NAVEGAÇÃO DE UM ROBÔ AUTÓNOMO SEMI-INDUSTRIAL USANDO DIFERENTES TÉCNICAS SENSORIAIS

Gil António Nunes Resende

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2008

Candidato: Gil António Nunes Resende, 1030329@isep.ipp.pt

Orientação: Mário Alves, mjf@isep.ipp.pt

Supervisão: Emmanuel Lomba, ECL@isep.ipp.pt

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

4 de Novembro de 2008

Agradecimentos

Desejo manifestar a minha gratidão perante o Prof. Doutor Mário Alves e o Engenheiro Emmanuel Lomba, por terem aceite orientar esta investigação e por o terem feito de uma forma tão motivadora. A orientação, a disponibilidade, o empenho, a confiança depositada e os incentivos dirigidos, contribuíram decisivamente para que a mesma tenha chegado a bom termo.

Resumo

Esta dissertação endereça a navegação de um robô móvel semi-industrial usando diferentes técnicas sensoriais. Foi desenvolvido um sistema de Navegação Autónoma com funcionalidades básicas de navegação e integrado um equipamento Laser Range Finder (LRF) para melhorar a navegabilidade num ambiente semi-estruturado interior. A plataforma robótica utilizada é um robô semi-industrial chamado Robuter. A solução desenvolvida mostra como a utilização do Laser Range Finder permite melhorar a navegação em ambientes com corredores e zonas estreitas, onde as manobras são difíceis. Foi implementado um algoritmo para navegação que pode ser adaptado para outros robôs. Os objectivos iniciais foram atingidos e validados experimentalmente.

Palavras-Chave

Laser Range Finder, Navegação Autónoma, Robuter, Trajectórias Paramétricas de 3º Grau, Robôs móveis

Abstract

This thesis addresses the navigation of a mobile robot semi-industrial sensory using different techniques. It was developed a system of autonomous navigation with basic features of an integrated navigation and equipment Laser Range Finder (LRF) to improve navigability in a semi-structured interior. The robotics platform used is a robot called semi-industrial Robuter. The solution developed shows such as the use of Laser Range Finder will improve navigation in environments with narrow corridors and areas where the maneuvers are difficult. It implemented an algorithm for navigation that can be adapted to other robots. The initial objectives have been met and validated experimentally.

Keywords

Laser Range Finder, Autonomous Navigation, Robuter, third degree parametric trajectory, Mobile Robots

Índice

AGRADECIMENTOS ..... **I**

RESUMO ..... **III**

ABSTRACT ..... **V**

ÍNDICE ..... **VII**

ÍNDICE DE FIGURAS ..... **IX**

ACRÓNIMOS ..... **XV**

1. INTRODUÇÃO **1**

1.1. OBJECTIVOS 1
1.2. CONTEXTO 1
1.3. ORGANIZAÇÃO DESTA DISSERTAÇÃO 2

2. PLATAFORMA TECNOLÓGICA **3**

2.1. ROBUTER 4
2.2. LASER RANGE FINDER 9

3. NAVEGAÇÃO **15**

3.1. NAVEGAÇÃO DE ROBÔS 15
3.2. CARACTERÍSTICAS NÃO-HOLONÓMICAS 17
3.3. PLANEAMENTO DE CAMINHOS 18
3.4. TRANSFORMAÇÃO DE CAMINHOS EM TRAJECTÓRIAS 24
3.5. MAP MATCHING 32

4. APLICAÇÃO DE NAVEGAÇÃO **36**

4.1. VISÃO GERAL 37
4.2. DESCRIÇÃO DA APLICAÇÃO DE NAVEGAÇÃO 39
4.3. ALARME DE COLISÕES 40
4.4. CLIENTE UDP 42
4.5. APLICAÇÃO DE MONITORIZAÇÃO 43
4.6. THREAD PLANEAMENTO_CAMINHOS 45

5. INTEGRAÇÃO DO LASER RANGE FINDER **50**

5.1. VANTAGENS DA UTILIZAÇÃO DO LASER RANGE FINDER 51
5.2. ROTINA DE MAP MATCHING 54

5.3. ROTINA DE RECUPERAÇÃO 56

6. CONCLUSÕES 59

REFERÊNCIAS DOCUMENTAIS 62

Índice de Figuras

Figura 1 Robuter e características 4
Figura 2 Sessão Remota 4
Figura 3 Arquitectura Física da Robuter 5
Figura 4 Blocos constituintes da Robuter 6
Figura 5 Varrimento Angular efectuado pelo Laser Range Finder 10
Figura 6 LMS 200 da SICK 10
Figura 7 Esquema do Setup da comunicação 11
Figura 8 Teste à conexão 12
Figura 9 Procedimentos para obter as Leituras 13
Figura 10 Estrutura de um Telegrama 14
Figura 11 Abordagem relativa à Navegação 16
Figura 12 Ambiente e respectivo *Configuration Space* 19
Figura 13 Trapezoidal Cell Decomposition 20
Figura 14 Método *Quadtrees* 20
Figura 15 Quadtrees e Correspondente Árvore 21
Figura 16 Divisão em Células Regulares 22
Figura 17 Método de *Wavefront* 23
Figura 18 Local Mínimo 23
Figura 19 Caminho com 2 arcos de círculo e um segmento de recta 24
Figura 20 Concatenação de Arcos de Círculo e Segmentos de Recta 24
Figura 21 Aproximação por um arco de elipse 25
Figura 22 Exemplos de Trajectórias obtidas por Polinómios Paramétricos 28
Figura 23 Gráfico referente à trajectória 1 31
Figura 24 Gráfico referente à Trajectória 2 32
Figura 25 Abordagem para *Map Matching* 33
Figura 26 Visão geral de Alto Nível 38
Figura 27 Esquema de Threads 39
Figura 28 Robuter_Nav 40
Figura 29 Rotina Alarme de Colisões 41
Figura 30 Cliente UDP 43

Figura 31 Screenshot da Consola e Aplicação de Monitorização 44
Figura 32 Aplicação em OpenGL pra visualização gráfica 44
Figura 33 Planeamento de Caminhos 46
Figura 34 Exemplo do Método de Wavefront 47
Figura 35 Suavização do caminho 48
Figura 36 Tamanho Virtual do Robô quando usa sonares 52
Figura 37 Tamanho Virtual do Robô quando usa o Laser Range Finder 53
Figura 38 Laser Range Finder montado na Robuter 54
Figura 39 Mapa Local 55
Figura 40 Áreas relevantes para a Rotina de Recuperação 57
Figura 41 Detecção de obstáculos 57
Figura 42 Exemplo do uso da Rotina de Recuperação 58

Acrónimos

API – Application Programming Interface

TOF – Time of Flight

RTAI – Real Time Application Interface

SSH – Secure Shell

CRC – Cyclic Redundancy Check

1. INTRODUÇÃO

1.1. OBJECTIVOS

Este projecto centrou-se no desenvolvimento dum sistema de Navegação Autónoma para uma plataforma robótica semi-industrial com funcionalidades básicas. Pretendeu-se também integrar um equipamento Laser Range Finder para aumentar a sua capacidade sensorial assim como servir de base para futuros trabalhos na mesma plataforma que usem parcial ou integralmente as aplicações desenvolvidas assim como o Hardware implementado. Neste projecto teve-se como premissa um ambiente semi-estruturado.

1.2. CONTEXTO

A plataforma robótica utilizada neste trabalho é um robô com características semi-industriais [9]. É caracterizado como um robô com características não-holonómicas, movido por 2 motores actuados por accionamento diferencial e com 6 sonares instalados na sua periferia. De forma a melhorar as suas capacidades de monitorização do ambiente foi instalado um Laser Range Finder. O sistema operativo é o Linux[13][14] e algumas das ferramentas são as normalmente utilizadas nesse ambiente.

Como este trabalho aborda conceitos de Robótica é importante abordar alguns conceitos relativamente gerais no desenvolvimento de aplicações robóticas. A primeira pergunta relativa à problemática da Navegação Autónoma em Robótica é: Como é que um robô pode decidir que movimentos deve efectuar para desenvolver certas tarefas no ambiente em que se encontra?

A robótica tende cada vez mais a ganhar uma dimensão maior na nossa vida, seja através da sua introdução mais generalizada nos processos industriais, exploração espacial, área de serviços, intervenção em ambientes hostis entre outros. A questão do planeamento de caminhos é uma das componentes mais importantes para a necessária autonomia dos robôs. Mas o planeamento de caminhos não pode ser entendido como uma simples verificação de possíveis colisões ou de evitar colisões. Envolve também a consideração de objectos que se movam, coordenação de vários robôs, lidar com a incerteza, restrições físicas, temporais e geométricas.

A pesquisa na área do planeamento de caminhos na Robótica teve o seu início na década de 60. No entanto, praticamente todos os trabalhos com relevo nesta área seguem princípios compilados por Latombe [2].

A abordagem de Latombe centra-se nos métodos mais comuns de planeamento de caminhos. Estes métodos usam um modelo bidimensional e aproximam o robô e os obstáculos a polígonos. Os principais métodos são *Potencial Fields Methods*, *Cell Decomposition* e *Roadmap Methods*. Uma descrição destes métodos e suas aplicações será apresentado no capítulo 2.

1.3. ORGANIZAÇÃO DESTA DISSERTAÇÃO

Esta dissertação está dividida em 6 capítulos. No primeiro capítulo será feita uma contextualização e declaração dos objectivos previstos para esta Tese. No segundo capítulo é abordada a plataforma robótica utilizada e as suas características principais. No terceiro capítulo serão introduzidos alguns conceitos relativos à Navegação de Robôs. O quarto capítulo endereça a aplicação de Navegação desenvolvida. No quinto capítulo descreve-se a integração do Laser Range Finder salientando as vantagens daí decorrentes. O capítulo seis apresenta algumas notas conclusivas e trabalho futuro.

2. PLATAFORMA TECNOLÓGICA

2.1. ROBUTER

2.1.1. ASPECTOS GERAIS

A plataforma robótica móvel utilizada nesta dissertação é a Robuter[9]. É um robô de forma rectangular com características não-holonómicas, desenvolvido pela empresa Robosoft [8].

A sua locomoção é assegurada por 2 motores DC controlados independentemente. Desta forma podemos usar um accionamento diferencial, permitindo que o veículo possa fazer arcos de círculo, sendo necessário apenas que as velocidades das rodas motrizes sejam diferentes. O controlo é efectuado por um MPC555.

- Comprimento: 102 cm
- Largura: 68 cm
- Altura: 44 cm
- Carga Máxima: 120 kg
- Peso: 150 kg
- Velocidade Máxima: 1.0 m/s

Figura 1 Robuter e características

Esta plataforma utiliza Linux (Red Hat) como sistema operativo, implementando características de tempo real ao usar o Real Time Application Interface (RTAI).

Figura 2 Sessão Remota

Para possibilitar a execução de aplicações remotamente, como podemos observar na Figura 2, possui uma interface Wireless (802.11) através da qual podemos executar uma sessão SSH (Secure Shell) onde se podem lançar aplicações desenvolvidas em linguagem C que estão orientadas para o sistema operativo Linux, usando algumas API's disponíveis.

2.1.2. CAPACIDADES SENSORIAIS

A Robuter está equipada com 6 sonares, sendo que 4 são de curto alcance (200 cm) e 2 são de médio alcance (500 cm). Como melhoramento da sua capacidade sensorial foi também integrado, no decorrer desta dissertação, um Laser Range Finder que permitiu expandir o leque de potenciais aplicações da Robuter. Passou a ter-se uma maior exactidão nas distâncias mediadas além de se poder obter uma percepção muito mais pormenorizada do ambiente circundante, dado que o Laser Range Finder permite fazer medições com uma abertura angular de 180º com resoluções de 1º, 0.5º e 0.25º e com um alcance de 80 metros.

A alimentação da Robuter é efectuada por 4 baterias de 12 V ligadas em série permitindo assim 48 V embora existam outros valores de alimentação a alimentar certos componentes.

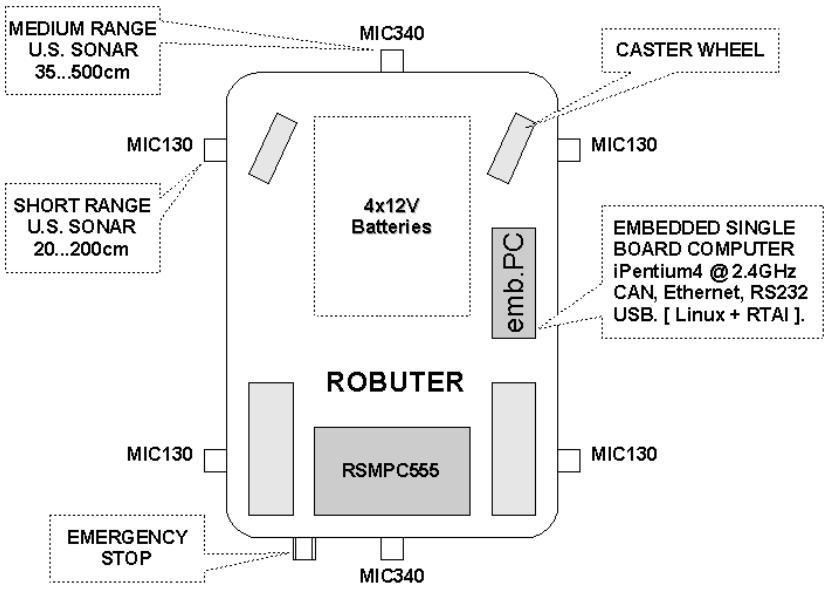

Figura 3 Arquitectura Física da Robuter[19]

2.1.3. DIAGRAMA REPRESENTATIVO DA ARQUITECTURA FÍSICA DA ROBUTER

A Figura 4 apresenta um esquema representativo da arquitectura electromecânica da Robuter.

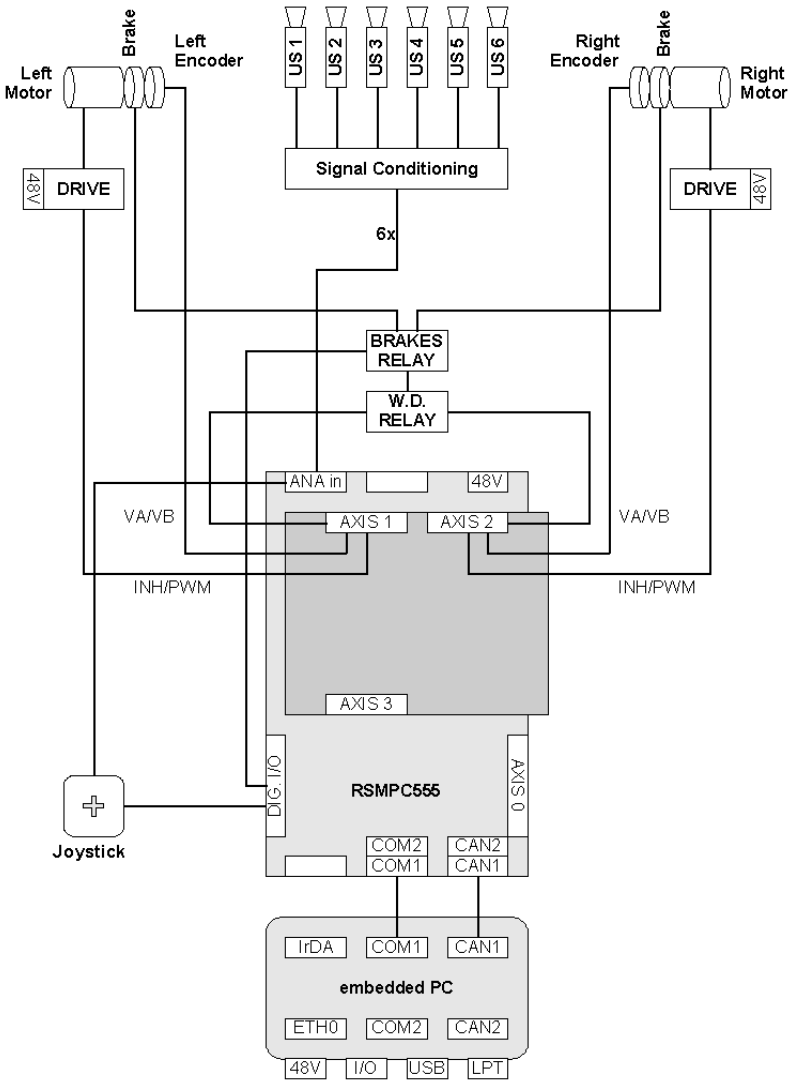

Figura 4 Blocos constituintes da Robuter[19]

Analisando a arquitectura, podemos destacar a placa RSMPC555, que pode fazer o controlo até 4 motores. Neste caso, controla apenas 2. A comunicação entre o RSMPC555 e o computador (*embedded pc*) é assegurada por uma ligação CAN e uma ligação por porta série. A comunicação efectuada com o Laser Range Finder é efectuada através duma porta série.

