

Faculdade de Engenharia da Universidade do Porto



**Estudo e Implementação de um Algoritmo de
Localização Baseado na Correspondência de
Mapas**

Manuel Carlos Monteiro Gouveia

Dissertação/Relatório de Projecto realizada(o) no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Automação

Orientador: Prof. Dr. António Paulo Gomes Mendes Moreira

Co-orientador: Prof. Dr. Paulo José Cerqueira Gomes da Costa

Julho de 2008

A Dissertação intitulada

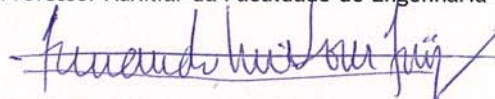
**“Estudo e Implementação de um Algoritmo de Localização
Baseado na Correspondência de Mapas”**

foi aprovada em provas realizadas em 15/Julho/2008

o júri

Presidente

Professor Doutor Armando Luís Sousa Araújo
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto



Professora Doutora Estela Guerreiro Silva Bicho Erlhagen
Professora Auxiliar da Escola de Engenharia da Universidade do Minho

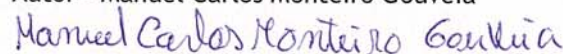


Professor Doutor António Paulo Gomes Mendes Moreira
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto



O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - Manuel Carlos Monteiro Gouveia



Faculdade de Engenharia da Universidade do Porto

Agradecimentos

Aos meus orientadores Prof. Dr. António Paulo Moreira e Prof. Dr. Paulo Gomes Costa pela oportunidade de investigação no grupo de Controlo e Robótica (5DPO) da FEUP, e pelo apoio e suporte durante a execução deste trabalho.

Ao Paulo Malheiros por partilhar os seus conhecimentos e tempo.

Aos meus companheiros e amigos de curso, que partilharem estes últimos anos de faculdade.

Aos meus pais que ao longo destes anos na faculdade sempre me apoiaram e me deram todas as condições para que tivesse sucesso.

Resumo

O problema da localização em robótica móvel é um problema chave que permite aos robôs serem totalmente autônomos. O robô necessita de saber a sua posição e orientação, em cada instante de tempo, para assim ser possível determinar o que fazer a seguir. Para se localizar, o robô tem acesso a informação relativa e absoluta sobre o ambiente que o rodeia, e assim deve ser capaz de determinar com a máxima precisão possível a sua localização. Para tal são necessários algoritmos eficientes que combinem essa informação para estimar a localização do robô.

Neste trabalho, foi implementado um algoritmo capaz de localizar robustamente e eficientemente o robô através de uma nova abordagem em termos de auto-localização de robôs. A abordagem é baseada na minimização de uma função custo que permite obter a estimativa da localização do robô de melhor qualidade, permitindo desta forma obter uma estimativa de forma robusta, precisa e computacionalmente eficiente.

É exemplificado como este algoritmo é versátil, podendo ser integrado com vários tipos de sensores e em variadas situações.

Abstract

The mobile robot localization problem is a key problem that allows robots to be truly autonomous. A robot needs to know its position and orientation, in each time stamp, so that it can determine what to do next. In order to localize itself, the robot has access to relative and absolute information about the environment surrounding it, then it must be able to determine its location as accurately as possible, and for that is needed an efficient algorithm that combines the information, in an optimal way, to find the robot location.

In this work it is implemented an algorithm able to locate the robot with robustness and efficiently through a new approach in robot self-localization. The approach is based on the minimization of an error function that allows us to obtain an estimate of better quality, allowing a more robust, precise and efficient localization of the robot.

It is shown how versatile this algorithm is, being able to be integrated with several types of sensors and in different situations. It's also identified how the fusion of different estimates is important for the robot localization.

Índice

Agradecimentos	v
Resumo	vii
Abstract.....	ix
Índice.....	xi
Lista de figuras	xiv
Abreviaturas e Símbolos	xviii
Capítulo 1 Introdução	1
1.1 Estado da Arte	1
1.1.1 Métodos Geométricos.....	2
1.1.2 Métodos Estatísticos	2
1.1.3 Métodos de correspondência de mapas	4
1.2 Enquadramento e Objectivos	5
1.3 Estrutura do documento	7
Capítulo 2 Algoritmo de Localização: Análise Matemática.....	9
2.1 Algoritmo de localização	9
2.2 Algoritmo “Resilient Propagation”, RPROP	12
2.3 Cálculo da primeira derivada da função custo	15
2.4 Cálculo da segunda derivada da função custo.....	16
2.5 Fusão da estimativa da localização	18
2.5.1 Variância da odometria	18
2.5.2 Variância da estimativa do algoritmo de localização	19
2.5.3 Fusão da estimativa com a odometria.....	20
2.6 Localização global	21
Capítulo 3 Sensores Utilizados	23

3.1	Ultra-sons	23
3.2	Sensores de Infra-vermelhos	25
3.3	Laser Range Finder	27
3.3.1	Teoria de operação	27
3.3.2	Hardware	28
3.3.3	Software	29
3.4	Visão artificial	32
Capítulo 4	Mapeamento	35
4.1	Criação do mapa de distâncias	36
4.1.1	Linhas	36
4.1.2	Arcos	37
4.2	Representação	39
4.3	Gradiente	40
Capítulo 5	Algoritmo de localização: Implementação	44
5.1	Minimização da função custo	44
5.2	Optimização da estimativa	46
5.3	Fusão da estimativa local com a odometria	48
5.3.1	Variância da estimativa do algoritmo de localização	48
5.3.2	Variância da odometria	50
5.3.3	Fusão das estimativas	51
5.4	Localização global	52
Capítulo 6	Resultados	55
6.1	Interface implementada	56
6.2	Futebol robótico	57
6.3	Medical Robot	61
6.3.1	Sonares	62
6.3.2	Infra-vermelhos	62
6.3.3	Localização	62
6.3.4	Variâncias e fusão das estimativas	66
6.4	Laser Range Finder	69
6.4.1	Resultados usando 115 pontos de medida	70
6.4.2	Resultados usando 34 pontos de medida	74
6.4.3	Resultados usando 12 pontos de medida	78
Capítulo 7	Conclusões e Trabalhos Futuros	79

Referências	81
Anexo A Funções do Algoritmo	85
A.1 Minimização da função custo.....	85
A.2 Optimização da estimativa da localização	86
A.3 Cálculo da segunda derivada e da variância a estimativa da localização	87
A.4 Fusão da estimativa da localização com a odometria	88
A.5 Localização global	89
A.6 Leitura de distância no mapa	90
A.7 Leitura do gradiente	90
Anexo B Funções de Mapeamento	91
B.1 Tratamento das Linhas	91
B.2 Tratamento dos Arcos.....	92
B.3 Actualização das distâncias no mapa	93
B.4 Cálculo do Gradiente.....	93
B.5 Desenho do mapa em forma de imagem	94
Anexo C Interface comunicação laser	97
C.1 Leitura do buffer de dados do laser	97
C.2 Extracção dos dados do buffer.....	97

Lista de figuras

Figura 2.1: Esboço do ponto da linha detectado, relativamente ao eixo de coordenadas do robô e ao eixo de coordenadas global [1]	10
Figura 2.2: Comparação da função de erro quadrático com a função mais robusta definida em (2.5) [11]	11
Figura 2.3: Diagrama de uma rede neuronal	12
Figura 2.4: Algoritmo “Resilient Propagation”	14
Figura 3.1: Sensor de Ultra-Sons	23
Figura 3.2: esquema simplificado do funcionamento do Sonar	24
Figura 3.3: Reflexão dos ultra-sons na parede	24
Figura 3.4: Disposição dos sonares no robô	25
Figura 3.5: Sharp's comerciais	25
Figura 3.6: Diferentes ângulos com diferentes distâncias medidos por um IR Sharp	26
Figura 3.7: Curva típica da saída analógica de um Sharp, em função da distância ao objecto.....	26
Figura 3.8: Exemplo de como fixar um IR no robô, para que a medida não tenha influência da “dead zone”	27
Figura 3.9: Hokuyo URG-04LX laser range finder.	27
Figura 3.10: Características das medidas do Hokuyo URG-04LX.....	28
Figura 3.11: Características físicas e de conexão do Hokuyo URG-04LX.	29
Figura 3.12: Configuração dos passos do laser Hokuyo URG-04LX	31
Figura 3.13: Interface de Leitura do “laser range finder” Hokuyo URG-04LX	32
Figura 3.14: Câmara de vídeo firewire	32
Figura 3.15: Imagem do campo de futebol obtida através da câmara de vídeo	33

Figura 3.16: Detecção das linhas por parte do algoritmo a partir de uma imagem fornecida pela câmara de vídeo.....	33
Figura 4.1: Distância de um ponto em relação a uma linha, horizontal	36
Figura 4.2: Distância de um ponto a uma linha diagonal	37
Figura 4.3: Distância de um ponto a uma linha rodada	37
Figura 4.4: Distância de um ponto a um arco de circunferência.....	38
Figura 4.5: Limite do ângulo do arco, para o qual se pode calcular normalmente a distância do ponto ao arco de circunferência.....	38
Figura 4.6: Aproximação da distância de um ponto a um arco de circunferência, caso o ponto se encontre fora do raio do arco	39
Figura 4.7: Interface de Criação de Mapas	40
Figura 4.8: Representação do gráfica gradiente segundo o eixo x do mapa do corredor DEEC	42
Figura 4.9: Representação gráfica do gradiente segundo o eixo y do mapa do corredor DEEC	42
Figura 4.10: Representação gráfica do gradiente do campo de futebol segundo x	42
Figura 4.11: Representação gráfica do gradiente do campo de futebol segundo y	42
Figura 5.1: Cálculo da função custo	45
Figura 5.2: Cálculo da estimativa da localização do robô	46
Figura 5.3: Algoritmo RPROP	47
Figura 5.4: Segunda derivada da função custo	48
Figura 5.5: Fusão das estimativas.....	51
Figura 5.6: Localização global, estimativa da localização para n posições aleatórias	52
Figura 5.7: Localização global, decisão sobre as poses aleatórias	53
Figura 6.1: Robô de futebol robótico da equipa 5DPO	55
Figura 6.2: <i>Medical Robot</i>	55
Figura 6.3: Representação da imagem do ambiente obtida pelo sensor laser	56
Figura 6.4: Interface do algoritmo de localização	57
Figura 6.6: Mapa de distâncias do campo de futebol.....	58
Figura 6.7: Imagem obtida pela câmara do robô, demonstrando também o processamento de imagem efectuado	59
Figura 6.7: Mapa do campo de futebol, onde é possível fazer visualização do estado dos robôs da equipa durante o jogo.	60

Figura 6.8: Mapa do campo de futebol robótico, com a representação da localização do robô (ponto vermelho), dos pontos da linha visíveis (pontos azuis) e das posições aleatórias para a localização global (pontos verdes)	60
Figura 6.9: Posicionamento dos sensores, sonares e sharps no Medical Robot	61
Figura 6.10: Mapa do corredor onde o robô circulou, a azul encontra-se o percurso efectuado pelo robô.	62
Figura 6.12: Percurso do robô usando apenas a odometria	64
Figura 6.13: Percurso do robô, resultante da localização usando o fusão da estimativa da odometria com a do algoritmo	64
Figura 6.14: Percurso do robô resultante da estimativa da posição obtida pelo algoritmo	64
Figura 6.15: Evolução da variação da odometria ao longo do deslocamento do robô, segundo o eixo x, o eixo y e a orientação ϕ do robô.	65
Figura 6.16: Evolução do deslocamento do robô segundo a odometria	65
Figura 6.17: Evolução do deslocamento do robô segundo o algoritmo de localização, antes da fusão de estimativas	66
Figura 6.18: Evolução do deslocamento do robô com a localização efectuada, depois da fusão de estimativas.....	66
Figura 6.18: Evolução da variância da odometria ao longo do deslocamento do robô	67
Figura 6.19: Evolução da variância do algoritmo de localização para o movimento no eixo x	67
Figura 6.20: Evolução da variância do algoritmo de localização para o movimento no eixo y	68
Figura 6.21: Evolução da variância do algoritmo de localização para a orientação do robô ...	68
Figura 6.22: Evolução das variâncias globais do robô, para o movimento x, y e orientação ϕ	69
Figura 6.23: Mapa e percurso efectuado pelo robô (linha a azul), usando 115 medidas do laser	70
Figura 6.25: Movimento do robô em pormenor na direcção positiva.....	71
Figura 6.25: Movimento do robô em pormenor na direcção negativa	71
Figura 6.26 Variação da posição (m) e orientação do robot (radianos) ao longo do tempo para todo o percurso efectuado	72
Figura 6.27: Variância da estimativa da posição segundo o eixo x	72
Figura 6.28: Variância da estimativa da posição segundo o eixo y	73
Figura 6.30: Variância da estimativa da orientação do robô	73
Figura 6.31: Variação do valor da função custo da estimativa ao longo do percurso efectuado pelo robô	74

Figura 6.32: Mapa e percurso efectuado pelo robô (linha a azul), usando 34 medidas do laser	74
Figura 6.32: Movimento do robô em pormenor na direcção positiva	75
Figura 6.33 Movimento do robô em pormenor na direcção negativa	75
Figura 6.34: Variação da posição (m) e orientação do robot (radianos) ao longo do tempo para todo o percurso efectuado	76
Figura 6.36 Variância da estimativa da posição segundo o eixo x	76
Figura 6.37: Variância da estimativa da posição segundo o eixo y	77
Figura 6.38: Variância da estimativa da orientação do robô.....	77
Figura 6.39: Variação do erro da estimativa ao longo do percurso efectuado pelo robô	78
Figura 6.39: Variação do valor da função custo da estimativa ao longo do percurso efectuado pelo robô	78

Abreviaturas e Símbolos

Abreviaturas

FEUP – Faculdade de Engenharia da Universidade do Porto

DEEC – Departamento de Engenharia Electrotécnica e de Computadores

5DPO – Equipa de Futebol Robótico da Faculdade de Engenharia da Universidade do Porto

IR – Infravermelhos

GPS - Global Positioning System (Sistema de Posicionamento Global, navegação através de uma rede de satélites universal)

FK – Filtro de Kalman

SLAM - Simultaneous Localization and Mapping

RPROP – Resilient Propagation

Símbolos

Símbolo	Significado
s_i	Posição (x,y) de um ponto no eixo do robô
p	Estimativa da posição do robô, segundo o algoritmo
ϕ	Estimativa da orientação do robô, segundo o algoritmo
d	Distância
err	Função custo
E	Erro da estimativa da localização do algoritmo
e	Aproximação da função custo
ω_{ij}	Peso do neurónio i ao j
s_i	Saída do neurónio i
net_i	Soma pesada das entradas do neurónio i
ϵ	Taxa de aprendizagem
μ	Parâmetro de momento
r_t	Estimativa da posição do robô, segundo a odometria
Ψ_t	Estimativa da orientação do robô, segundo a odometria
$\sigma_{r_x,t}^2$	Variância da estimativa da posição segundo x (odometria)
$\sigma_{r_y,t}^2$	Variância da estimativa da posição segundo y (odometria)
$\sigma_{\psi,t}^2$	Variância da estimativa da orientação (odometria)
$\sigma_{p_x,t}^2$	Variância da estimativa da posição segundo x (algoritmo)
$\sigma_{p_y,t}^2$	Variância da estimativa da posição segundo y (algoritmo)
$\sigma_{\phi,t}^2$	Variância da estimativa da orientação (algoritmo)
v	Velocidade linear
w	Velocidade angular

Capítulo 1

Introdução

Os robôs autónomos necessitam de saber a sua posição e orientação, em cada instante de tempo, para serem capazes de executarem tarefas simples como, por exemplo, movimentarem-se até uma determinada posição. Para isso são necessários métodos e algoritmos de localização que dependem do meio ambiente onde o robô se encontra.

Assim os pontos seguintes apresentam uma breve apresentação de vários algoritmos de localização utilizados em robótica móvel, bem como fazem uma apresentação dos objectivos e direcções tomadas no decorrer do projecto.

1.1 Estado da Arte

Vários métodos foram já estudados, no que diz respeito à localização absoluta de robôs. Os métodos mais interessantes são aqueles que não requerem preparação do ambiente, mas também são os de mais difícil tratamento. Nestes é comum o uso de algoritmos baseados em correspondência de mapas e estatísticos, por outro lado existem métodos que requerem a preparação do ambiente, o que por vezes é inviável, sendo comum serem usados algoritmos de localização baseados em trilateração e/ou triangulação com balizas. No primeiro caso é normal o uso de sensores como a bússola magnética, os marcos naturais, balizas e sistema GPS, entre outros. No segundo caso é comum usarem-se marcos artificiais, tais como etiquetas reflectoras, códigos de barras, formas geométricas, códigos de cores, sensores de efeito de hall e transponders embutidos no chão. Já os métodos de localização relativa são focados na informação obtida por sensores do robô que permitem medir a posição relativa deste através de odometria, sensores de Doppler, acelerómetros e giroscópios, [J. S. Esteves \(2005\), \[1\]](#).

A real dificuldade encontra-se nos algoritmos a aplicar para o tratamento dos dados obtidos através de algoritmos de localização, pelo que um vasto conjunto de trabalhos tem vindo a ser desenvolvido nesta área, ou seja, obter a melhor estimativa possível da posição e orientação do robô.

Os algoritmos de localização mais comuns podem ser divididos em três categorias:

- métodos geométricos
- métodos estatísticos
- métodos de correspondência de mapas.

Os pontos seguintes apresentam, com algum detalhe, cada um destes métodos.

1.1.1 Métodos Geométricos

Os algoritmos baseados em métodos geométricos são aqueles que usam a medida da distância e do ângulo a uma baliza, ou marco, e a partir dessa informação são capazes de localizar o robô no ambiente em que este se insere. Geralmente estes algoritmos, quando utilizam balizas, recorrem a métodos de triangulação e/ou trilateração sendo necessário balizas artificiais ou mesmo naturais desde que seja conhecida a sua correcta posição no sistema global de coordenadas. [Cohen et al \(1992\)](#), [2] explica como funciona o processo de localizar um robô a partir de três balizas, o qual é designado por “*Three Object Localization*”.

Por sua vez [Betke et al \(1997\)](#), [3] descrevem um método eficiente para a localização de robôs, usando balizas, assumindo que o robô é capaz de identificá-las e medir a orientação entre elas, pelo que o tempo de processamento é linearmente dependente do número de balizas visíveis.

[J. S. Esteves et al \(2003\)](#), [4] propõem o uso generalizado da triangulação geométrica para a localização absoluta do robô. Note-se que este método está sujeito a algumas restrições que o tornam inviável em muitos dos casos. Isto porque:

- é necessário que o robô observe pelo menos três balizas distintas;
- a localização falha sempre que o robô se encontra sobre a circunferência definida pelas três balizas;
- o ângulo entre balizas deve ser menor que 180°;
- o método só é consistente se o robô se encontrar no interior do triângulo definido pelas três balizas.

