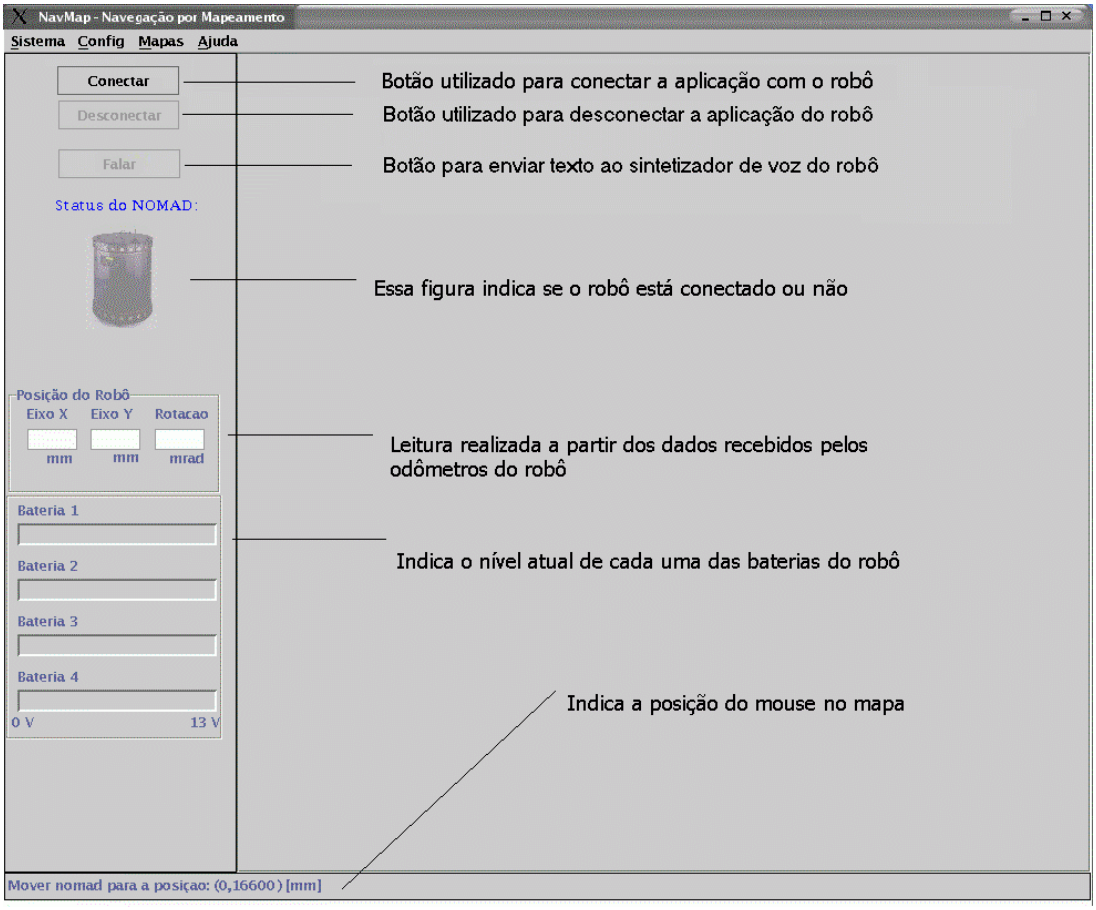


Figura 32 - Interface gráfica da aplicação NavMap

A Figura 32 mostra a tela inicial da aplicação. Inicialmente é necessário abrir uma conexão com o robô. Essa conexão pode ser feita pelo botão conectar ou através do menu Sistema. A foto do robô Nomad XR4000 indica se aplicação está conectada com o robô ou não. A foto do robô apagada indica que a aplicação não está conectada com o robô. Já a foto com um destaque maior indica que a aplicação está conectada (Figura 34).

Em seguida, são mostradas algumas informações de posição do robô coletadas a partir da leitura do odômetro do robô e da posição do robô no mapa. Informações como o nível das baterias do robô e a posição do mouse sobre o mapa também podem ser visualizados.

Antes de a conexão ser efetivada, é necessário que o usuário forneça algumas informações. A posição inicial do robô, em relação ao referencial global posicionado no canto inferior esquerdo do mapa, deve ser passada nessa fase para que o sistema de navegação possa localizar o robô dentro do mapa.

Outro parâmetro a ser passado pelo usuário é o mapa do ambiente. Um mapa existente pode ser aberto por meio da tela de conexão (Figura 33) escolhendo o arquivo referente ao mapa com a extensão.map. No caso da omissão desse parâmetro, a aplicação entende que o usuário deseja obter um mapeamento do ambiente. Dessa forma inicia-se o mapeamento a partir de um mapa totalmente desconhecido.

A Figura 33 mostra a tela de conexão com o robô.

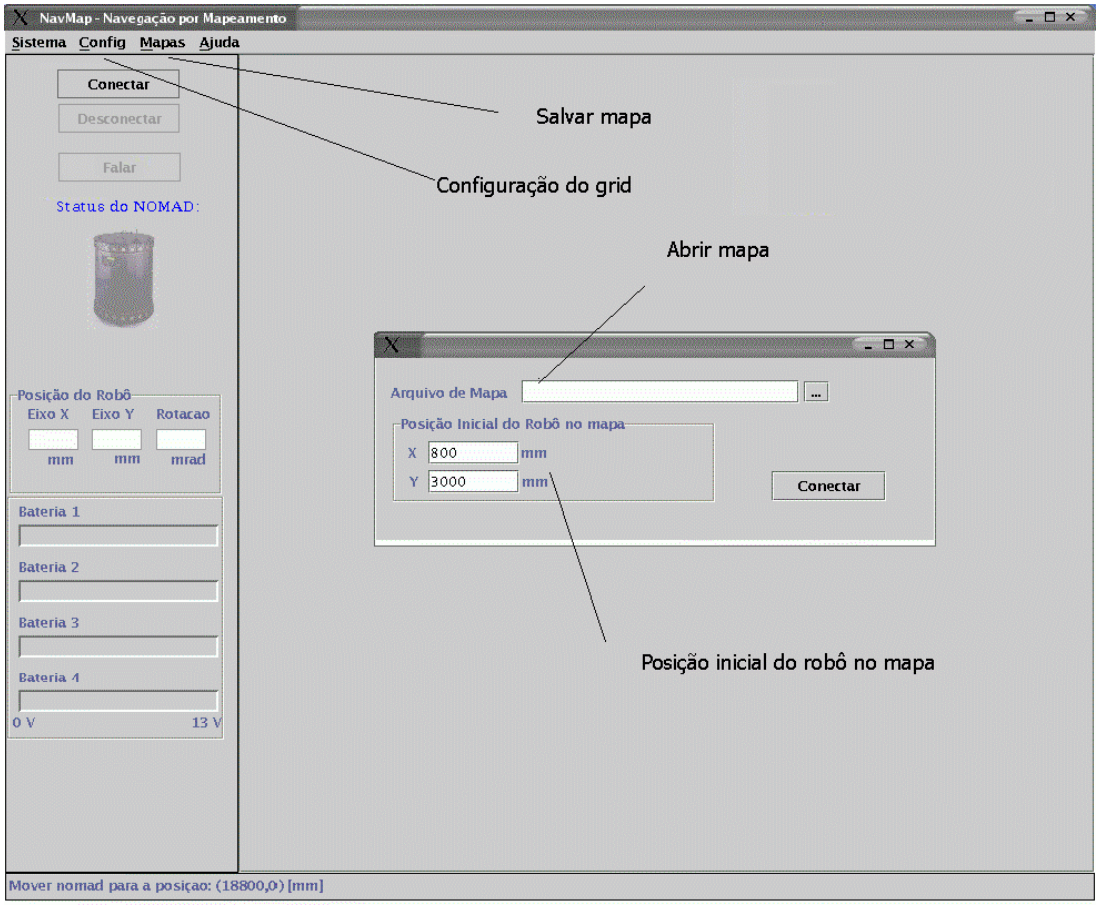


Figura 33 - Interface de conexão

Caso a conexão seja realizada com sucesso, aplicação está apta a se comunicar com o robô. Existem várias *threads* tanto em linguagem C/C++ (três) quanto em Java (três). Estas

threads ficam executando paralelamente, cada uma encarregada de executar uma determinada tarefa.

No lado Java, essas *threads* são encarregadas de atualizar na tela, cada uma com a sua devida taxa de atualização, o mapa, as informações de posição do robô e o estado das baterias.

No lado "C", as *threads* são encarregadas de atualizar a área de memória onde são armazenados o mapa, as informações de posição do robô e o estado das baterias.

A Figura 34 mostra a aplicação NavMap em funcionamento.

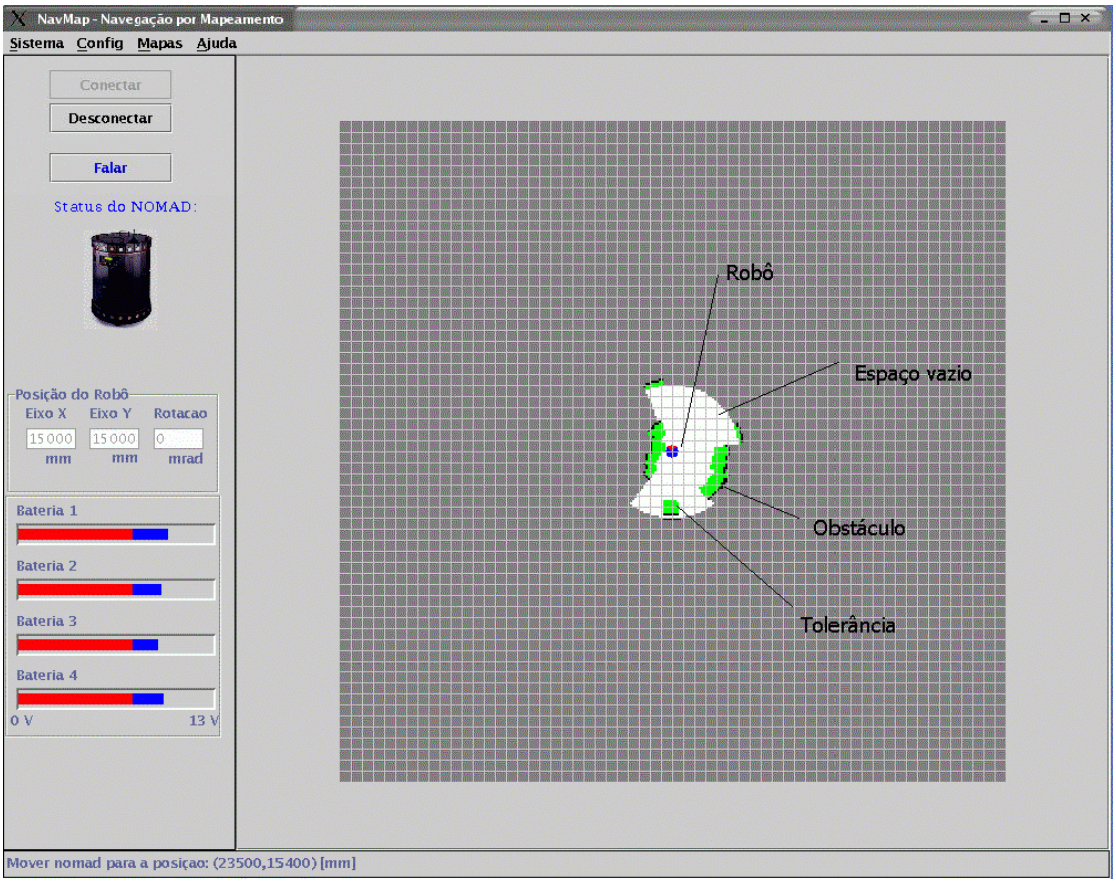


Figura 34 - Aplicação em funcionamento

Pode-se observar pela Figura 34 a aplicação NavMap em funcionamento. O mapa se localiza na parte direita da aplicação. A grade pode ser ajustada de acordo com a preferência do usuário através do menu Config (Figura 33). Ela serve para se ter uma noção da distância dos obstáculos ao robô.

As células em cinza representam a região do espaço que ainda não foi mapeada, ou seja, a região desconhecida. As células em preto representam a região do espaço onde foram detectados obstáculos. A região em branco representa o espaço não ocupado. As células em

verde representam a tolerância que deve ser adicionada aos obstáculos por questões de navegação conforme descrito na seção 4.1.

A Figura 35 mostra o detalhe do applet de emergência e como é simples enviar um texto ao sintetizador de voz do robô Nomad. O usuário pode enviar um texto ao sintetizador de voz do robô com um simples click no botão “Falar” da interface que mostrará uma janela para o usuário preencher o campo com o texto a ser enviado.

O applet de emergência é usado para a segurança do robô. Com um click no botão o robô é parado imediatamente.