Como podemos observar na Figura 3, o computador utilizado tem um processador *Pentium* 4, e foi desenhado para ocupar pouco espaço. Estes componentes fornecem uma capacidade de processamento elevada comparativamente a outras soluções adoptadas em projectos semelhantes.

Existe ainda um botão de emergência na Robuter que permite fazer o Reset à placa MPC555. Isto pode ser feito por acção humana, quando existe necessidade de parar o Robô para evitar possíveis colisões durante testes, por exemplo.

2.1.4. PLATAFORMA DE DESENVOLVIMENTO DE SOFTWARE

O sistema operativo utilizado é o Linux Red Hat 9.0 onde foi instalado um "patch" que permite obter características de um sistema de tempo real. A linguagem utilizada é a Linguagem C, sendo essa a linguagem em que o sistema operativo foi construído de base. Ao usarmos o Linux, passamos a ter à nossa disponibilidade uma API para implementação de Threads, uso de sinais do sistema operativo, semáforos, memória partilhada, entre outras. A possibilidade de ter todas as capacidades dum sistema operativo para o desenvolvimento de aplicações permitiu uma integração de conhecimentos anteriores e também a aprendizagem de outros conhecimentos necessários para o desenvolvimento da aplicação de navegação assim como a integração dos blocos fundamentais desenvolvidos. Em certo momento tornou-se necessário desenvolver uma ferramenta de depuramento de erros e de situações anómalas. Nesse particular, utilizou-se OpenGL para se desenvolver uma aplicação gráfica que permitisse obter melhor e diferente informação sobre o que acontecia com a plataforma robótica.

2.1.5. ESTRUTURAS QUE PERMITEM INTERACÇÃO COM AS CAMADAS INFERIORES

A Robosoft desenvolveu uma camada de abstracção sobre o Hardware da Robuter. Foram criadas estruturas que, através da leitura das mesmas, permite obter os valores da odometria, da distância lida pelos sonares entre outros dados relevantes. A estrutura onde esses valores se encontram está definida a seguir:

```
typedef struct {
  int inputs_o0;
  int encR_encoderR_o0;
  int encL_inv_encL_o0;
  int odom_X;
  int odom_Y;
  int odom_T;
  int jstk_mul2_o0;
  int jstk_mul0_o0;
  int count_cnt;
  int mic_USF_o0;
  int mic_USFL_o0;
```

```
  int mic_USRL_o0;
  int mic_USR_o0;
  int mic_USRR_o0;
  int mic_USFR_o0;
  int time;
  unsigned char sem;
} shm_seg0_def;
```

Existe ainda uma estrutura que permite a alteração dos valores de velocidade angular, velocidade linear do Robô e aceleração. A estrutura foi definida como:

```
typedef struct {
  int shmRDRobuLAB_v;
  int shmRDRobuLAB_w;
  int shmRDRobuLAB_acc;
  int shmRDRobuLAB_rst;
  int time;
  unsigned char sem;
} shm_seg1_def;
```

Escrevendo nesta estrutura, podemos actuar alguns dos parâmetros do robô, existindo obviamente algumas limitações no que toca às acelerações. Esta estrutura está protegida com um semáforo, impedindo acessos simultâneo. Uma variável do tipo desta estrutura é depois declarada num local de memória conhecido e estático.

A Robosoft criou um grupo de funções que permite ler e escrever nas estruturas definidas anteriormente. A função que permite escrever a estrutura relativa aos parâmetros de velocidade e aceleração da Robuter é:

```
void LAB150Move(toLAB150ShmDef * to, double v,
double w, double acc)
```

A partir desta estrutura torna-se mais simples modificar os seus parâmetros na aplicação de navegação desenvolvida. Torna-se necessário focar o cuidado na escolha dos valores da aceleração, sendo que o sistema físico não consegue fornecer todos os valores possíveis de aceleração. Todos os factos relevantes para a utilização destas estruturas e das funções foram estudados em Pomiers [10].

2.2. LASER RANGE FINDER

2.2.1. PRINCÍPIO DE FUNCIONAMENTO

O LRF proporciona uma capacidade sensorial muito desenvolvida e com alta resolução. Sendo uma tecnologia usada em robôs de maior custo já há muitos anos, a sua utilização está cada vez mais difundida em robôs de baixo custo. A sua alta resolução e alta frequência torna-os ideais para muitas aplicações tais como *Map Matching* entre outras. O equipamento LRF utilizado tem potencialidades tanto em ambientes fechados (ex: salas, laboratórios) como em ambientes *outdoor*. Esta capacidade de adaptação ao ambiente pode ser executada alterando a sua resolução e a distância máxima que pode medir. Em ambientes fechados, podemos medir até 8 metros com uma resolução na ordem dos mm. No caso de ambientes abertos, podemos efectuar medições até 80 metros e com uma resolução na ordem dos cm. Logo, o espectro de aplicações deste equipamento é muito vasto. Neste caso, foi usado um Laser Range Finder da SICK que pode observar-se na Figura 6.

Um dos pontos a considerar quando pretendemos usar um Laser Range Finder é a sua necessidade de capacidade de processamento para usar a plenitude das suas capacidades.

O princípio de funcionamento do LRF baseia-se no TOF (*Time of Flight*). Um único pulso é enviado e ao ser reflectido num obstáculo é detectado. O tempo entre a emissão e recepção permite calcular a distância entre o objecto e o emissor. Através dum espelho integrado no Laser, podemos fazer um varrimento angular obtendo uma medição em 2 dimensões. As opções de varrimento dependem da aplicação que queremos implementar. Podemos obter um varrimento angular de 180º ou de 100º, com intervalos de 1º e 0.5º para o caso de 180º e com intervalos de 0.5º e 0.25º para o caso de 100º. Como o LRF só consegue fazer intervalos de 1º, quando pretendemos usar a opção de intervalos de leituras de 0.5º em 0.5º, na realidade estamos a esperar que o LRF faça 2 varrimentos angulares desfasados de 0.5º no início.

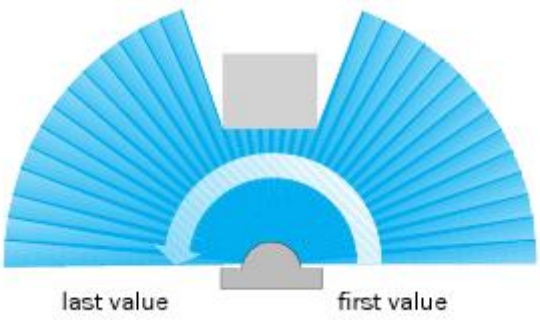

Figura 5 Varrimento Angular efectuado pelo Laser Range Finder

Os principais benefícios são:

- A detecção dos objectos independentemente da cor e da textura da superfície
- Resolução obtida e distância máxima
- Fiabilidade

Figura 6 LMS 200 da SICK

2.2.2. COMUNICAÇÃO COM O LASER RANGE FINDER

A comunicação com o Laser Range Finder pode ser efectuada usando 2 protocolos. Através dum jumper, podemos escolher entre RS232 (por defeito) e RS422. O RS422 deve ser usado quando pretendemos obter elevadas velocidades de transmissão.

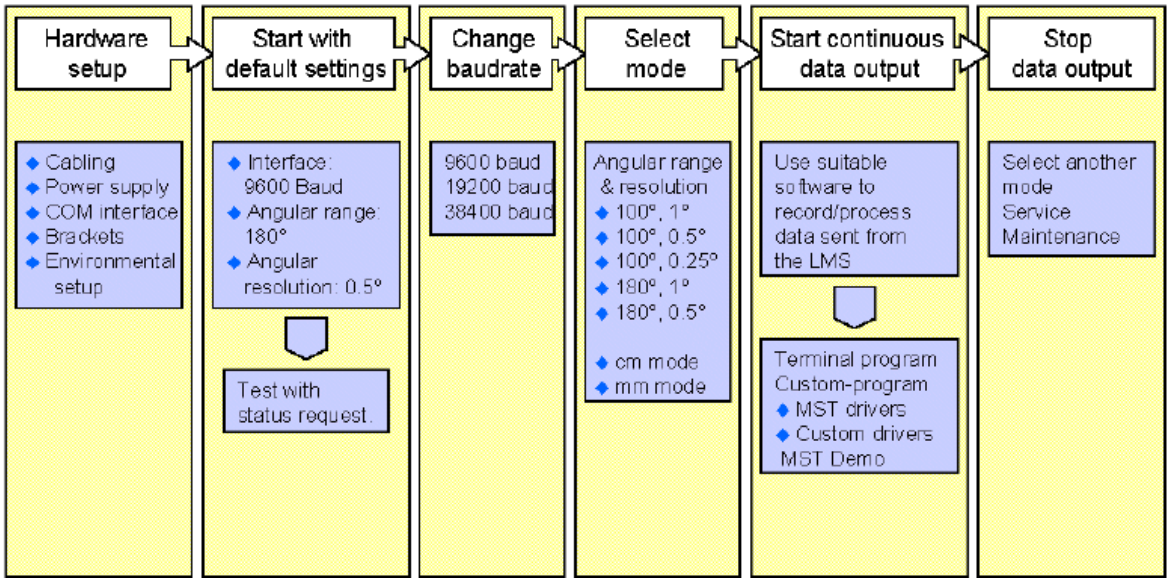

Figura 7 Esquema do Setup da comunicação

A comunicação do LRF com o computador faz-se através do envio de tramas que estão completamente descritas no manual do fabricante [6]. Existem 2 tipos de tramas: PC para o Laser e do Laser para o PC. As primeiras tramas têm sempre comandos que serão executados pelo LMS. As segundas tramas são sempre uma resposta a telegramas enviados pelo PC. Qualquer trama termina com um CRC (*Checksum*).

Pode-se observar na Figura 7 o método de utilização do LRF. Antes de tudo, testa-se a conexão física ao LRF. Configurámos a porta Série para a velocidade 9600 bps que é a escolhida por defeito quando o LRF sofre um Reset ou se liga pela primeira vez.

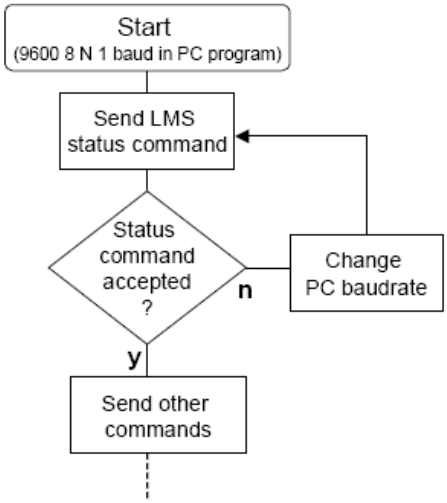

Figura 8 Teste à conexão

Após o teste da conexão, podemos enviar comandos para configurar outras velocidades de comunicação, outros modos de funcionamento, entre outros como se observa na Figura 8. Após as configurações pretendidas, colocamos o LRF em modo de envio contínuo de dados. A partir deste momento, interessa-nos fazer o *parsing* da trama, verificar o CRC e retirar apenas os dados que nos interessam.

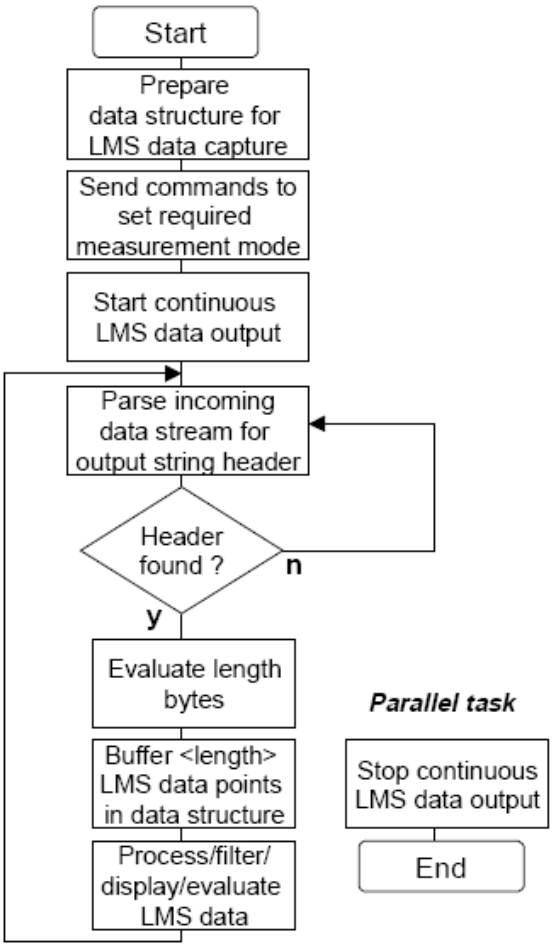

Figura 9 Procedimentos para obter as Leituras

Seguindo a diagrama representado na Figura 9, podemos obter as leituras do LRF e utilizá-las como entendermos.

2.2.3. ESTRUTURA DAS TRAMAS

As tramas têm sempre a mesma estrutura quer sejam enviados pelo PC ou pelo LRF. A estrutura da trama pode ser observada na Figura 10. O primeiro campo é um *Start Byte*. Permite identificar o inicio da trama. Como a *Data* é um campo de tamanho variável consoante o tipo de comando e resposta, torna-se necessário o campo *Length* que nos permite saber o número de bytes do campo *Data*.

Para confirmação dos dados recebidos temos o campo CRC (*Checksum*). O cálculo deste CRC pode ser confirmado no manual [6].

	Frame				Commands and data		Frame	
Description	STX	Address	Length		Command/ Response	Data	Checksum	
Byte position	1	2	3	4	5	6 to n	n+1	n+2

Figura 10 Estrutura de um Telegrama

3. NAVEGAÇÃO

3.1. NAVEGAÇÃO DE ROBÔS

A navegação de robôs é um tema estudado já há muitas décadas. Existem várias técnicas compiladas por Latombe [2] onde são apresentados os métodos mais comuns que usam uma representação bidimensional, permitindo aproximar o robô e os obstáculos a polígonos.

A navegação pode ser classificada como global quando o ambiente de trabalho é totalmente conhecido, ou local quando o ambiente só é conhecido parcialmente ou eventualmente totalmente desconhecido. É composta por etapas que envolvem medições do modelo do ambiente, a localização, o planeamento e execução da trajectória. O modelo do ambiente é apoiado pelo uso de sensores ou pode estar guardado em memória. No entanto, durante a execução de trajectórias é importante usar uma ou mais capacidades sensoriais para detectar alterações no mapa e para permitir uma resposta adequada ao novo obstáculo detectado. Uma tarefa muito

comum dos robôs móveis é a navegação em ambientes interiores. Nestas situações é comum manipular as informações lidas nos sensores de forma a construir ou corrigir mapas, determinar a posição no mapa ou atingir certas configurações. O uso de vários tipos de sensores está intimamente ligado ao objectivo proposto para o robô móvel, sendo possível obter robôs móveis com capacidades interessantes a baixo custo.

Uma das formas de resolver o problema da navegação é adoptar um algoritmo que é usado em diversos trabalhos e que é descrito pela Figura 11.