[A. Sousa et al \(2005\)](#), [5] propõem o uso de balizas coloridas combinadas com cálculo geométrico e odometria, que, na prática, se o ambiente não for correctamente tratado, tem a sua robustez posta em causa pois as balizas coloridas são facilmente confundidas e misturadas com outros objectos coloridos nas imediações da mesma. Assim, quanto maior for a área de funcionamento do robô e menor a quantidade de balizas disponíveis, menor a viabilidade da utilização de balizas coloridas na auto-localização.

1.1.2 Métodos Estatísticos

Desde o seu aparecimento que os métodos estatísticos têm sido largamente utilizados na localização de robôs. Bons exemplos são o Filtro de Bayes usado como base em muitos algoritmos por ser um bom estimador de estado de sistemas dinâmicos

baseados na informação sensorial, e o Filtro de Kalman, muito usado não só em métodos de localização, mas também em métodos de fusão da informação sensorial, como evidenciado em [6] (nesta secção será seguida a notação desta referência).

O Filtro de Kalman (FK) é um algoritmo de processamento de dados recursivo que estima o estado de um sistema dinâmico com ruído. Para estimar o estado, o FK tem acesso às medidas do sistema que estão linearmente relacionadas com o estado e são afectadas por ruído. Assim, o FK processa todas as medidas disponíveis para estimar o estado, usando o conhecimento do sistema, da dinâmica dos sensores, da distribuição probabilística do ruído (quer dos sensores quer do próprio estado), das medidas com ruído e qualquer outra informação sobre o estado inicial do sistema.

J. J. Leonard and H. F. Durrant-Whyte (1992), [7] demonstram que a implementação do Filtro de Kalman, usando sensores de ultra-sons, na localização de robôs provou ser robusta e exacta. Contudo não consegue representar ambiguidades e sofre da falta de habilidade para relocalizar globalmente o robô no caso de falhas de localização. R. Negenborn et al (2003), [8] propõem o uso do filtro de Kalman, não como função para estimar a posição do robô, mas como meio de combinação das várias estimativas da posição dadas por dois ou mais sensores (fusão sensorial), pois o filtro de Kalman é o melhor estimador possível para uma larga classe de sistemas com incerteza. É ainda um óptimo estimador para uma classe de sistemas ainda mais abrangente, sendo um dos mais conhecidos e usados na estimação de processos estocásticos para sensores com ruído. Da mesma forma D. Cabecinhas et al (2006), [9] demonstram que a fusão sensorial usando o Filtro de Kalman permite obter uma melhor estimativa da localização do robô.

O algoritmo de Monte Carlo usa um Filtro de Partículas combinado com modelos probabilísticos da percepção do robô e do movimento deste. O princípio de funcionamento utiliza o Filtro de Bayes como estimador do estado do sistema dinâmico, baseado na informação sensorial disponível D. Fox et al (2003), [10].

Neste algoritmo o primeiro passo é o “lançamento” de um dado número de partículas pelas posições de interesse, do mapa, onde o robô pode estar, pelo que com o decorrer das iterações (cálculo da probabilidade de cada partícula), as partículas com menor probabilidade são eliminadas e criadas (“lançadas”) de novo em outros locais de interesse, geralmente próximos dos locais onde existem partículas com probabilidade maior. Desta forma, dá-se uma convergência de quase todas as partículas para o local de maior probabilidade, pois garante-se que existe um número mínimo de partículas espalhadas pelo espaço admissível de forma a garantir uma localização global mesmo quando o robô é “raptado”. Por outro lado, o algoritmo de Monte Carlo, embora robusto, despende muito tempo na avaliação das posições de interesse, uma vez que segue o paradigma da pesquisa cega e tal como alguns testes práticos já o demonstraram, aproximadamente 98% das posições examinadas não contribuem para a posição final estimada, Isto porque as posições são examinadas mesmo que as posições vizinhas tenham sido consideradas más estimativas. Contudo consegue respeitar os requisitos de tempo-real para uma parte das aplicações de auto-localização de robôs, M. Lauer et al (2005), [11].

I. Rekleitis (2004), [12] publicou um tutorial onde demonstra como usar o Filtro de Partículas aplicado à robótica móvel. Mais tarde F. G. Bravo et al (2007), [13] . estudaram a utilização do Filtro de Partículas na cooperação entre robôs. Devido à forma como faz a sua representação o Filtro de Partículas combina eficiência com a habilidade para representar uma larga quantidade de densidades de probabilidade. A sua eficiência reside na forma como usa os recursos computacionais disponíveis, obtendo amostras em proporção à zona provável. O Filtro de Partículas foca a capacidade computacional nas regiões do espaço de estados com maior probabilidade.

A abordagem de Markov para a localização considera que, para todo o instante de tempo, é mantida uma densidade de probabilidade da posição do robô, baseada nos dados dos sensores e nas acções realizadas por este. Segundo este algoritmo, o mapa é dividido numa grelha em que cada posição contém uma densidade de probabilidade da posição do robô. Assim, em cada iteração, para todas as células da grelha é calculada a respectiva densidade de probabilidade. Esta rede de probabilidades aplica distribuições multimodais para a densidade de probabilidade da posição, permitindo a representação de situações ambíguas por considerar múltiplas hipóteses em paralelo. C. Penedo et al (2003), [14] implementaram o método de localização de Markov usando a distribuição de probabilidade ao longo de uma grelha de granulação fina que define as posições possíveis do robô de forma a este se localizar globalmente, mesmo na presença de ruído. Tal como a mesma afirma, o facto de ser necessário um grande espaço de estados torna este algoritmo problemático e pouco fiável, até porque mesmo com certas optimizações apenas resolve o problema dentro de uma determinada extensão, continuando a haver limitações na escolha do tamanho adequado para a grelha.

O vSLAM, variante do algoritmo SLAM (*Simultaneous Localization and Mapping*), algoritmo este que tem por objectivo prover o robô de total autonomia, ou seja, o robô deve ter a habilidade de explorar o ambiente sem intervenção de um utilizador externo, bem como ser capaz de construir um mapa do ambiente explorado e auto-localizar-se nele. A variante vSLAM utiliza um algoritmo de reconhecimento de objectos, baseado em visão, e um algoritmo de factorização para construir e manter um mapa baseado em marcadores. Para proceder à localização utiliza o Filtro de Partículas.

N. Karlsson (2005), [15] afirma que o vSLAM é um algoritmo de baixo custo, pois permite a utilização de sensores baratos, sem que haja grande influência na estimativa calculada. Afirma ainda que este algoritmo é extremamente robusto no que toca à auto-localização do robô após mudança de ambiente, como mudanças de luminosidade, objectos ou pessoas a movimentarem-se. O robô recupera rapidamente de uma situação de “rapto” logo que o mapa fique preenchido de marcadores, contudo em ambientes amplos a base de dados de marcadores torna-se demasiado grande e se o ambiente não contiver elementos visuais suficientes, os marcadores não podem ser criados, não sendo possível produzir uma boa estimativa da posição.

1.1.3 Métodos de correspondência de mapas

Os métodos de correspondência de mapas usam um mapa 2D do ambiente onde o robô vai actuar, e através da posição estimada do robô em 2D, proveniente dos sensores,

são capazes de calcular a função de “erro” dessa posição e determina a localização óptima, segundo um critério, do robô no mapa, ou seja, fazem a correspondência entre a informação dos sensores do robô e o mapa.

X. Yun et al (1998), [16] defendem o uso da detecção de linhas através da transformada de Hough usando também redes neuronais. O uso da transformada de Hough necessita de efectuar cálculos complexos usando um acumulador tridimensional. Como são efectuadas grandes quantidades de cálculos não necessários para a determinação da posição, qualquer aumento de precisão implica um aumento de tempo computacional.

Mais recentemente M. Lauer et al (2005), [11] propuseram, para o futebol robótico, um algoritmo baseado no princípio de minimização da distância entre as linhas brancas do campo de futebol, vistas pela câmara, e a posição esperada dessas linhas no mapa. O método proposto usa um algoritmo de convergência (Resilient Propagation - RPROP) que também costuma ser utilizado em redes neuronais, M. Riedmiller and H. Braun 1993, [17]. Este permite uma convergência mais rápida, da localização estimada do robô, para a localização mais provável deste. Adicionalmente, é possível obter uma medida da confiança da posição estimada analisando a estrutura da função custo, de forma a ser possível aplicar um processo de fusão sensorial para aumentar a precisão da estimativa.

Devido à versatilidade, eficiência, reduzido peso computacional e pela possibilidade de poder integrar diversos sensores, o algoritmo proposto por M. Lauer et al (2005), [11] foi o seleccionado para se implementar.

1.2 Enquadramento e Objectivos

Um robô móvel é uma combinação de vários componentes físicos (hardware e mecânica) e computacionais (software). O principal objectivo da robótica móvel é o desenvolvimento de sistemas autónomos e inteligentes, capazes de actuarem em ambientes reais e situações desconhecidas. Dentre as diversas linhas de pesquisa, exploradas na área da robótica móvel, as funções relativas à mobilidade, como localização e navegação, têm grande importância no desenvolvimento de sistemas inteligentes autónomos.

A localização de robôs móveis é motivo de intensa investigação científico-tecnológica, pelo que foram já desenvolvidas, aplicadas e estudadas uma grande diversidade de técnicas, baseadas em diferentes princípios físicos e diferentes algoritmos para localização.

A localização em ambiente estruturado socorre-se, em geral, de elementos externos que são denominados de marcos. É possível utilizar marcos naturais já existentes no ambiente. Outra possibilidade passa por acrescentar propositadamente, ao ambiente, marcos artificiais, ou balizas para a localização do robô. Alguns trabalhos utilizam medida de distância a marcos naturais tais como paredes. Outros exploram a localização por rádio frequência, havendo ainda outros autores que utilizam técnicas baseadas em

sistemas de visão que permitem a identificação de diversos elementos visuais da imagem obtida.

Assim, um dos maiores problemas da robótica móvel é estimar, de forma fiável e exacta, a posição global do robô num determinado ambiente, ou seja, o robô é colocado num ambiente conhecido e deve ser capaz de se auto-localizar, sem conhecimento da sua posição inicial, bem como de fazer o rastreamento de sua posição local em função do tempo, isto é, ter a percepção do seu movimento. Conhecendo a sua posição global, o robô pode navegar de maneira confiável. Rastreando a sua posição local, o robô pode executar uma navegação eficiente e realizar as tarefas para as quais foi criado. Para que isso seja possível é necessário estimar o estado do robô, em tempo real, com base no estado inicial e em todas as medidas obtidas pelos seus sensores.

Em suma: O problema da localização de robôs é um “problema fundamental para dotar os robôs das verdadeiras capacidades de autonomia”.

Quando se fala de localização, pose ou posição referimo-nos às coordenadas X e Y do robô e à sua orientação no referencial global de coordenadas.

Para se obter a percepção de localização vários métodos têm sido propostos, desde métodos baseados em mapas, construídos pelo próprio robô, em mapas pré-programados, e também no reconhecimento de balizas, ou marcos, com posicionamento pré-definido. Tipicamente, estes métodos apresentam algumas características menos desejáveis, isto é, são computacionalmente exigentes, requerem um tempo exploratório considerável e são inflexíveis.

Os algoritmos de localização baseados no algoritmo (original) de Monte Carlo são aqueles que têm apresentado os melhores resultados na localização de robôs quando comparados com outros algoritmos existentes. Em contraste com os algoritmos baseados no Filtro de Kalman, este algoritmo tem a capacidade de localizar um robô globalmente. Além disso, reduz drasticamente o uso de memória, comparado com algoritmos baseados na *Localização de Markov* e pode integrar medidas numa frequência consideravelmente mais elevada. É mais exacto que os algoritmos baseados na *Localização de Markov*, com um tamanho fixo de célula, já que os estados representados pelos exemplos não são discretizados. É também mais simples de implementar, por manipular menos dados e exigir equações e cálculos mais simples. Apesar disso, os requisitos temporais aumentam com a necessidade do aumento da exactidão, ou seja, aumento do número de partículas necessárias.

O método proposto em [M. Lauer et al \(2005\), \[11\]](#) vem colmatar alguns dos problemas encontrados noutros métodos, no sentido de se obter uma localização mais precisa, computacionalmente mais eficiente e ainda com melhor exactidão e robustez.

Finalmente, é também de grande importância a combinação da informação da posição inicial, da relativa e da absoluta para calcular estimativas da posição do robô num certo espaço temporal. Se as medidas forem leituras directas de sensores, o problema será como combiná-las.

A fusão da informação de múltiplos sensores é importante, pois a informação combinada é mais precisa. Alguma informação pode não estar disponível a alguns sensores, enquanto está visível para outros. Em conjunto, os sensores podem fornecer uma imagem completa do ambiente, num determinado instante de tempo. A fusão multi-sensorial é pois importante pois permite reduzir a variância da estimativa.

O presente trabalho pretende estudar e implementar o algoritmo de localização proposto em [M. Lauer et al \(2005\), \[11\]](#) para um robô móvel num ambiente estruturado baseado na correspondência entre o mapa do mundo e a informação dos sensores. Os sensores utilizados são a câmara de vídeo (localização baseada em visão), ultra-sons, infra-vermelhos e um *laser range finder*, não sendo estes usados em simultâneo, uma vez que se pretende verificar o seu desempenho independentemente. Pretende-se usar ainda odometria que embora não seja objecto de estudo neste trabalho, é usada na fusão com a estimativa da localização.

O algoritmo implementado deverá ser rápido, com características de tempo real e, ao mesmo tempo, aplicável a diferentes situações e tipos de robôs, tais como em ambientes que podem ir de um campo de futebol robótico a um piso de um Hospital, e com robôs que podem ser um jogador de futebol robótico (com características de movimento omnidireccional) ou uma cadeira de rodas (com características de movimento diferencial), sem necessidade de balizas ou marcadores artificiais.

Desenvolveu-se o algoritmo para um ambiente bem estruturado, no caso um campo de futebol robótico. O sensor usado foi uma câmara de vídeo, montada no robô de futebol robótico, e colocada de forma a ser possível obter uma imagem de 360° em torno do robô.

Posteriormente aplicou-se o algoritmo usando um robô com movimento omnidireccional num corredor, com os restantes sensores, *laser range finder*, sensores de ultra-sons e sensores de infra-vermelhos. Concluída esta fase efectuou-se uma fusão das estimativas dada pelo algoritmo desenvolvido com a dada pela odometria do robô usando uma média pesada, com base na variância das diferentes estimativas, para, assim, aprimorar a estimativa da posição do robô.

1.3 Estrutura do documento

Este documento descreve o trabalho realizado e encontra-se dividido em sete capítulos, incluindo introdução e conclusões.

O Capítulo 1 encontra-se dividido em três secções. A secção 1.1 ilustra o estado da arte, com a apresentação dos algoritmos de localização já implementados e estudados. Segue-se o enquadramento e objectivos do trabalho em questão na secção 1.2. O capítulo termina nesta secção, que apresenta a estrutura deste documento e serve para o leitor ver se percebeu o focus do Capítulo 1.

No Capítulo 2 apresenta-se uma abordagem matemática do algoritmo. Assim na secção 2.1 define-se o algoritmo. A secção 2.2 descreve depois como convergir a estimativa para a localização do robô usando um algoritmo de gradiente descendente. O

capítulo prossegue com o cálculo da segunda derivada da estimativa, na secção 2.3. Para estudar os locais de possível ambiguidade, passando pela fusão da estimativa da localização calculada com a estimativa da odometria, na secção 2.4, e finalizando com a localização global do robô na secção 2.5.

Em seguida, o Capítulo 3 apresenta os sensores utilizados durante este trabalho e a forma como estes obtêm as medidas de distâncias necessárias ao algoritmo. Apresentam-se os sensores de ultra-sons, na secção 3.1, os sensores de infra-vermelhos, nomeadamente os Sharp, na secção 3.2, os sensores laser, em específico o sensor Hokuyo URG40-LX, na secção 3.3 e o processamento de imagem da câmara de vídeo, na secção 3.4.

O Capítulo 4 trata, na secção 4.1, da criação de mapas, identificando como são construídos os mapas necessários ao algoritmo, e as funções implementadas para a sua criação. A representação destes, na secção 4.2, e ainda o cálculo do gradiente na secção 4.3.

O Capítulo 5 apresenta a implementação do algoritmo do Capítulo 2. Começando pela minimização da função custo, na secção 5.1 e tratando a optimização da estimativa, na secção 5.2. A fusão das estimativas é focada na secção 5.3 e por fim a implementação do algoritmo para a localização global é apresentada na secção 5.4.

O Capítulo 6 ilustra os resultados obtidos da implementação do algoritmo em diferentes ambientes, com diferentes robôs e com diferentes sensores. Na secção 6.1 apresentam-se a interface de localização implementada. Os resultados obtidos no futebol robótico, são apresentados na secção 6.2. Seguidamente apresenta-se os resultados obtidos com o *medical robot*. Sem um robô em específico, apresenta-se o comportamento do algoritmo usando o sensor laser, na secção 6.3. Por fim apresenta-se a análise feita à cadeira de rodas autónoma na secção 6.4.

Finalmente, e para além das referência bibliográficas citadas e dos anexos, o relatório termina com um capítulo de conclusões onde são realizados comentários acerca do projecto e de possíveis melhoramentos futuros.

Capítulo 2

Algoritmo de Localização: Análise Matemática

O algoritmo de localização implementado tem por base o algoritmo desenvolvido pela equipa de futebol robótico Tribots, [18]. Após um primeiro estudo do trabalho desenvolvido por estes, [M. Lauer et al \(2005\), \[11\]](#) e [A. Merke et al \(2005\), \[19\]](#), verificou-se, a partir das conclusões e comparações com outros algoritmos já implementados, nomeadamente o algoritmo de localização de Monte Carlo, [F. Dellaers et al \(1999\), \[20\]](#), que o algoritmo de minimização ultrapassa a maioria das limitações desses algoritmos, em termos de robustez, eficiência, exactidão e menor tempo de processamento.

Esta abordagem é baseada num modelo da qualidade da estimativa, usando uma função custo e uma minimização numérica desta. Assim, o algoritmo é do tipo de correspondência de mapas, pelo que usa uma lista de pontos com a qual faz a correspondência entre as coordenadas destes com a respectiva posição no mapa. A melhor estimativa da localização do robô é dada calculando uma função custo que pesa a posição dos pontos visíveis, em relação ao robô, com a sua posição esperada no mapa. Estes pontos correspondem a paredes, obstáculos, ou linhas no chão, cuja posição no mapa é conhecida, conforme os sensores utilizados, pelo que a minimização desse erro permite encontrar a estimativa da posição com menor erro.