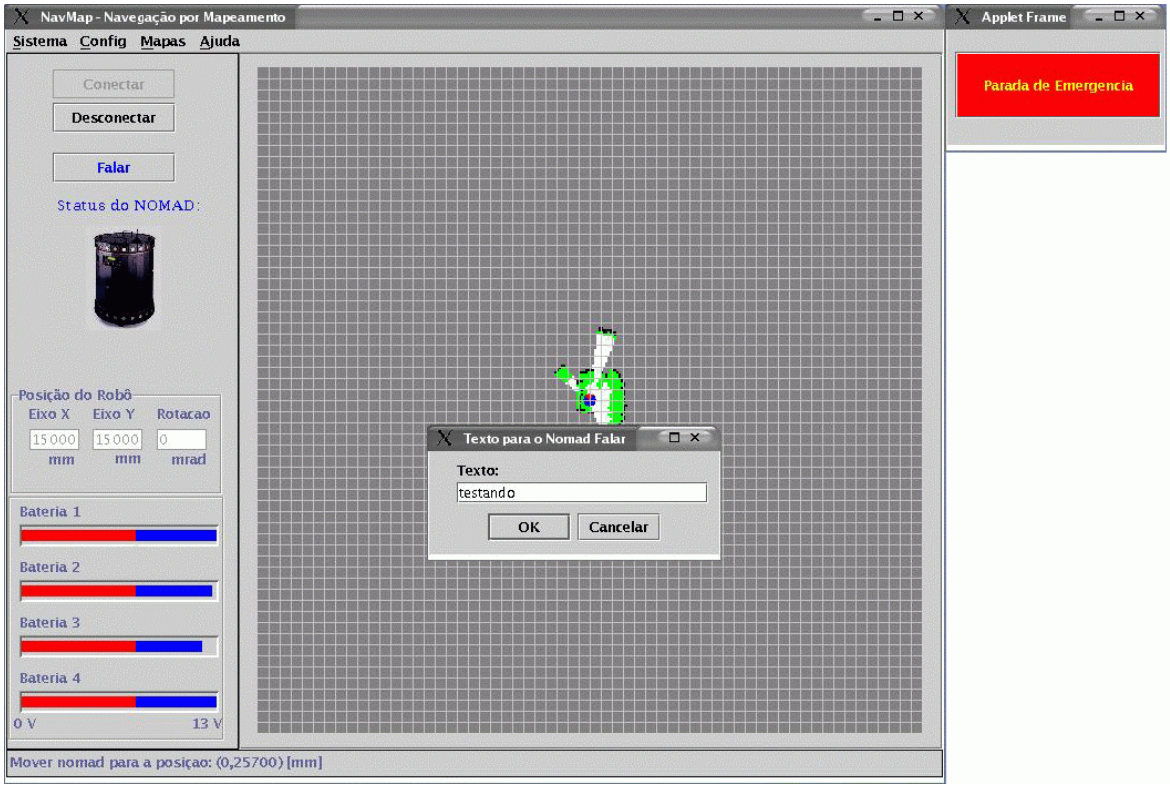


Figura 35 - Enviar texto ao robô

Quando a aplicação é executada remotamente via WEB, o usuário necessitaria de uma realimentação real do robô. Desse modo, quando foi desenvolvida a pagina HTML que contém o applet, criou-se janelas contendo imagens de algumas câmeras existentes no laboratório e da câmera existente no próprio robô. Foram criadas três janelas para as câmeras para que o *browser* seja capaz de gerenciar com maior eficiência as imagens a serem mostradas. Além das três janelas para as câmeras, mais duas janelas foram criadas: uma para o applet principal e uma para o applet de emergência.

Foram utilizados servidores de imagens desenvolvidos em trabalhos anteriores por membros do GRACO. Esses servidores realizam a animação das imagens por meio de

mecanismos de envio de dados do Netscape [30]. Por esse motivo, as janelas das câmeras funcionam exclusivamente nessa família de *browser*.

A Figura 36 mostra a janela da câmera na ocasião em que o robô estava em sua garagem. A Figura 37 mostra a imagem da câmera do Graco e a Figura 38 a imagem da câmera da garagem.

Figura 36 - Câmera do Nomad

Figura 37 - Câmera do Ambiente do Laboratório

Figura 38 - Câmera da Garagem do Nomad

4.2.3 Navegação Utilizando Mapa

Os mapas de ambiente armazenados ou pré-processados são utilizados para realizar a navegação do robô e planejamento de trajetória. O usuário escolhe o mapa a ser utilizado pela aplicação no momento da conexão com o robô e pode salvar um mapa apenas quando a aplicação está conectada ao robô. As funcionalidades de salvamento e carregamento de mapas não estão habilitadas no modo remoto da aplicação (WEB). O planejamento da trajetória é realizado pelo módulo responsável pelo calculo da trajetória ótima. Esse módulo (PathPlanner seção 3.8.2.3) utiliza técnicas de IA (ver seções 2.7.2 e 3.5).

O usuário através de um comando de alto nível determina o ponto objetivo no mapa que o robô deve alcançar. Caso seja possível uma trajetória sem colisões, o robô é comandado pelo sistema de navegação até atingir o objetivo proposto pelo usuário.

O comando é realizado através da interface gráfica (Figura 34). O usuário especifica o destino a ser alcançado pelo robô, ou seja, as coordenadas de destino, com um simples toque do *mouse* em cima do mapa. O sistema de navegação se encarrega de executar todos os passos para que o robô alcance o objetivo.

O sistema de navegação, à medida que conduz o robô até o objetivo, executa o sensoriamento de obstáculos não previstos pelo mapa. Caso o sistema de navegação detecte um obstáculo muito próximo ao robô (distâncias iguais ou inferiores a 200 mm), uma nova trajetória é calculada de forma a desviar do obstáculo, evitando assim possíveis colisões.

Como medida de segurança, o sistema de navegação pára totalmente a movimentação do robô caso este se aproxime muito dos obstáculos. A distância mínima permitida entre o robô e os obstáculos foi limitada a 150 mm. Os sensores de colisão também são constantemente monitorados para que, em caso de uma colisão não esperada, o robô pare totalmente.

4.2.4 Mapeamento

Realizou-se o mapeamento do laboratório do Graco (Grupo de Automação e Controle) com o objetivo de se observar a eficiência da representação do espaço real. A exploração do ambiente foi realizada utilizando a própria aplicação desenvolvida. O robô foi inicialmente posicionado em um ponto fixo no qual as coordenadas desse ponto em relação ao referencial global eram conhecidas. Dessa forma realizou-se a localização do robô no ambiente.

O resultado pode ser observado pela Figura 39.

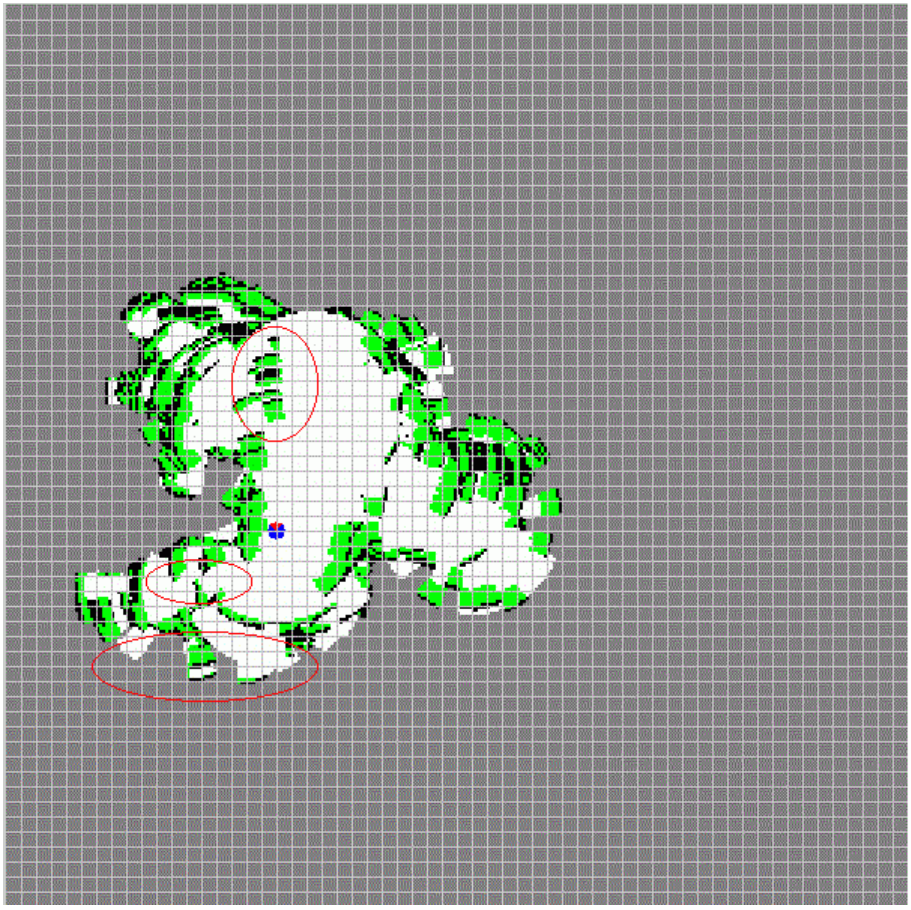


Figura 39 - Mapeamento do laboratório Graco

Observa-se que devido ao modelo sensorial adotado (ver seção 3.3), algumas células apresentam obstáculos que na realidade são inexistentes. Esse fato pode ser atribuído às características dos sensores utilizados.

Podemos observar na Figura 40 o mapa do ambiente do GRACO feito manualmente para comparação de resultados. O tamanho total da área cinza é de 30.000 x 30.000 mm. Dessa forma os mapas da Figura 39 e da Figura 40 estão na mesma escala.

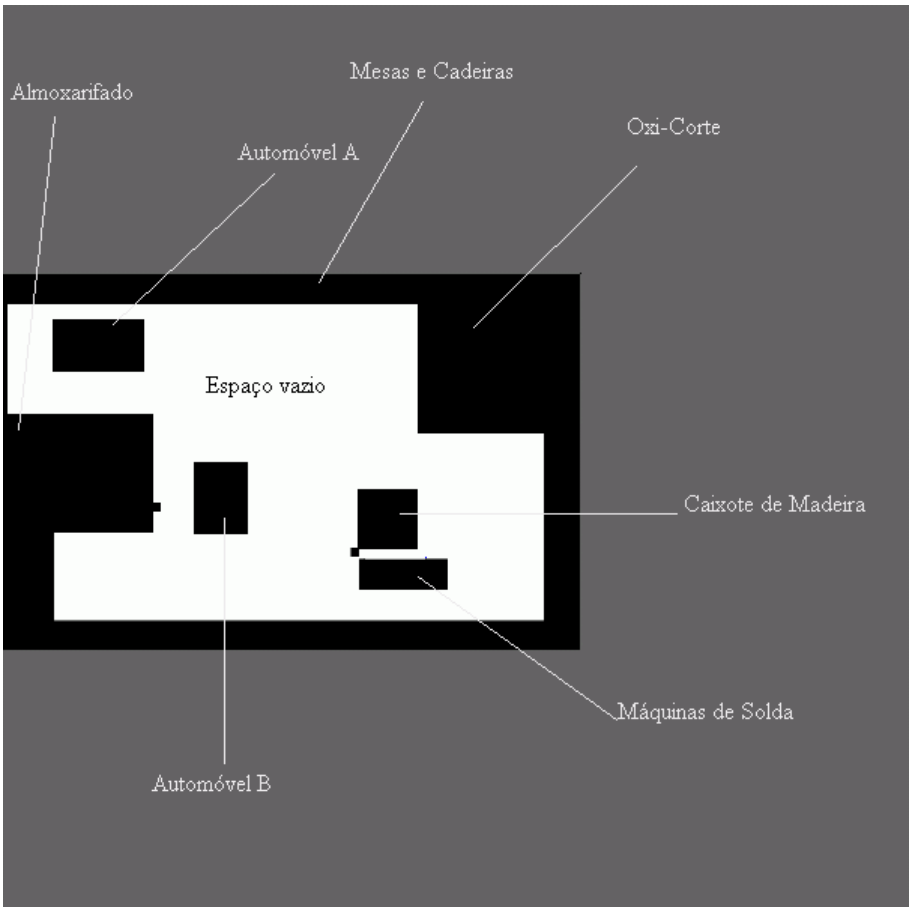


Figura 40 - Mapa do Graco

Na Figura 41 é feita uma sobreposição de imagens da Figura 39 e Figura 40 - Mapa do Graco. Verificou-se que uma determinada região do espaço é fielmente representada quando o robô está próximo a essa região. À medida que o robô se afasta, observa-se a atualização errônea de algumas células.

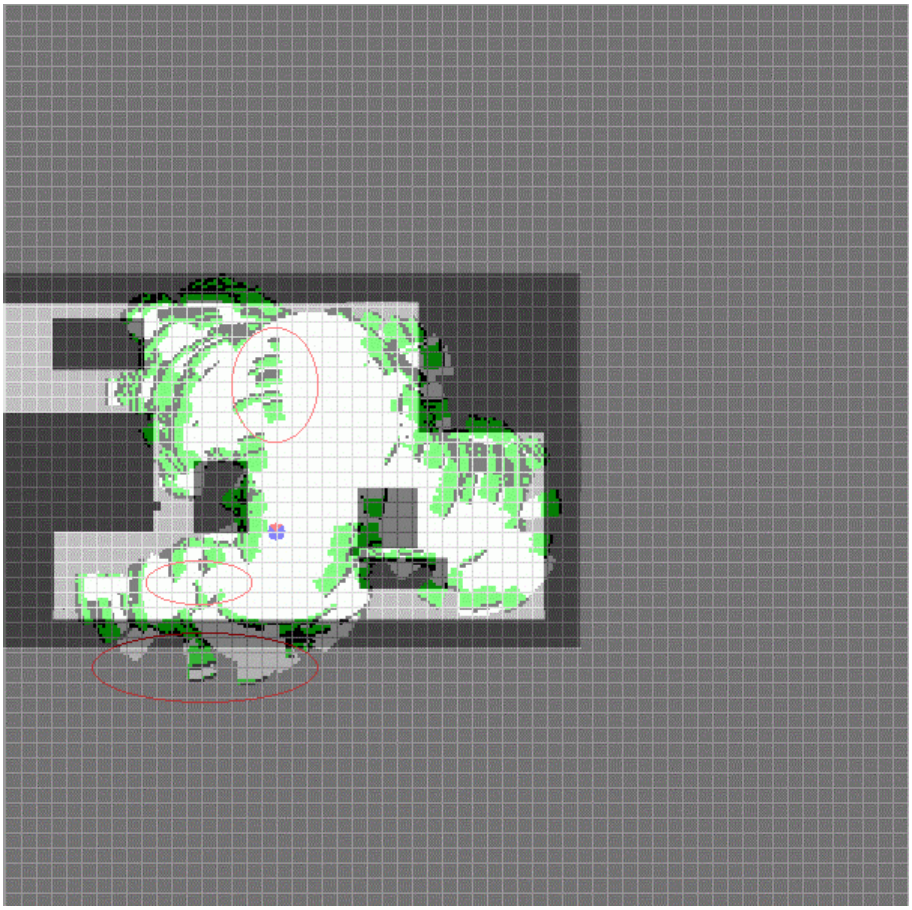


Figura 41 - Sobreposição dos mapas

4.3 Análise dos Resultados do Projeto

De acordo com os objetivos propostos, implementou-se uma interface gráfica bastante amigável. Tal interface disponibiliza informações como nível de carga das baterias do robô, mapa do ambiente e posição do robô no mapa. Todas essas informações são constantemente atualizadas para que o usuário receba as informações mais fiéis possíveis.