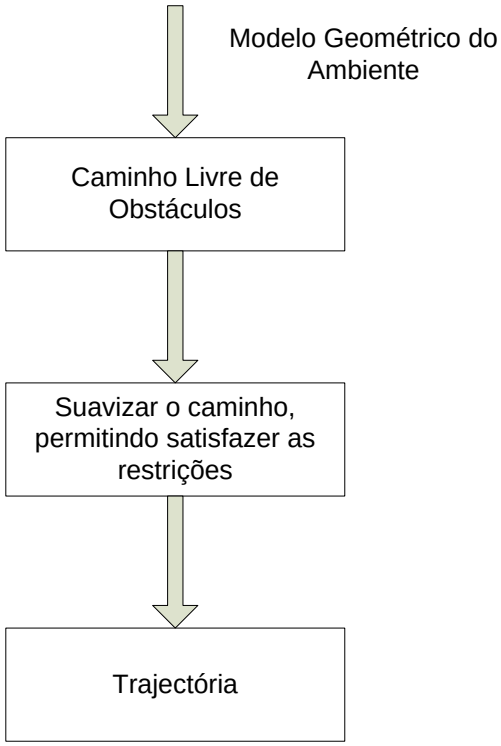

Figura 11 Abordagem relativa à Navegação

Primeiro, tendo como entrada o Modelo Geométrico do Ambiente, podemos obter um caminho livre de obstáculos utilizando diversos métodos: *Roadmaps*, *Potencial Fields* e *Cell Decomposition*. Todos estes métodos estão descritos em Latombe [2], sendo que alguns têm sido implementados de formas diferentes e podemos observar ainda que existem outros que não estão descritos aqui. Assim que obtemos um caminho livre de obstáculos, torna-se importante adaptar esse caminho à realidade do robô. Existem variados tipos de robôs móveis, com restrições com raios mínimos de curvatura entre outras. Noutra secção, serão discutidas as restrições não-holonómicas. Após obtermos um

caminho suavizado que possa ser efectuado pelo robô em causa, devemos obter perfis de velocidade para o caminho em questão. Estes perfis de velocidade terão de ter em conta o tempo da viagem e poderão considerar ainda pontos intermédios para fragmentar o caminho. Mais adiante, serão explicadas algumas formas de obtenção das trajectórias.

3.2. CARACTERÍSTICAS NÃO-HOLONÓMICAS

As técnicas de planeamento de caminhos levam em consideração os obstáculos existentes de forma a obtermos um caminho livre de colisões, sendo chamado de caminho geométrico. Como é conhecido, os robôs móveis possuem restrições cinemáticas. Estas restrições têm o nome de restrições não-holonómicas. Devido a estas restrições, os caminhos calculados pelos métodos comuns podem ser caminhos que o robô não consegue executar. A navegação autónoma em veículos com restrições não-holonómicas é mais difícil do que em veículos que só possuem restrições holonómicas. Por exemplo, um robô móvel com restrições idênticas às de um carro tem 2 controlos: Linear e Angular. No entanto, ele move-se num ambiente com 3 dimensões : X , Y e θ (Orientação). Logo, qualquer caminho encontrado pode não corresponder a um caminho executável pelo robô em questão. Podemos obter caminhos livres de colisões, no entanto, considerando estas restrições podemos obter caminhos que se tornem impossíveis de executar pelo robô.

Desta forma, ao abordar a realidade dos veículos não-holonómicos, temos que ter em conta que poder-se-á não atingir exactamente o objectivo (posição e orientação) mas sim uma vizinhança nesse ponto.

Matematicamente, as restrições não-holonómicas são caracterizadas por uma equação não integrável que envolve as derivadas no tempo dos parâmetros de configuração.

Equação cinemática:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}$$

Restrição Não-Holonómica:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0$$

Um dos métodos mais comuns de planeamento para robôs condicionados por este tipo de restrição é dividir o problema em várias fases. Primeiro, construímos um caminho geométrico, não tendo em conta as restrições não-holonómicas, utilizando qualquer um dos métodos propostos por Latombe [2]. De seguida, devemos transformar o caminho geométrico em um caminho que possa ser percorrido pelo robô. Este tipo de abordagem pode levar a um caminho ineficiente. A escolha da heurística pode ser determinante consoante o ambiente. Este algoritmo está representado na Figura 11.

Existem variadas formas para transformar os caminhos geométricos em caminhos que compreendam as restrições não-holonómicas. Muito do trabalho sobre planeamento de trajectórias para robôs móveis não-holonómicos foi desenvolvido por Laumond [1].

3.3. PLANEAMENTO DE CAMINHOS

Configuration Space

Neste trabalho presume-se que o ambiente em questão é totalmente conhecido. No entanto, nem todos os pontos do ambiente podem ser atingidos, mas neste caso devemos ter em consideração que além dos pontos a atingir temos de considerar a orientação do robô. Como podemos ver na Figura 12, o modelo do ambiente é adaptado consoante o tipo de

robô para depois podermos calcular os caminhos livres de obstáculos e apenas com configurações que o robô possa atingir. É com este modelo do ambiente que a rotina de *Path Planning* irá trabalhar.

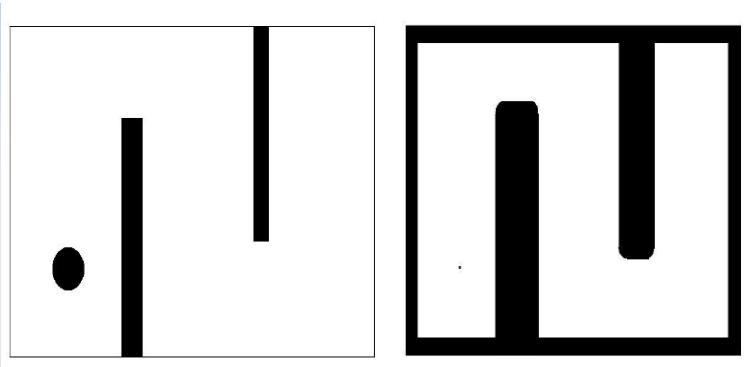

Figura 12 Ambiente e respectivo *Configuration Space*

A partir deste momento, o robô é considerado como se de um ponto se tratasse mas apresentando uma orientação que restringe os movimentos possíveis.

Existem muitos métodos para obter caminhos livres de obstáculos estáticos. Entre os métodos descritos por Latombe, temos 3 grandes grupos: *Potencial Fields Methods*, *Cell Decomposition* e *Roadmap Methods.* Após um estudo aprofundado dos diversos métodos compilados por Latombe, escolheu-se como método principal a divisão do ambiente em células regulares. Dentro deste grande grupo que é a Decomposição em Células, existem várias formas de abordar este problema. As principais são: *Trapezoidal Cell Decomposition ,Regular Grids e Quadtrees.*

No método *Trapezoidal Cell Decomposition* o ambiente é dividido em secções críticas livres formando trapezóides. O objectivo passa por partir de uns para os outros obtendo um caminho livre de obstáculos. Uma representação deste método está demonstrada na Figura 13.

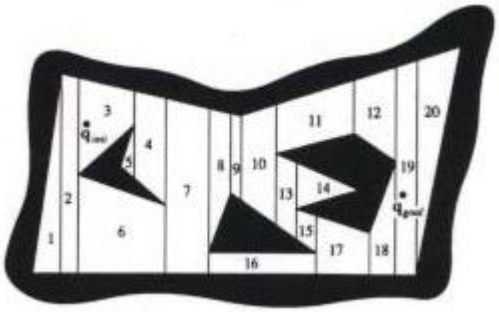

Figura 13 Trapezoidal Cell Decomposition

No método dos *Quadtrees* conseguimos obter resoluções muito maiores nos locais mais importantes: periferia dos obstáculos. Para isso usam-se *Quadtrees* que são estruturas de árvores que permitem aumentar o número de células para representar certas zonas do ambiente. Como podemos observar na figura abaixo, compreende-se que a periferia dos obstáculos é representada por muito mais células que as zonas livres do mapa. Isto permite obter um caminho livre de obstáculos, caso ele exista.

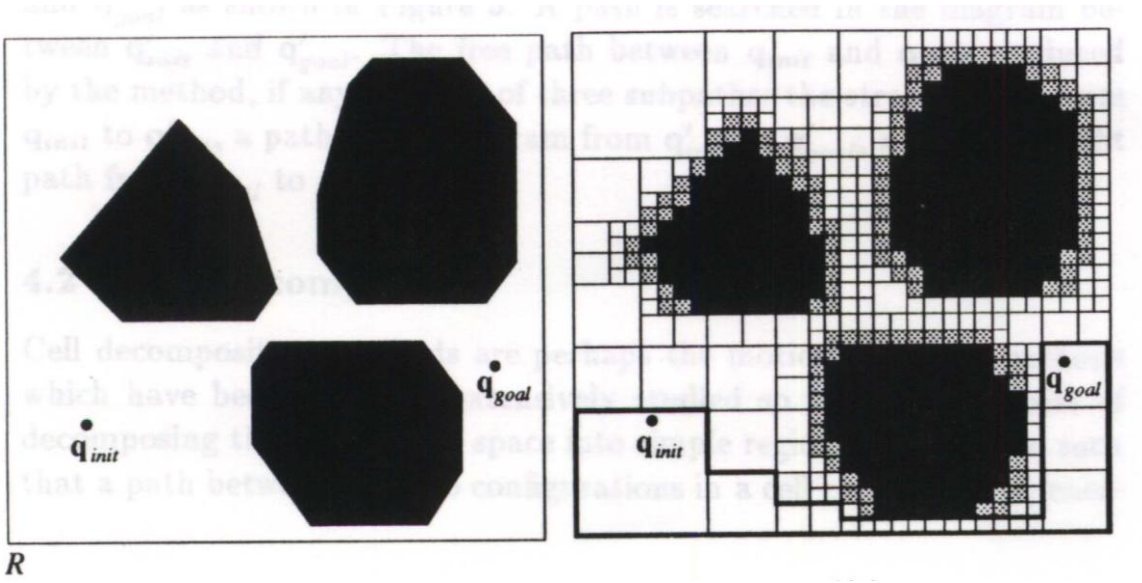

Figura 14 Método *Quadtrees*

Para construir o mapa existem vários algoritmos. Pode-se estipular um limite mínimo do tamanho da célula ou um número máximo de divisões das células em questão de forma a limitar o esforço computacional do método. No fundo, ter-se-á uma árvore representativa do respectivo mapa que permite outra forma de visualizar a divisão do ambiente.

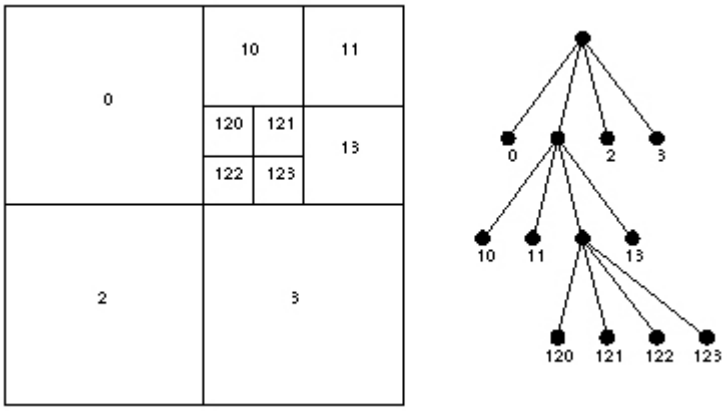

Figura 15 Quadtrees e Correspondente Árvore

Outra forma de obter a divisão do mapa em células é utilizar células regulares. Desta forma, todas as células do mapa têm o mesmo tamanho. Isto implica que quanto maior o número de células melhor será a representação do mapa. Consoante o tipo de ambiente, o número de células que representam o mapa pode influenciar a obtenção de caminhos livres. Se as células forem muito grandes, poderemos não obter um caminho livre. No entanto, se o número de células for muito grande, o algoritmo torna-se mais pesado.

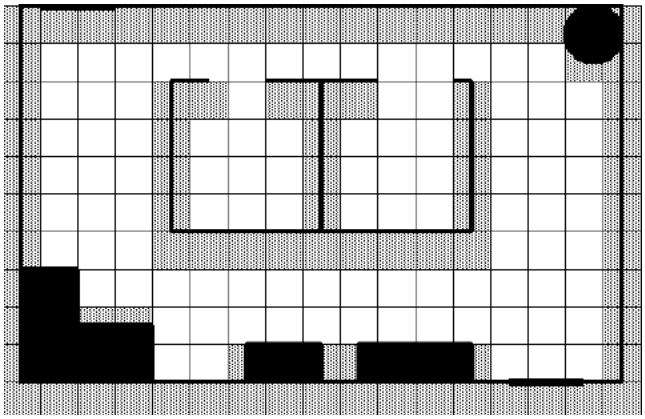

Figura 16 Divisão em Células Regulares

O ambiente usado no robô foi decomposto em Células Regulares. No entanto, a partir do momento que dividimos o ambiente em células regulares, podemos adoptar diversos métodos para calcular o caminho. Após a análise de vários métodos, foi escolhido o método *Wavefront[12]*. Este método tem o seu funcionamento representado na Figura 17. Consideramos que cada célula tem um valor numérico associado. O primeiro passo é atribuir um valor muito alto às células que representam obstáculos. De seguida atribui-se um valor muito baixo ao ponto objectivo e atribui-se um valor intermédio ao ponto actual (de partida). Após estes passos, atribuem-se valores às restantes células, sendo que as células mais longe do ponto objectivo terão valores cada vez maiores. O objectivo passa por obter uma distribuição de valores nas células idêntica à da Figura 17. A partir do momento em que temos todas as células com valores, partimos da célula inicial e vamos avaliando a sua periferia, procurando a célula com valor mais baixo. Começamos a construir o caminho que nos poderá permitir atingir a célula final. Então, utilizando o raciocínio acima descrito, procurando sempre a célula com menor valor na periferia da célula actual podemos construir um vector com as células que temos que percorrer para atingir o objectivo.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15
1	36	350	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14
2	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13
3	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12
4	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
5	22	21	20	19	18	17	500	500	500	500	500	500	500	500	500	500	10	10	10	10	10	10	10
6	22	21	20	19	18	17	500	500	500	500	500	500	500	500	500	500	9	9	9	9	9	9	9
7	22	21	20	19	18	17	500	500	500	500	500	500	500	500	500	500	8	8	8	8	8	8	8
8	22	21	20	19	18	17	500	500	500	500	500	500	500	500	500	500	7	7	7	7	7	7	7
9	22	21	20	19	18	17	500	500	500	500	500	500	500	500	500	500	6	6	6	6	6	6	6
10	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	5	5	5	5	5
11	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4	4	4
12	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3	3	4
13	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	3	4

Ponto Objectivo

Ponto Actual

Figura 17 Método de *Wavefront*

No entanto, nem sempre se consegue atingir o ponto objectivo. Se tivermos uma situação em que estamos perante a situação apresentada na figura abaixo, podemos ficar presos num Mínimo Local. Sendo assim, recorreu-se a pontos intermédios no mapa que só são chamados em caso de ficarmos presos num local mínimo. O objectivo desses pontos é desviar o robô desse local mínimo identificado e depois calcular o caminho normalmente.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	23	22	21	20	19	18	15	16	17	18	19	20	19	18	17	16	19	19	19	19	19	19	19
1	22	21	20	19	18	17	15	16	17	18	350	20	19	18	17	16	18	18	18	18	18	18	18
2	21	20	19	18	17	16	15	16	17	18	19	20	19	18	17	16	17	17	17	17	17	17	17
3	20	19	18	17	16	15	15	16	17	18	18	20	19	18	17	16	16	16	16	16	16	16	16
4	19	18	17	16	15	14	15	16	17	18	17	20	19	18	17	16	15	15	15	15	15	15	15
5	18	17	16	15	14	13	500	500	14	15	16	15	14	13	500	500	14	14	14	14	14	14	14
6	17	16	15	14	13	12	500	500	13	14	15	14	13	12	500	500	13	13	13	13	13	13	13
7	16	15	14	13	12	11	500	500	12	13	14	13	12	11	500	500	12	12	12	12	12	12	12
8	15	14	13	12	11	10	500	500	500	500	500	500	500	500	500	500	11	11	11	11	11	11	11
9	14	13	12	11	10	9	500	500	500	500	500	500	500	500	500	500	10	10	10	10	10	10	10
10	13	12	11	10	9	8	7	6	6	4	4	4	4	6	7	8	9	10	11	12	13	14	15
11	13	12	11	10	9	8	7	6	5	3	3	3	4	6	7	8	9	10	11	12	13	14	15
12	13	12	11	10	9	8	7	6	5	3	2	3	4	6	7	8	9	10	11	12	13	14	15
13	13	12	11	10	9	8	7	6	5	3	3	3	4	6	7	8	9	10	11	12	13	14	15

Ponto Objectivo

Ponto Actual

Figura 18 Local Mínimo

3.4. TRANSFORMAÇÃO DE CAMINHOS EM TRAJECTÓRIAS

Após a geração de caminhos que evitem obstáculos, o objectivo é utilizar um método de adaptação do caminho geométrico obtido em trajectórias executáveis pelo robô em causa.