Assim, as secções seguintes apresentam o algoritmo estudado em termos matemáticos.

2.1 Algoritmo de localização

O algoritmo de localização utiliza os dados provenientes de vários sensores, informação que é relativa ao robô onde estão inseridos, essencialmente medidas de distâncias. O objectivo é fazer com que os dados obtidos coincidam com a informação, disponibilizada num mapa, assumindo, desta forma, a melhor posição e orientação possível, de acordo com os dados. Daí ser um algoritmo de localização, pois calcula a

posição e orientação do robô que corresponde ao menor erro de correspondência com o mapa.

Seja $(\mathbf{p}, \phi)$ o par que define a posição $\mathbf{p} = (p_x, p_y)$ e orientação ϕ , do robô no sistema de coordenadas global. Para uma lista de pontos, $\mathbf{s}_1, \dots, \mathbf{s}_n$, detectados pelos sensores do robô relativamente a este e expresso no referencial do robô, temos que posição de cada um desses pontos, $\mathbf{P}$, no sistema de coordenadas global é dada por:

$$\mathbf{p} + \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \mathbf{s}_i \quad (2.1)$$

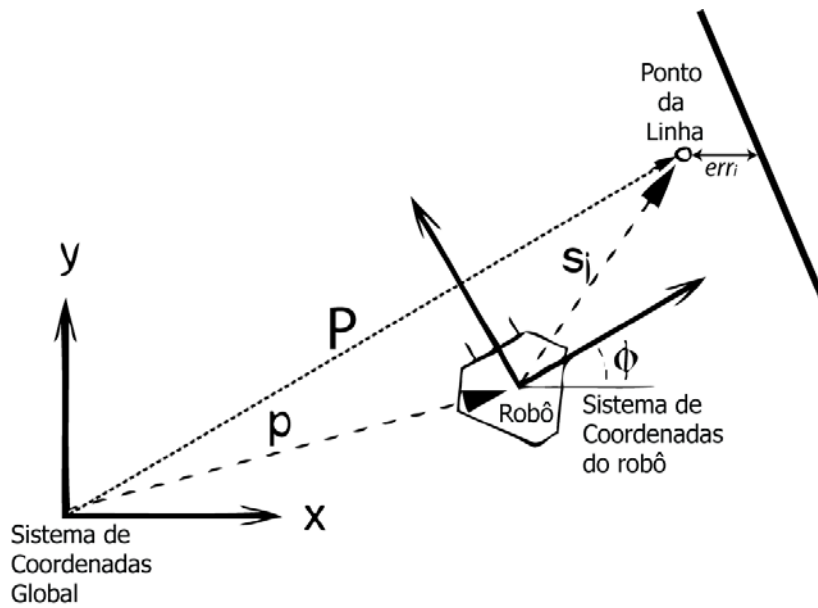


Figura 2.1: Esboço do ponto da linha detectado, relativamente ao eixo de coordenadas do robô e ao eixo de coordenadas global [1]

Seja

$$d \left(\mathbf{p} + \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \mathbf{s}_i \right) \quad (2.2)$$

a distância, uma função contínua, seccionalmente diferenciável e que pode ser obtida através do conhecimento das medidas do mapa (parede, linhas de um campo de futebol robótico, um objecto), e seja a função custo, que minimiza o err_j , isto é, o desvio entre o ponto do mapa e o ponto da linha, equação (2.1), dada por :

$$\sum_{i=1}^n \text{err} \left(d \left(\mathbf{p} + \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \mathbf{s}_i \right) \right) \quad (2.3)$$

em que err é a função que dá o custo de um ponto.

Para estimar a posição e orientação do robô calcula-se a minimização desta função custo, pois a correspondência com menor erro é a melhor estimativa possível calculada com os pontos adquiridos pelos sensores:

$$\text{minimize } E = \sum_{i=1}^n \text{err} \left(d \left(\mathbf{p} + \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \mathbf{s}_i \right) \right) \quad (2.4)$$

Devido ao ruído inerente à imperfeição da informação sensorial, que nunca é exacta, por diversos motivos, dependendo do tipo de sensor utilizado, somos confrontados com uma quantidade substancial de pontos mal detectados, o que influencia e distorce a estimativa que se está a calcular. Por esse motivo é usada uma aproximação da função custo que limita a influência dos pontos que têm demasiado erro. Assim, a aproximação da função custo é:

$$\text{err}() = 1 - \frac{c^2}{c^2 + \left(d \left(\mathbf{p} + \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \mathbf{s}_i \right) \right)^2} \quad (2.5)$$

Esta função fica similar a uma função de erro quadrática, que está limitada superiormente pela constante c , para erros elevados, pelo que a influência de pontos mal adquiridos é reduzida [11].

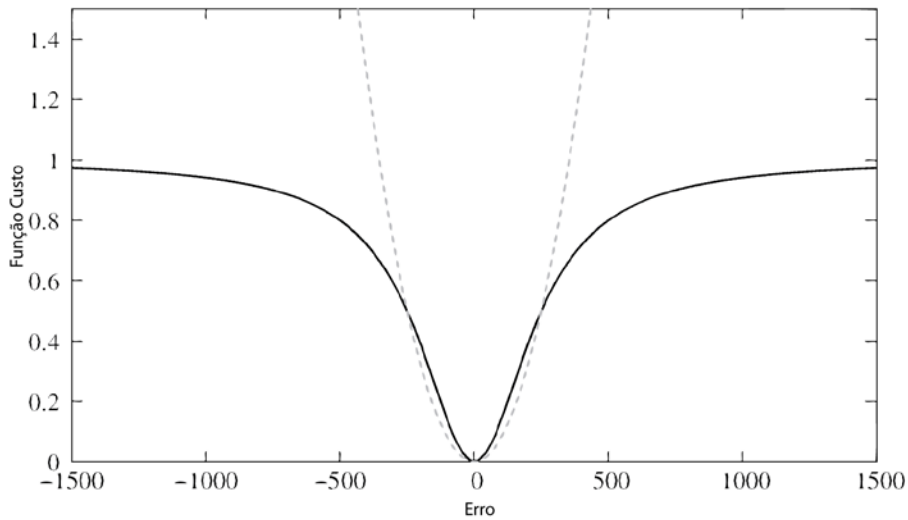


Figura 2.2: Comparação da função de erro quadrático com a função mais robusta definida em (2.5) [11]

Devido à não linearidade da função de minimização dada pela equação (2.4) não é possível calcular analiticamente a sua solução. Como a equação (2.2) é diferenciável, em praticamente todo o mapa, é também possível calcular o seu gradiente na globalidade do mapa, sendo necessário interpolar apenas a pequena quantidade de pontos não diferenciáveis, que se encontram nos limites do mapa. Para resolver o problema da não linearidade da função de minimização podemos recorrer ao método do gradiente descendente, isto é, usar o algoritmo de treino “Resilient PROPagation” (RPROP), para o qual é necessário o gradiente da função custo. Devido à rápida convergência deste algoritmo e à sua elevada robustez, é possível resolver convenientemente, com algumas iterações, a tarefa de minimização, obtendo-se desta forma uma convergência rápida para a melhor estimativa da posição e orientação do robô [11].

2.2 Algoritmo “Resilient Propagation”, RPROP

A retropropagação (BackPropagation) [17] é o algoritmo mais usado em aprendizagem supervisionada em redes neuronais de múltipla camada. A ideia básica do algoritmo de aprendizagem de retropropagação é a repetição da aplicação de uma cadeia de regras para calcular a influência de cada peso da rede com respeito à arbitrariedade da função custo E :

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial net_j} \frac{\partial net_j}{\partial \omega_{ij}} \quad (2.6)$$

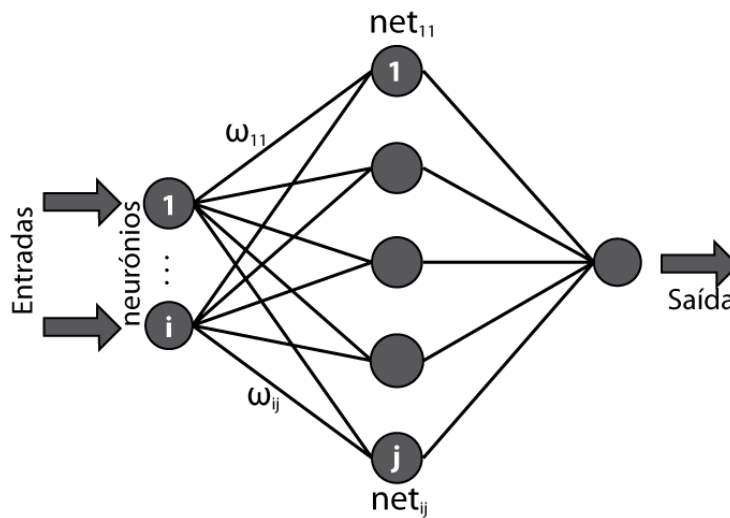


Figura 2.3: Diagrama de uma rede neuronal

Onde ω_{ij} é o peso do neurónio j ao neurónio i , s_i é a saída, e net_i é a soma pesada das entradas do neurónio i . Uma vez conhecida a derivada parcial para cada peso, o objectivo de minimizar a função custo é atingido resolvendo o gradiente descendente:

$$\omega_{ij}(t+1) = \omega_{ij}(t) - \epsilon \frac{\partial E}{\partial \omega_{ij}}(t) \quad (2.7)$$

Obviamente a escolha da taxa de aprendizagem, que escala a derivada, tem um efeito importante no tempo necessário até a convergência ser alcançada. Se este valor for demasiado pequeno são necessários muitos passos para se alcançar uma solução aceitável; pelo contrário, uma taxa de aprendizagem elevada leva à oscilação, evitando que o erro caia abaixo de um determinado valor. Uma das primeiras formas propostas para resolver este problema foi a introdução de um termo de momento em que o parâmetro de momento μ escala a influência do passo anterior no corrente passo [17]:

$$\Delta\omega_{ij}(t) = -\epsilon \frac{\partial E}{\partial \omega_{ij}}(t) + \mu\Delta\omega_{ij}(t-1) \quad (2.8)$$

Acredita-se que o termo de momento torna o processo de aprendizagem mais estável e que acelera a aprendizagem nas regiões superficiais da função custo. No entanto, a experiência prática mostra que isso nem sempre é verdade, pois o valor óptimo para o parâmetro de momento μ tem igualmente um problema de dependência, como a taxa de aprendizagem, pelo que a melhoria geral não é conseguida, [17].

Assim, foram desenvolvidos vários algoritmos de adaptação da aprendizagem para lidar com o problema de calcular o valor apropriado dos pesos. Um desses algoritmos é o Resilient Propagation (RPROP), desenvolvido com o intuito de evitar o problema da “adaptabilidade obscurecida”, assim o RPROP altera o valor da actualização do peso, $\Delta\omega_{ij}$, directamente sem considerar o valor da derivada parcial.

O RPROP efectua a adaptação directa do peso de cada passo baseado na informação do gradiente local, não sendo o esforço de adaptação, obscurecido pelo comportamento do gradiente.

Para atingir isto, cada peso tem o seu próprio valor de actualização Δ_{ij} , que somente determina o tamanho da actualização do peso. Este valor evolui durante o processo de aprendizagem baseado na visão sobre a função custo E de acordo com a seguinte regra de aprendizagem:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ * \Delta_{ij}^{(t-1)}, & \text{se } \frac{\partial E}{\partial \omega_{ij}}(t-1) * \frac{\partial E}{\partial \omega_{ij}}(t) > 0 \\ \eta^- * \Delta_{ij}^{(t-1)}, & \text{se } \frac{\partial E}{\partial \omega_{ij}}(t-1) * \frac{\partial E}{\partial \omega_{ij}}(t) < 0 \\ \Delta_{ij}^{(t-1)}, & \text{se } \frac{\partial E}{\partial \omega_{ij}}(t-1) * \frac{\partial E}{\partial \omega_{ij}}(t) = 0 \end{cases} \quad (2.9)$$

Deste modo, sempre que a derivada do peso correspondente ω_{ij} muda de sinal, o que indica que a última actualização foi muito alta e que o algoritmo saltou para um mínimo local, pelo que, o valor de actualização Δ_{ij} é diminuído pelo factor η^- . Se a derivada mantiver o sinal, o valor de actualização é ligeiramente aumentado com o objectivo de acelerar a convergência nas regiões superficiais.

Quando o valor de actualização para cada peso estiver adaptado, a actualização dos pesos segue uma regra simples: se a derivada é positiva, o erro aumentou, logo o peso é diminuído pelo valor de actualização; se a derivada é negativa, o valor de actualização é somado:

$$\Delta_{\omega_{ij}}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}, & \text{se } \frac{\partial E}{\partial \omega_{ij}}(t) > 0 \\ +\Delta_{ij}^{(t)}, & \text{se } \frac{\partial E}{\partial \omega_{ij}}(t) < 0 \\ 0, & \text{se } \frac{\partial E}{\partial \omega_{ij}}(t) = 0 \end{cases} \quad (2.10)$$

Note-se que existe uma excepção, que ocorre quando a derivada parcial muda de sinal, ou seja, quando o passo anterior foi tão grande que o mínimo desaparece, nesse caso a actualização do peso é revertida:

$$\Delta_{\omega_{ij}}^{(t)} = -\Delta_{\omega_{ij}}^{(t-1)}, \text{ se } \frac{\partial E}{\partial \omega_{ij}}(t-1) * \frac{\partial E}{\partial \omega_{ij}}(t) < 0 \quad (2.11)$$

Assim, vem o seguinte algoritmo de RPROP [11]:

```

If  $\frac{\partial E}{\partial \omega_{ij}}(t) \neq 0$  then begin
    if  $\frac{\partial E}{\partial \omega_{ij}}(t-1) * \frac{\partial E}{\partial \omega_{ij}}(t) > 0$  then
         $\Delta_{ij}^{(t)} = \text{minimo}(\Delta_{ij}^{(t-1)} * \eta^+, \Delta_{max})$ 
    Else if  $\frac{\partial E}{\partial \omega_{ij}}(t-1) * \frac{\partial E}{\partial \omega_{ij}}(t) < 0$  then
         $\Delta_{ij}^{(t)} = \text{máximo}(\Delta_{ij}^{(t-1)} * \eta^-, \Delta_{min})$ 
End
If  $\frac{\partial E}{\partial \omega_{ij}}(t) > 0$  then
     $\omega_{ij}(t+1) = \omega_{ij}(t) - \Delta_{ij}^{(t)}$ 
Else if  $\frac{\partial E}{\partial \omega_{ij}}(t) < 0$  then
     $\omega_{ij}(t+1) = \omega_{ij}(t) + \Delta_{ij}^{(t)}$ 
 $\frac{\partial E}{\partial \omega_{ij}}(t-1) = \frac{\partial E}{\partial \omega_{ij}}(t)$ 

```

Figura 2.4: Algoritmo “Resilient Propagation”

2.3 Cálculo da primeira derivada da função custo

O algoritmo RPROP necessita da informação da primeira derivada da função custo, apresentada na secção 2.1, em ordem à posição e à orientação do robô. A função custo é dada pela equação (2.4), sendo os cálculos da primeira derivada efectuados, usando a aproximação da função custo, equação (2.5).

Seja então a derivada da função custo em ordem à distância:

$$\frac{\partial err}{\partial d} = \frac{\partial}{\partial d} \left(1 - \frac{c^2}{c^2 + d^2} \right) \quad (2.12)$$

$$\frac{\partial err}{\partial d} = \frac{2 * c^2 * d}{(c^2 + d^2)^2} \quad (2.13)$$

A derivada da distância em ordem à posição do robô, ou seja, o gradiente em ordem a x e y, é obtida através da matriz de gradiente definida no Capítulo 4:

$$\frac{\partial d}{\partial \mathbf{p}}(x, y) \cong \text{gradiente}(\mathbf{p}) \quad (2.14)$$

Temos que a derivada da função custo em ordem à posição do robô é:

$$\frac{\partial err}{\partial \mathbf{p}}(x, y) = \frac{\partial err}{\partial d} * \frac{\partial d}{\partial \mathbf{p}}(x, y) \quad (2.15)$$

E a derivada da função custo em ordem orientação do robô é:

$$\frac{\partial err}{\partial \phi} = \frac{\partial err}{\partial d} * \frac{\partial d}{\partial \mathbf{p}} * \frac{\partial \mathbf{p}}{\partial \phi} \quad (2.16)$$

$$\frac{\partial err}{\partial \phi} = \frac{\partial err}{\partial d} * \begin{bmatrix} \frac{\partial d}{\partial p_x} \\ \frac{\partial d}{\partial p_y} \end{bmatrix} * \begin{bmatrix} -\sin(\phi) & -\cos(\phi) \\ \cos(\phi) & -\sin(\phi) \end{bmatrix} * \mathbf{s}_i \quad (2.17)$$

Em que:

$$\begin{bmatrix} -\sin(\phi) & -\cos(\phi) \\ \cos(\phi) & -\sin(\phi) \end{bmatrix} * \mathbf{s}_i = \frac{\partial d}{\partial \phi} \quad (2.18)$$

2.4 Cálculo da segunda derivada da função custo

Na maior parte das situações é possível obter um mínimo local distinto, ou seja, a tarefa de otimização é bem definida e é possível estimar com bastante fiabilidade a posição e orientação do robô. Outras situações há em que ocorre ambiguidade na estimativa da localização do robô, tanto devido à pouca informação disponibilizada pelos sensores (número reduzido de medidas), como a uma fraca estrutura dos pontos medidos, e até ao local onde o robô se encontra no mapa, ou seja, locais do mapa que são idênticos. Isto acontece quando o robô tem, por exemplo, informação suficiente para estimar a sua posição robustamente no eixo de coordenadas Ox , mas o mesmo não acontece para o eixo de coordenadas Oy . Desta forma a função custo é caracterizada por um vale com valores semelhantes.

Para resolver este problema de ambiguidade, é necessário reconhecer três situações possíveis:

- (a) A função custo exhibe um mínimo global distinto, assim é possível estimar $\mathbf{p}$ e ϕ com fiabilidade;
- (b) A função custo é totalmente plana, devido ao reduzido número de pontos, pelo que não é possível estimar qualquer parâmetro;
- (c) A função custo exhibe uma estrutura tipo vale em volta do mínimo. Neste caso podem-se estimar, robustamente, os parâmetros referentes ao eixo de coordenadas ortogonal ao vale, mas não se podem estimar os parâmetros referentes ao eixo de coordenadas paralelo ao vale, ou seja, é possível, estimar $\mathbf{p}_y$ e ϕ mas não $\mathbf{p}_x$, por exemplo.