No entanto, durante a movimentação do robô, a detecção de obstáculos e o planejamento da trajetória são prioritários. Nesse caso, as informações disponibilizadas ao usuário pela interface gráfica não são atualizadas até que o movimento do robô se complete.

O sistema de navegação implementado apresenta ao usuário mapas obtidos do ambiente remoto a partir dos dados sensoriais do robô, e permite ao mesmo o controle de movimentos do robô.

Entretanto a movimentação do robô e a detecção de obstáculos dinâmicos e estáticos são extremamente dependentes do mapeamento realizado pelo sistema de navegação. Os valores coletados pelos sensores de proximidade são bastante importantes para a eficiência do mapeamento do ambiente e conseqüente navegação do robô.

Desse modo verifica-se a necessidade da utilização de filtros de Kalman [12] para eliminar as medidas errôneas comumente apresentadas na estimativa de distância utilizando sensores ultra-sônicos.

Observou-se em alguns casos que a luminosidade do ambiente afetavam as leituras de alguns sensores infravermelhos. Apesar de não haver obstáculos dentro da faixa de operação desse tipo de sensor (ver seção 2.3.2.2), alguns sensores os detectavam. Esse fato pode ser atribuído aos raios infravermelhos provenientes da iluminação solar.

Desse modo, com a utilização do modelo sensorial adotado (ver seção 3.3) e a detecção de obstáculos inexistentes por alguns sensores infravermelhos, o sistema de navegação ou executava paradas de segurança na movimentação do robô ou recalculava novas trajetórias dependendo da leitura do infravermelho conforme descrito na seção 4.2.3. Esse problema não foi solucionado até o presente momento, devendo-se então, estudar algum método para minimizar os efeitos de leituras errôneas de sensores infravermelhos.

Verificou-se também a necessidade da implementação de um algoritmo de atualização do mapa local (limitado pelo alcance dos sensores ultra-sônicos), centrado na posição atual do robô, e a partir desse mapa local realizar a atualização do mapa global. Neste trabalho atualizam-se todas as células do mapa global, o que ocasiona em um maior custo computacional.

Um problema ainda não solucionado, mas essencial à eficiência do sistema de navegação do robô, é o acúmulo do erro odométrico (ver experimento descrito no Apêndice A.2). Essa degradação na estimativa da posição do robô ocasiona distorções no mapa comprometendo a navegação do robô.

Apesar dos erros gerados pelo *dead-reckoning* ainda não terem afetado significativamente os resultados obtidos pelo sistema de navegação devido à baixa velocidade de deslocamento do robô e às dimensões do ambiente explorado serem relativamente pequenas, observa-se a necessidade da implementação de um módulo de auto-localização. Este módulo de auto-localização poderia ser implementado com a utilização de sensores externos ao robô ou com a utilização de câmera de vídeo.

No entanto, o mapa adquirido pode ser utilizado como uma primeira aproximação do ambiente. Com a utilização desse mapa o robô foi capaz de alcançar os objetivos propostos sem colidir com nenhum obstáculo.

É importante observar que a exploração do ambiente desconhecido foi realizada utilizando-se o próprio sistema de navegação implementado. De acordo com o mapa local adquirido pelos sensores em um determinado instante, o usuário da interface gráfica envia ao

sistema de navegação as coordenadas de destino do robô. Ao mesmo tempo em que o robô se movimenta realiza-se o mapeamento do ambiente desconhecido.

Dessa maneira, propõe-se, como melhoria do sistema de navegação, a implementação de uma estratégia de exploração autônoma no qual o robô possua a inteligência de mapear um ambiente desconhecido sem a intervenção humana.

5 CONCLUSÃO

A abordagem utilizada para representação dos mapas ambientais permitiu ao sistema desenvolvido nesse projeto a navegação em um ambiente conhecido ou não. Com essa representação, os mapas adquiridos podem ser utilizados como uma primeira aproximação do ambiente.

Apesar das dificuldades encontradas devido ao modelo sensorial e à representação do ambiente adotados, o robô foi capaz de alcançar, com pequenas limitações, os objetivos propostos pelo usuário do sistema de navegação sem colidir com nenhum obstáculo.

Conclui-se por fim, de acordo com a análise dos resultados alcançados, que alguns trabalhos podem ser realizados para a melhoria do sistema de navegação implementado, tais como:

- Atualização do mapa local (limitado pelo alcance dos sensores ultra-sônicos) e casamento das células desse mapa com o mapa global;
- Desenvolvimento de um módulo de auto-localização, utilizando sensores externos ao robô ou câmera de vídeo, para correção do erro odométrico evitando possíveis distorções no mapa;
- Desenvolvimento de uma maneira mais eficiente para a transmissão dos mapas através do protocolo TCP;
- Utilização de filtros de Kalman para minimização dos erros das leituras dos sonares.
- Representação do ambiente utilizando grades de evidência.
- Implementação de uma estratégia de exploração autônoma.

Todos os arquivos de código, binários e *papers* utilizados como referência nesse projeto podem ser acessados pelo CDROM em anexo no projeto ou através do endereço eletrônico: http://gospel.graco.unb.br/~nomad

REFERÊNCIAS BIBLIOGRÁFICAS

[1] MCKERROW, P. J. (1991). *Introduction to Robotics*, University of Wollongong, Australia. Addison-Wesley publishing company.

[2] FUKE, Y. and E. KROTKOV. *Dead Reckoning for a LunarRover on Uneven Terrain*. Proceedings of the 1996 IEEE International Conference on Robotics andAutomation, Minneapolis, Minnesota, 1996, pp. 411-416.

[3] MURATA, S. and T. HIROSE. *Onboard Locating SystemUsing Real-Time Image Processing for a Self-Navigation Vehicle*. IEEE Trans. Industrial Electronics,Vol. 40, No. 1, 1993, pp. 145-154.

[4] LAGES, W.F.; E. M. HEMERLY e L. F. A. PEREIRA. *Controle Linerizante de uma Plataforma Móvel Empregando Realimentação Visual*.Anais do XICongresso Brasileiro de Automática, São Paulo, 1996, pp.537-542.

[5] BORENSTEIN, J. and L. FENG. *Measurement andCorrection of Systematic Odometry Errors in MobileRobots*. IEEE Trans. Robotics Automation, Vol. 12,No. 6, 1996, pp. 869-880.

[6] BARSHAN, B. and H. F. DURRANT-WHYTE. *Inertial Navigation System for Mobile Robots*.IEEE Trans.Robotics and Automation, Vol. 11, No. 3, 1995, pp. 328-342.

[7] TACHI, S. and KOLMORIYA, K. (1995). *Guide Dog Robot*. In: Hanafusa, H. and Inoue, H. (eds) *Robotics Research*: *The Second International Symposium*, MIT Press, pp. 333-340.

[8] JÚNIOR, E. P. S. *Navegação exploratória baseada em problemas de Valores de Contorno*. 2002. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Instituto de Informática, Porto Alegre, Brasil, 2002.

[9] ELFES, *A. Sonar-based real world mapping and navigation*. IEEE Journal of Robotics and Automation, v.RA-3, n.3, p.249-265, 1987.

[10] MORAVEC, H. P.; ELFES, A. *High resolution maps from wide angle sonar*. IN: IEEE INTERNATIONAL CONFERENCE OF ROBOTICS AND AUTOMATION (ICRA), 1985. Proceedings…[S.1.:s.n.], 1985.

[11] CHATILA, R.; LAUMOND, J.P. *Position referencing and consistent world modeling*. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 1989. Proceedings…[S.1.:s.n.], 1989.

[12] ROMERO, L. *Construcción de mapas y localización de robots móviles: un enfoque híbrido. 2001*. Tese (Doutorado em Ciencia da Computação) – TEC de Monterrey, Cuernavaca, Morelos, 2001.

[13] DAM, J.V. *Environment modeling for móbile robots: neural learning for sensor fusion*. 1998. Tese (Doutorado em Ciência da Computação) – Universiteit van Amsterdam, Faculteit WINS, Amsterdam, The Netherlans, 1998.

[14] HEBERT, P.; BETGÉ-BREZETS, S.; CHATILA, R. *Probabilist map learning: necessity and difficulties*. Toulose Cedex, France: Laboratoire d'Analyse et d'Architecture des Systèmes, 1995. (95542).

[15] DISSANAYAKE, M. W. M. G. et al. *A solution to the simultaneous localization and map building problem*. IEEE Transaction on Robotics and Automation, v.17, n.3, p.229-241, 2001.

[16] THRUN, S. *Learning maps for indoor mobile robot navigation*. Artificial Intelligence, v.99, n.1, p.21-71, 1998.

[17] Nomad XRDEV Software Manual Release 1.0, Nomadic Technologies, Inc. – 03/1999

[18] NAIRA, J. ;J; Hon; J. D. Tardos; G. Schmidt. *Multisensor Mobile Robot Localization.* . IN: IEEE International Conference of Robotics and Automation (ICRA), 1996. Proceedings…[S.1.:s.n.], 1996.

[19] NASCIMENTO, C. L.;YONEYAMA, T. *Inteligência artificial em controle e automação*, Ed. Edgard Blucher e FAPESP, São Paulo, 1a. Edição, Julho 2000.

[20] WINSTON, P. H. *Artificial Intelligence*, Addison-Wesley, 1984.

[21] PAGAC, D.; NEBOT, E.; DURRANT, W. *An Evidential Approach to Probabilistic Map-Building*. IN: IEEE International Conference of Robotics and Automation (Minneapolis), 1996, pp745-750.

[22] BILGIÇ, T; BURHAN, I. *Model-based Localization for an Autonomous Mobile Robot Equipped with Sonar Sensors*. Depart. of Industrial Engineering, Toronto Ontario M5S 1A4 Canada,1995.

[23] Sun Microsystems - *System Interfaces Guide*. California 94043-1100 U.S.A. 1995

[24] TANENBAUM, A. S. *Sistemas Operacionais Modernos*. Tradução: Nery Machado, Editora LTC,1995

[25] Nomad XR4000 Hardware Manual Release 1.0, Nomadic Technologies, Inc. – 03/1999

[26] SCHULTZ, A. C.; ADAMS, W. and YAMAUCHI, B. *Integrating, Exploration, Localization, Navigation and Planning with a Common Representation*. Navy Center for Applied Research in artificial intelligence, Naval Research Laboratory Washington, DC – USA – 1999.

[27] YAMAUCHI, B.; SCHULTZ, A. C. and ADAMS, W. *Mobile Robot Exploration and Map-Building with Continuous Localization*, Navy Center for Applied Research in artificial intelligence, Naval Research Laboratory Washington, DC – USA – 1998.

[28] RIBEIRO, I. S. *Localização em Robótica Móvel*, Instituto Superior Técnico – Portugal – 1999.

[29] FRANZ, A. S.; Elder, M. H. e WALTER, F. L. *Sistema para Navegação e guiagem de robôs móveis autônomos*, Instituto Tecnológico de Aeronáutica – ITA – 1998.

[30] TOURINO, S. R. G. *Guiagem do Robô Móvel XR4000 para Inspeção via Internet de Tubulações Industriais Soldadas*, Projeto final de graduação. GRACO – UnB – 2000.

[31] YAMAUCHI, B; SCCHULTS, A; ADAMS, W; GRAVES, K; GREFENSTETTE, J; PERZANOWSKI, D. ARIEL: *Autonomous Robot for Integrated Exploration and Localization*. Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, 1997.

[32] JANDI, P. Jr. *Introdução ao Java. Núcleo de Educação a Distância*. Universidade São Francisco, 1999.

[33] Sun MicroSystem – *Java Native Interface: Programmer's Guide and Specification*

[34] S. KOENIG and M. LIKHACHEV. *Improved Fast Replanning for Robot Navigation in Unknown Terrain*. In Proceedings of the International Conference on Robotics and Automation, 2002..

APÊNDICE A

A.1 Experimento 1 – Teste Realizados com os Sensores Ultra-sônicos

Neste experimento manteve-se o robô em uma posição fixa enquanto os sonares eram disparados e suas medidas coletadas. Escolheu-se um sonar de cada conjunto para a observação de sua leitura, totalizando 06 sonares. As leituras entre os sonares superiores e dos seus correspondentes inferiores (sonar na mesma posição, porém em conjuntos diferentes) foram comparadas e as medidas que possuíam os menores valores foram coletadas.