Após um estudo meticuloso das formas de transformação de caminhos em trajectórias, foram encontradas 3 abordagens principais: Abordagem Clássica, Aproximação por elipse e Geração de Caminhos Ponto – a – Ponto [3] usando polinómios paramétricos.

3.4.1. ABORDAGEM CLÁSSICA

A abordagem clássica para obter o caminho geométrico entre duas configurações consiste em concatenar arcos de círculo e segmentos de recta. Este comportamento foi inicialmente proposto por *Reeds* e *Shepp[18]*. Pressupõe que o robô se movimenta para a frente e para trás e que possua limitação no raio de curvatura.

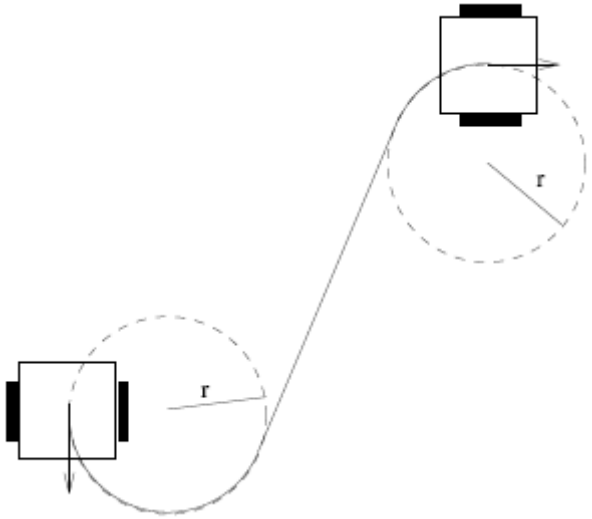

Figura 19 Caminho com 2 arcos de círculo e um segmento de recta

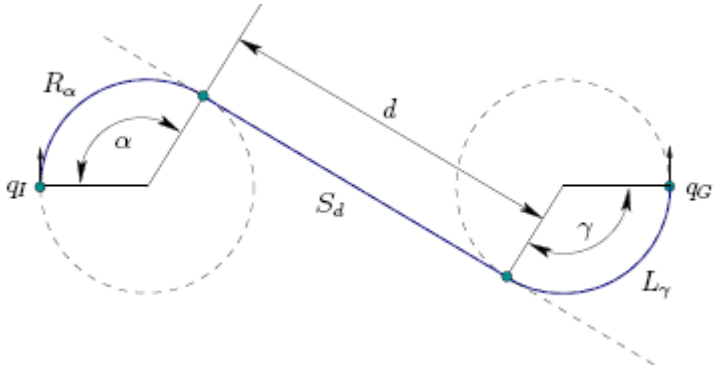

Figura 20 Concatenação de Arcos de Círculo e Segmentos de Recta

A abordagem clássica, quando aplicada a robôs móveis com accionamento diferencial, engloba 3 fases.

- Rotação em torno do próprio eixo de forma a apontar para a posição desejada
- De seguida, efectua um movimento em linha recta até atingir a posição final
- Novamente, executa uma rotação em torno do próprio eixo até atingir a orientação desejada

Podemos observar as fases descritas na Figura 20 e na Figura 19.

Neste caso, o robô em causa tem restrições não-holonómicas, o que implica que não possa rodar sobre si próprio. No entanto, pode efectuar um arco de círculo com um raio que equivale a metade da distância entre as rodas motrizes. Desta forma, adapta-se o raciocínio acima descrito para a realidade deste robô. No entanto, torna-se importante obter caminhos que permitam efectuar arcos de círculo com raios iguais ou preferencialmente superiores ao raio mínimo.

3.4.2. APROXIMAÇÃO POR ELIPSE

Outra forma de atingir o objectivo seria usar um arco de elipse.

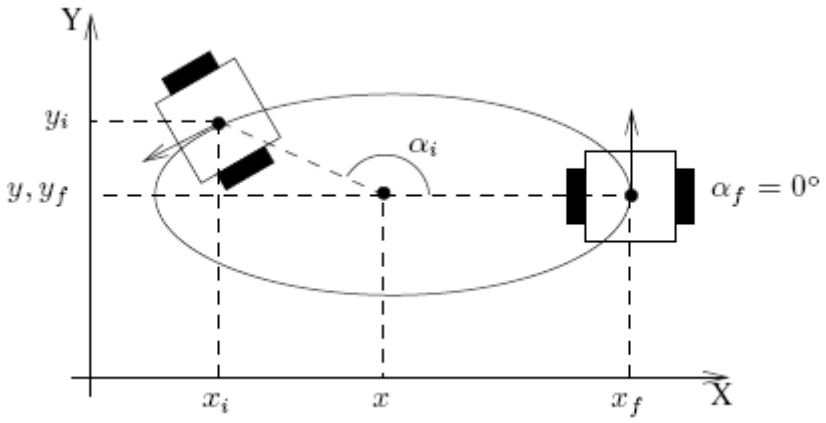

Figura 21 Aproximação por um arco de elipse

A equação duma elipse é dada por:

$$\left(\frac{(x - xc)}{a}\right)^2 + \left(\frac{(y - yc)}{b}\right)^2 = 1$$

Após definida a equação da elipse, o objectivo passa por conhecer as configurações inicial e final, x_i, y_i, θ_i e x_f, y_f, θ_f, respectivamente. A pergunta que se deve fazer é: qual a elipse que passa pelos pontos x_i, y_i e x_f, y_f e que apresentem as orientações pretendidas naqueles pontos.

$$x = x_c + a\cos\alpha$$
$$y = y_c + b\sin\alpha$$
$$\theta = \arctan\frac{(dx/d\alpha)}{(dy/d\alpha)} = \arctan\frac{(a\sin\alpha)}{(-b\cos\alpha)}$$

Substituindo os valores das configurações inicial e final:

$$x_i = x_c + a\cos\alpha_i$$
$$y_i = y_c + b\sin\alpha_i$$
$$\theta_i = \arctan\frac{(a\sin\alpha_i)}{(-b\cos\alpha_i)}$$
$$x_f = x_c + a\cos\alpha_f$$
$$y_f = y_c + b\sin\alpha_f$$
$$\theta_f = \arctan\frac{(a\sin\alpha_f)}{(-b\cos\alpha_f)}$$

O uso desta abordagem não é simples, devido à existência de equações não-lineares.

3.4.3. POLINÓMIOS PARAMÉTRICOS

O problema no planeamento de caminhos é a divisão de um caminho maior livre de obstáculos em conjuntos de configurações que possam ser atingidas respeitando as restrições cinemáticas do robô. Para podermos resolver este problema podemos usar o método dos polinómios paramétricos. Estes polinómios apresentam configurações

intermédias que dependem de parâmetros de entrada. Neste caso, foram usadas trajectórias paramétricas de terceiro grau. São mais complexas que as trajectórias paramétricas de segundo grau, no entanto, apresentam melhores resultados, como seria de esperar. As diversas configurações estão ligadas a um parâmetro λ adimensional e não relacionado com o tempo.

O parâmetro λ varia entre os valores 0 e 1. Quando λ é igual a 0 significa que estamos na presença da configuração inicial e quando λ é igual a 1 estamos na presença da configuração final.

Os polinómios paramétricos de terceiro grau são obtidos por:

$$\begin{aligned} x(\lambda) &= a_0 + a_1\lambda + a_2\lambda^2 + a_3\lambda^3 \\ y(\lambda) &= b_0 + b_1\lambda + b_2\lambda^2 + b_3\lambda^3 \\ \theta(\lambda) &= \tan^{-1}\left(\frac{b_1 + b_2\lambda + b_3\lambda^2}{a_1 + a_2\lambda + a_3\lambda^2}\right) \end{aligned}$$

Para λ igual a 0 obtemos (x_i, y_i, θ_i) e para λ igual a 1 obtemos (x_f, y_f, θ_f).

$$(x(0), y(0), \theta(0)) = (x_i, y_i, \theta_i) \quad para \ \lambda = 0$$

$$(x(1), y(1), \theta(1)) = (x_f, y_f, \theta_f) \quad para \ \lambda = 1$$

Para obter os coeficientes:

$$\begin{cases} x(0) = a_0 \\ y(0) = b_0 \\ d(0) = \tan\theta(0) = \dfrac{b_1}{a_1} \\ x(1) = a_0 + a_1 + a_2 + a_3 \\ y(1) = b_0 + b_1 + b_2 + b_3 \\ d(1) = \tan\theta(1) = \dfrac{b_1 + 2b_2 + 3b_3}{a_1 + 2a_2 + 3a_3} \end{cases}$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3d_f & 2d_f & d_f \\ 0 & 0 & 1 & 0 & 0 & -d_i \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_3 \\ b_2 \\ b_1 \\ a_3 \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} \Delta y \\ 3\Delta y \\ 0 \\ \Delta x \end{bmatrix}$$

Este sistema está definido para situações onde $\theta_i \neq \pi/2$ e $\theta_f \neq \pi/2$. Como é possível de observar, temos 4 equações mas temos 6 parâmetros. Logo temos que arbitrar 2 dos 6 parâmetros. Com parâmetros diferentes podemos obter trajectórias com características diferentes. É aqui que reside a flexibilidade deste método e a possibilidade de optimização para trajectórias que se aproximam das características que pretendemos.

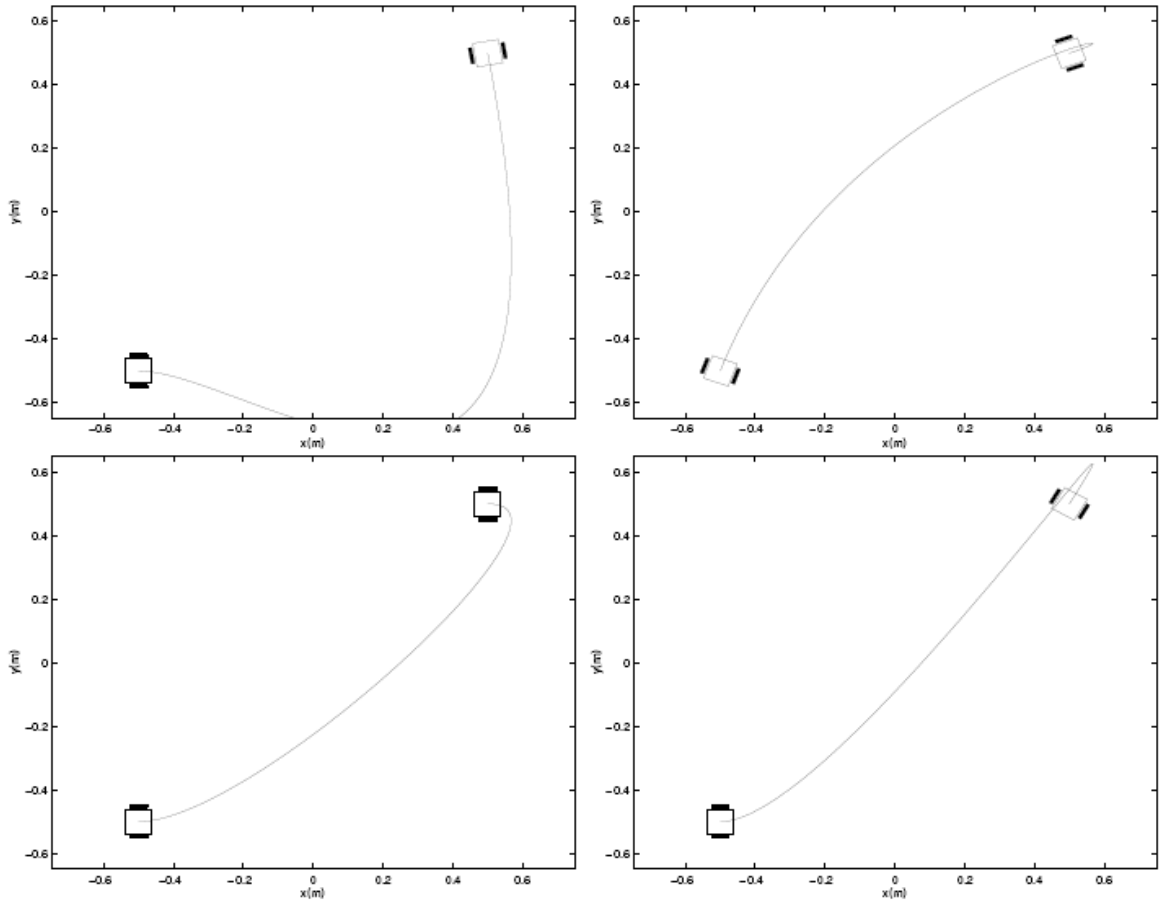

Figura 22 Exemplos de Trajectórias obtidas por Polinómios Paramétricos

Como é possível observar na Figura 22, podemos obter vários tipos de trajectórias consoante os parâmetros de entrada e as configurações iniciais e finais pretendidas. No

entanto, torna-se complicado obter trajectórias porque a optimização depende das configurações pretendidas assim como da possibilidade de efectuar a trajectória, quer seja por a trajectória provocar colisões com obstáculos ou por obrigar o robô a efectuar arcos de círculo cujo raio é menor do que o raio mínimo que o robô consegue efectuar.

Existe ainda outro cuidado a ter que são os casos especiais que aparecem devido às orientações de entrada e de saída serem iguais a $\pm \pi/2$.

No caso de $\theta_i = \pm\pi/2$ e $\theta_f = \pm\pi/2$ ocorrem singularidades uma vez que $\tan(\theta_i)$ e $\tan(\theta_f)$ tendem para infinito.

Como nem sempre é possível obter bons caminhos, torna-se importante criar uma rotina que permita escolher as melhores trajectórias e os melhores coeficientes e aqui reside uma dificuldade acrescida. Pode escolher-se uma faixa de valores para os coeficientes que vão ser arbitrados. E deve ainda fazer-se uma filtragem das trajectórias através dos raios mínimos da trajectória em questão, de forma a podermos aplicar a trajectória ao robô. Outro critério de optimização passa pelo custo da trajectória. Aplicando o critério do custo e dos raios mínimos conseguem-se obter melhores resultados. No entanto, estes nem sempre são úteis devido às configurações obtidas terem orientações despropositadas nalguns casos. Para melhorar isto, foi implementada uma rotina para filtrar as trajectórias que tenham orientações contrárias às pretendidas. Logo se pretende que os valores da orientação cresçam ou diminuam ao longo da trajectória. É necessário rejeitar as trajectórias obtidas que não respeitam esse critério.

Como a obtenção de boas trajectórias nem sempre é possível, a aplicação desenvolvida usa um método idêntico ao de *Reeds & Sheep[18]* quando se falha obtenção de uma boa trajectória.

Exemplos de Trajectórias Paramétricas:

Valores de Entrada no algoritmo desenvolvido que calcula as trajectórias paramétricas:

Xi = 3.35	Yi = 2.4	Ti = 0
Xf = 3.85	Yf = 3.45	Tf= 1.57

A trajectória calculada foi:

X (m)	Y (m)	T - Orientação (Radianos)
3,35	2,4	0
3,49	2,43	0,48
3,59	2,52	0,86
3,68	2,65	1,1
3,74	2,8	1,25
3,79	2,97	1,35
3,82	3,13	1,42
3,84	3,27	1,47
3,85	3,39	1,5
3,85	3,45	1,53
3,85	3,45	1,57

Na rotina que foi implementada, foram introduzidas a configuração inicial e a configuração final. Foi então calculada uma trajectória paramétrica. Pode observar-se o seu resultado na Figura 23. Esta trajectória é o resultado da refinação de muitas trajectórias onde o custo da trajectória é um dos factores a ter em consideração.