Para determinar a estrutura da função custo em volta do mínimo propõe-se a análise da segunda derivada da função. Assim, o valor de $\frac{\partial^2 err^2}{(\partial p_x)^2}$:

- é baixo no caso de o vale ser paralelo ao eixo Ox e totalmente plano
- é elevado se a função custo tiver um mínimo distinto em relação ao eixo dos Ox .

Da mesma forma se pode analisar $\frac{\partial^2 err^2}{(\partial p_y)^2}$ e $\frac{\partial^2 err^2}{(\partial \phi)^2}$.

Como a maioria das linhas que constituem o mapa são paralelas a um dos eixos de coordenadas, a distância, (2.2), é seccionamento linear, na maior parte do mapa, e a sua segunda derivada é zero em todas essas zonas, ou seja: $\frac{\partial^2 err}{(\partial p)^2} = 0$.

A função custo aproximada (2.5), não é totalmente definida positiva e, portanto, o critério de curvatura pode ser enganoso, isto é, a segunda derivada parcial pode ser pequena para um mínimo local. No entanto, este problema é apenas causado pelos *outliers*, uma vez que a função custo é definida positiva no intervalo $\left(-\frac{c}{\sqrt{3}}, \frac{c}{\sqrt{3}}\right)$. Para evitar este problema a função custo original, (2.5) é substituída por uma função custo quadrática, equação (2.19), e são ignorados os *outliers*, apenas no cálculo das segundas derivadas.

Desta forma a função fica definida positiva.

$$err2() = \frac{1}{2} \left(\frac{d0}{c} \right)^2 \quad (2.19)$$

As primeiras derivadas parciais são dadas por:

$$\frac{\partial E}{\partial p_x} = \sum_i^n \frac{\partial err2}{\partial d}(s_i) * \frac{\partial d}{\partial x}(s_i) \quad (2.20)$$

$$\frac{\partial E}{\partial p_y} = \sum_i^n \frac{\partial err2}{\partial d}(s_i) * \frac{\partial d}{\partial y}(s_i) \quad (2.21)$$

$$\frac{\partial E}{\partial \phi} = \sum_i^n \frac{\partial err2}{\partial d}(s_i) * \frac{\partial d}{\partial p}(s_i) * \frac{\partial p}{\partial \phi}(s_i) \quad (2.22)$$

As segundas derivadas parciais ficam:

$$\frac{\partial^2 E}{(\partial p_x)^2} = \sum_i^n \frac{\partial^2 err2}{(\partial d)^2}(s_i) * \left(\frac{\partial d}{\partial x}(s_i) \right)^2 \quad (2.23)$$

$$\frac{\partial^2 E}{(\partial p_y)^2} = \sum_i^n \frac{\partial^2 err2}{(\partial d)^2}(s_i) * \left(\frac{\partial d}{\partial y}(s_i) \right)^2 \quad (2.24)$$

$$\begin{aligned} \frac{\partial^2 E}{(\partial p_\phi)^2} = \sum_i^n & \left(\frac{\partial^2 err2}{(\partial d)^2}(s_i) \left(\frac{\partial d}{\partial x}(s_i) * (-\sin\phi \quad -\cos\phi) * s_i + \frac{\partial d}{\partial y}(s_i) * (\cos\phi \quad -\sin\phi) * \right. \right. \\ & \left. \left. s_i \right)^2 + \frac{\partial err2}{\partial d}(s_i) \left(\frac{\partial d}{\partial x}(s_i) * (-\cos\phi \quad \sin\phi) * s_i + \frac{\partial d}{\partial y}(s_i) * (\cos\phi \quad -\sin\phi) * s_i \right) \right) \end{aligned} \quad (2.25)$$

2.5 Fusão da estimativa da localização

A abordagem até agora descrita estima a posição e orientação do robot que melhor se adequa à informação extraída dos sensores. Devido às vibrações do robô e aos diversos erros introduzidos pelas medidas dos sensores, que aumentam muito com a velocidade do robô, a posição estimada é afectada por uma quantidade considerável de ruído e imprecisão.

Para reduzir o ruído propõe-se avaliar a dependência temporal das posições estimadas através da informação sensorial subsequente. Como as posições subseqüentes do robô são vizinhas e ligadas por alguma transição, dependendo da velocidade do robô, pode-se usar uma média estocástica ponderada, que é, na realidade, uma aplicação simplificada do Filtro de Kalman.

2.5.1 Variância da odometria

A variância da estimativa da posição do robô, através da odometria, pode ser modelada pelo seu grau de incerteza. Considerando $(\mathbf{r}_t, \boldsymbol{\psi}_t)$ a estimativa da posição e orientação do robô no instante de tempo t e $\sigma_{r_x,t}^2, \sigma_{r_y,t}^2, \sigma_{r_\psi,t}^2$ as respectivas variâncias. Após o robô se mover a estimativa da posição e as respectivas variâncias são actualizadas usando o modelo do robô omnidireccional com velocidade $\mathbf{v}$ e velocidade de rotação $\boldsymbol{\omega}$, [1]:

$$\hat{\boldsymbol{\psi}}_t = \boldsymbol{\psi}_{t-\tau} + \boldsymbol{\omega} \cdot \tau \quad (2.26)$$

$$\hat{\mathbf{r}}_t = \begin{cases} \mathbf{r}_{t-\tau} + \mathbf{v} \cdot \tau & \text{se } \boldsymbol{\omega} = 0 \\ \mathbf{r}_{t-\tau} + \frac{1}{\omega} R_{\boldsymbol{\psi}_{t-\tau}} \begin{pmatrix} \sin(\omega\tau) & \cos(\omega\tau) - 1 \\ 1 - \cos(\omega\tau) & \sin(\omega\tau) \end{pmatrix} R_{-\boldsymbol{\psi}_{t-\tau}} \mathbf{v} & \text{se } \boldsymbol{\omega} \neq 0 \end{cases} \quad (2.27)$$

Sendo $R_{\boldsymbol{\psi}}$ a matriz de rotação pelo ângulo $\boldsymbol{\psi}$. A velocidade linear $\mathbf{v}$ e a velocidade de rotação $\boldsymbol{\omega}$ são dadas pela odometria e a actualização das variâncias tem em conta a imprecisão do movimento:

$$\sigma_{\hat{\boldsymbol{\psi}},t}^2 = \sigma_{\boldsymbol{\psi},t-\tau}^2 + \alpha(\hat{\boldsymbol{\psi}}_t - \boldsymbol{\psi}_{t-\tau})^2 \quad (2.28)$$

$$\sigma_{\hat{r}_x,t}^2 = \sigma_{r_x,t-\tau}^2 + \alpha(\hat{r}_{x,t} - r_{x,t-\tau})^2 \quad (2.29)$$

$$\sigma_{\hat{r}_y,t}^2 = \sigma_{r_y,t-\tau}^2 + \alpha(\hat{r}_{y,t} - r_{y,t-\tau})^2 \quad (2.30)$$

Em que o parâmetro $\alpha > 0$ traduz a maior ou menor precisão do movimento. Em (2.29) e (2.30) a dependência não linear entre o movimento rotacional e o de translação é ignorada, pois, desta forma, mantém-se o modelo estatístico eficientemente tratável considerando que essa dependência requereria técnicas estatísticas que consomem muito tempo de processamento. Desde que se mantenha uma elevada frequência de amostragem o erro adicional originado mantém-se pequeno.

2.5.2 Variância da estimativa do algoritmo de localização

Após processar a informação dos sensores e calcular a estimativa respectiva a essa informação, $(\mathbf{p}, \phi)$, é possível determinar-se uma estimativa da posição mais exacta combinando $(\mathbf{p}, \phi)$ e $(\mathbf{r}, \psi)$. No entanto, é necessário calcular também as variâncias, para $(\mathbf{p}, \phi)$, que modelam a incerteza da sua estimativa, que é calculada usando a estimativa obtida pelo algoritmo de localização.

A confiança na estimativa do algoritmo de localização obtida através dos sensores é influenciada por vários aspectos, que vão desde a vibração mecânica do robô, ao erro da medida do sensor, por não estar perpendicular com o objecto que está a medir por exemplo, passando pela precisão do sistema óptico, bem como pela estrutura de pontos medidos pelos sensores.

A estimativa calculada pelo algoritmo de localização pode ser de confiança para uns parâmetros e não para outros, o que quer dizer que a suposição de incerteza é diferente para cada um dos parâmetros p_x, p_y e ϕ . Usa-se deste modo a função custo para determinar, individualmente, a variância de cada parâmetro. Assim temos que uma segunda derivada pequena equivale a uma elevada variância enquanto uma segunda derivada elevada equivale a uma pequena variância.

$$\sigma_{p_{x,t}}^2 = \frac{K_x}{\frac{\partial^2 E}{(\partial p_x)^2}} \quad (2.31)$$

$$\sigma_{p_{y,t}}^2 = \frac{K_y}{\frac{\partial^2 E}{(\partial p_y)^2}} \quad (2.32)$$

$$\sigma_{\phi,t}^2 = \frac{K_\phi}{\frac{\partial^2 E}{(\partial p_\phi)^2}} \quad (2.33)$$

2.5.3 Fusão da estimativa com a odometria

A fusão da estimativa é dada pela média pesada, supondo duas distribuições Gaussianas independentes.

$$\mathbf{r}_{x,t} = \frac{\sigma_{p_{x,t}}^2 \cdot \hat{\mathbf{r}}_t + \sigma_{\hat{\mathbf{r}}_{x,t}}^2 \cdot p_{x,t}}{\sigma_{p_{x,t}}^2 + \sigma_{\hat{\mathbf{r}}_{x,t}}^2} \quad (2.34)$$

$$\sigma_{\mathbf{r}_{x,t}}^2 = \frac{\sigma_{p_{x,t}}^2 \cdot \sigma_{\hat{\mathbf{r}}_{x,t}}^2}{\sigma_{p_{x,t}}^2 + \sigma_{\hat{\mathbf{r}}_{x,t}}^2} \quad (2.35)$$

$$\mathbf{r}_{y,t} = \frac{\sigma_{p_{y,t}}^2 \cdot \hat{\mathbf{r}}_t + \sigma_{\hat{\mathbf{r}}_{y,t}}^2 \cdot p_{y,t}}{\sigma_{p_{y,t}}^2 + \sigma_{\hat{\mathbf{r}}_{y,t}}^2} \quad (2.36)$$

$$\sigma_{\mathbf{r}_{y,t}}^2 = \frac{\sigma_{p_{y,t}}^2 \cdot \sigma_{\hat{\mathbf{r}}_{y,t}}^2}{\sigma_{p_{y,t}}^2 + \sigma_{\hat{\mathbf{r}}_{y,t}}^2} \quad (2.37)$$

$$\boldsymbol{\psi}_t = \frac{\sigma_{\phi,t}^2 \cdot \hat{\boldsymbol{\psi}}_t + \sigma_{\hat{\boldsymbol{\psi}}_t}^2 \cdot \phi_t}{\sigma_{\phi,t}^2 + \sigma_{\hat{\boldsymbol{\psi}}_t}^2} \quad (2.38)$$

$$\sigma_{\boldsymbol{\psi},t}^2 = \frac{\sigma_{\phi,t}^2 \cdot \sigma_{\hat{\boldsymbol{\psi}}_t}^2}{\sigma_{\phi,t}^2 + \sigma_{\hat{\boldsymbol{\psi}}_t}^2} \quad (2.39)$$

Sendo $\sigma_{p_{x,t}}^2, \sigma_{p_{y,t}}^2$ e $\sigma_{\phi,t}^2$, a variância de p_x, p_y e ϕ respectivamente, obtidos no algoritmo de localização; $\sigma_{\hat{\mathbf{r}}_{x,t}}^2$, $\sigma_{\hat{\mathbf{r}}_{y,t}}^2$ e $\sigma_{\hat{\boldsymbol{\psi}}_t}^2$ a variância de $\hat{\mathbf{r}}_{x,t}$, $\hat{\mathbf{r}}_{y,t}$ e $\hat{\boldsymbol{\psi}}_t$ respectivamente, obtidos através da odometria do robô.

A filtragem das estimativas, ajuda tanto no aumento da robustez, como no aumento da precisão. Deste modo medidas erradas não levam o robô a perder-se, uma vez que o filtro não permite que o robot salte para uma localização completamente diferente, o que seria possível quando se usava apenas o algoritmo de localização.

2.6 Localização global

Resolver o problema da localização global significa encontrar um mínimo global para o erro, E , a partir de (2.4), mesmo que não exista nenhuma estimativa inicial.

Propõe-se assim resolver o problema da localização global, calculando a estimativa da localização para n posições aleatórias, do robô, no mapa. Para cada uma é aplicado o algoritmo de localização usando a informação sensorial actual, sendo calculadas n estimativas da localização, alternativas à actual. Deste modo espera-se que estas estimativas convirjam, rapidamente, para um mínimo local. Efectuando a comparação do erro E de cada estimativa é possível escolher a melhor. Desta forma é verificado todo o mapa para a possível localização do robô.

Em cada iteração é calculada a estimativa actual da localização do robô a partir da posição actual do robô, ou seja, a partir da localização anterior acrescida da deslocação medida pela odometria.

Ao efectuar a localização global, para que o que o robô não salte constantemente de localização, a melhor estimativa aleatória é guardada e comparada continuamente com todas as novas estimativas aleatórias calculadas. Se essa mesma estimativa for melhor que a estimativa actual, dentro de uma dada gama (por exemplo 5% melhor), durante n iterações consecutivas, o robô salta para a nova localização estimada.

Neste capítulo foi apresentado matematicamente o algoritmo de localização, começando pela função custo, passando pela minimização desta mesma função utilizando o algoritmo RPROP, que assim permite obter a estimativa da localização do robô. Foi ainda demonstrada o cálculo da primeira e segunda derivada da função custo, derivadas estas que são utilizadas no cálculo da variância que nos permite fazer a fusão das estimativas. Concluindo-se com a localização global que através de poses aleatórias que utiliza o algoritmo de minimização e o valor da função custo para localizar o robô globalmente. No capítulo seguinte serão apresentados os sensores estudados e utilizados para testar o algoritmo implementado.

Capítulo 3

Sensores Utilizados

Na robótica móvel há necessidade de medir as características ambiente envolvente, usando, preferencialmente sensores livres de contacto com os objectos presentes. De seguida são abordadas as características dos sensores estudados e utilizados, eles são: o sensor de ultra-sons, o sensor de infra-vermelhos, o *laser range finder* e a câmara de vídeo.

3.1 Ultra-sons

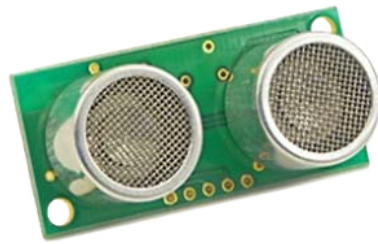


Figura 3.1: Sensor de Ultra-Sons

Os sensores de ultra-sons, também conhecidos por sonares, usam o som e o tempo de voo para medir a distância a um objecto. Estes sensores podem ser usados em qualquer tipo de ambiente e permitem efectuar medidas da distância com alguma precisão embora a sua medida possa ser ruidosa devido a obstáculos capazes de absorverem o som, como, por exemplo, materiais esponjosos.

O funcionamento do sonar usa um microcontrolador, que efectua o comando e o cálculo da medida da distância. Assim, o microcontrolador gera uma onda de 40KHz, para o caso do sonar utilizado, que é transmitida por um emissor de ultra-sons. Quando o sensor receptor de ultra-sons detecta o eco do sinal enviado, o microcontrolador, que mantém o registo do tempo decorrido, desde que o sinal foi enviado até que o eco foi detectado, sabendo a velocidade do som, calcula a distância a que o objecto se encontra, ou seja [21]:

$$dist = c * \Delta t / 2 \quad (3.1)$$

Onde c é a velocidade do som e Δt o tempo decorrido entre o envio e a detecção do ultra-som.

Porque o sinal sonoro é espalhado radialmente, a amplitude do sinal diminui em ordem ao inverso do quadrado da distância. Isto significa que a distância máxima mensurável diminui rapidamente. Normalmente, o sinal é amplificado electronicamente, para contrariar este efeito, mas o máximo alcance típico, num sonar de custo médio, encontra-se entre os 2 a 7 metros. Existe ainda uma distância mínima, não detectável pelo sonar, que, neste caso, é de cerca de 10 cm. Tal deve-se á velocidade de processamento do microcontrolador e da electrónica envolvida.

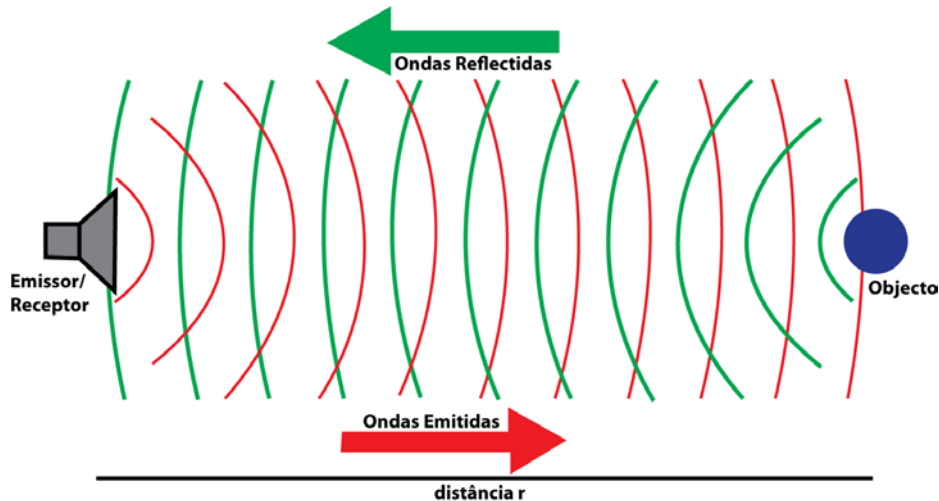


Figura 3.2: esquema simplificado do funcionamento do Sonar

Existem vários factores que alteram a correcta medida do sonar. As propriedades da superfície do objecto fazem a diferença, uma carpete e um espelho originam diferentes medidas. O som reflectido por uma almofada e por uma parede não é o mesmo. Se o objecto não estiver perpendicular ao sonar, o mais provável é o som não ser convenientemente reflectido. O som pode também ser reflectido de parede em parede até voltar a sonar, aumentando desta forma o tempo de medida e originando falsas medidas.

Outro problema é o eco fantasma. Na Figura 3.3 pode-se observar como o som é reflectido nas diferentes paredes causando o efeito do eco fantasma.