O atraso entre o disparo de um sonar e o disparo do próximo sonar (*firedelay*) foi ajustado inicialmente em 01 ms (milisegundos). Foram realizadas duas observações sendo que, em cada observação, foram coletadas 100 medidas para cada sonar. Em seguida, ajustando o *firedelay* em 03 ms e posteriormente em 05 ms, repetiu-se o mesmo procedimento descrito acima.

A Figura 42 mostra o gráfico das leituras dos sonares obtidas utilizando-se o *firedelay* igual a 01 ms. A linha tracejada é referente à primeira observação dos sensores enquanto a linha contínua se refere à segunda observação dos sensores. As cores representam as medidas de 03 sonares distintos.

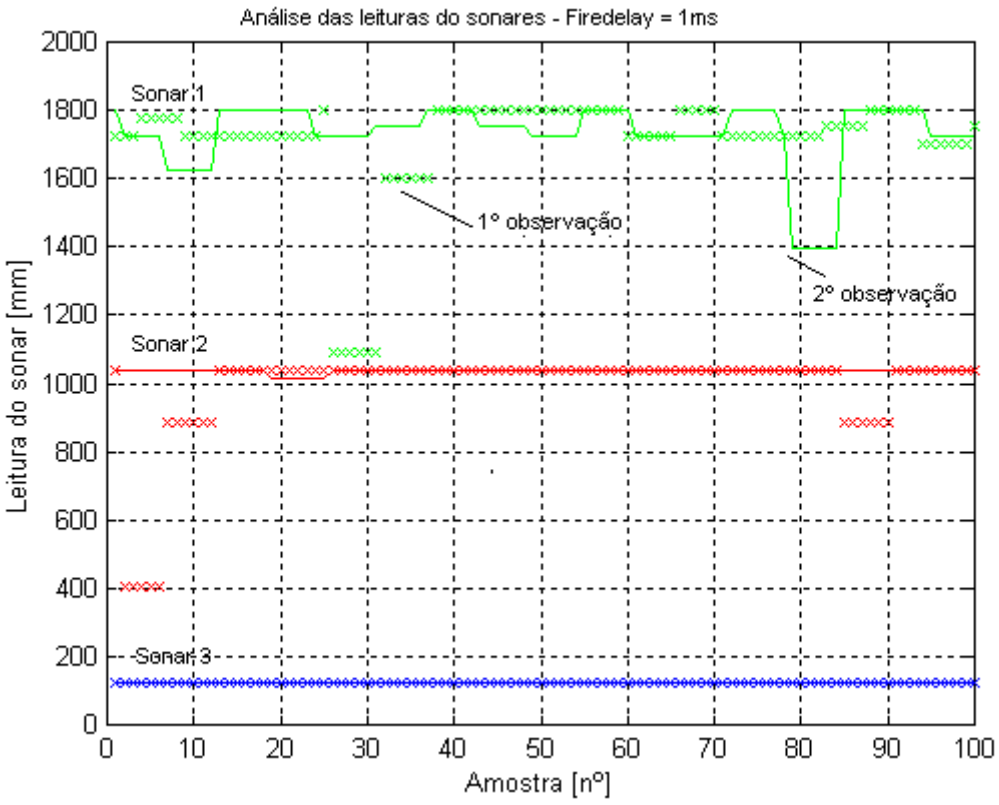


Figura 42 - Leituras dos sensores sonares – Firedelay 01 ms

Da Figura 42 observa-se uma grande variação nas leituras dos sensores à medida que se aumenta a distância do obstáculo ao sonar. Verificam-se em algumas leituras muito distantes das reais (sonar 01 e sonar 02 representados pelas linhas verdes e vermelhas respectivamente).

Como pode ser visto pela linha azul do gráfico da Figura 42, as medidas desse sonar não apresentaram variações, observando-se uma grande repetição dos valores medidos.

Em seguida, ajustou o *firedelay* em 03 ms. Observou-se nesse caso uma repetitibilidade nas leituras dos sensores. Os resultados podem ser vistos no gráfico da Figura 43. Somente o sonar 01 apresentou algumas variações nas leituras. Entretanto, os resultados encontrados utilizando um *firedelay* de 03 ms, foram muito mais satisfatórios do que os resultados utilizando-se um *firedelay* de 01 ms.

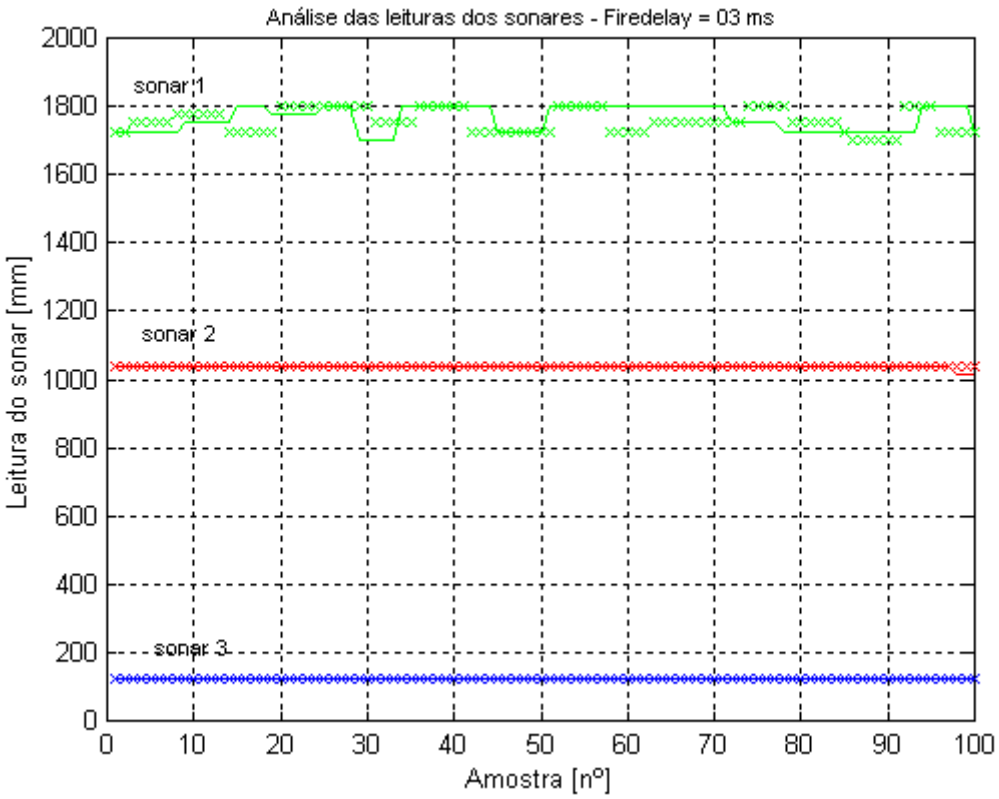


Figura 43 - Leituras dos sensores sonares – Firedelay 03 ms

Logo após, ajustando-se o *firedelay* para 5 ms, observou-se um aumento nas variações das leituras dos sensores sonares. Os resultados observados podem ser vistos no gráfico da Figura 44. Apesar desses resultados terem sido melhores que no caso do *firedelay* 01 ms (Figura 42), foram piores do que se utilizando um *firedelay* de 03 ms (Figura 43).

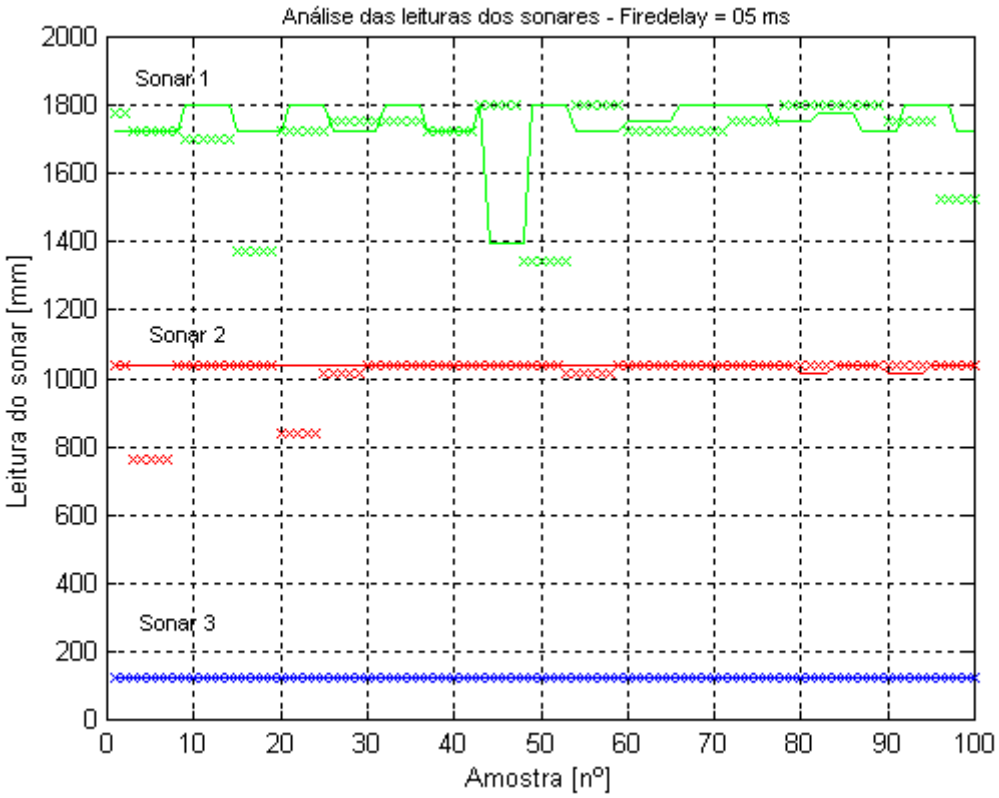


Figura 44 - Leituras dos sensores sonares – Firedelay 05 ms

Com esse experimento conclui-se que as variações nas leituras dos sensores sonares tendem a aumentar com o aumento da distância do obstáculo sentido e o ajuste do *firedelay* de 03 ms apresenta resultados satisfatórios, minimizando tais variações nas leituras dos sonares. Dessa forma, limitando o alcance dos sonares e ajustando o *firedelay* para um valor adequado, pode-se diminuir alguns efeitos indesejáveis que geram leituras errôneas.

A.2 Experimento 2 – Medição do Erro Odométrico

Realizou-se o seguinte experimento com o objetivo de verificar o erro de odometria do robô e estudar uma forma de corrigi-los. O experimento consistiu basicamente em programar o robô para que este, partindo de uma posição inicial, seguisse um caminho pré-definido e retornasse à posição de partida. A diferença medida entre a posição inicial e a posição final do robô caracteriza o erro de posição. Dessa forma mediu-se o erro translacional no eixo x e no eixo y em relação ao referencial do robô. Nesse experimento não foi considerado o erro rotacional.

O robô percorreu 4 metros na direção do eixo x (2 metros no sentido positivo e 2 metros no sentido negativo) e 4 metros na direção do eixo y(2 metros no sentido positivo e 2 metros no sentido negativo), totalizando 8 metros percorridos em uma volta completa no circuito. A Figura 45 representa o caminho percorrido pelo robô.

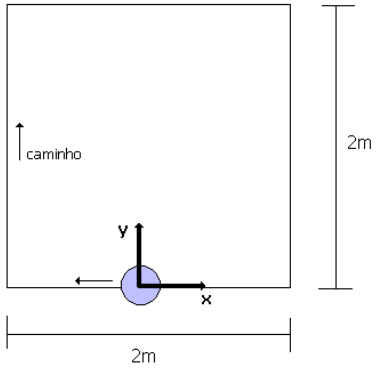


Figura 45 - Caminho percorrido pelo robô

Com os dados coletados traçou-se o gráfico da Figura 46:

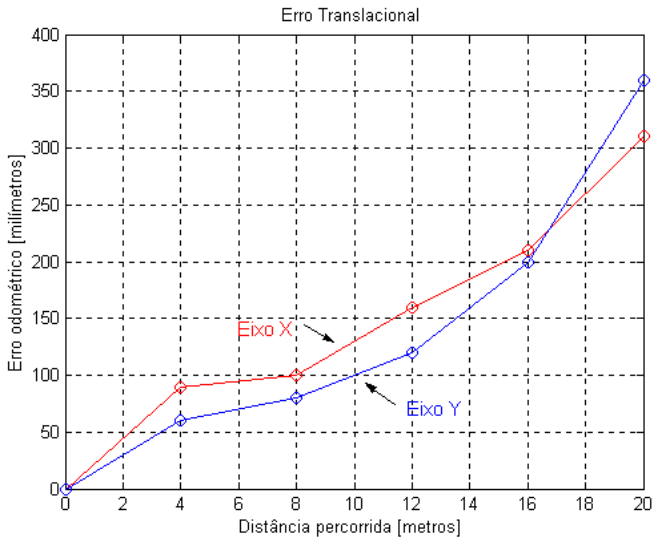


Figura 46 - Erro acumulado pela odometria do robô

Verificou-se, a partir do gráfico da Figura 46, o erro odométrico acumulado à medida que o robô se movimentava. Esse erro acumulado ficou dentro de valores esperados. O acumulo desse erro podem ser atribuídos a erros sistemáticos característicos dos sensores odométricos e a erro não sistemáticos causados principalmente por irregularidade dos solos.