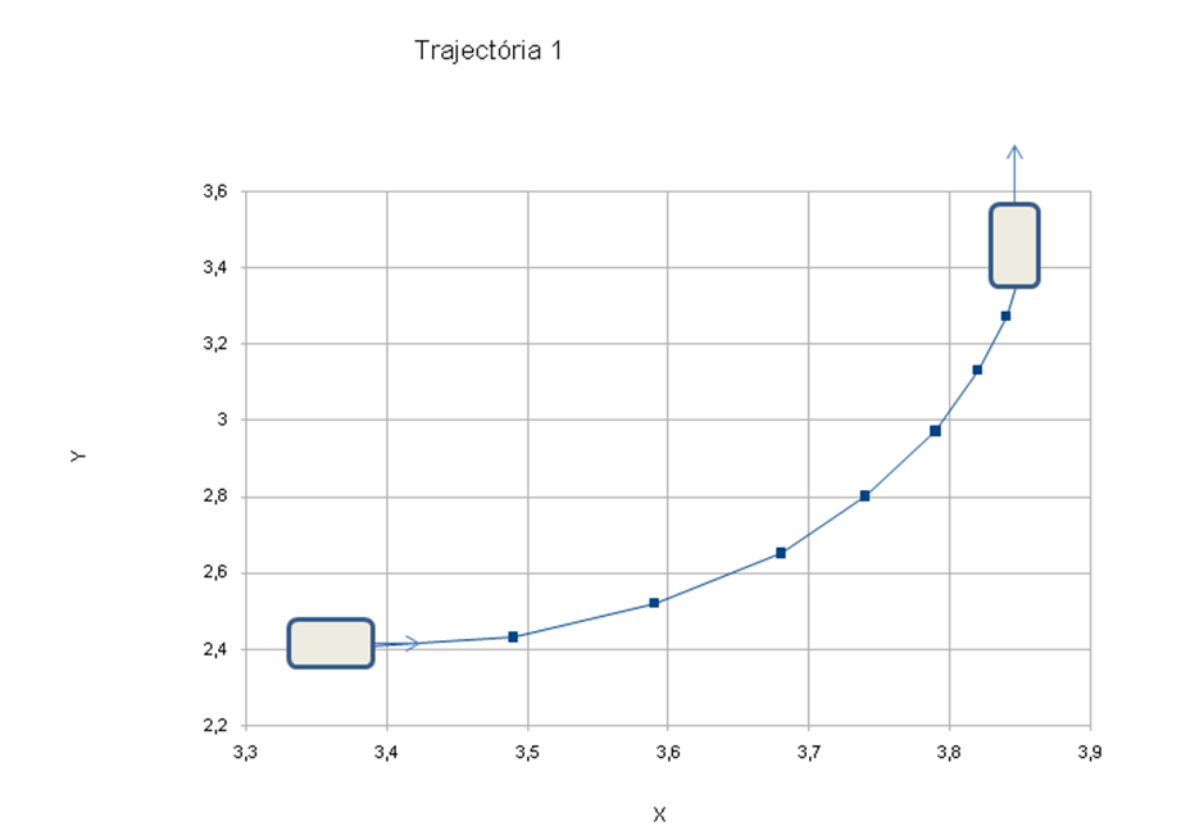

Figura 23 Gráfico referente à trajectória 1

Valores de entrada para a Trajectória 2:

Xi = 3,05	Yi = 2.4	Ti = 1.57
Xf = 3.95	Yf = 3.45	Tf = 0

A trajectória obtida após o uso do algoritmo desenvolvido:

X (m)	Y (m)	T - Orientação (Radianos)
3,05	2,4	1,57
3,05	2,64	1,53
3,07	2,84	1,45
3,1	3,01	1,33
3,15	3,14	1,13
3,21	3,25	0,87
3,3	3,33	0,59
3,42	3,39	0,34
3,56	3,42	0,17
3,74	3,44	0,06
3,95	3,45	0

Na Figura 24 pode observar-se outro exemplo de uma trajectória paramétrica obtida através da rotina desenvolvida nesta dissertação. Esta rotina pode ser facilmente adaptada a outros robôs.

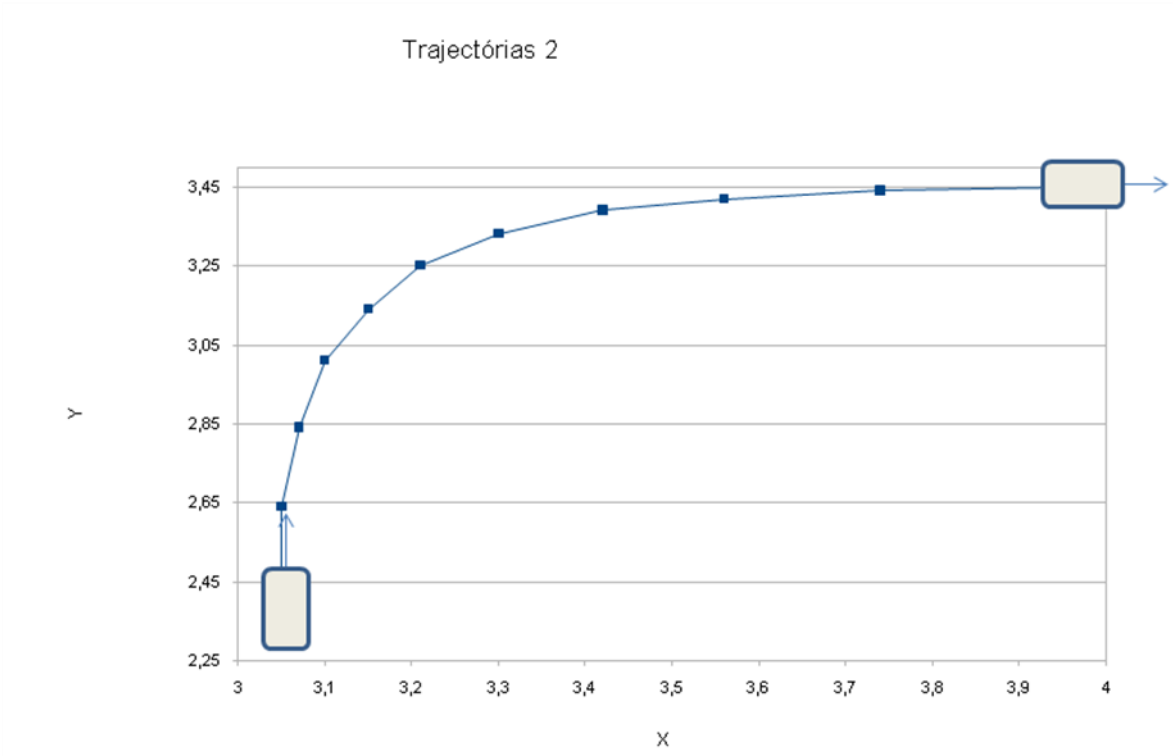

Figura 24 Gráfico referente à Trajectória 2

O uso das trajectórias paramétricas permite a obtenção de trajectórias em virtude do custo da trajectória e onde se tenta adaptar as mesmas às restrições dos robôs.

3.5. MAP MATCHING

Map Matching. *Map Matching* ou *Map-based positioning* é uma técnica aplicada na robótica que usa sensores para criar um mapa local [7]. O objectivo passa por obter uma correspondência entre a representação local e o ambiente global que está representado em memória. Como forma de simplificação, para reduzir o número de cálculos, é necessário que o robô consiga estimar a sua posição actual evitando processar zonas do mapa que estão muito longe do local de acção. Para construção da representação local é necessário possuir sensores que permitam obter leituras sobre o mundo, tais como câmaras, sonares, Laser Range Finder, GPS e bússolas. Estes sensores estão sujeitos a erros de medição e limitações tais como não atravessar paredes e outros objectos. Um dos pontos mais

importantes passa por ao obter diversas leituras de sensores diferentes, conseguir obter a partir destes uma leitura mais rigorosa.

O procedimento mais comum para *Map matching* é:

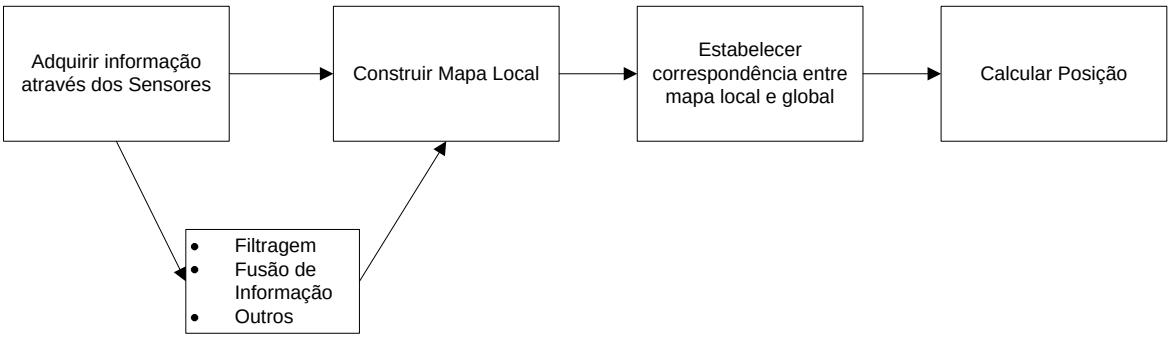

Figura 25 Abordagem para *Map Matching*

Com este método podemos aproveitar o facto de ambientes interiores serem normalmente estruturados ou semi-estruturados. Desta forma, podemos obter leituras e procurar correspondências no mapa. É possível ainda efectuar a actualização do mapa que está em memória a partir das leituras que obtemos dos sensores. Permite também afastar-nos de mínimos locais no mapa quando se usa uma forma local para evitar obstáculos.

No entanto, algumas das desvantagens ou necessidades da aplicação de *Map Matching* é que o ambiente precisa de ser relativamente estático e possuir algumas características fáceis de identificar. É necessário também ter sensores com uma boa exactidão que pode variar com o tipo de tarefa a implementar. O mapa global guardado em memória tem que ser uma representação relativamente boa do mapa real, caso contrário as leituras poderão não servir para identificar o local do mapa onde está o robô. Para tratar uma grande quantidade de dados e aplicar algumas formas de obter resultados da fusão das leituras dos sensores é necessário uma grande capacidade de processamento.

A capacidade do robô de construir mapas depende muito das capacidades sensoriais instaladas no robô.

Os dois principais métodos baseiam-se em que o robô já tem um mapa previamente gravado em memória ou que terá de construir primeiro o seu mapa para depois poder localizar-se.

Considerando as leituras dos sensores, procura-se:

- Extrair as características determinantes do mapa;
- Fusão das leituras;
- Geração automática dum mapa com diferentes níveis de abstracção.

No caso de não existir um modelo do mapa previamente gravado, podemos usar as capacidades sensoriais e algoritmos de exploração para construir ou melhorar constantemente o modelo.

É normal usar-se o maior número de características possíveis do mapa tais como paredes, esquinas entre outros de forma a obter uma maior correlação. No entanto, o custo computacional aumenta muito.

Autores como Schiele and Crowley[20] consideram algumas formas de *Map Matching*:

- Correspondência entre segmentos;
- Correspondência entre Grelha e segmento;
- Correspondência entre segmento e Grelha;
- Correspondência entre Grelhas – Aqui pretende-se criar uma máscara do mapa local. Procura-se sobrepor a grelha local sobre a grelha global. A correlação aumenta quando as células têm o mesmo estado e diminui quando têm um estado diferente. Neste caso, procura-se obter a posição com maior correlação. A incerteza pode ser considerada em conta neste processo.

Devemos ter em consideração que quanto maior a incerteza da posição e orientação do robô mais difícil fica obter uma correspondência positiva. Este tipo de problemas ainda é mais grave quando se usam sonares como o método principal de obter as leituras devido à sua fraca resolução angular.

Para representar um mapa podemos usar várias formas de representação. Uma delas, obviamente, é a representação geométrica do mapa. Desta forma, obtemos uma representação absoluta dos objectos.

Existem diversas formas de representar geometricamente o mapa. Uma dessas formas é usar uma grelha onde se representam as células ocupadas e desocupadas. A resolução das células é um dos factores a ter em conta na representação do mapa. Uma resolução muito pequena aumenta o custo computacional embora também melhore a representação do mapa. Uma resolução muito grande pode representar o mapa duma forma prejudicial à aplicação de *Map Matching*.

As vantagens de usar mapas baseados em grelhas são:

- Permitem uma maior densidade;
- Têm um custo computacional relativamente baixo e podem ser criados rapidamente;
- Permitem uma maior integração de leituras de sensores diferentes.

As desvantagens de usar mapas baseados em grelhas são:

- Têm áreas com grande incerteza e dependente do tipo de características do mapa;
- Têm dificuldades no tratamento de ambientes dinâmicos.

Neste trabalho foi implementada uma forma de *Map Matching* que usa um modelo global do ambiente baseado numa grelha onde cada célula tem 2 estados possíveis: ocupada ou desocupada. A partir deste momento cria-se uma grelha local a partir das leituras do Laser Range Finder e aplica-se uma correspondência entre as 2 grelhas.

Este método é referido em Borenstein [7]. Existem diversos métodos, no entanto a opção recaiu sobre este devido ao uso de células na representação do ambiente. Tornou-se importante obter as leituras do Laser Range Finder e procurar construir uma grelha com a informação obtida para poder efectuar uma comparação com o mapa guardado em memória.

Alguns resultados e conclusões serão descritos no capítulo 5.

4. APLICAÇÃO DE NAVEGAÇÃO

4.1. VISÃO GERAL

No âmbito desta dissertação foi desenvolvido um sistema de navegação autónoma com algumas funcionalidades básicas e integrado um Laser Range Finder de forma a permitir uma navegação em ambientes mais estreitos e a implementação de uma rotina de *Map Matching*, entre outras aplicações.

Após um levantamento das especificações do sistema de navegação a implementar, optou-se por desenvolver uma aplicação no sistema operativo Linux. Recorreu-se à distribuição Red Hat com um "patch" que permite obter características de Tempo Real. O "patch" utilizado é uma extensão do kernel que permite escrever aplicações que conseguem satisfazer certos requisitos temporais.

As características de tempo real são, sobretudo, aplicadas na leitura regular dos sonares e odometria. As restantes tarefas a decorrer não têm restrições de tempo real.

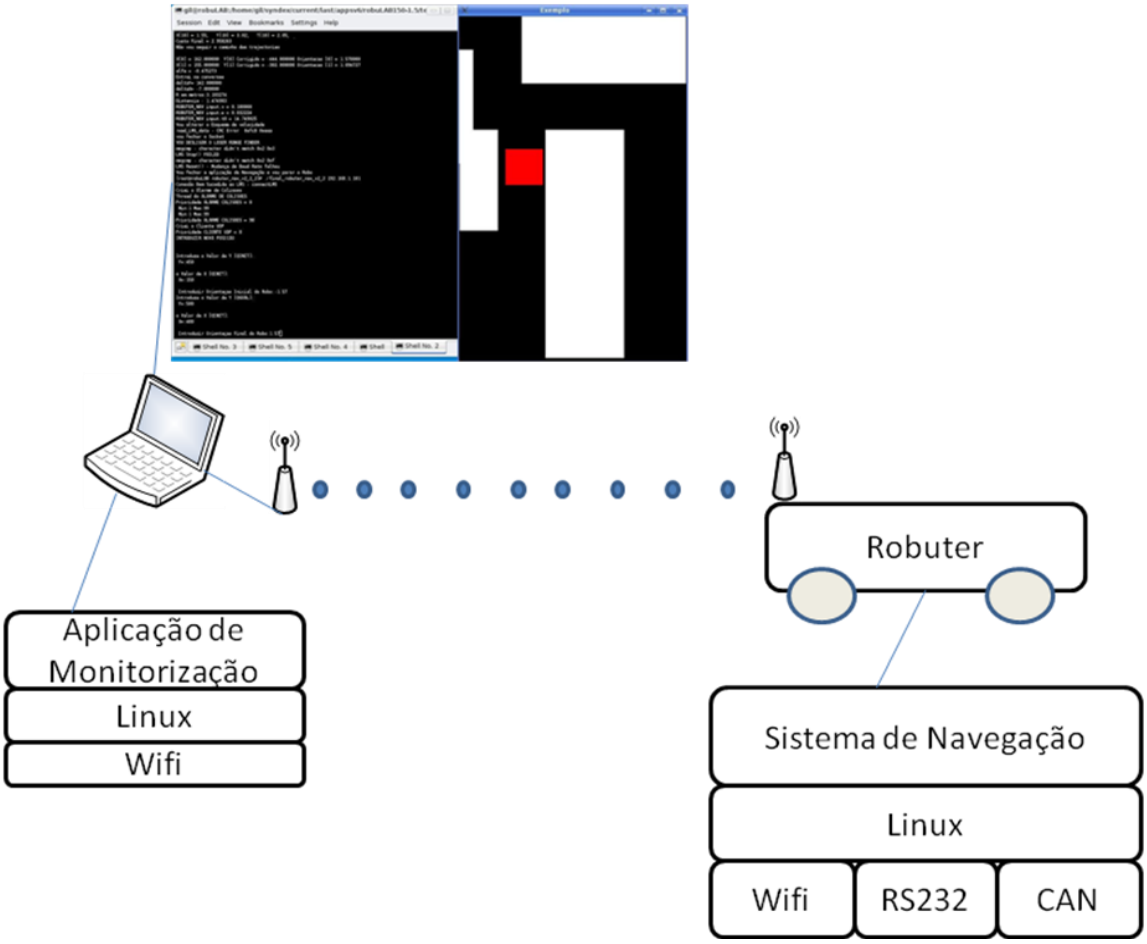

Figura 26 Visão geral de Alto Nível

Podemos observar, na Figura 26, uma perspectiva de alto nível desta dissertação. Após um estudo autónomo de OpenGL foi desenvolvida uma aplicação que permitisse visualizar graficamente o planeamento de caminhos efectuado pelo robô assim como a posição que ele pensa ocupar naquele instante. A aplicação desenvolvida em OpenGL corre num dispositivo portátil que corre Linux. A informação necessária para a representação gráfica é enviada pela Robuter.