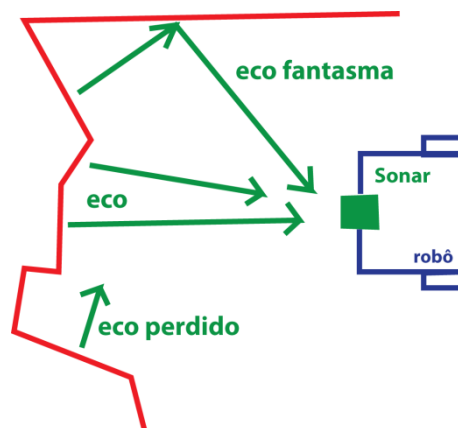


Figura 3.3: Reflexão dos ultra-sons na parede

O uso de múltiplos sensores também tem os seus problemas, pois existe sempre a possibilidade de um sonar detectar a onda emitida por outro sonar, e, para estes casos, a

abordagem a seguir é colocar os sonares apontados em diferentes ângulos. Assim, de modo a aumentar a probabilidade de a onda emitida não ser detectada pelo sonar vizinho, devem ser espaçados no mínimo por 20°. Para aumentar a referida probabilidade os sonares são usados alternadamente.

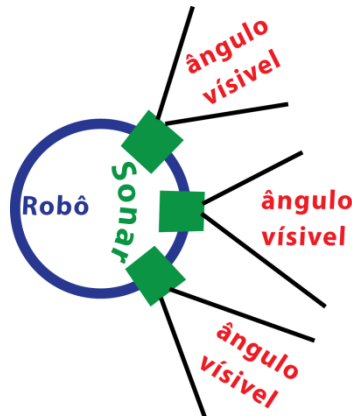


Figura 3.4: Disposição dos sonares no robô

3.2 Sensores de Infra-vermelhos



Figura 3.5: Sharp's comerciais

Existem três tipos de sensores de infravermelhos (IR). Os que são integrados num circuito e fornecem uma saída binária, os que fornecem uma saída analógica e os que fornecem uma saída digital, [22]. Os que têm saída binária são bons para detectar a proximidade de um obstáculo, e não a distância a este, ou seja, este tipo de sensor IR apenas consegue determinar se existe ou não um obstáculo no seu alcance de medida. Os sensores IR que têm saída analógica ou digital, em tensão, são sensores de medida em que a saída é função da distância ao objecto medido.

Os sensores IR têm algumas limitações. Uma vez que emitem luz infravermelha, este tipo de sensores não funcionam correctamente em locais abertos, devido à influência da luz solar. Para os sensores IR Sharp utilizados este problema não foi detectado. Para além disso, estes sensores IR são exactos, fiáveis e fornecem uma saída analógica de fácil tratamento.

Existem vários modelos, em que a principal diferença é o alcance mínimo e máximo que são capazes de medir, pelo que a escolha do modelo apropriado deve ser feita de acordo com os objectivos pretendidos para o seu uso.

O modelo utilizado funciona usando a técnica de triangulação tendo um cone de abertura relativamente baixo. É emitido um pulso de luz infra-vermelha, com comprimento de onda de $850 \pm 70\text{nm}$, e o mesmo é reflectido num obstáculo (ou simplesmente, não é reflectido). Quando a luz retorna é medido o ângulo, entre a luz emitida e reflectida, que depende da distância do obstáculo que a reflectiu. Conhecido este ângulo é possível determinar a distância.

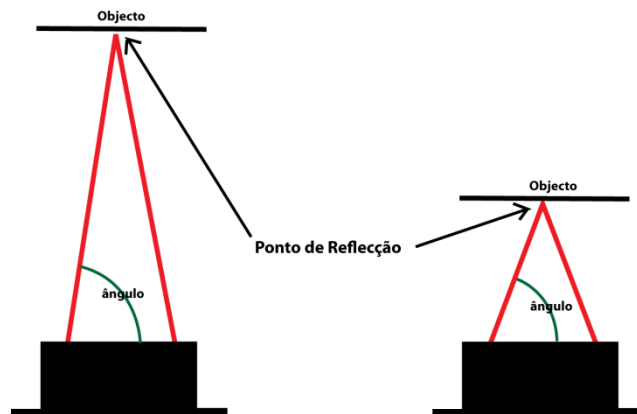


Figura 3.6: Diferentes ângulos com diferentes distâncias medidos por um IR Sharp

Estes modelos têm uma lente especial, de precisão, que transmite a luz reflectida para um CCD linear. O CCD determina o ângulo sendo calculado o correspondente valor analógico da distância medida. Além disso o circuito integrado, destes IR, aplica uma frequência modulada ao raio infra-vermelho emitido. Estas características são as que tornam este sensor quase imune à luz ambiente, e indiferente à cor do obstáculo.

Como a luz não é reflectida igualmente em todas as superfícies, a medida do IR será diferente para diferentes tipos de superfícies e para uma mesma distância. Na maioria das vezes a medida não se afasta muito da realidade podendo ser considerada válida, pelo que tendo conhecimento, à partida, deste facto é possível precaver a existência de erros na medida.

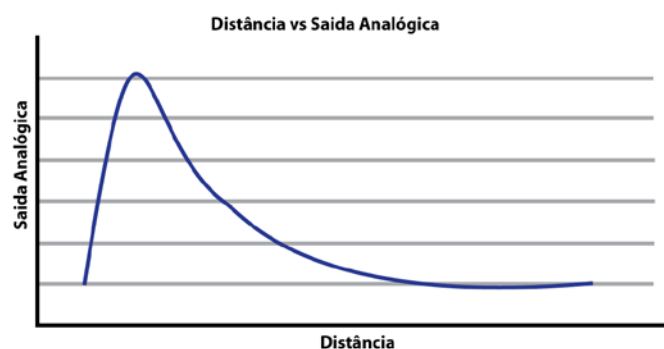


Figura 3.7: Curva típica da saída analógica de um Sharp, em função da distância ao objecto

Este modelo de IR têm uma saída não linear, o que significa que à medida que a distância aumenta linearmente, a saída analógica aumenta ou diminui não linearmente. Na Figura 3.7 é possível observar a curva típica da saída analógica de um IR Sharp. A subida rápida no início do gráfico deve-se à incapacidade deste modelo efectuar

medidas a curtas distâncias, pelo que cada tipo de modelo tem uma medida predefinida da distância que este é capaz de medir, ou seja a zona que é possível ser linearizada, zona esta que é possível aproximar por uma função logarítmica. Para evitar a zona de medida errada deve-se colocar, fisicamente, os sensores de forma que a zona de medida errada se encontre dentro do limite do robô, ver Figura 3.8

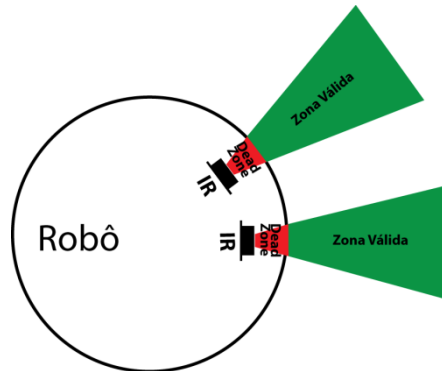


Figura 3.8: Exemplo de como fixar um IR no robô, para que a medida não tenha influência da “dead zone”

3.3 Laser Range Finder



Figura 3.9: Hokuyo URG-04LX laser range finder.

Um laser de varrimento ou *laser range finder* é um dispositivo que usa um raio laser para determinar a distância a um objecto fornecendo uma “imagem” do que o rodeia com uma resolução de milímetros.

Este tipo de sensor gera uma grande quantidade de dados. Assim, é necessária capacidade de processamento, para o tratamento dos mesmos em tempo real, pelo que um microcontrolador não é suficiente, sendo necessário, no mínimo, um computador. O processamento dos dados do *laser range finder* é mais simples que um sistema de visão artificial, mas mais complexo que o processamento dos dados de um sensor de ultras-sons ou um IR.

3.3.1 Teoria de operação

A maioria dos *laser range finder* funciona de acordo com o princípio de tempo de voo, ou seja, envia um pulso de raio laser, durante um curto espaço de tempo, em

direcção ao objecto e mede o tempo que o pulso leva a ser reflectido pelo objecto. Algumas técnicas usam a triangulação, outras há, que medem o desfasamento entre pulsos dos raios laser emitidos reduzindo, significativamente, as necessidades temporais requeridas pelo circuito de controlo do sensor laser.

A forma como o *laser range finder* faz o varrimento pode ser comparada a um sonar usando luz em vez de ultra-sons para criar uma mapa 2D dos objectos na sua proximidade.

Com os sensores laser é possível efectuar medidas com rapidez e com um campo de visão extremamente reduzido. O termo varrimento significa que o *laser range finder* após fazer uma leitura, move o sensor uma fracção da sua resolução angular e faz uma nova leitura, repetindo, continuamente, este processo até ter percorrido toda a sua gama de detecção. Devido às limitações físicas e mecânicas existe uma zona que o sensor não é capaz de medir. Os dados obtidos do *laser range finder* mapeiam directamente um gráfico polar a cada uma das medidas, ao longo de uma circunferência de 360°, em relação ao scanner que se encontra na origem, com uma zona sem medidas, chamada *dead zone*, Figura 3.10.

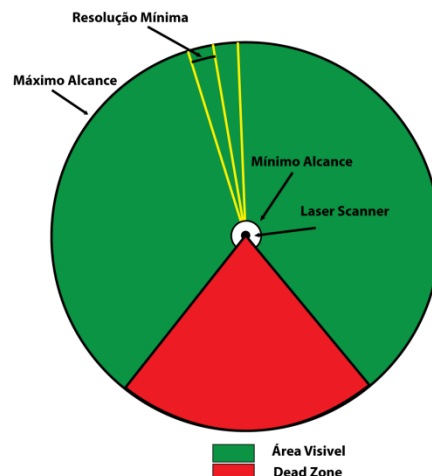


Figura 3.10: Características das medidas do Hokuyo URG-04LX.

Os lasers são normalmente caracterizados pela sua resolução, frequência e resolução angular. A resolução refere-se à exactidão da distância, medida numa determinada direcção, e situa-se, normalmente entre 1 e 3mm. A frequência refere-se a quantas vezes é percorrido o arco de medida, de 270 graus, pelo sensor. Esta é, tipicamente, 10Hz ou mais. A resolução angular refere-se ao numero de amostras possível de obter nos 360 graus de varrimento do sensor, este valor é geralmente maior que 500 passos, o que significa uma medida abaixo do grau, entre passos. O sensor tem também um alcance mínimo e máximo, sendo o mínimo de alguns milímetros e o máximo na ordem dos metros ou dezenas de metros [23].

3.3.2 Hardware

Dentro do compartimento que aloja o laser, existe um disco giratório que altera a direcção na qual o laser está a efectuar a medida, permitindo desta forma que seja medido todo o espaço angular. Os dados das medidas são processados no circuito do

laser, colocados em pacotes e enviados pela porta de comunicação, na qual se encontra ligado ao computador.

Existem vários fabricantes de lasers sendo os mais comuns na robótica móvel criados pela “SICK” e pela “Hokuyo”. Estes variam no alcance, na performance, no consumo energético e no preço. Neste trabalho temos disponível um Hokuyo URG-04LX, Figura 3.9, [24].

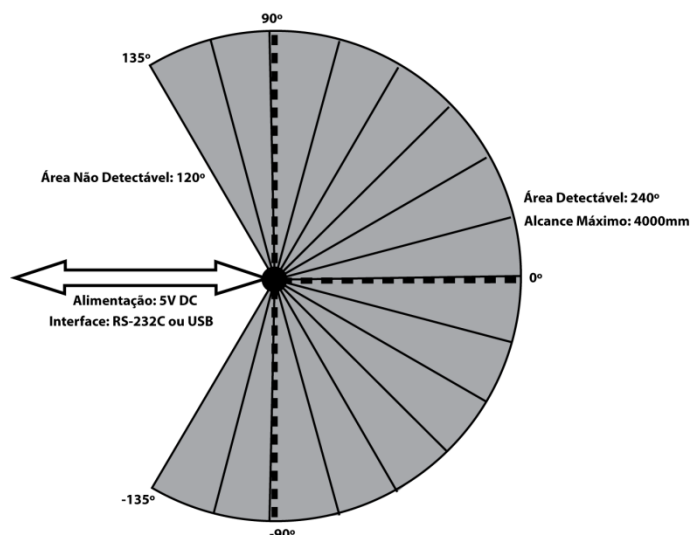


Figura 3.11: Características físicas e de conexão do Hokuyo URG-04LX.

O Hokuyo URG-04LX usa um laser de luz infra-vermelha com um comprimento de onda de 785nm e é um laser de classe 1 de segurança. A área de varrimento é um semicírculo de 240° e tem um alcance máximo de 4000mm. A resolução angular é de cerca de 0.3515625°, ou seja, na gama de detecção do sensor, 240°, é possível medir 681 distâncias diferentes. O diâmetro do raio laser emitido é menor que 20mm até a uma distância de 2000m e tem uma divergência máxima de 40mm a uma distância de 4000mm. O princípio de medida da distância é baseado no cálculo do desfasamento entre pulsos, pelo que é possível obter medidas estáveis com influência mínima da cor e da opacidade do objecto, [24].

3.3.3 Software

Para poder utilizar o *laser range finder* disponível foi necessário implementar um software capaz de comunicar com este e processar os dados provenientes do mesmo, e ainda com a capacidade de interagir com o restante software de localização no que toca à troca de dados.

De seguida apresenta-se o protocolo de comunicação, protocolo este que utiliza caracteres ASCII para a comunicação.

Host->sensor:	Comando	Parâmetros	LF ou CR					
Sensor->Host:	Comando	Parâmetros	LF	Estado	LF	Dados	LF	LF

A comunicação é sempre iniciada pelo lado do Host, computador. Esta inicia com um **Comando** identificado por um dos seguintes caracteres:

- “V”: Informação da versão;
- “L”: Controlo de iluminação do laser;
- “S”: Alteração dos parâmetros de comunicação;
- “G”: Requisição de medidas da distância

Os **Parâmetros** dependem do comando que está a ser usado e são explicados nos pontos seguintes.

Por fim é enviado um código de terminação que pode ser um “Carriage Return”, **CR** (caracter 13 em ASCII), ou “Line Feed”, **LF** (caracter 10 em ASCII)

Os comandos e os parâmetros são sempre retornados pelo sensor. É ainda retornado o **Estado** que se for “0” indica que o processamento foi concluído com êxito, caso contrário o valor do **Estado** corresponde a um código de erro. Os **Dados** são recebidos em tramas de 64bytes, pelo que se o tamanho destes for superior a 64bytes estes são recebidos em tramas de 64bytes separados por um caracter LF, no fim da trama completa de dados são recebidos dois caracteres LF.

Informação da versão, “V”:

Host->sensor:	V	LF ou CR							
Sensor->Host:	V	LF	Estado	LF	I.V.	I.P.	V.F.	V.P.	

Onde I.V. é a Informação do vendedor, I.P. é a informação do produto, V.F. é a versão do *firmware*, V.P. é a versão do protocolo e N.S. é o número de série do sensor.

Controlo de iluminação do laser, “L”:

Host->sensor:	L	Byte de Controlo	LF ou CR				
Sensor->Host:	L	Byte de Controlo	LF	Estado	LF	LF	

O sensor laser é iluminado quando o byte de controlo é “0” e é apagado quando o byte de controlo é “1”, outro código de controlo retorna erro. Por definição o laser é automaticamente iluminado quando o sensor é ligado, mesmo que o laser tenha sido apagado antes.

Alteração dos parâmetros de comunicação, “S”:

Host->sensor:	S	Baud Rate (6 dígitos)	Área Reservada (7 dígitos)	LF ou CR				
Sensor->Host:	S	Baud Rate (6 dígitos)	Área Reservada (7 dígitos)	LF	Estado	LF	LF	

Por defeito a velocidade da comunicação é 19200bps e pode ser alterada para 57600bps e 115200bps, inserindo um destes valores no campo **Baud Rate**, apenas para a porta série pois no caso de se estar a usar USB este comando não influencia o Baud Rate. A alteração apenas é efectuada se no campo **Estado** for retornado o valor “0”. Os dados inseridos no campo **Área Reservada** não são usados, apenas são reenviados de volta.

Pedido de requisição de medidas da distância, “G”:

Host->sensor:	G	P.P.	P.C.	T.C.	LF ou CR				
Sensor->Host:	G	P.P.	P.C.	T.C.	LF	Estado	LF	Bloco de Dados 1	LF
								...	LF
								Bloco de Dados N	LF
									LF

Onde P.P. é o passo de partida, ou seja o passo da área onde começa a leitura do sensor (valor entre 0 e 768), P.C. é o passo de chegada, ou seja, o passo onde acaba a leitura do sensor (valor entre 0 e 768). Estes dois campos devem ter 3 dígitos. As leituras do sensor são agrupadas em clusters, ou seja, múltiplos pontos vizinhos são agrupados de acordo com o tamanho cluster, T.C., em que o mínimo valor desse grupo é o enviado para o Host. Este campo T.C. deve ter 2 dígitos.

As distâncias medidas são expressas em milímetros, sendo o valor máximo mensurável pelo sensor 4095mm com uma resolução de 1mm. Os dados são expressos em 12 bits (entre 0 e 4095), mas para ser transmitido o código de 12 bits é separado em dois códigos de 6 bits que são depois convertidos para código ASCII com 1byte. O processo de codificação é simples, os códigos de 12 bits são separados em dois de 6 bit, cada, convertidos para hexadecimal e é-lhes adicionado 30H, sendo então convertidos para o respectivo código ASCII e enviados via porta de comunicação. O processo de decodificação é simplesmente o inverso do processo de codificação.

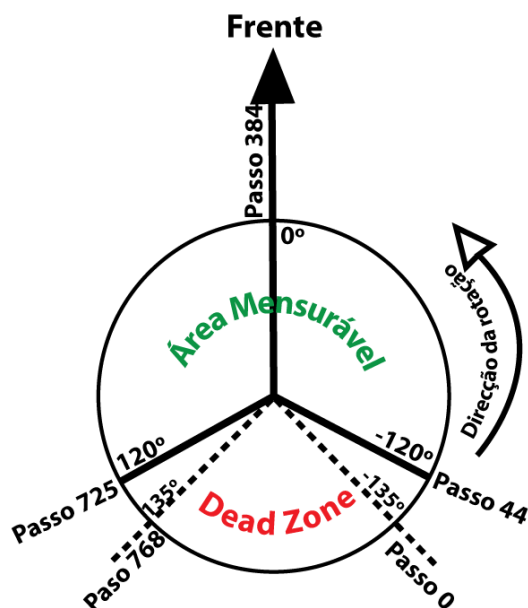


Figura 3.12: Configuração dos passos do laser Hokuyo URG-04LX

Tendo conhecimento do protocolo foi desenvolvida uma aplicação capaz de ler os dados do laser via porta série. Esta envia a trama de dados necessária à leitura do laser e processa os dados extraindo toda a informação útil.