APÊNDICE B

DIAGRAMAS DE CLASSES DA INTERFACE COM O USUÁRIO

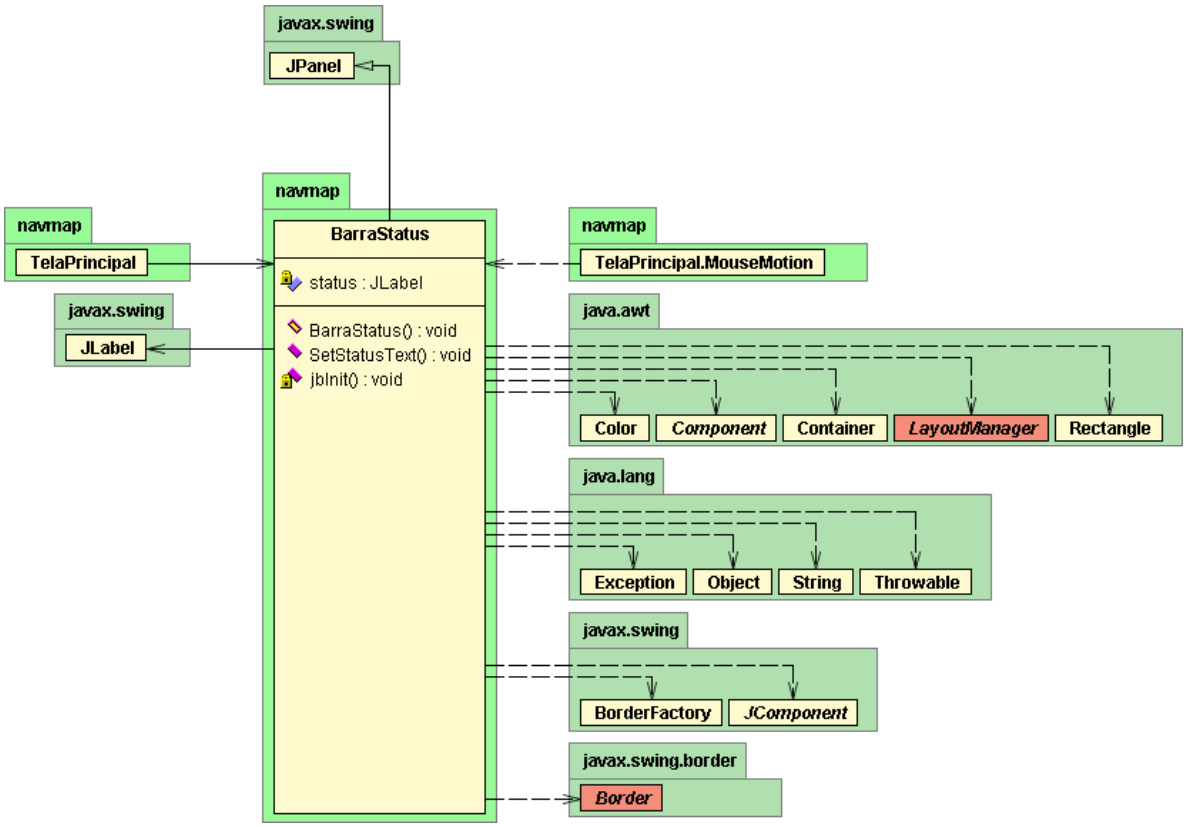


Figura 47 - BarraStatus

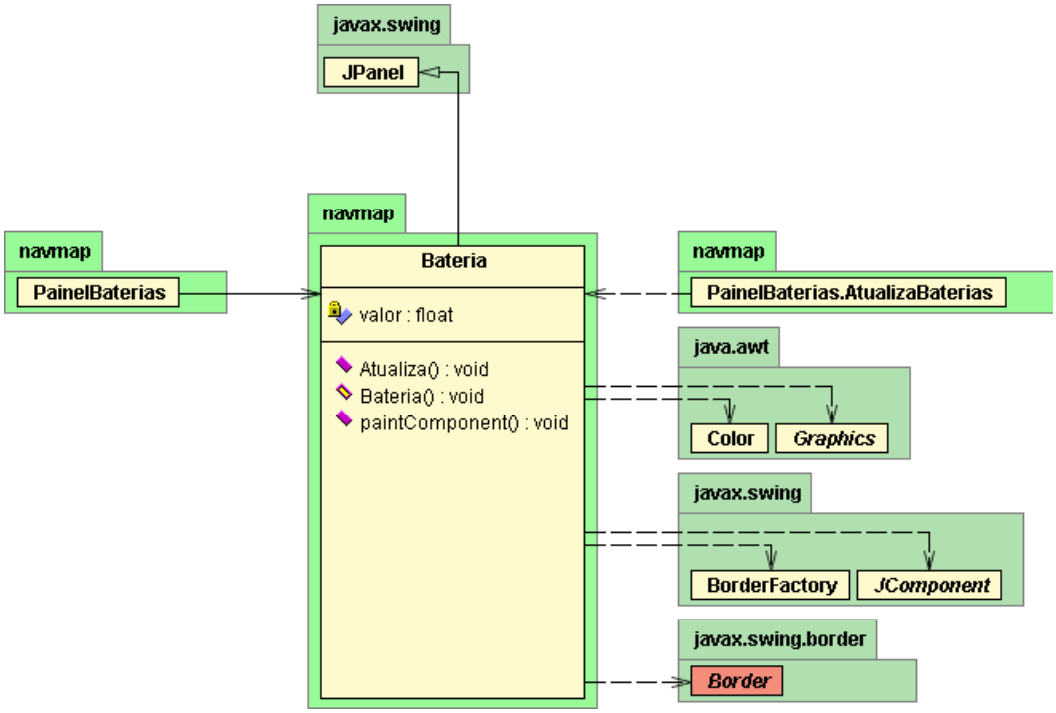


Figura 48 - Bateria

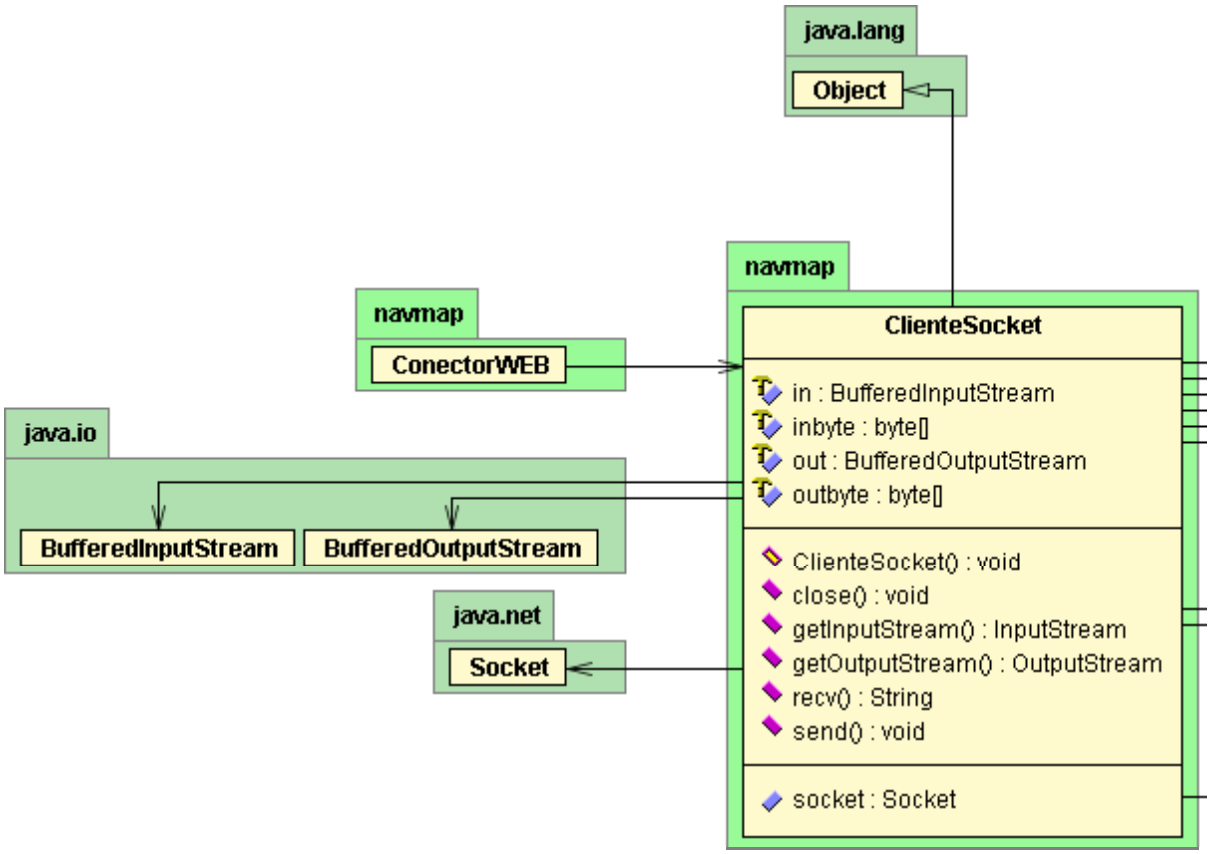


Figura 49 - ClienteSocket

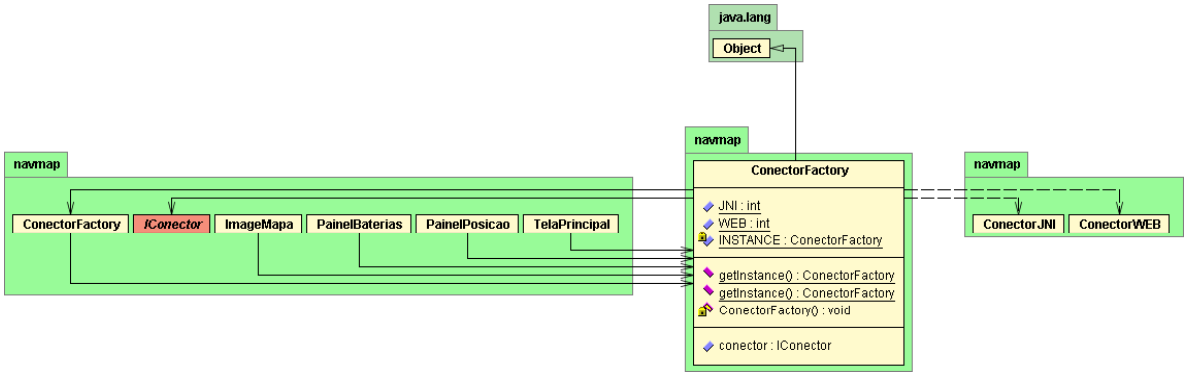


Figura 50 - ConectorFactory

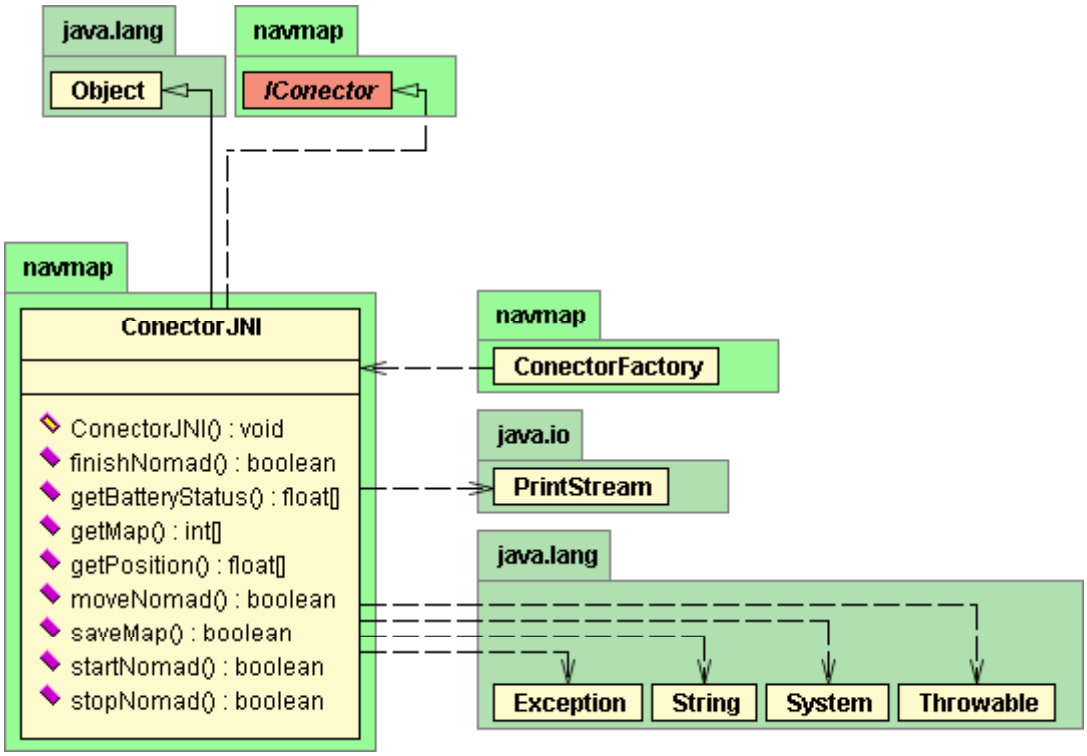


Figura 51 - Conector JNI

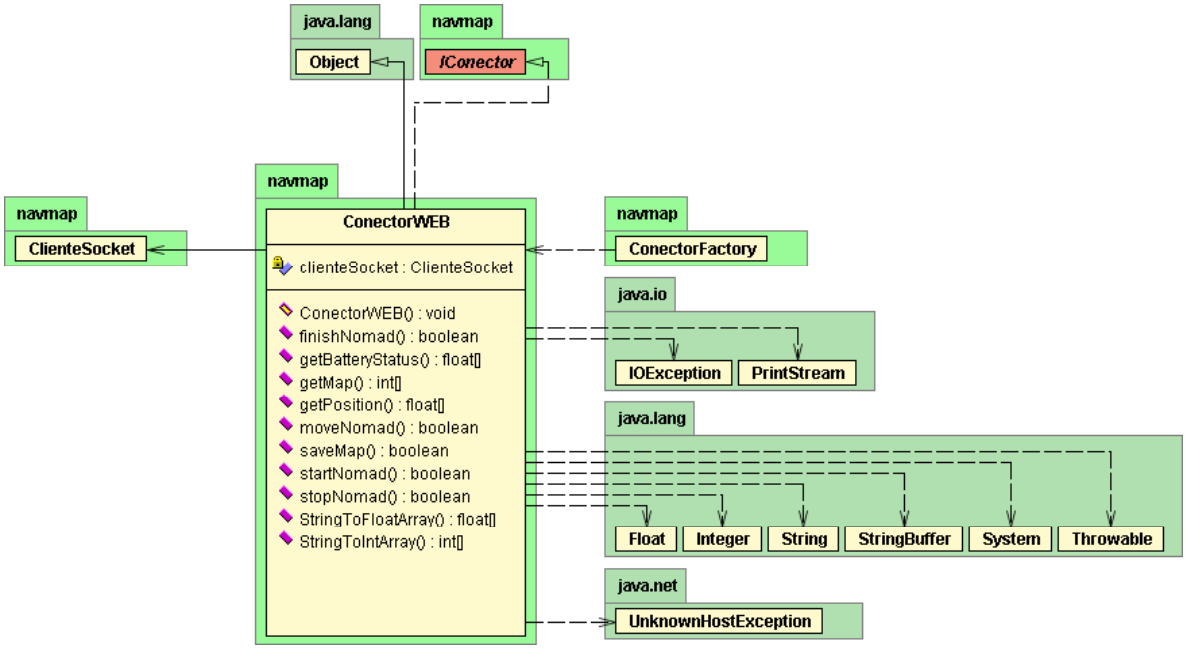


Figura 52 - ConectorWeb

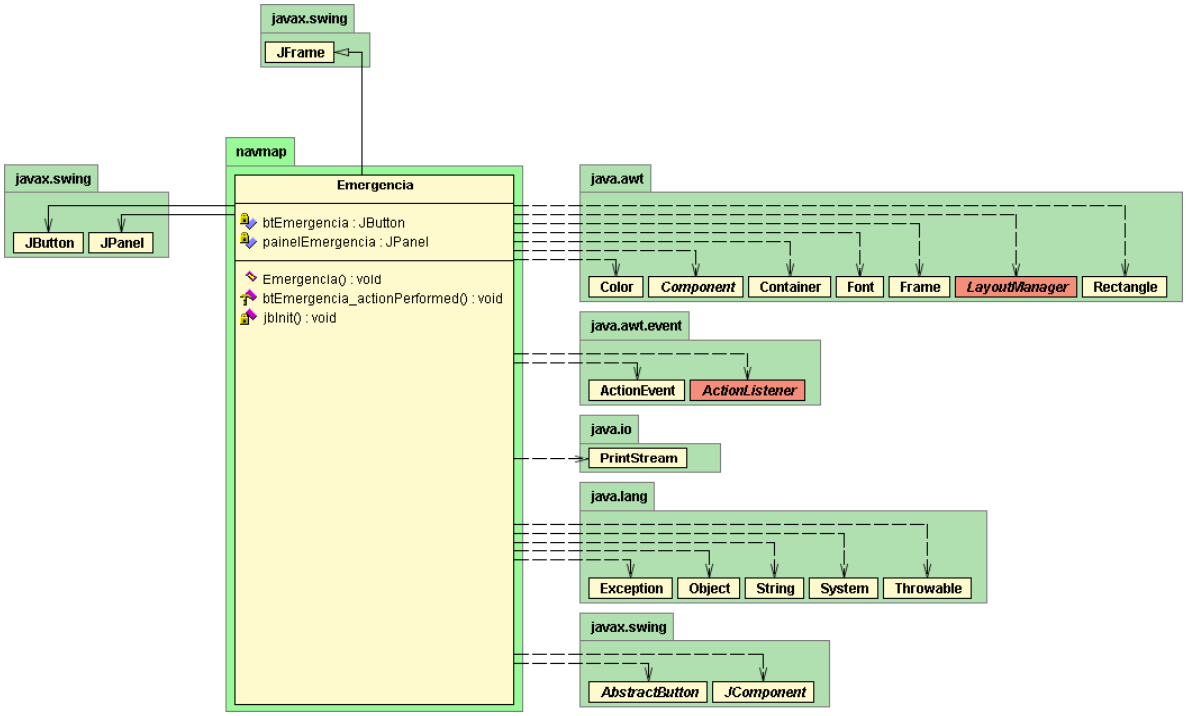


Figura 53 - Emergência

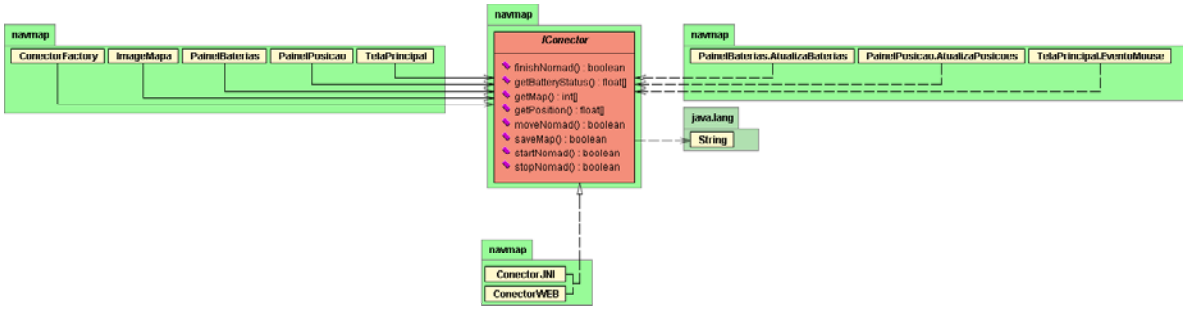


Figura 54 - IConector

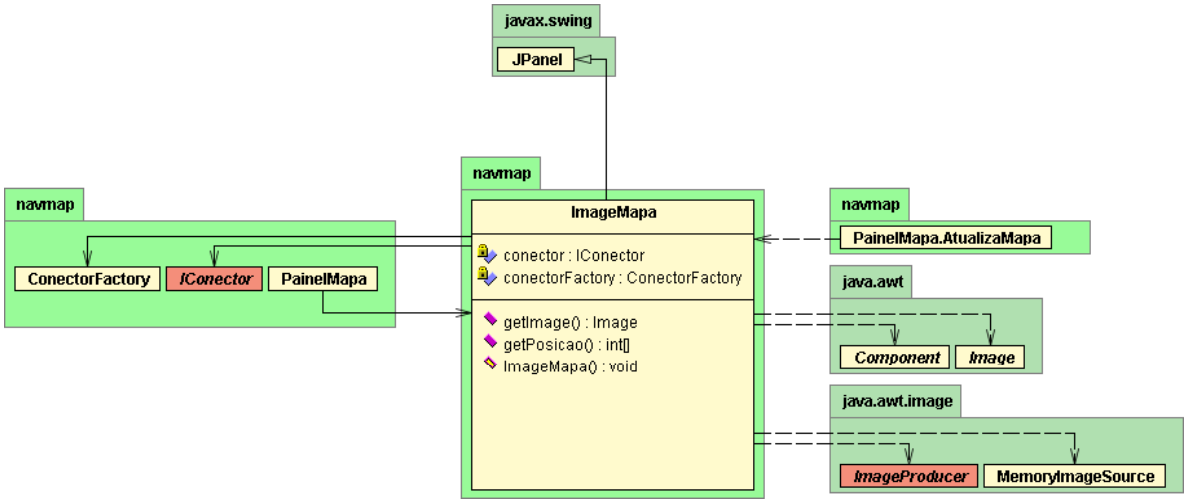


Figura 55 - ImageMapa

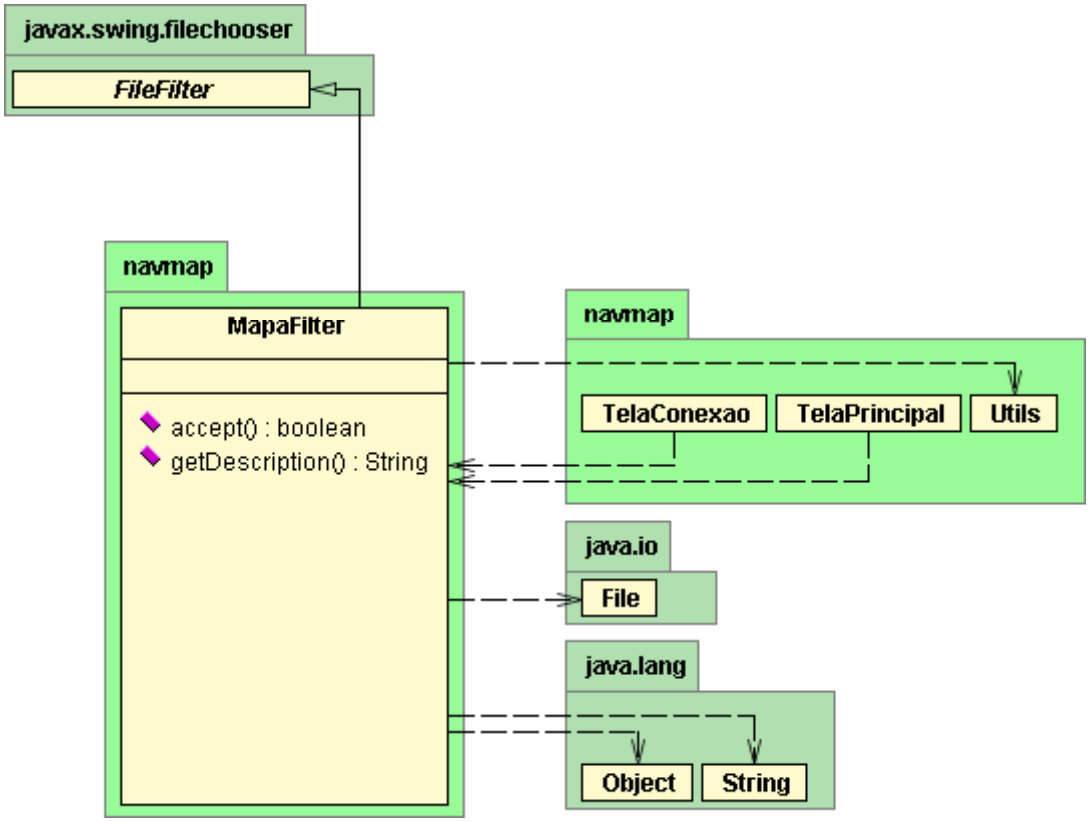


Figura 56 - MapaFilter

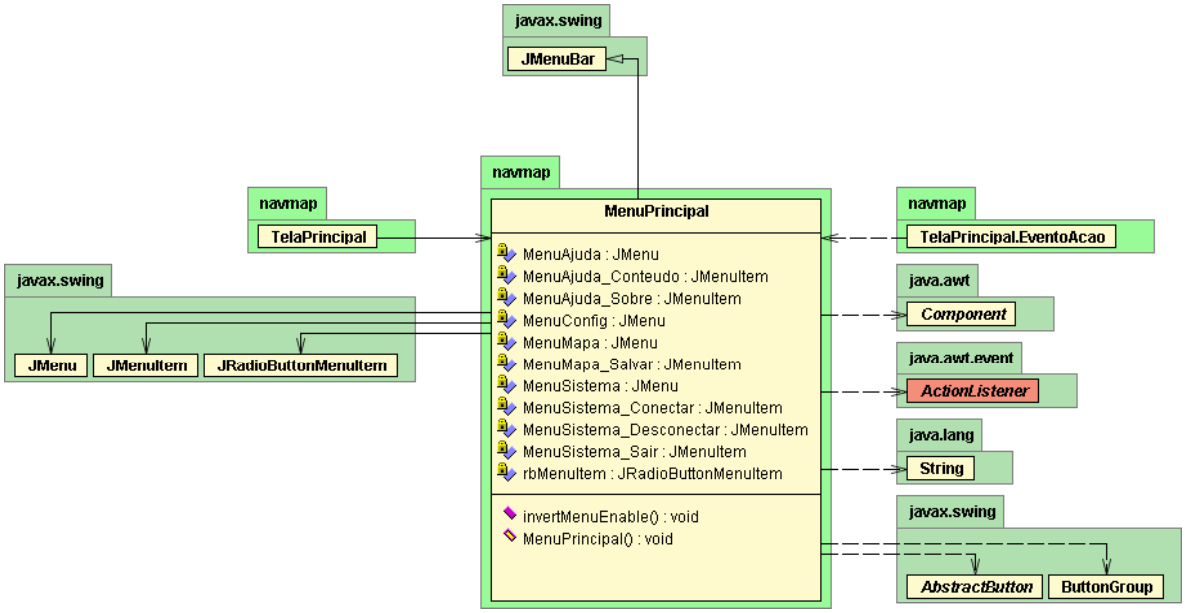