Como as tarefas têm diferentes necessidades de processamento e até pela sua importância no funcionamento da aplicação, foi decidido criar várias *Threads* de forma às várias tarefas decorrerem em paralelo sempre que necessário.

Para a atribuição do número de *Threads* que correm em paralelo foi tido em consideração o objectivo de cada uma das tarefas e a necessidade de realmente correrem em paralelo. Foram identificadas 4 tarefas principais que estão representadas na Figura 27:

- Alarme de Colisões
- Thread_Principal
- Planeamento de Caminhos
- Cliente_UDP

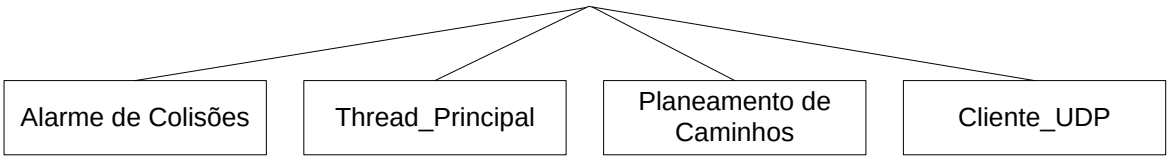

Figura 27 Esquema de Threads

A importância de cada Thread assim como o seu papel na aplicação, será explicada neste capítulo.

4.2. DESCRIÇÃO DA APLICAÇÃO DE NAVEGAÇÃO

Inicialmente é criada a Thread_Principal, que é a *thread* original do processo. Ao iniciar esta *thread*, é efectuada a conexão ao Laser Range Finder e testada. É também criada a *thread* referente ao Cliente_UDP. Após estes procedimentos de inicialização, estamos preparados para receber os parâmetros da missão, a configuração inicial e a configuração a atingir. A partir deste momento, cria-se uma *thread* que irá calcular os caminhos livres de obstáculos, vulgarmente descrita como rotina de *Path Planning*. De seguida, procede-se a uma série de rotinas que permitem adaptar o caminho, obter trajectórias e executá-las de forma a atingir a configuração final.

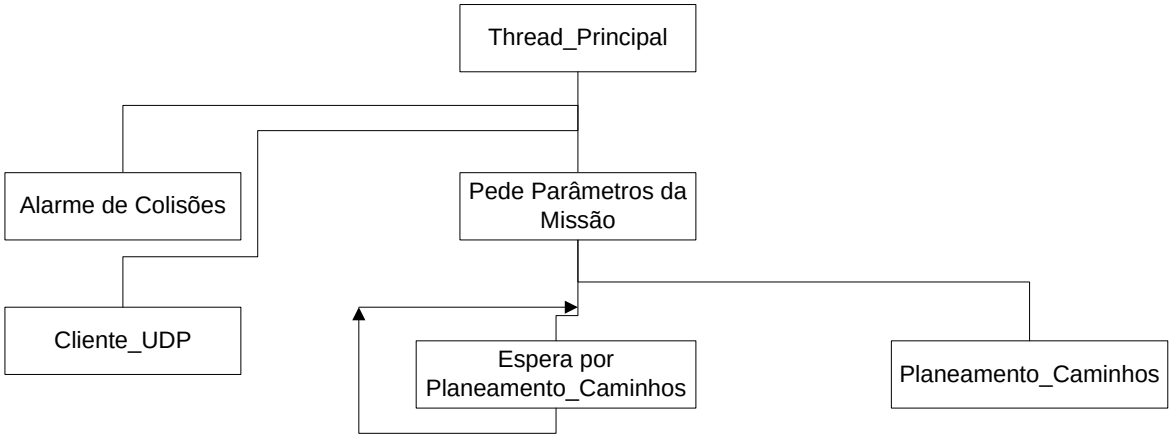

Figura 28 Robuter_Nav

A partir do momento que se usam *threads*, torna-se importante restringir o acesso das variáveis comuns às *threads*. No entanto, apenas se efectuam leituras nas mesmas não sendo necessária a implementação de *Mutexes* ou de semáforos.

Nos casos em que efectua a escrita em estruturas de dados, existem semáforos que condicionam o acesso à estrutura.

4.3. ALARME DE COLISÕES

Tendo em conta que a *thread* do Alarme de Colisões é a que apresenta as restrições temporais mais apertadas, é a *thread* com mais prioridade na aplicação.

A *thread* Alarme de Colisões tem como função ler regularmente, com um período de 10 ms, uma estrutura de dados onde estão acessíveis os valores lidos pelos sonares e pela odometria do robô. Esta thread torna-se muito importante na navegação devido a avaliar a possibilidade de colisão e ainda tentar recuperar dessa situação. Para uma avaliação da situação, são utilizadas as leituras fornecidas pelo LRF, que por sua vez substituiu os sonares na detecção de obstáculos devido à sua resolução ser muito mais alta. Adicionalmente obtém-se uma enorme quantidade de dados com qualidade, o que permite uma resposta muito mais adequada do que se usassem apenas os sonares para a avaliação do ambiente local. Através do uso do LRF, pode obter-se uma navegabilidade muito

melhorada pois é possível detectar o obstáculo e parar o robô a apenas uns cms deste, algo que não seria possível com os sonares disponíveis.

Quando é detectada a possibilidade de colisão é disparado um sinal que chamará uma função que terminará a *thread* correspondente ao Planeamento de Caminhos, de forma a acabar com qualquer movimento e processamento. O Planeamento de Caminhos só será reiniciado quando já tiver recuperado da situação de colisão. Isto significa que tentar-se-á obter uma configuração que permita entrar numa rota que leve a atingir o objectivo final. Mas mais à frente será discutida a rotina que tenta recuperar o robô para uma configuração aceitável.

Para envio de sinais quando na presença de *Threads*, deve ter-se o cuidado de usar `pthread_kill(pthread_t` *`thread`*`, int` *`sig`*`)` para assegurar a entrega do sinal.

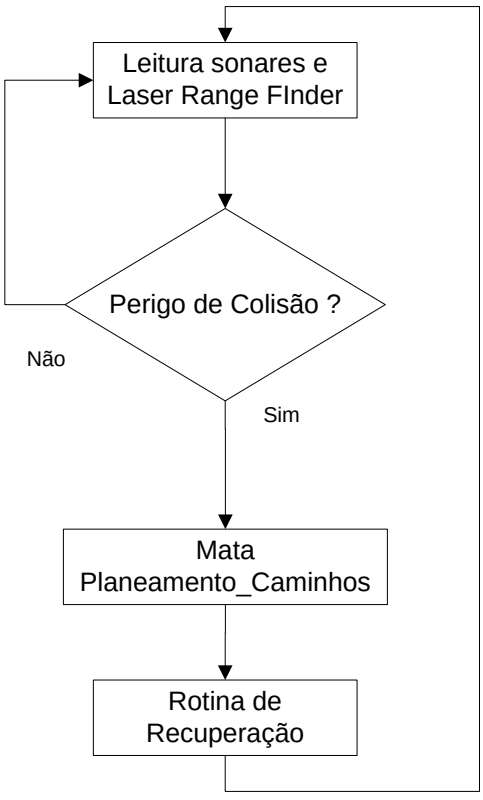

Figura 29 Rotina Alarme de Colisões

Pode observar-se um fluxograma na Figura 29 referente à rotina Alarme de Colisões. A rotina de Recuperação será explicada adiante.

4.4. CLIENTE UDP

A *thread* referente ao Cliente_UDP foi criada após verificar a importância de ter uma representação visual gráfica do que é processado pelo robô. O debug da aplicação de Navegação foi claramente melhorado a partir do momento em que se criou uma aplicação em *OpenGL[15][16]* num PC. Desta forma, o ambiente e o robô estão representados permitindo compreender algumas das reacções que aconteceram ao longo dos testes desenvolvidos. Esta abordagem é muito útil pois disponibiliza muito mais informação quando se pretendem identificar e corrigir certos problemas. O facto dessa informação ser fornecida por uma aplicação gráfica torna muito mais fácil a sua interpretação, tendo como auxílio o *output* que é enviado para o terminal. Fundindo as informações obtidas, em grande parte das situações torna-se simples identificar o problema. É um bom método que ao gastar bastante tempo no seu desenvolvimento, permite poupar muito mais tempo na correcção de erros que afectam seriamente o desempenho da aplicação e também melhorar a qualidade da aplicação desenvolvida.

No entanto, tornou-se imprescindível receber alguns dados no computador onde corre a aplicação gráfica. A opção escolhida foi criar um servidor UDP nesse computador e criar um cliente UDP no robô. Como a informação que é enviada serve apenas para visualização e não é relevante para cálculos ou controlo, UDP é uma solução perfeitamente adaptada à necessidade desta aplicação. O UDP é uma escolha adequada para fluxos de dados em tempo real, especialmente aqueles que admitem perda de parte de seu conteúdo, tais como vídeos ou voz. Como neste caso, existe alteração da posição constantemente não é grave perder alguns pacotes.

Na Figura 30 temos um fluxograma que descreve o funcionamento do cliente UDP.

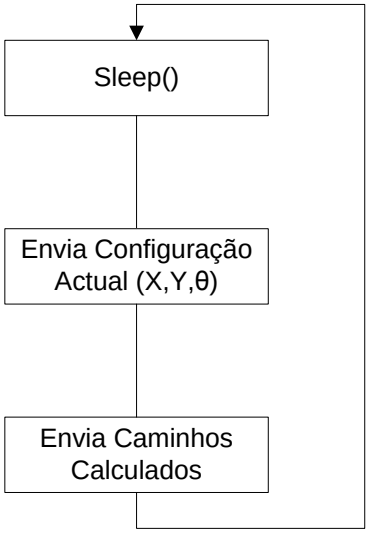

Figura 30 Cliente UDP

4.5. APLICAÇÃO DE MONITORIZAÇÃO

A aplicação de monitorização desenvolvida permite obter informação sobre o caminho calculado pela Robuter e a posição que ocupa naquele instante. Esta aplicação é constituída por um servidor UDP e por uma aplicação desenvolvida em OpenGL no PC. Temos também, como descrito anteriormente, o cliente UDP a correr na Robuter. O OpenGL tem a vantagem de ser implementado num paradigma multi-plataforma, logo pode ser executado em vários Sistemas Operativos.

Na Figura 31 podemos ver a fusão da informação recebida na consola e a visualização da Aplicação de Monitorização. Ao juntar estas duas fontes de informação de carácter distinto, conseguimos obter uma melhor percepção do que está a ser executado pelo robô.

Esta aplicação pode ser adaptada para outros projectos em robótica sendo necessário alterar a geometria do robô na aplicação e o Layout do ambiente de testes.

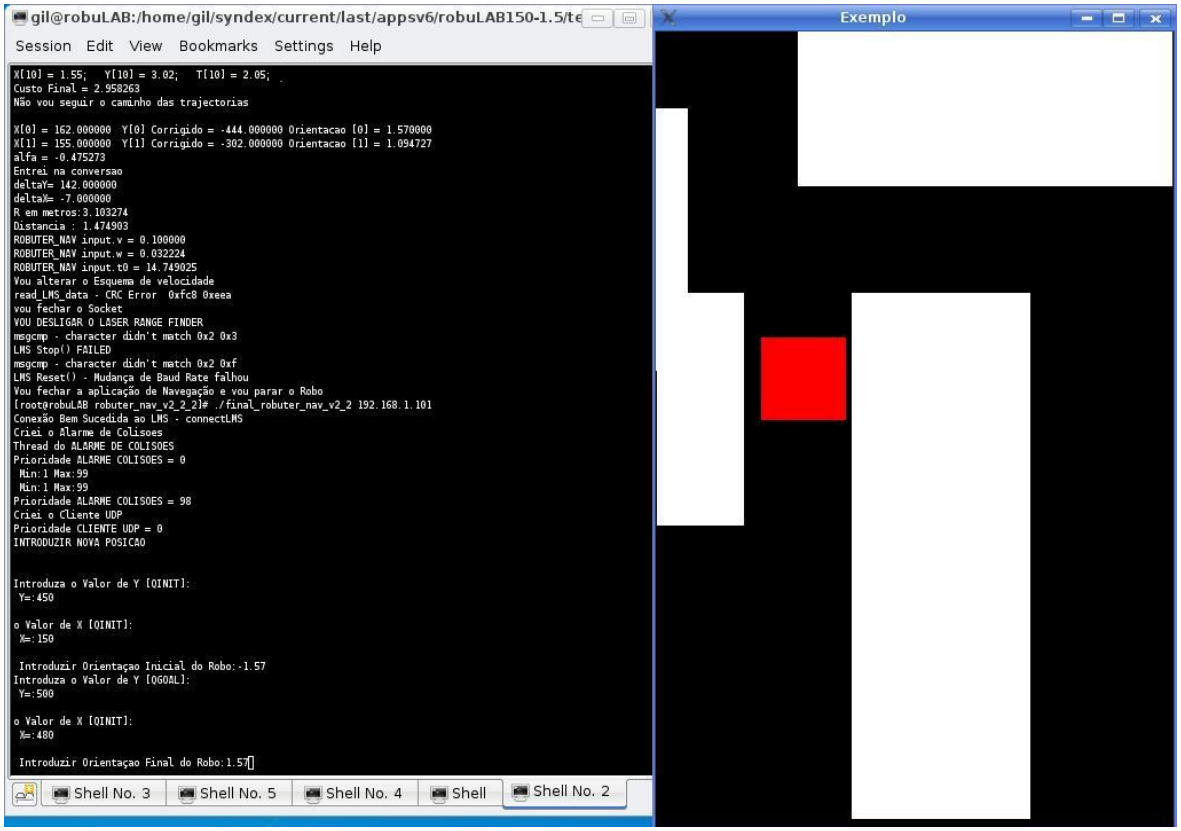

Figura 31 Screenshot da Consola e Aplicação de Monitorização

Na Figura 32 podemos ver alguns *screenshots* da Aplicação de Monitorização em diversos momentos.

Screenshots da Aplicação de Monitorização em *OpenGL*

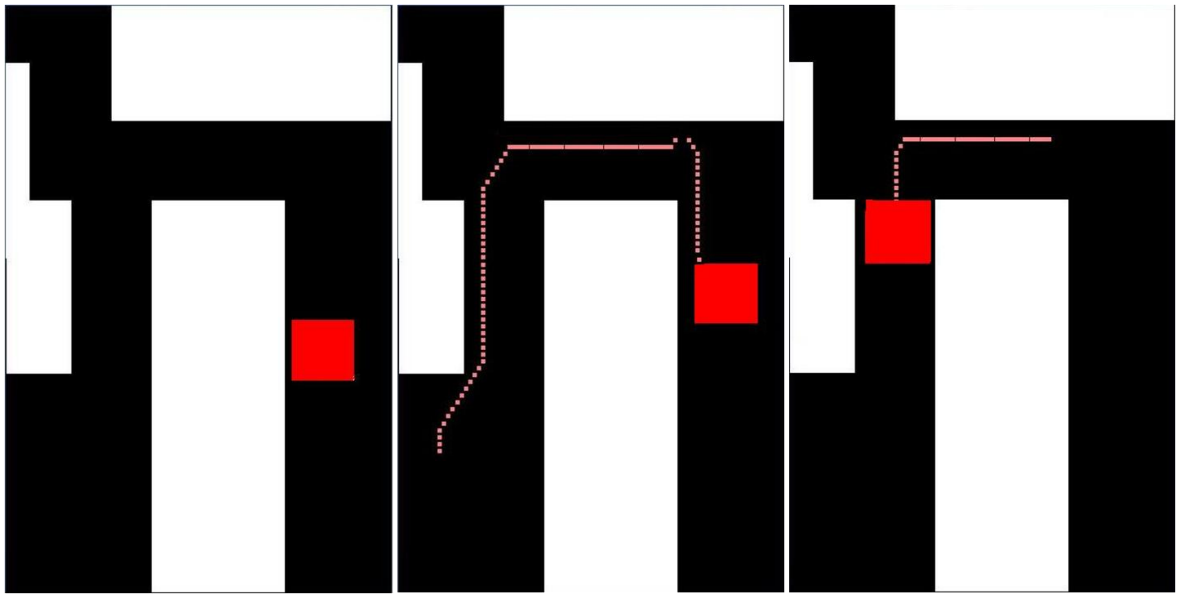

Figura 32 Aplicação em OpenGL pra visualização gráfica

4.6. THREAD PLANEAMENTO_CAMINHOS

Como é possível de observar, as *threads* apresentadas até ao momento ainda não tratam nenhum problema da navegação. Apenas são blocos secundários e que apoiam o sistema de navegação que foi criado.