Na Figura 3.13 é possível observar a interface criada. Esta após receber uma trama completa de dados processa-os. Sabendo a distância e o ângulo a que essa distância foi medida, através do passo em que foi medida, é possível calcular o respectivo ponto de

medida no plano 2D e, desta forma, desenhar uma imagem do que é observado pelo sensor. Note-se que toda esta informação é referenciada ao sensor.

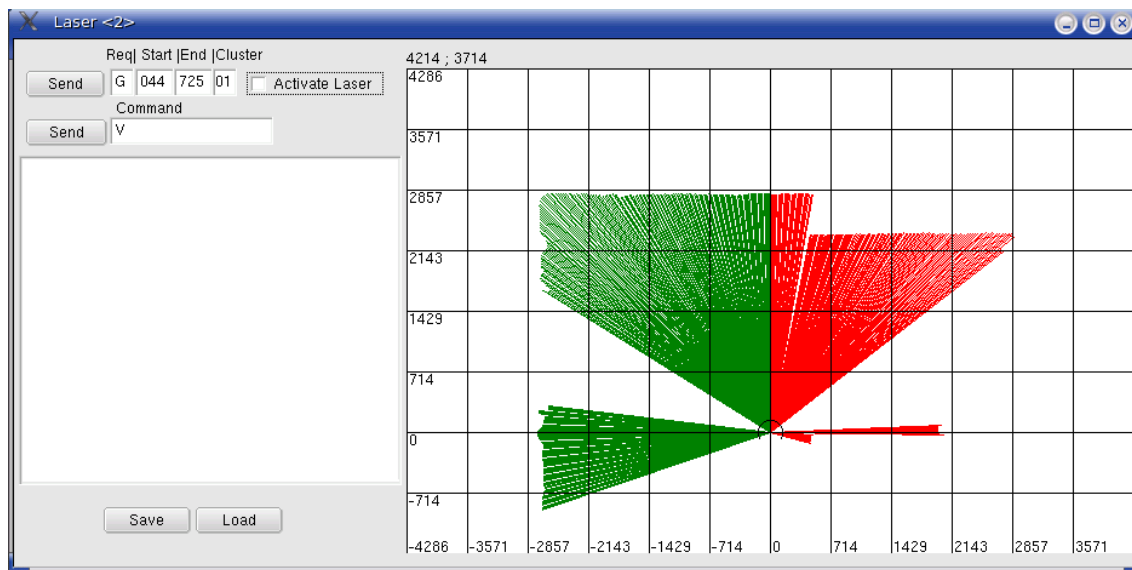


Figura 3.13: Interface de Leitura do “laser range finder” Hokuyo URG-04LX

3.4 Visão artificial



Figura 3.14: Câmara de vídeo firewire

As câmaras de vídeo são já um sensor muito usado na robótica, muito devido à grande quantidade de informação que é possível extrair da cada imagem (frame) que esta adquire. Para isso é necessário efectuar processamento da imagem obtida focado na informação que se pretende extrair dessa mesma imagem. Aqui será abordada apenas a forma como é extraída a informação relevante para este projecto, em específico para o futebol robótico, ou seja, na detecção das linhas de um campo de futebol. A informação pretendida resume-se a uma lista de pontos, que definem as linhas do campo que estão presentes na imagem processada, [19].

No futebol robótico a abordagem da montagem da câmara no robô é também importante. A combinação da câmara de vídeo com um espelho concavo omnidireccional permite à câmara obter uma imagem omnidireccional de tudo o que rodeia o robô, que, dependendo do formato do espelho permite medir correctamente objectos até cerca de 6 metros de distância do robô, [19].

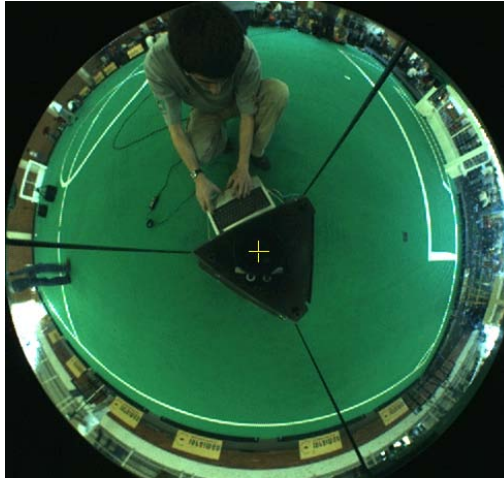


Figura 3.15: Imagem do campo de futebol obtida através da câmara de vídeo

É efectuado o seguinte processamento à imagem para se obter uma lista de pontos das linhas brancas que o robô tem no seu campo de visão. São lançadas várias linhas radialmente colocadas em torno do centro da imagem omnidireccional, ver Figura 3.16. É determinada a intersecção de cada uma dessas linhas com as linhas brancas do campo, essa intersecção é obtida pela variação brusca das cores, ou seja, pela transição de cor verde para branco e/ou de branco para verde. Visto que é conhecida a relação entre os pixéis da imagem e a distância correspondente em metros, após ter sido efectuada a respectiva calibração da imagem, para cada transição, ponto, detectada é possível saber a distância e o ângulo em relação ao robô.

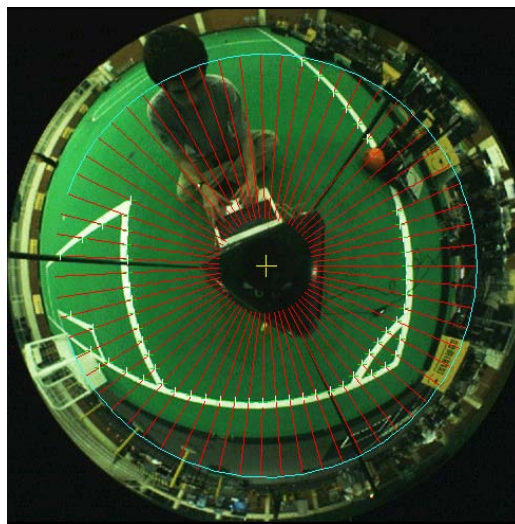


Figura 3.16: Detecção das linhas por parte do algoritmo a partir de uma imagem fornecida pela câmara de vídeo

Desta forma obtém-se uma lista de pontos/distâncias às linhas que se encontram na vizinhança do robô. Pode-se comparar esta lista, no futebol robótico, a uma lista de pontos/distâncias num outro tipo de ambiente (corredor, por exemplo) em que os sensores usados são sonares, IR, e/ou laser, a diferença é que é apenas usado um sensor para obter todas as distâncias mensuráveis. Notar que não seria viável usar sonares e/ou IR num jogo de futebol robótico para detectar as linhas do campo, bem como não seria viável usar uma câmara de vídeo num corredor para determinar a distância a paredes,

portas ou outros objectos, devido a terem cores muito idênticas, e as condições de iluminação não serem geralmente as mais favoráveis.

Neste capítulo foi feita uma apresentação de quatro sensores utilizados, em termos das suas características principais, mas essencialmente da qualidade da medida que estes permitem efectuar. No capítulo seguinte será abordado o cálculo da matriz de distâncias bem como da matriz de gradiente desta.

Capítulo 4

Mapeamento

Um passo fulcral para o bom funcionamento de todo o algoritmo é a criação do mapa, não do ponto de vista da exactidão com que é criado, mas sim da forma como é criado, isto porque, deste modo, permite que o mapa possa ser criado de forma independente do algoritmo.

Para o algoritmo utilizado é necessário um mapa de distâncias a objectos, paredes, linhas, obstáculos, dependendo do objectivo e do tipo de ambiente em que o robô vai actuar.

O mapa é algo parecido a uma imagem do mundo que o robô pode medir com os seus sensores, ou seja, um mapa formado por distâncias. O mapa criado é um mapa 2D formado por células com um determinado tamanho, tamanho esse que define a resolução da estimativa da posição do robô. Assim, sabendo a dimensão, em metros, do mapa, largura e comprimento, e sabendo a resolução pretendida, tamanho de cada célula, é construído um mapa em que cada célula contém a distância à parede/linha mais próxima, que se assemelha a uma imagem em que cada pixel contém a informação da cor. Por exemplo, um mapa de 20m de comprimento por 10m de largura, com uma resolução de 4cm equivale a um mapa de células com 500 células de comprimento e 250 células de largura, ou seja, no mapa são definidas 125000 posições possíveis onde o robô se pode localizar, com um erro de 4cm.

De facto este mapa pode, e deve, ser criado antes de o robô começar a funcionar. Assim, durante o seu funcionamento, o robô não tem de recriar o mapa mas apenas ler as posições (x,y) deste, o que, em termos computacionais, é um passo extremamente importante.

Deste modo este capítulo apresenta os cálculos necessários para se criar uma matriz de distâncias do mapa que está a ser utilizado. Além da matriz de distâncias é também demonstrado como é calculada a matriz de gradiente, também essencial ao algoritmo.

4.1 Criação do mapa de distâncias

Para criar o mapa é necessário ter conhecimento prévio do local onde o robot vai actuar, sendo necessário saber as medidas, com a maior exactidão possível, de todo o tipo de objectos e/ou obstáculos ou linhas (no caso do futebol robótico). Assim, e visto que qualquer objecto pode ser aproximado por linhas ou arcos de circunferência, são exactamente estes dois tipos de geometria que são usados para criar o mapa, ou seja, imagine-se uma parede que se encontra a 3 metros da origem do mapa, no eixo dos Ox e tem 5m de comprimento na direcção Oy. A linha que une o ponto (3,0) ao ponto (3,5) é suficiente para definir esta parede em 2D. Como o que se pretende não é a linha mas sim o que ela define, ou seja, um objecto ao qual são medidas distâncias de todas as células do mapa, às células onde esta linha está a ser definida, isto quer dizer que, para determinar as distâncias mínimas, a todos os objectos presentes no ambiente, onde o robô se insere, é necessário calcular a distância de todas as células do mapa às linhas e arcos que definem esses objectos. Os pontos seguintes ilustram o método utilizado para estes cálculos.

4.1.1 Linhas

Para as linhas o cálculo das distâncias resume-se ao cálculo geométrico da distância de um ponto a uma linha. Neste caso, não a qualquer ponto da linha, mas ao ponto da linha que está à menor distância do ponto a ser estudado. Defina-se o ponto de início ($p1$) e o ponto de fim ($p2$) da linha, o ponto de estudo (p) e o ponto (pp) onde uma recta perpendicular à linha, definida por $p1$ e $p2$, que passa por p , a intersecta.

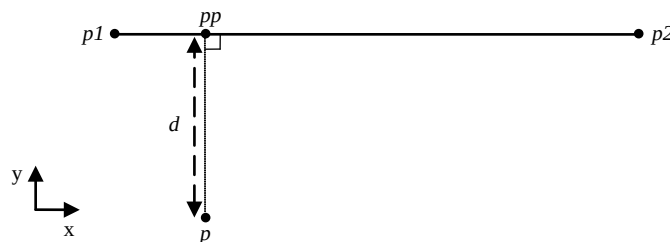


Figura 4.1: Distância de um ponto em relação a uma linha, horizontal

Para simplificar os cálculos, o eixo de coordenadas é rodado de acordo com o ângulo que a linha, definida por $p1$ e $p2$, faz com o eixo de coordenadas global, para que a linha fique horizontal, tal como na Figura 4.1. Assim, o cálculo da distância do ponto, p , à linha $p1 p2$, resume-se à distância segundo o eixo y :

$$d = |p.y - p1.y| \quad (4.1)$$

desde que o ponto pp se encontre dentro da zona definida pelos dois pontos da linha, ver Figura 4.2. Quando tal não acontece é necessário calcular a distância em relação ao

ponto $p1$ ou ao ponto $p2$, dependendo a qual dos dois a distância é menor, Figura 4.3. Nesse caso a distância resume-se a um cálculo de distâncias entre dois pontos:

$$d = \sqrt{((p1.x - p.x)^2 + (p1.y - p.y)^2)} \quad (4.2)$$

Tal como anteriormente, como o eixo se encontra rodado, para identificar se o ponto pp se encontra fora da linha definida por $p1$ e $p2$, verifica-se a orientação da linha, comparando $p1.x$ com $p2.x$, a direcção será do que tiver menor valor para o que tiver maior valor, e verifica-se assim se o ponto se encontra entre esses dois valores. Se sim aplica-se o cálculo usando a equação (4.1). Se não calcula-se a distância a cada um dos extremos da linha usando a equação (4.2) e verifica-se qual destes é menor.

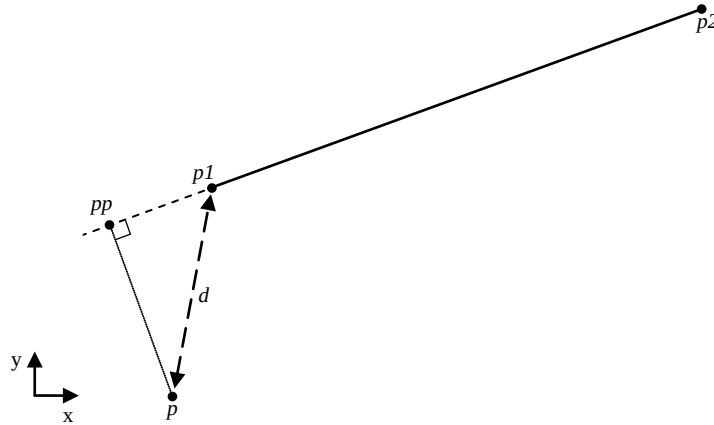


Figura 4.2: Distância de um ponto a uma linha diagonal



Figura 4.3: Distância de um ponto a uma linha rodada

4.1.2 Arcos

Analogamente às linhas também os arcos têm uma zona definida à qual é possível calcular a distância ao arco, mas, por outro lado, existem zonas em que a menor distância ao arco é dada pela distância a uma das extremidades do arco.

Considere-se um arco de circunferência centrado em C , de raio R , iniciando-se o arco em TS e terminando em TE . Considere-se o ponto p , ao qual se pretende medir a distância ao arco de circunferência definido, Figura 4.4.

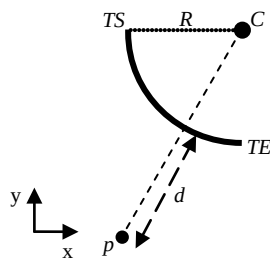


Figura 4.4: Distância de um ponto a um arco de circunferência

A distância de um ponto a um arco é dada pela distância do ponto ao centro do arco subtraída do raio do arco. Visto que o ponto pode estar no interior do arco aplica-se o módulo:

$$d = \left| \sqrt{((p.x - C.x)^2 + (p.y - C.y)^2)} - R \right| \quad (4.3)$$

Isto apenas é aceitável quando o ponto se encontra localizado na zona da amplitude definida pelo arco TS-TE, Figura 4.5. Para determinar se está dentro ou fora dessa área é calculado o ângulo do ponto ao centro do arco, se esse ângulo se encontrar entre TS e TE então é calculado a distância ao arco normalmente.

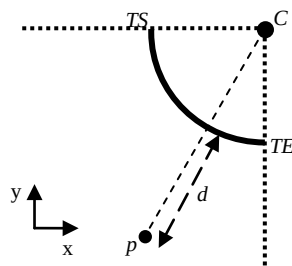


Figura 4.5: Limite do ângulo do arco, para o qual se pode calcular normalmente a distância do ponto ao arco de circunferência

Para os outros casos é necessário calcular a distância a um dos extremos do arco, ou seja, aquele que menor dista do ponto em estudo. Assim a distância a cada um dos extremos do arco fica:

$$d1 = \sqrt{(p.x - C.x - R * \cos(TS))^2 + (p.y - C.y - R * \sin(TS))^2} \quad (4.4)$$

$$d2 = \sqrt{(p.x - C.x - R * \cos(TE))^2 + (p.y - C.y - R * \sin(TE))^2} \quad (4.5)$$

Deste modo a distância mínima ao arco é a menor distância da comparação entre d_1 e d_2 .

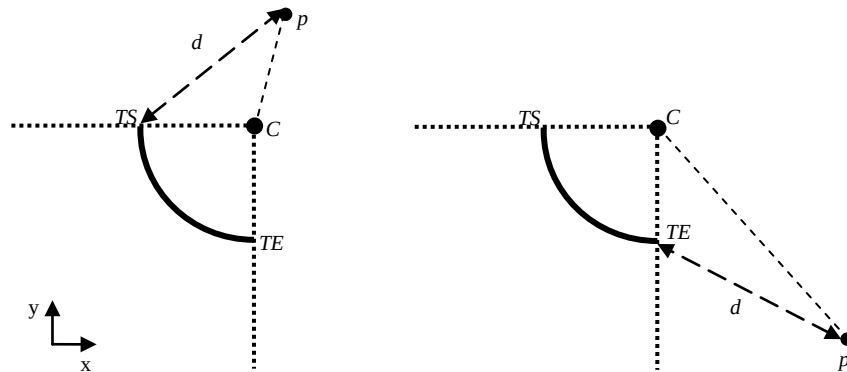


Figura 4.6: Aproximação da distância de um ponto a um arco de circunferência, caso o ponto se encontre fora do raio do arco

Assim, no mapa final, tem-se distâncias, junto às linhas, paredes, objectos, de valor baixo, e distâncias, em zonas onde não existam objectos, de valor elevado. Por exemplo, num corredor, junto às paredes o valor das distâncias será baixo. Por sua vez no centro do corredor os valores das distâncias serão os mais elevados. Note-se que o valor máximo possível da distância depende das dimensões reais do mapa.

4.2 Representação

Para tornar acessível a criação do mapa foi desenvolvida uma interface, que permite adicionar linhas e/ou arcos ao mapa, manualmente. Permite ainda a representação em forma de imagem do mapa que está a ser criado, ou seja, cada vez que se adiciona uma linha ou arco ao mapa, a matriz que contém os valores das distâncias é representada graficamente, de acordo com as actualizações efectuadas sendo possível observar se se está a proceder correctamente à criação do mapa. Uma vez que o mapa é uma matriz de distâncias, nas suas células contêm distâncias, a representação destas em forma de imagem é efectuada em gradientes de cinzento, ou seja, sabendo o valor da máxima distância possível de encontrar no mapa, faz-se uma relação entre as distâncias e o correspondente byte de um pixel, de valor entre 0 e 255, que é escrito na estrutura da imagem e desenhado. A imagem tem tamanho fixo pelo que dependendo do tamanho do mapa que se está a criar, a sua representação é adaptada dinamicamente à imagem, ou seja, por exemplo, considerando a relação directa célula do mapa/pixel da imagem, é digitalmente impossível desenhar por completo um mapa de 1280 células por 1280 células, numa imagem com 800 pixels por 600 pixels, considerando uma relação de 1:1. Por esse motivo, existe um factor de escala que permite desenhar por completo o mapa na imagem. Assim, caso o mapa seja maior que a imagem, teremos um pixel a representar várias células adjacentes. Por outro lado, se o mapa for menor que a imagem terá vários pixels adjacentes a representar a mesma célula. Logo as zonas escuras, na imagem, representam as menores distâncias e, conseqüentemente, a zona de cor preta representa as paredes, objectos ou linhas; as zonas mais claras representam as maiores distâncias. Para facilitar a visualização do local onde as paredes, objectos ou linhas são definidas, estas zonas são “pintadas” a vermelho.