Figura 57 - MenuPrincipal

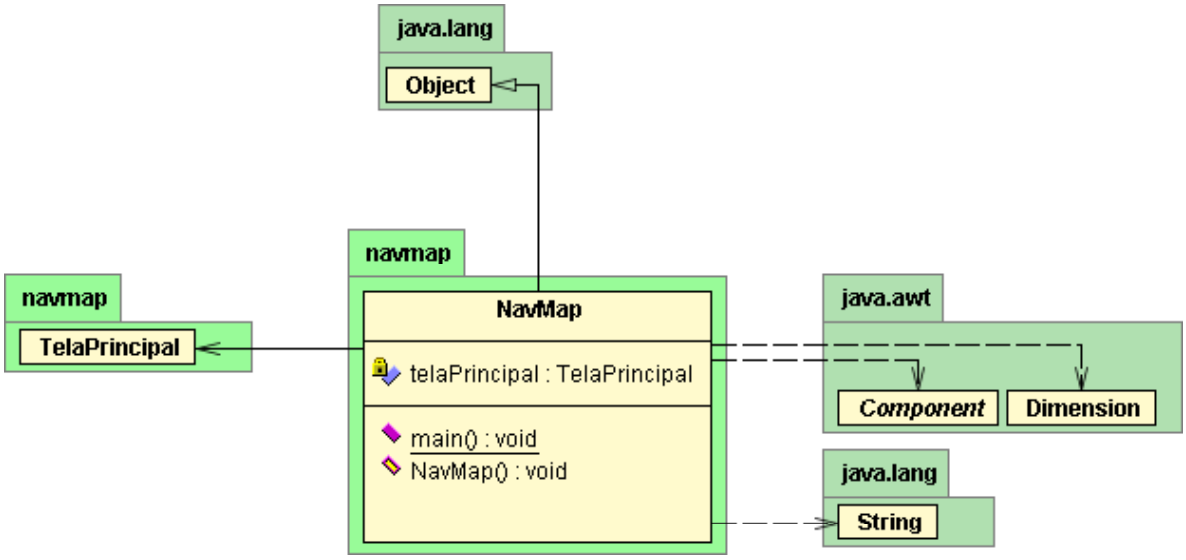


Figura 58 - NavMap

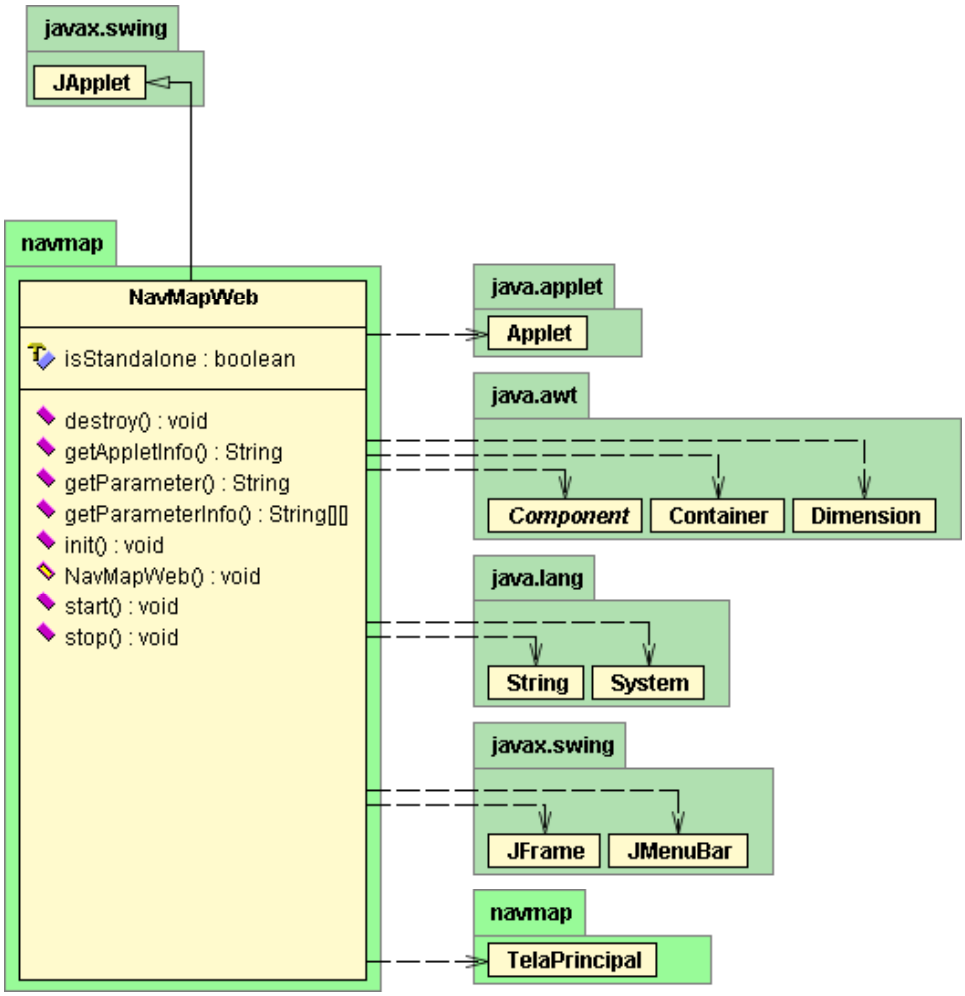


Figura 59 - NavmapWeb

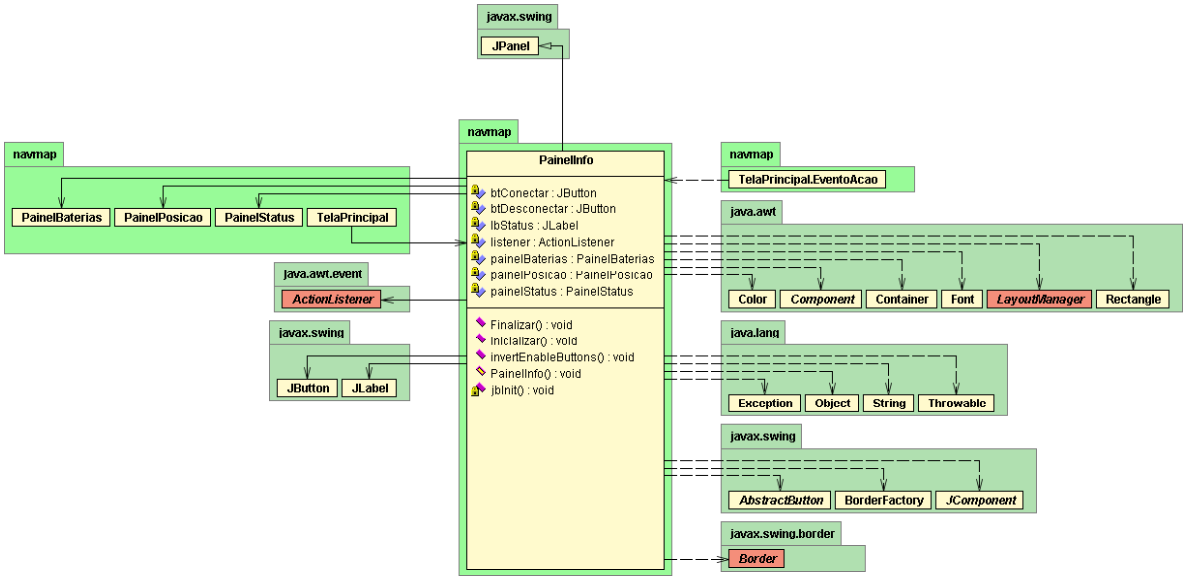


Figura 60 - PainelInfo

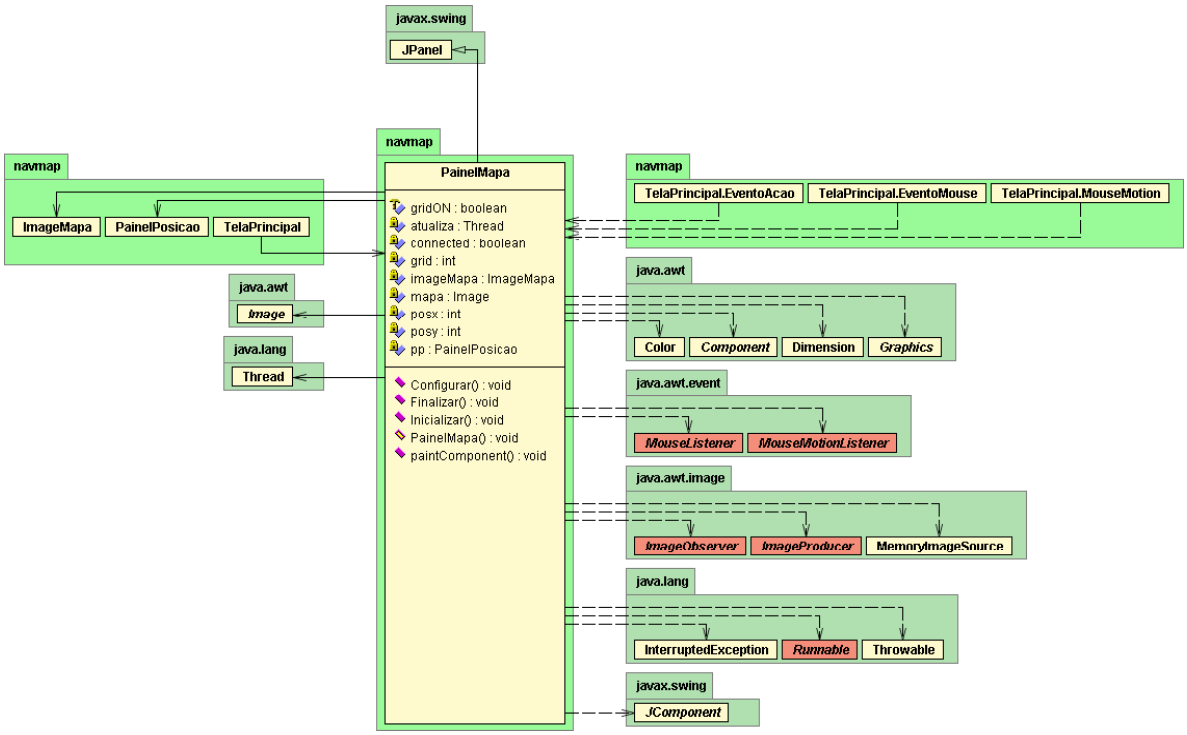


Figura 61 - PainelMapa

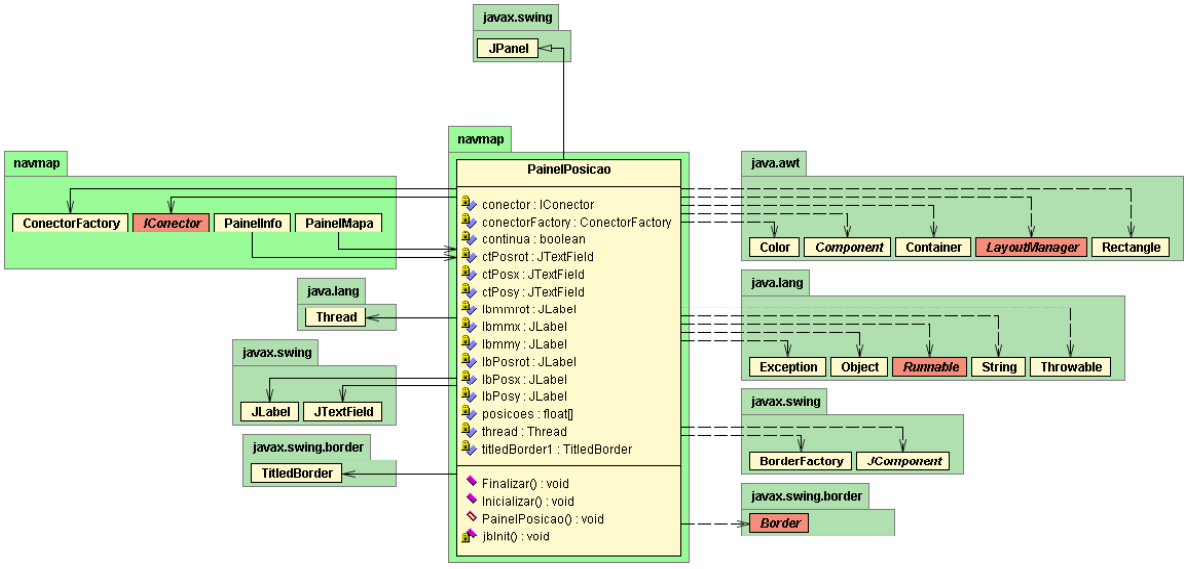


Figura 62 - PainelPosicao

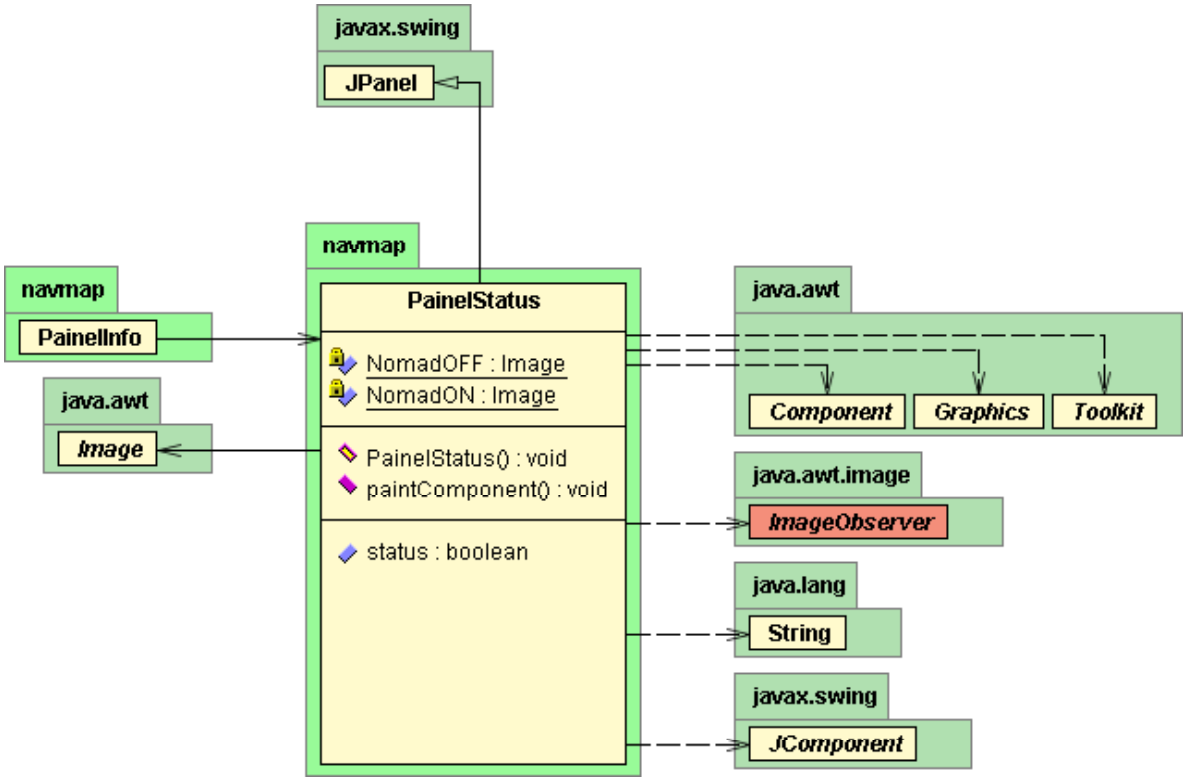


Figura 63 - PainelStatus

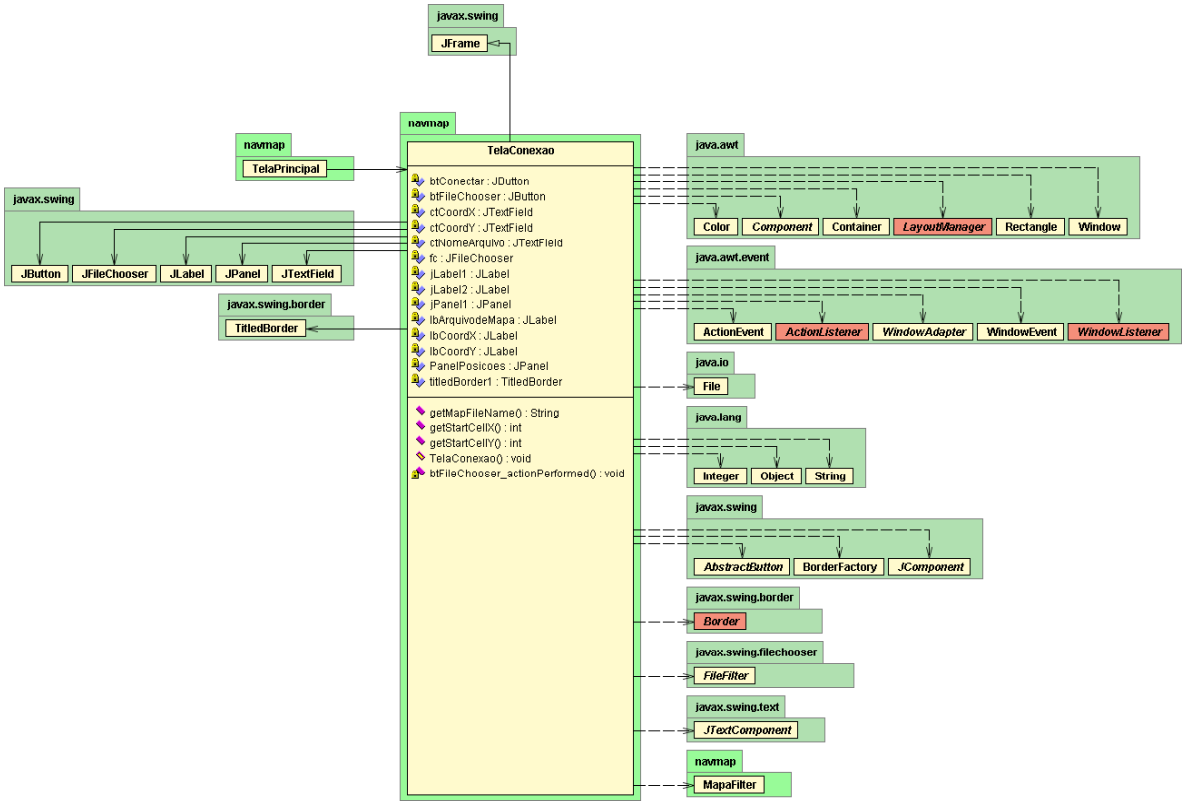


Figura 64 - TelaConexao

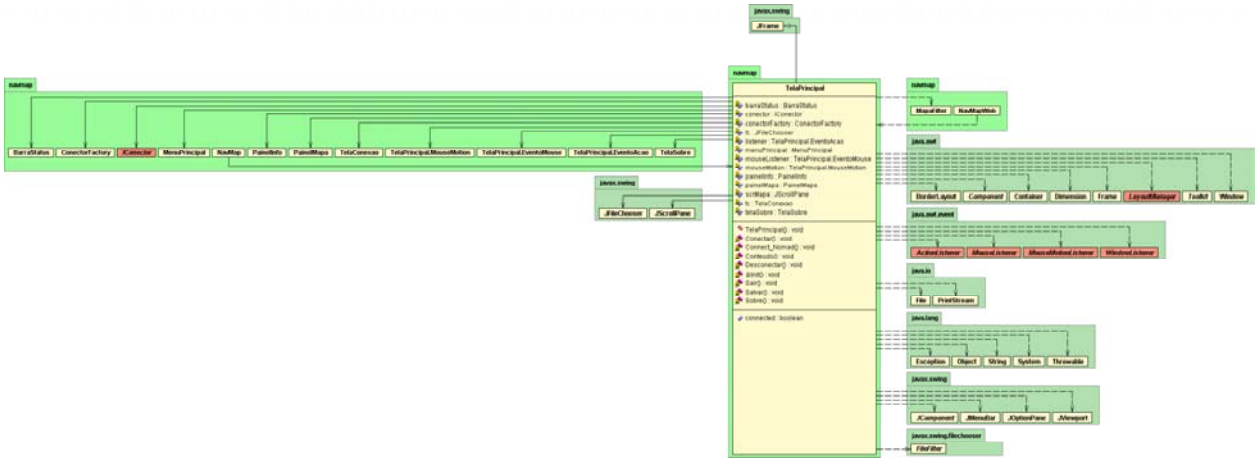


Figura 65 - TelaPrincipal

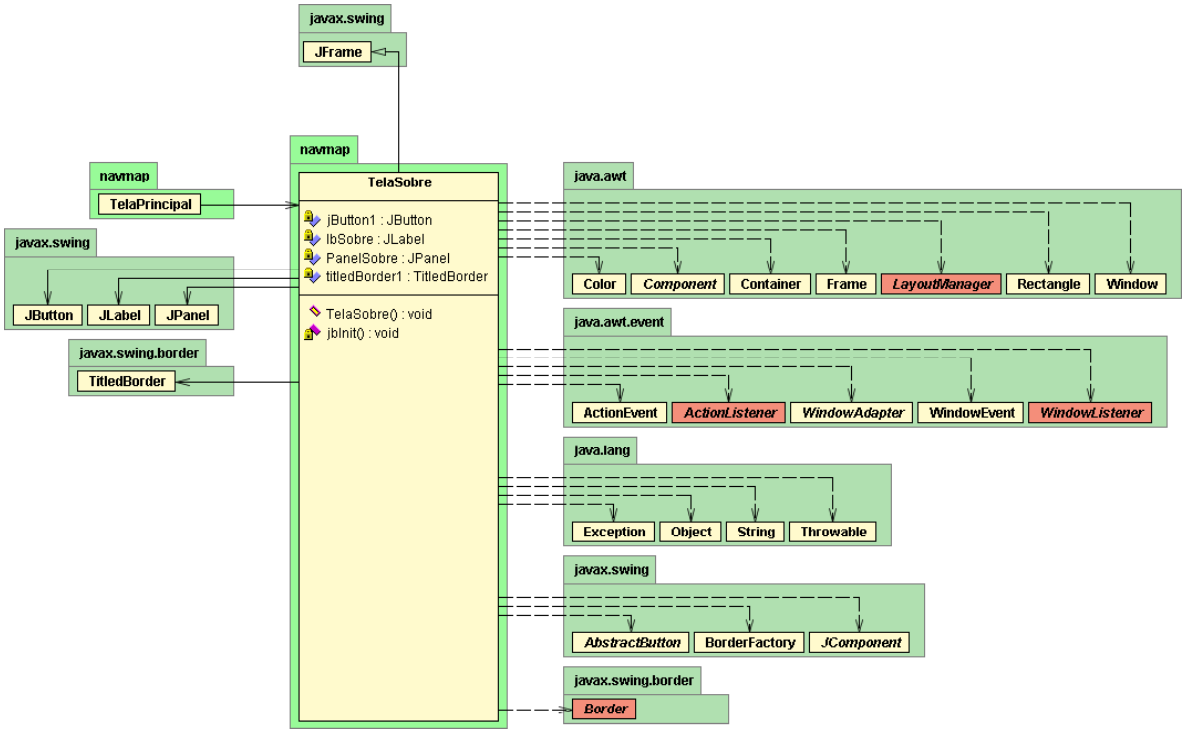


Figura 66 - TelaSobre

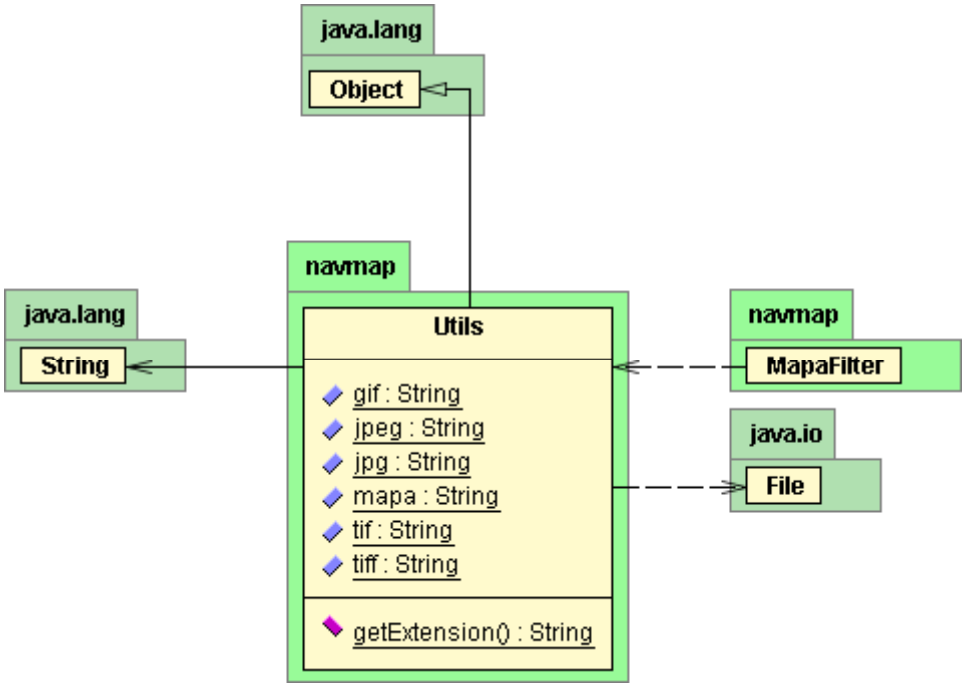


Figura 67 - Utils

GLOSSÁRIO

Terminologias

[inteligência artificial] são métodos computacionais que visam desenvolver um nível de raciocínio e inferência em máquinas.