Para conferir capacidades de navegação ao Robô foi criada uma *thread* que trata de todo o processo de navegação desde o cálculo de caminhos livres de obstáculos, cálculos de trajectórias paramétricas de terceiro grau, execução da trajectória, entre outras rotinas. Após um estudo de soluções existentes, foi decidido implementar uma solução cujo processo pode ser observado na Figura 33.

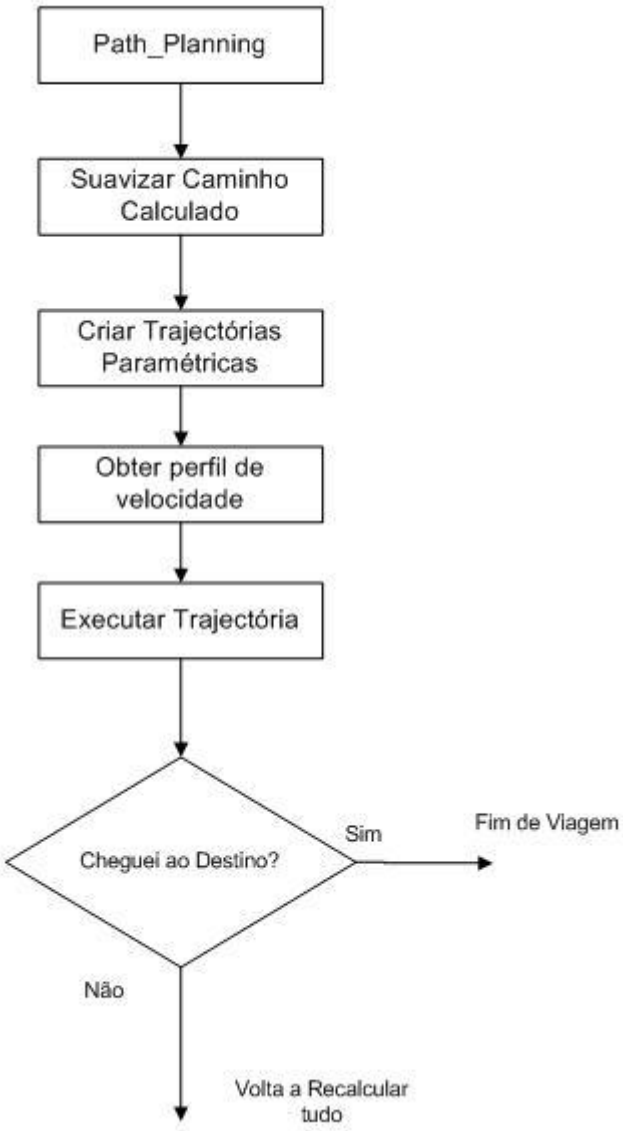

Figura 33 Planeamento de Caminhos

Como é observável, a rotina implementada tem em consideração a realidade não-holonómica pois suaviza o caminho calculado e ainda implementa uma filtragem das trajectórias paramétricas que estão adaptadas ao robô em causa.

A primeira rotina é o *Path Planning*. Esta rotina recebe como parâmetros o ponto inicial e o ponto final (ponto objectivo) e calcula um caminho livre de obstáculos. É a *Thread* mais importante e que necessita de maior capacidade de processamento pois é onde ocorre o cálculo de trajectórias paramétricas e de caminhos livres de obstáculos.

Primeiro o ambiente é dividido numa grelha e os obstáculos guardados em memória são colocados na grelha. Depois, através do método de *Wavefront*, procura obter-se um caminho livre de obstáculos como se pode observar na Figura 34.

			500	500	500	500	500	500	500	500	500	500	500
			500	500	500	500	500	500	500	500	500	500	500
			500	500	500	500	500	500	500	500	500	500	500
			500	500	500	500	500	500	500	500	500	500	500
		4	**4**	**4**	**4**								
		4	**3**	**3**	**3**								
		4	3	2	**3**								
500	500	4	**3**	**3**	**3**								
500	500	4	**4**	**4**	500	500	500	500	500	500			
500	500	5	**5**	**5**	500	500	500	500	500	500			
500	500	6	**6**	**6**	500	500	500	500	500	500			
500	500	7	**7**	**7**	500	500	500	500	500	500			
500	500	8	**8**	**8**	500	500	500	500	500	500			
500	500	9	**9**	**9**	500	500	500	500	500	500			
500	500	10	**10**	**10**	500	500	500	500	500	500			
500	500	11	**11**	**11**	500	500	500	500	500	500			
500	500	12	**12**	**12**	500	500	500	500	500	500			
500	500	13	**13**	**13**	500	500	500	500	500	500			
500	500	14	**14**	**14**	500	500	500	500	500	500			
500	500	15	**15**	**15**	500	500	500	500	500	500			
500	500	16	**16**	**16**	500	500	500	500	500	500			
		17	**17**	**17**	500	500	500	500	500	500			
		18	**18**	**18**	500	500	500	500	500	500			
	250	19	**19**	**19**	500	500	500	500	500	500			
					500	500	500	500	500	500			
					500	500	500	500	500	500			
					500	500	500	500	500	500			

- Ponto Objectivo

- Ponto Actual

Figura 34 Exemplo do Método de Wavefront

Os mapas obtidos através do *Wavefront* são dependentes do ponto inicial e do ponto final. Pode acontecer que quando se procuram atingir certos pontos do mapa através de outros que estão muito longe, se tenha dificuldades em se obter um caminho. Para poder atingir pontos que estão muito longe uns dos outros, criou-se uma rotina que tem pontos intermédios no mapa e procura usar o método *Wavefront* para encontrar caminhos entre os pontos em questão e para as posições iniciais e finais. Desta forma, já é possível atingir grande parte das zonas do ambiente.

Após o cálculo do caminho livre de obstáculos, é necessário analisar o caminho e verificar se existem indícios de movimentos difíceis de fazer para o robô em causa. Um dos exemplos é a existência de movimentos em que obtemos um ângulo recto no caminho calculado.

A rotina de *Path Planning* retorna um caminho constituído por células adjacentes onde por vezes existem alterações que obrigariam o robô a fazer uma variação brusca de orientação sem se mexer. Para contornar este problema, o caminho é adaptado de forma a se escolherem pontos intermédios. Na escolha dos pontos intermédios, tem-se o cuidado de detectar as hipóteses de efectuar caminhos com ângulos rectos e afasta-se os pontos intermédios desse ponto. Desta forma, a trajectória do robô já pode ser efectuada tendo como ponto de entrada um ponto a uma certa distância do ângulo recto e o ponto final à mesma distância do ângulo recto detectado. Assim, podemos contornar o problema com alguma antecipação. Este comportamento pode ser observado na Figura 35.

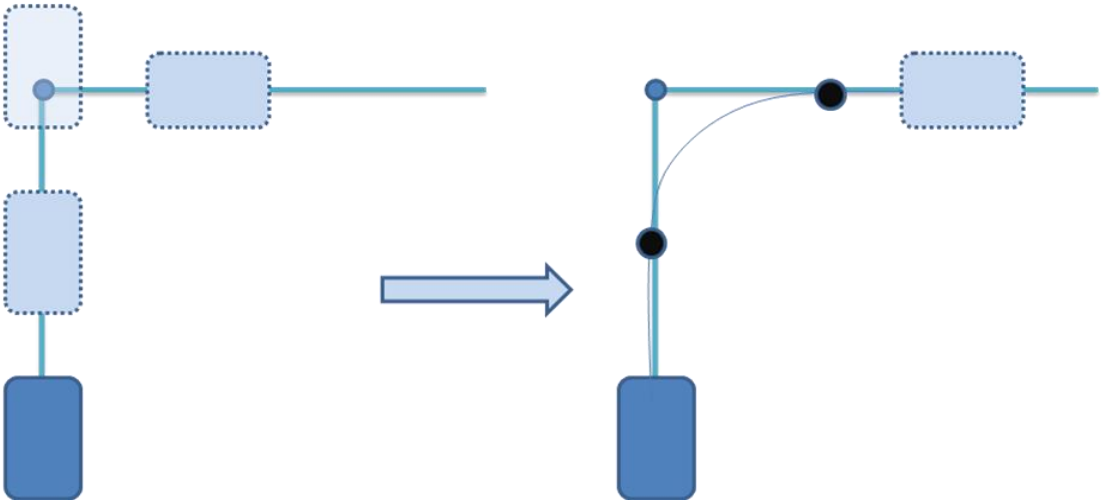

Figura 35 Suavização do caminho

O planeamento de caminhos não tem em consideração a orientação inicial e a orientação final, sendo este um dos pontos mais criticáveis deste método. Logo, como entrada deste bloco temos apenas o ponto inicial e o ponto final. De seguida, é colocado um valor numérico alto na célula actual e um valor baixo e conhecido na célula final. É atribuído um valor muito elevado a células ocupadas com obstáculos. Constrói-se então um mapa com valores para todas as células do mapa. Sendo que quanto mais perto se estiver do ponto final, menor será o valor da célula. Corre-se então um algoritmo que analisa a periferia da nossa célula actual e determina qual tem o valor mais baixo. Efectuando este

tipo de raciocínio podem acontecer 2 tipos de situação. Atingimos o ponto objectivo com sucesso ou ficámos presos num mínimo local. Quando ficámos presos num mínimo local, essa situação é detectada pela aplicação. Tenta-se então recorrer a pontos intermédios no mapa que permitam ligar o ponto inicial e o ponto final através de um ou mais pontos intermédios. No caso de conseguirmos calcular um caminho livre de obstáculos desde o ponto inicial até ao ponto final, então vamos tentar suavizar o caminho e dividi-lo em vários segmentos tal como foi descrito acima.

Após termos um conjunto de segmentos, vamos aplicar um cálculo para geração de trajectórias paramétricas que foram introduzidas na secção 3.4.3. Este cálculo está inserido num problema de optimização que nem sempre apresenta resultados aplicáveis. No caso de não se conseguir atingir um resultado que se possa utilizar, é calculado um arco de círculo que permita atingir o ponto em consideração através do ponto onde estamos.

Para filtragem das trajectórias obtidas e adequação ao robô em questão é efectuado um teste ao raio das trajectórias para conferir se o robô consegue efectuar aquele movimento. É ainda efectuado o cálculo do custo da trajectória em que o critério é a distância entre os pontos que constituem a trajectória obtida. É feita uma comparação entre a trajectória que se considera a melhor naquele momento e no caso de respeitar a condição de raio e se tiver um custo inferior, então substituímos a trajectória pela trajectória calculada.

Após a execução da trajectória, é feito um teste procurando ver se atingimos o ponto objectivo. Se sim, a viagem acabou e é dada como bem sucedida. No caso de não atingirmos, testa-se se há mais trajectórias para calcular. No caso de estarmos próximos do ponto final, é efectuado um teste para verificar se a actual configuração do robô permite atingir o ponto final. Caso não se consiga atingir o ponto final, a missão é dada como falhada.

5. INTEGRAÇÃO DO LASER RANGE FINDER

5.1. VANTAGENS DA UTILIZAÇÃO DO LASER RANGE FINDER

No inicio deste trabalho, a Robuter contava apenas com sonares como sensores externos, de forma a monitorar o ambiente local. Embora os sonares sejam elementos muito comuns e em conjunto com outras tecnologias sejam boas opções, neste caso particular e devido ao espaço em laboratório ser relativamente pequeno para um robô com estas dimensões, torna-se bastante complicado navegar apenas considerando as leituras dos sonares. Algumas manobras da Robuter necessitam de espaço e os sonares têm uma zona cega onde não obtemos leituras devido ao tempo que demora a comutar de emissor para receptor. Além desta limitação temos de considerar a fraca resolução angular comum nos sonares. Outro facto a considerar é que uma boa parte das aplicações mais robustas que usam somente sonares como sensores externos têm 16, 24 ou até 32 sonares, de forma a poderem monitorar o ambiente com muito melhor qualidade. Neste caso temos 6 sonares espalhados por um robô que possui um tamanho considerável. Logo, existe uma grande área em redor do robô sobre a qual nada sabemos em cada instante. Uma das hipóteses seria utilizar informação de instantes anteriores, se possível.

O cenário visível quando se usam apenas os sonares para monitorização do ambiente é que temos de considerar o robô como podemos observar na Figura 36. Neste caso, podemos ver que o tamanho do robô fica virtualmente maior pois no espaço correspondente às zonas cegas dos sonares não temos informação nem podemos assumir tendo em consideração o mapa global dado que podem existir novos objectos no mapa ou eventualmente um humano a deslocar-se pela instalação. O robô passa a ter umas dimensões virtuais 172 x 104 cm.

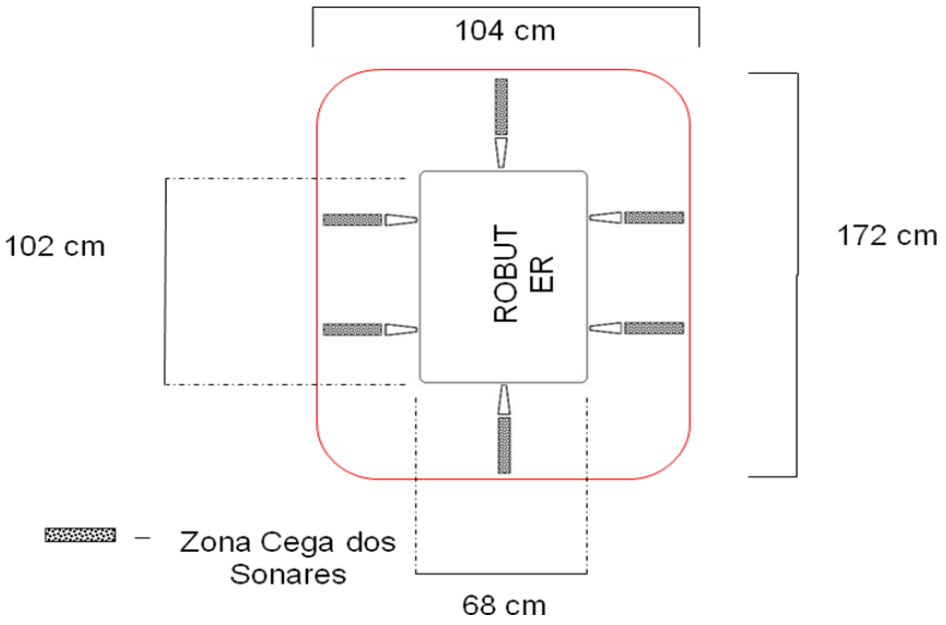

Figura 36 Tamanho Virtual do Robô quando usa sonares

Ao ser instalado um Laser Range Finder no robô, podemos resolver muitos dos problemas que tínhamos ao usar os sonares como únicos elementos sensoriais. A sua instalação permite o desenvolvimento de novas aplicações, de uma maior fiabilidade das medições e neste caso permitiu a navegação no laboratório com muito maior facilidade, dado que o ambiente é relativamente apertado para manobras, em alguns sítios.

Ao instalar o Laser Range Finder no local que podemos observar na Figura 37 passámos a ter leituras referentes a quase toda a área que o robô precisa de conhecer. Uma das assumpções é que o robô só navegará para a frente, por falta de condições de monitorização. A única excepção é na rotina de recuperação de posição onde se movimenta um pouco para trás para ganhar maior amplitude de movimentos, como será explicado noutra secção.