Note-se que a imagem serve para representação do mapa de distâncias, pois só este último é necessário para o algoritmo. A Figura 4.7 ilustra o exposto.

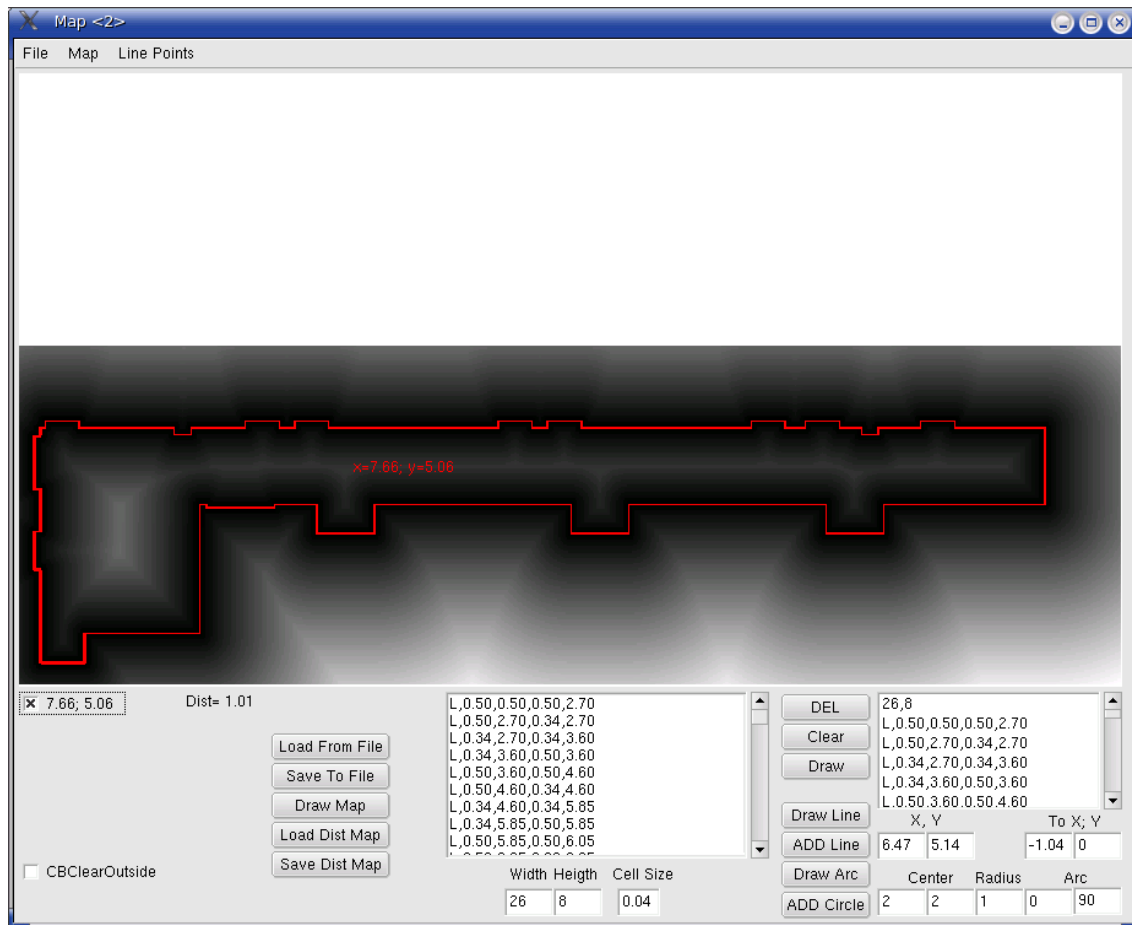


Figura 4.7: Interface de Criação de Mapas

4.3 Gradiente

Um outro tipo de mapa, matriz, necessário ao algoritmo é a derivada da distância em ordem à posição (x e y): $\frac{\partial d}{\partial x}$ e $\frac{\partial d}{\partial y}$ que se guarda numa matriz para posterior processamento.

Uma vez calculada a matriz de distâncias que define o mapa, é possível usar este para calcular a matriz de derivadas parciais, para isso aplica-se o Filtro de Sobel [25].

O Filtro Sobel é aplicado sobretudo em algoritmos de detecção de contornos. Em termos técnicos, consiste num operador que calcula diferenças finitas, dando uma aproximação do gradiente da intensidade dos pixels da imagem. Em cada ponto da imagem, o resultado da aplicação do filtro Sobel devolve o gradiente desta matriz [25].

O Filtro Sobel calcula o gradiente da função distância em cada ponto, dando a direcção da maior variação de distâncias altas para as baixas e a quantidade de variação nessa direcção. Assim, obtém-se uma noção de como varia a distância em cada ponto,

se de forma mais suave ou abrupta. Com isto consegue-se estimar a presença de uma transição brusca de distâncias e qual a orientação desta.

Este operador utiliza duas matrizes 3×3 que são convoluídas com a matriz original para calcular aproximações das derivadas: uma para as variações horizontais e outra para as verticais. A derivada segundo x e a derivada segundo y serão duas matrizes que, em cada ponto, contêm uma aproximação às derivadas horizontal e vertical, respectivamente, do mapa de distâncias.

O gradiente de x é igual à média pesada de acordo com a seguinte matriz [25]

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

O gradiente de y é igual à média pesada de acordo com a seguinte matriz [25]

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Assim, para cada ponto p , considerando a seguinte matriz:

$$\begin{bmatrix} lo & mo & ro \\ lm & p & rm \\ lu & mu & ru \end{bmatrix}$$

temos oito células adjacentes que serão usadas para calcular os gradientes, de x e de y, do ponto em questão, de acordo com a média pesada representada pelas matrizes acima:

$$gradient.x = \frac{-lo+ro-2lm+2rm-lu+ru}{8} \quad (4.6)$$

$$gradient.y = \frac{-lo+lu-2mo+2mu-ro+ru}{8} \quad (4.7)$$

Note-se que como nas extremidades, cantos, junto ao limite vertical e horizontal do mapa, as células não têm 8 pontos adjacentes, ou seja, não existe uma matriz 3x3 definida em torno do ponto não é possível aplicar directamente o filtro de Sobel a todo o mapa. Assim, para esses casos, o valor do gradiente é extrapolado, isto é, para as linhas verticais e horizontais, copia-se o valor adjacente já calculado, para os cantos faz-se uma média pesada dos dois valores adjacentes ao ponto.

As figuras 4.8 e 4.9 ilustram graficamente, os dois gradientes para o mapa do corredor DEEC;

A Figura 4.10 e a Figura 4.11 apresentam o resultado da aplicação do método ao campo de futebol robótico.

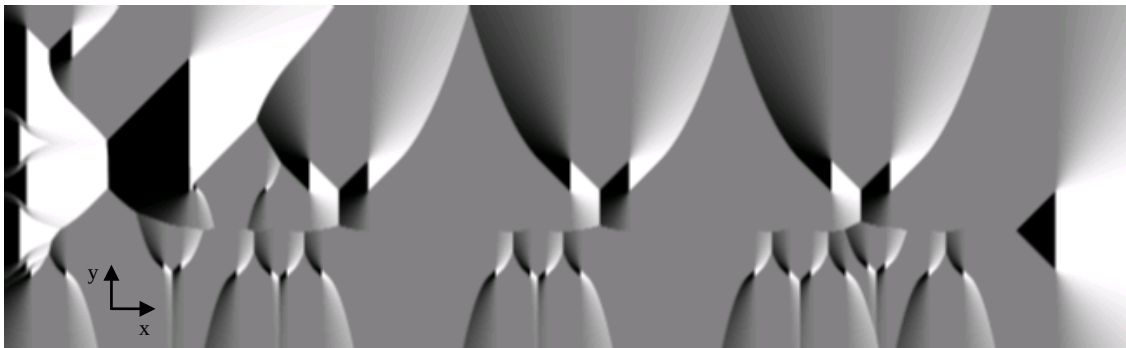


Figura 4.8: Representação do gráfico de gradiente segundo o eixo x do mapa do corredor DEEC



Figura 4.9: Representação gráfica do gradiente segundo o eixo y do mapa do corredor DEEC

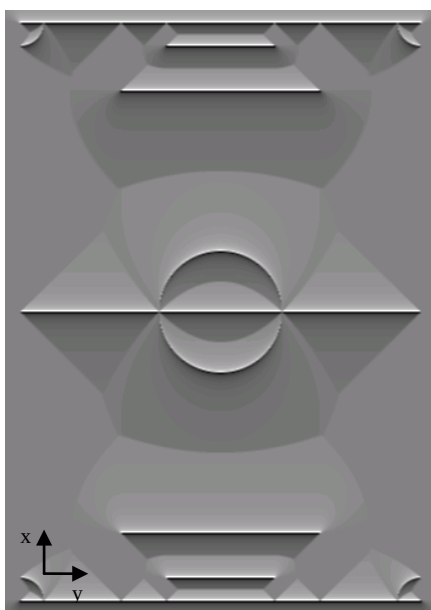


Figura 4.10: Representação gráfica do gradiente do campo de futebol segundo x



Figura 4.11: Representação gráfica do gradiente do campo de futebol segundo y

Neste capítulo foi exposto como se efectua o cálculo da matriz de distâncias e da matriz de gradiente. Na primeira aplicando cálculos geométricos aplicados a linhas e arcos, na segunda utilizando o Filtro de Sobel aplicado á matriz de distâncias. No capítulo seguinte será apresentada a implementação do algoritmo em estudo, nos seus diversos passos, para chegar ao objectivo pretendido: a localização do robô.

Capítulo 5

Algoritmo de localização: Implementação

O algoritmo de localização, descrito matematicamente no Capítulo 2, funciona com base nos dados obtidos a partir dos sensores, descritos no Capítulo 3, e do mapa, matriz de distâncias, do local onde o robô vai operar, descrito no Capítulo 4.

O primeiro, e importante passo, é a criação do mapa, pois todo o algoritmo se baseia nos seus valores, que são os definidos pela equação (2.2), a distância ao objecto mais próximo, pelo que, no algoritmo, é necessário saber qual dos pontos do mapa, se pretende ler, e isso é indicado pela equação (2.1), a qual depende da informação sensorial.

Nas secções deste capítulo será introduzida a implementação do algoritmo efectuada. Começando pela minimização da função custo, em que o seu resultado nos indica o quão exacta é a estimativa da localização actual do robô, passando pela optimização da estimativa calculada, em que se utiliza um algoritmo de minimização que permite convergir a estimativa da localização para uma estimativa com função custo de valor mínimo. A fusão das estimativas, entre a estimativa calculada e a odometria, sendo assim possível obter-se uma estimativa mais precisa, e, por fim, a localização global que dá ao algoritmo a capacidade de localizar o robô nas situações em que este “perde” a sua correcta localização.

5.1 Minimização da função custo

A minimização da função custo é feita começando por adquirir os dados provenientes dos sensores, ou do sensor, que estão a ser utilizados. Posteriormente, esses dados são definidos em forma de pontos com coordenadas (x, y) relativamente ao referencial do robô. Alguns dos sensores fornecem valores em forma de distância ao robô mas, sabendo a posição e ângulo, do respectivo sensor, relativamente ao referencial do robô é possível determinar o ponto que define essa distância relativamente a este e ao referencial de coordenadas global.

Após a lista de pontos, $s_1, \dots, s_n$, estar definida a posição destes é calculada no referencial de coordenadas global através da equação (2.1).

O valor da distância, equação (2.2), que permite pesar cada um dos pontos, é obtido através do mapa de distâncias, previamente calculadas, e guardadas numa matriz. Nessa matriz as coordenadas dos pontos são usadas para determinar a célula do mapa a que esse ponto corresponde, sendo aí lida a distância ao objecto mais próximo. Este valor permite calcular o erro correspondente, para essa coordenada, indicada pelo sensor, erro que é determinado pela aproximação da função custo, equação (2.5).

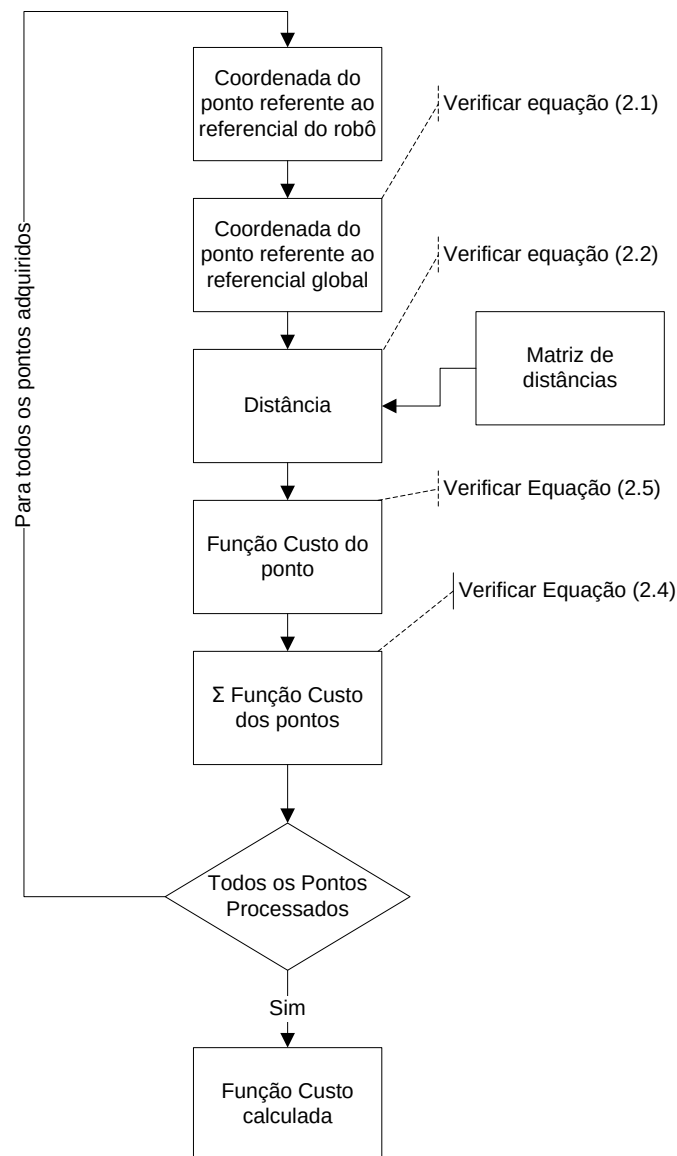


Figura 5.1: Cálculo da função custo

A soma do custo de cada um dos pontos da lista corresponde ao erro da estimativa, ou seja, à função custo a minimizar, equação (2.4). Se a informação dos sensores for correcta esse erro será mínimo pois cada um dos pontos aponta para uma célula em que a distância é aproximadamente zero, ou seja, cada célula corresponde à aproximação da distância a um objecto que existe no mapa, aquele que é medido pelo sensor.

A figura 5.1 ilustra o algoritmo utilizado. O código implementado encontra-se no Anexo A.1.

5.2 Optimização da estimativa

O cálculo da função custo discutida no ponto anterior, não estima a localização do robô, mas apenas fornece informação sobre quão correcta é a estimativa da localização actual do robô. Para calcular a estimativa da localização, do robô, é necessário optimizar a estimativa actual, ou seja, garantir a convergência da localização actual do robô para aquela onde o robô se encontra na realidade. Com base nos dados dos sensores, essa localização é aquela em que a função custo possui o erro mais baixo possível de se obter. Para estimar a localização do robô é usado o algoritmo RPROP.

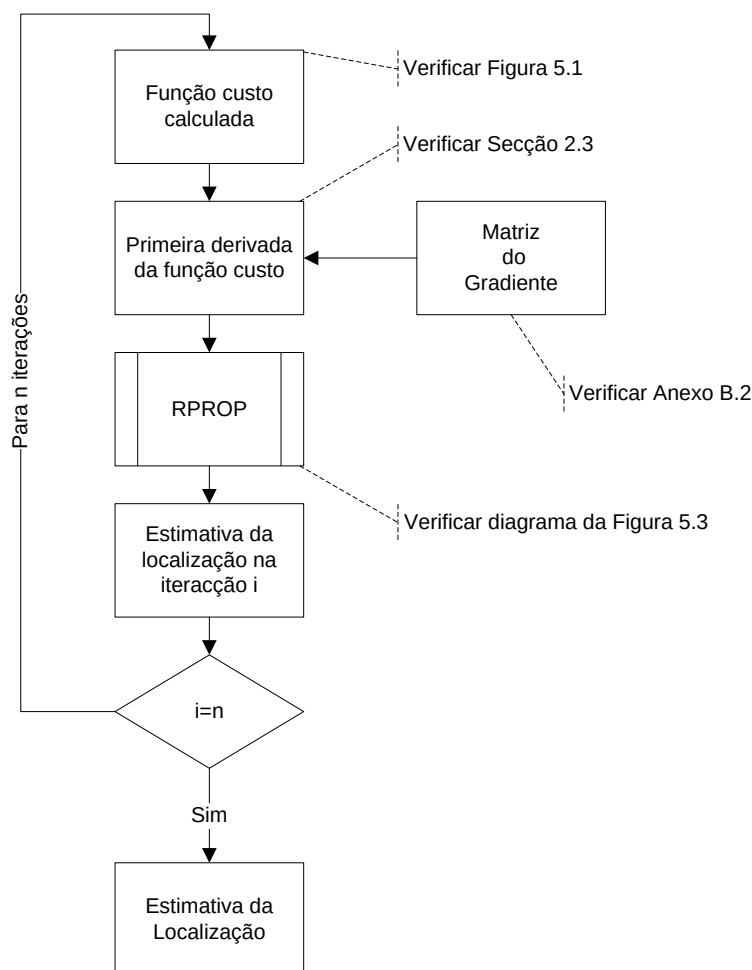


Figura 5.2: Cálculo da estimativa da localização do robô

A Figura 5.2 ilustra o algoritmo utilizado.