[mapeamento] é a organização dos dados sensoriais em uma forma apropriada para serem utilizados pelo robô móvel para a navegação autônoma no ambiente.
[navegação] é a movimentação do robô no ambiente, com base em dados sensoriais, odométricos e de mapas armazenados.
[odometria] é a estimativa com base em modelos cinemáticos da posição e orientação do robô em seu ambiente.
[planejamento de trajetória] é o método no qual um robô escolhe a trajetória ótima com base na tarefa a ser realizada, definida ou por um operador ou por um sistema de planejamento de tarefas.
[robô] são sistemas capazes de realizar tarefas repetitivas de forma flexível e programável.
[robô móvel] são robôs que permitem movimentação própria sobre o solo, modificando sua posição em relação a um referencial fixo.
[sensores] são dispositivos que realizam a transformação de formas de energia com o objetivo de obter informações úteis a um sistema produtivo ou robótico.
[sensores por infravermelho] são sensores que utilizam luz infravermelha como sinal emitido e lido para medição de distâncias ou temperatura.
[sensores táteis] são sensores ativados através do toque ou colisão, sendo utilizados em robôs como sistema de segurança do sistema.
[sensores ultra-sônicos] são sensores que utilizam pulsos de som de alta freqüência para medir, através do tempo de viagem da onda, a distância do sensor até um objeto próximo.
[tele-operação] é a metodologia utilizada para controlar dispositivos à distância, usualmente recebendo informações do ambiente remoto.