Ao usar o Laser Range Finder, o tamanho virtual do robô diminui significativemente se compararmos com a situação em que apenas se usam os sonares como sensores externos. Ao instalar o LRF permitiu que o tamanho virtual do robô passasse de 172x104 cm para 115x80 cm. Esta alteração permitiu uma navegabilidade muito melhor. Primeiro, pode navegar-se em espaços mais apertados, dado que podemos ajustar os valores da rotina que evita colisões para os limites do tamanho do robô e mesmo assim evitar as colisões. Logo, podemos aproveitar muito melhor os espaços existentes no ambiente. Com os sonares teríamos de ter tolerâncias maiores para evitar que existissem colisões. Enquanto os

sonares têm uma resolução angular fraca, o Laser Range Finder tem uma resolução muito boa e até é possível obter várias resoluções, se necessário. Passámos a obter 181 leituras referentes às zonas em redor do robô quando anteriormente só tínhamos 5 sonares a trabalhar nas zonas laterais e frontal do robô. Isto permite uma monitorização muito mais rigorosa das manobras do robô ou de algum objecto novo no ambiente.

O Laser Range Finder proporciona um aumento considerável nas capacidades sensoriais disponíveis na Robuter. E isso reflecte-se na melhoria das aplicações e tarefas possíveis para o robô. O facto de proporcionar uma excelente resolução angular além de medições muito fiáveis permitiu criar uma rotina de recuperação de posição para melhorar a navegabilidade em ambientes apertados. Esta rotina será discutida na secção 5.3.

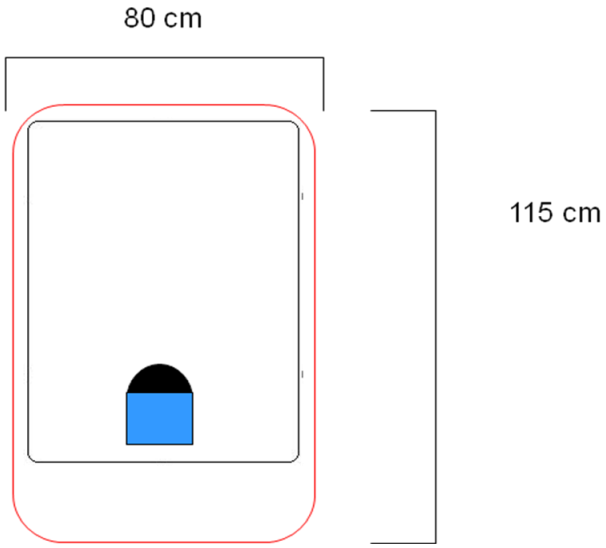

Figura 37 Tamanho Virtual do Robô quando usa o Laser Range Finder

Figura 38 Laser Range Finder montado na Robuter

5.2. ROTINA DE MAP MATCHING

É fundamental obter uma localização fiável do robô no mapa quando se procura desenvolver um sistema de navegação autónoma. Neste trabalho foi implementada uma forma de *Map Matching[17]* referida em Borenstein[7]. O objectivo passa por procurar correspondências através duma grelha construída através das leituras do mapa e entre um mapa local guardado em memória. Estas leituras são fundidas com as leituras provenientes da odometria de forma a limitar a necessidade de processamento de zonas do mapa onde seria muito difícil nos encontrarmos, dado que seriam dados muito contraditórios em relação aos obtidos pela odometria. O mapa construído através das leituras do Laser Range Finder é uma grelha de ocupação em que existem 3 estados: livre, ocupado e desconhecido. Após um varrimento do LRF é processado um algoritmo que verifica a correspondência entre as zonas próximas da posição dada pela odometria.

Para seguir o método indicado por Borenstein [7], no seu capítulo sobre *Map Matching*, devem obter-se dados dos sensores disponíveis no robô. O laser fornece uma excelente precisão para a tarefa em questão embora uma câmara ou outros sensores pudessem ser adicionados de forma a obter mais leituras de diferentes fontes para fundir a informação recebida. O próximo passo é filtrar os dados, fundir informação e construir o mapa local ou passar logo para a construção do mapa local sem a filtragem de dados e fusão das leituras de diferentes fontes.

O mapa criado é um mapa local onde devemos ter em consideração a orientação do robô de forma a podermos comparar com o mapa global guardado ou construído previamente.

Na Figura 39 podemos observar como o mapa em questão é construído através das leituras do Laser.

Figura 39 Mapa Local

O estado de desconhecido ocorre pela impossibilidade da luz atravessar os objectos em questão. No entanto, para obter uma correspondência mais eficaz foi considerado construir apenas uma matriz (grelha) de 2 metros por 4 metros. Desta forma podemos fazer algumas simplificações devido aos objectos em questão serem relativamente grandes, o que permite supor que no caso de detectar um obstáculo dentro do limite desta matriz, o espaço atrás desse mesmo objecto estará também ocupado. É preciso notar que esta simplificação é possível de fazer neste ambiente, embora noutros ambientes o tamanho da grelha e resolução das células podem ser reconsiderados. Assim, esta simplificação poderá não ser válida para outros tipos de ambiente.

Resultados do *Map Matching*:

Orientação no momento dos testes: 0 Radianos

	Valores Reais (cm)	Valores Obtidos (cm)	Percentagem	Correspondências	Falhas
X	215	215			
Y	260	260	85,78%	5500	912
X	270	260			
Y	270	270	87,35%%	4808	696
X	270	260			
Y	260	270	87,61%	4822	682

Como é possível de observar, os resultados obtidos são úteis quando o robô já percorreu um caminho relativamente longo e já tem a sua posição com uma incerteza que pode afectar a sua navegação. Nesse caso, esta rotina de *Map Matching* pode ser útil na correcção da posição. Os valores testados pertencem a uma zona do mapa que se julga ser a zona com mais rigor na sua representação. Existem zonas do mapa cuja representação no ambiente criado para o robô não é aproximada o suficiente para permitir resultados razoáveis. A qualidade da rotina está intimamente ligada à resolução de cada célula e mais ainda à qualidade da representação do ambiente. Se a resolução das células for relativamente pequena, influenciará a resolução com que pretendemos identificar a posição actual do robô. No caso de conjugarmos outros sensores tais como câmaras poderia ser possível a aplicação duma rotina de *Map Matching* mais rigorosa.

5.3. ROTINA DE RECUPERAÇÃO

A rotina de recuperação tornou-se possível de implementar a partir do momento que o Laser Range Finder foi implementado na Robuter. O LRF permitiu aumentar as capacidades sensoriais do robô, facilitando assim o desenvolvimento duma rotina para tentar recuperar duma situação em que a navegação foi interrompida por ser detectada a hipótese de colisão. Para contornar este tipo de problemas, adaptou-se uma solução que foi testada e validada experimentalmente.

Esta solução tem como base a criação de áreas bem definidas em redor do robô. Estas áreas são associadas a um esquema de células que pode ser observado na Figura 40. Estas células permitem reagir a novos obstáculos ou trajectórias mal executadas que podem dar origem à interrupção da viagem. Foram criadas 7 células cujos estados serão avaliados como ocupados ou livres. Em função de quais as células estão ocupadas e livres, tentar-se-á recuperar uma configuração que facilite o retorno à viagem.

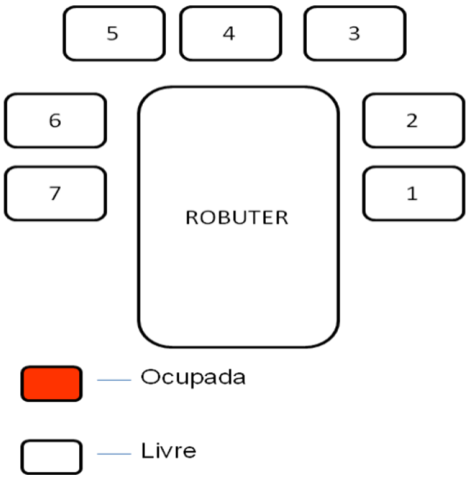

Figura 40 Áreas relevantes para a Rotina de Recuperação

Quando acontece uma interrupção por detecção de hipótese de colisão, a rotina de recuperação é executada e os estados das células são avaliados nesse instante. Na Figura 41 podemos ver uma representação do que acontece na avaliação dos estados das células.

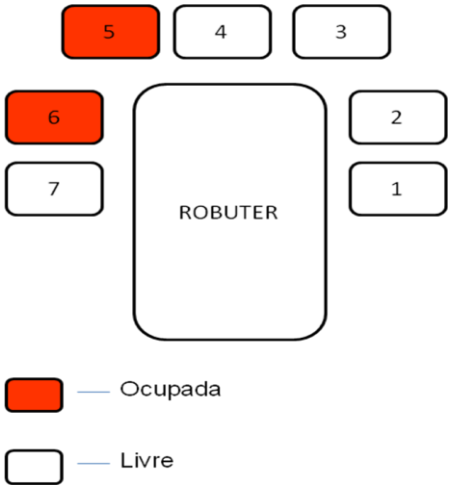

Figura 41 Detecção de obstáculos

Na Figura 42 podemos ver um exemplo da aplicação desta rotina. Neste caso, a trajectória executada não é a mais indicada. No entanto, ao interromper a viagem a rotina de recuperação é executada. Após a verificação dos estados das células, é adoptada uma

estratégia de recuperação para uma posição mais favorável, para retornar a viagem em melhores condições.

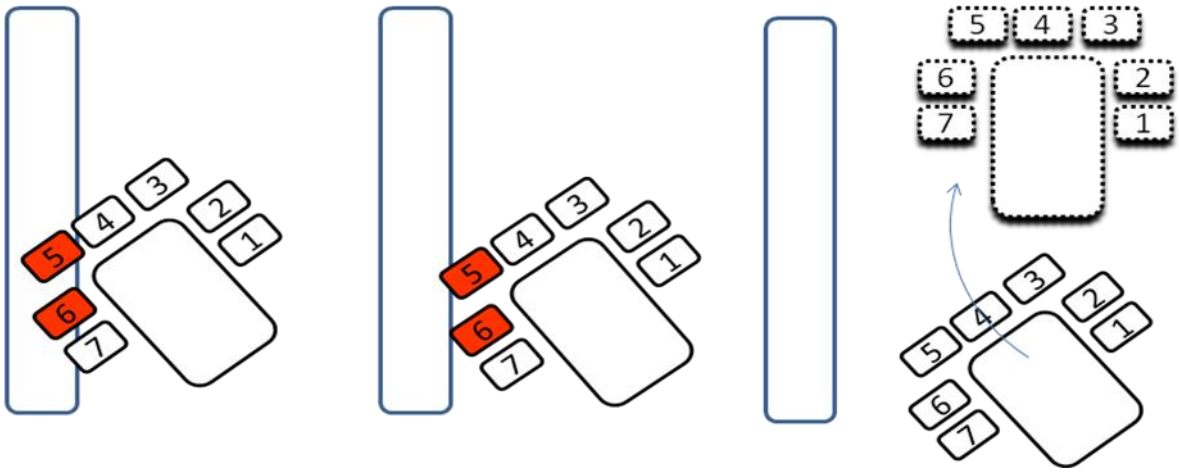

Figura 42 Exemplo do uso da Rotina de Recuperação

Esta rotina revelou-se útil quer para evitar pequenos objectos que se encontram no caminho quer para a navegação em ambientes estreitos onde uma trajectória mal executada condenava o robô a ficar numa situação desconfortável no que toca ao retorno à sua tarefa. Com esta rotina, a amplitude de movimentos do robô foi aumentada dado que mesmo executando mal uma trajectória, passa a ter uma hipótese de recuperação da mesma situação.

6. CONCLUSÕES

Neste trabalho, foi proposto o desenvolvimento do sistema de navegação autónoma para uma plataforma robótica semi-industrial e num ambiente semi-estruturado em que se integrasse um Laser Range Finder. Os objectivos foram atingidos na plenitude. O Laser Range Finder tornou-se num elemento muito importante na plataforma robótica. Permitiu implementar aplicações que sem ele não seriam possíveis com as anteriores condições sensoriais instaladas na Robuter. Para a instalação do Laser Range Finder e para a sua integração na plataforma foi necessária a aprendizagem de novos conhecimentos e aprofundamento de conhecimentos agregados durante o percurso académico. Foi feito um largo estudo sobre planeamento de caminhos, algoritmos de navegação e *Map Matching* dado que no inicio deste trabalho, a plataforma robótica possuía apenas Hardware e nenhuma capacidade de se deslocar por ela própria. Neste momento, tem implementado um sistema de Navegação Autónoma que incorpora recursos que antes não estavam disponíveis. O melhor exemplo disso é o Laser Range Finder. Foi efectuado um estudo autónomo de OpenGL de forma a implementar uma aplicação gráfica que permitisse visualizar o comportamento do robô e conferir o seu funcionamento ao mesmo tempo. Foram aprofundados conhecimentos de Programação em Ambiente Unix, mais propriamente, o sistema operativo Linux. Toda a aplicação cresceu com base no estudo efectuado. A aplicação do Laser Range Finder foi testada em várias componentes. Permitiu

reduzir o tamanho virtual do robô sendo que substituiu quase na integralidade os sonares no que toca a evitar colisões. Esta mudança proporcionou uma melhoria na navegabilidade do robô no ambiente de testes. O potencial do Laser Range Finder em aplicações robóticas é muito grande e pode ser bem explorado com a implementação de outras aplicações. Com a validação em laboratório da aplicação desenvolvida pode afirmar-se que os objectivos foram atingidos com sucesso.

No futuro, possíveis melhoramentos poderiam passar pela integração de outros sensores tais como câmaras para fundir informação com o Laser, permitindo uma técnica de *Map Matching* mais evoluída e eventualmente navegação em ambientes desconhecidos construindo o mapa através da fusão de informação obtida pelos diversos sensores. Para isso, implementar-se-ia um algoritmo de exploração de ambientes desconhecidos.

Poder-se-ia também integrar a plataforma robótica com a célula flexível de fabrico já existente, para efectuar a carga e descarga de buffers de peças.

Referências Documentais

[1] J. Paul Laumond — *Robot Motion Planning and Control*. Laboratoire d'Analye et d'Architecture des Syst`mes e Centre National de la Recherche Scientifique LAAS report 97438, 1997

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

[3] Pedrosa, D. F., Alsina, P. J. & Medeiros, A. A. D., "Geração de Caminhos Ponto-a-Ponto para Robôs Móveis com Rodas". Anais do Congresso Brasileiro de Automática - CBA2002, Natal, Brasil, Set., 2002, pp. 1519-1524.

[4] *Planning Algorithms*. S. M. LaValle. Cambridge University Press, Cambridge, U.K., 2006

[5] LMS200/211/221/291 Laser Measurement Systems – Technical Description

[6] Telegrams for Configuring and Operating the LMS2xx Laser Measurement Systems

[7] Feng, L., Borenstein, J., and Everett, B., 1994, "*Where am I? Sensors and Methods for Autonomous Mobile Robot Localization.*" Technical Report, The University of Michigan UM-MEAM-94-21, December 1994

[8] http://www.robosoft.fr/eng/

[9] Robuter Rectangular Base User Information, Robosoft, April 2005 http://212.208.189.50/PROJECTS/ROBUTER_RECT/Project/User_Information.html

[10] Pomiers, P., The SynDEx "linuxIO_" Macro: An Easy C/C++ Linux User's Application Interface, Robosoft S.A., April 2004

[11] Pomiers, P., Robosoft Development Toolchain, Robosoft S.A., June 2004.

[12] Choset, H. (2006, 2/14). Motion Planning (Wavefront Algorithm) Retrieved 10/July, 2006

[13] *Advanced Programming in the UNIX Environment*, Addison-Wesley, 1992, ISBN 0-201-56317-7

[14] Mark Mitchell, Jeffrey Oldham, and Alex Samuel, Advanced Linux Programming, June 2001

[15] http://www.opengl.org/documentation/

[16] Toby Howard, An Introduction to Graphics Programming with OpenGL, January 2004

[17] João da Silva Gomes Mota, Localisation of a Mobile Robot using Laser Scanner and Reconstructed 3D Models, November 2001

[18] Reeds, J. A. e Shepp, L. A. (1990). Optimal paths for a car that goes both forwards and backwards, *Pacific Journal of Mathematics* 145

[19] Emmanuel Lomba, Mário Alves, On the Hardware and Software Architecture of the Robuter Mobile Platform: a Hands-On Approach, November 2005

[20] Schiele, B. and Crowley, J., 1994, "A Comparison of Position Estimation Techniques Using Occupancy Grids." Proceedings of IEEE International Conference on Robotics and Automation, San Diego, CA, May 8-13, pp. 1628-1634