Após se obter minimização da função custo, como indicado na secção 5.1, é calculada a derivada dessa mesma função, em ordem à posição e orientação do robô, para, assim, se poder utilizar o algoritmo RPROP, uma vez que este necessita do conhecimento da evolução da função custo.

Assim, as primeiras derivadas da função custo em ordem à localização do robô (x, y, ϕ) são obtidas efectuando os cálculos apresentados nas equações (2.15) e (2.17). O código implementado encontra-se no Anexo A.1.

Em seguida aplica-se o algoritmo RPROP de modo a ser efectuada a convergência da estimativa da localização.

Considere-se o algoritmo RPROP apresentado na secção 2.2. Seja $grad(t) = \frac{\partial E}{\partial \omega_{ij}}(t)$, $grad(t-1) = \frac{\partial E}{\partial \omega_{ij}}(t-1)$, $passo = \Delta_{ij}^{(t)}$ e a estimativa da localização = $\omega_{ij}(t+1)$. Assim, vem:

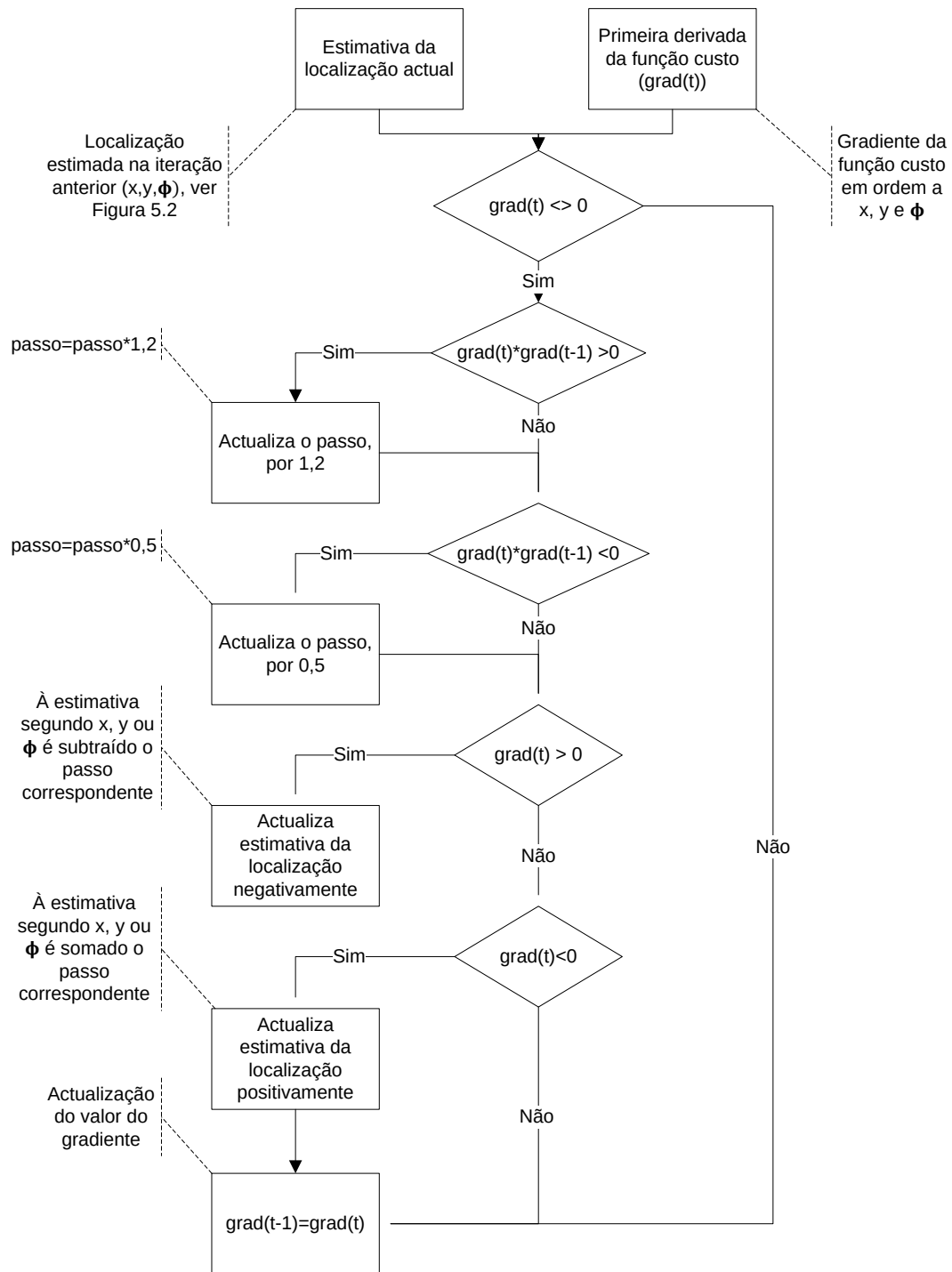


Figura 5.3: Algoritmo RPROP

e para cada uma das componente da localização, x, y e ϕ , é aplicado o algoritmo da Figura 5.3. Para que se garanta a convergência da estimativa da localização, são

aplicadas 10 iterações, valor obtido pelos testes efectuados. Em cada iteração a função custo é recalculada usando a mesma informação dos sensores mas usando agora a posição do robô actualizada. O código implementado pode ser consultado no anexo A.2.

5.3 Fusão da estimativa local com a odometria

Após calcular a estimativa do robô, usando a informação sensorial, revelou-se necessário suavizar essa estimativa da localização, pois, apesar de esta ser robusta, numa mesma zona podem existir várias células onde o robô se pode localizar, bem como, no mapa podem existir várias localizações suficientemente idênticas para que o robô salte de umas para as outras.

Nesse sentido, é feita a fusão da estimativa sensorial com a estimativa do deslocamento do robô obtida a partir da odometria. Deste modo, evita-se que o robô possa saltar de uma posição para outra, constantemente, quando este pouco se deslocou, e garante que o robô tem noção de que se está a movimentar, quando a estrutura da informação dos sensores não o permite.

5.3.1 Variância da estimativa do algoritmo de localização

O cálculo da variância da estimativa é realizado a partir da segunda derivada da função custo.

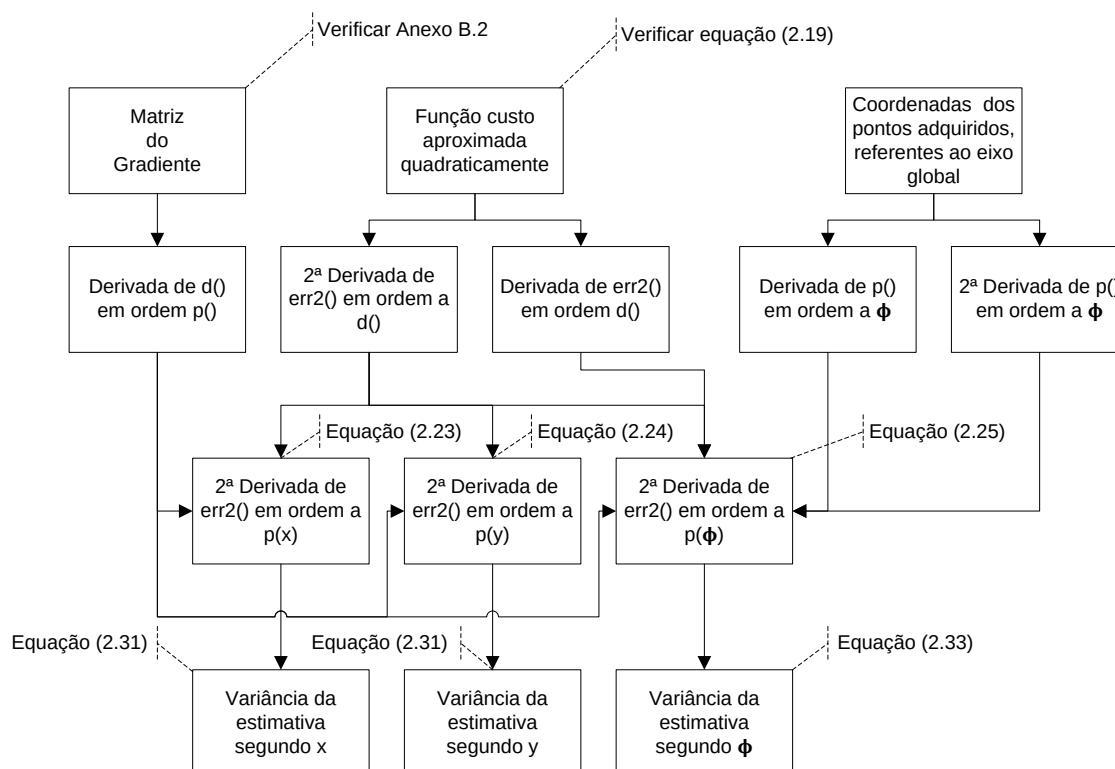


Figura 5.4: Segunda derivada da função custo

O diagrama da Figura 5.4, demonstra as dependências para o cálculo das segundas derivadas da função custo em ordem à posição, o código correspondente pode ser encontrado no anexo A.3.

No diagrama da Figura 5.4, temos:

- Derivada de $d()$ em ordem a $p()$

$$\mathbf{ddist} = \frac{\partial dist}{\partial pos} = \text{gradient}(pos) \quad (5.1)$$

$$\boxed{\text{ddist} := \text{Gradient}(p);}$$

em que $\text{gradien}()$ é uma matriz do gradiente da posição, derivada de $d()$ em ordem a $p()$, pré-calculada tal como descrito na secção 4.3.

- Primeira derivada de $\text{err2}()$ em ordem a $d()$, onde dist é a distância do ponto lida do mapa:

$$\mathbf{derr2} = \frac{\partial err2}{\partial dist} = \frac{dist}{c^2} \quad (5.2)$$

$$\boxed{\text{derr2} := \text{dist}/(c*c);}$$

- Segunda derivada de $\text{err2}()$ em ordem a $d()$:

$$\mathbf{d2err2} = \frac{\partial^2 err2}{(\partial dist)^2} = \frac{1}{c^2} \quad (5.3)$$

$$\boxed{\text{d2err2} := 1/(c*c);}$$

- Derivada de $p()$ em ordem a ϕ , onde o vector vis é a lista de pontos:

$$\mathbf{dposdphi} = \frac{\partial pos}{\partial \phi} = \begin{bmatrix} -\sin\phi & -\cos\phi \\ \cos\phi & -\sin\phi \end{bmatrix} * s_i \quad (5.4)$$

$$\boxed{\text{dposdphi.x} := -\sin(\phi)*\text{vis}[i].x - \cos(\phi)*\text{vis}[i].y;}$$

$$\boxed{\text{dposdphi.y} := \cos(\phi)*\text{vis}[i].x - \sin(\phi)*\text{vis}[i].y;}$$

- Segunda derivada de $p()$ em ordem a ϕ , onde o vector vis é a lista de pontos:

$$\mathbf{d2posdphi} = \frac{\partial^2 pos}{(\partial \phi)^2} = \begin{bmatrix} -\cos\phi & \sin\phi \\ -\sin\phi & -\cos\phi \end{bmatrix} * s_i \quad (5.5)$$

$$\boxed{\text{d2posdphi.x} := -\cos(\phi)*\text{vis}[i].x + \sin(\phi)*\text{vis}[i].y;}$$

$$\boxed{\text{d2posdphi.y} := -\sin(\phi)*\text{vis}[i].x - \cos(\phi)*\text{vis}[i].y;}$$

- A segunda derivada de $err2()$ em ordem a $p()$, ver equações (2.23) e (2.24), fica:

$$d2xy = \frac{\partial^2 err2}{(\partial pos)^2} = d2err2 * ddist^2 = \frac{1}{c^2} * (gradient(p))^2 \quad (5.6)$$

$d2xy.x := d2err2 * ddist.x * ddist.x;$

$d2xy.y := d2err2 * ddist.y * ddist.y;$

- A segunda derivada de $err2()$ em ordem à orientação, ver equação (2.25) fica:

$$d2phi = \frac{\partial^2 err2}{(\partial \phi)^2} = d2err2 * (ddist * dposodphi)^2 + derr2 * d2posdphi \quad (5.7)$$

$$d2phi = \frac{1}{c^2} * \left(gradient(p) * \begin{bmatrix} -\sin\phi & -\cos\phi \\ \cos\phi & -\sin\phi \end{bmatrix} * s_i \right)^2 + \frac{dist}{c^2} * gradient(p) * \begin{bmatrix} -\cos\phi & \sin\phi \\ -\sin\phi & -\cos\phi \end{bmatrix} * s_i \quad (5.8)$$

$$d2phi := d2err * (ddist.x * dposdphi.x + ddist.y * dposdphi.y) * (ddist.x * dposdphi.x + ddist.y * dposdphi.y) + derr2 * (ddist.x * d2posdphi.x + ddist.y * d2posdphi.y);$$

A variância é assim aproximada pelo inverso da segunda derivada do erro em ordem à posição. Tendo em conta o valor obtido para cada uma das variâncias verificou-se que estes eram muito superiores às variâncias calculadas para a odometria, pelo que o facto de não serem da mesma ordem de grandeza faz com que estas não tenham influência pretendida na localização aquando da fusão das estimativas. Assim, foi determinada uma relação heurística, resultado de vários testes, que coloca a variância da estimativa do algoritmo na mesma ordem de grandeza da variância da odometria. O valor determinado, $VAR_CONST=1.4e-10$, permite actualizar a localização do robô da forma adequada.

5.3.2 Variância da odometria

A variância da odometria, tal como já demonstrado no ponto 2.5.1, é aproximada pelo quadrado da variação da odometria, em x , y e ϕ . Aproximação que é comum para os robôs omnidireccionais, [11].

Visto que os valores da variação da odometria são calculados relativamente ao deslocamento do robô é necessário transportar esses valores para o eixo de coordenadas globais. Para isso faz-se uma rotação das coordenadas, pelo ângulo que o robô tem nesse momento.

Visto que, em cada instante de tempo, se está a considerar que a odometria do robô apenas possui uma pequena quantidade de erro, no seu deslocamento, presume-se que esta tem um erro de 10% do seu deslocamento, limitado pelo erro máximo admissível,

em metros, ou seja, em cada instante apenas é possível que o robô se tenha deslocado uma certa quantidade de metros. Este erro de 10% traduz a maior, ou menor, precisão do movimento.

5.3.3 Fusão das estimativas

A fusão das estimativas não é mais que uma média pesada entre as duas distribuições. Ver equações (2.34) a (2.39). O código correspondente encontra-se no anexo A.4.

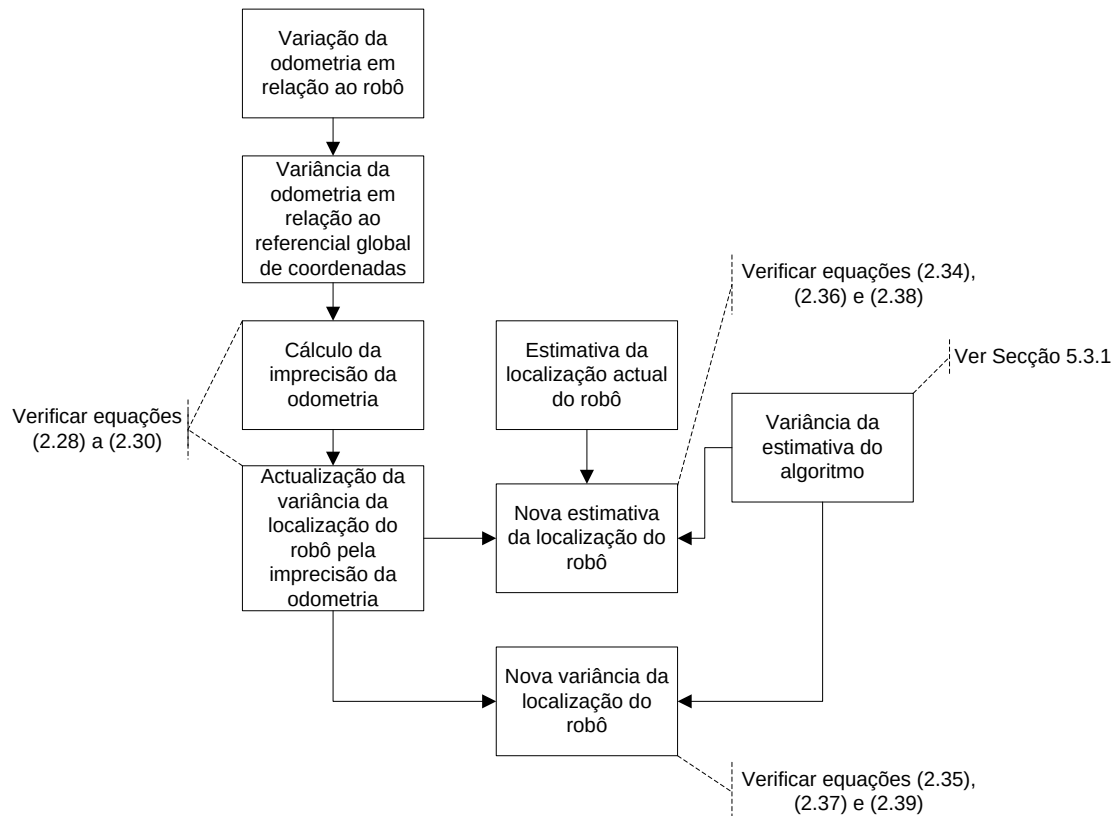


Figura 5.5: Fusão das estimativas

Como indicado na Figura 5.5, em cada instante de tempo, o robô movimentou-se, havendo variação na sua odometria. Essa variação é em relação ao referencial do robô, pelo que precisa de ser convertida em variação no referencial global de coordenadas, efectuando-se para isso uma rotação desta variação pela orientação do robô.

Após isto é calculada a imprecisão do movimento do robô, inserida pela odometria, e actualizada a variância global do robô com essa imprecisão. É então fundida a informação dada pela odometria com a informação dada pelo algoritmo de localização, e calculada a “correcta” localização do robô. Quanto maior for a variância de uma estimativa mais peso será dado à outra estimativa.

Após a localização do robô ter sido actualizada a variância da estimativa é também actualizada pesando igualmente a variância da estimativa da localização com a variância anteriormente actualizada pela odometria.

5.4 Localização global

O problema da localização global foi tratado, estimando a localização do robô a partir de qualquer ponto do campo, isto é, considera-se que o robô se pode localizar em qualquer célula, verificando-se em qual das células é possível ele estar. Visto que é incomportável em termos temporais e computacionais verificar sempre todas as células, em cada instante de tempo apenas é verificada uma determinada quantidade de células.

Seja: $pose = posição(x,y) + orientação$ do robô.

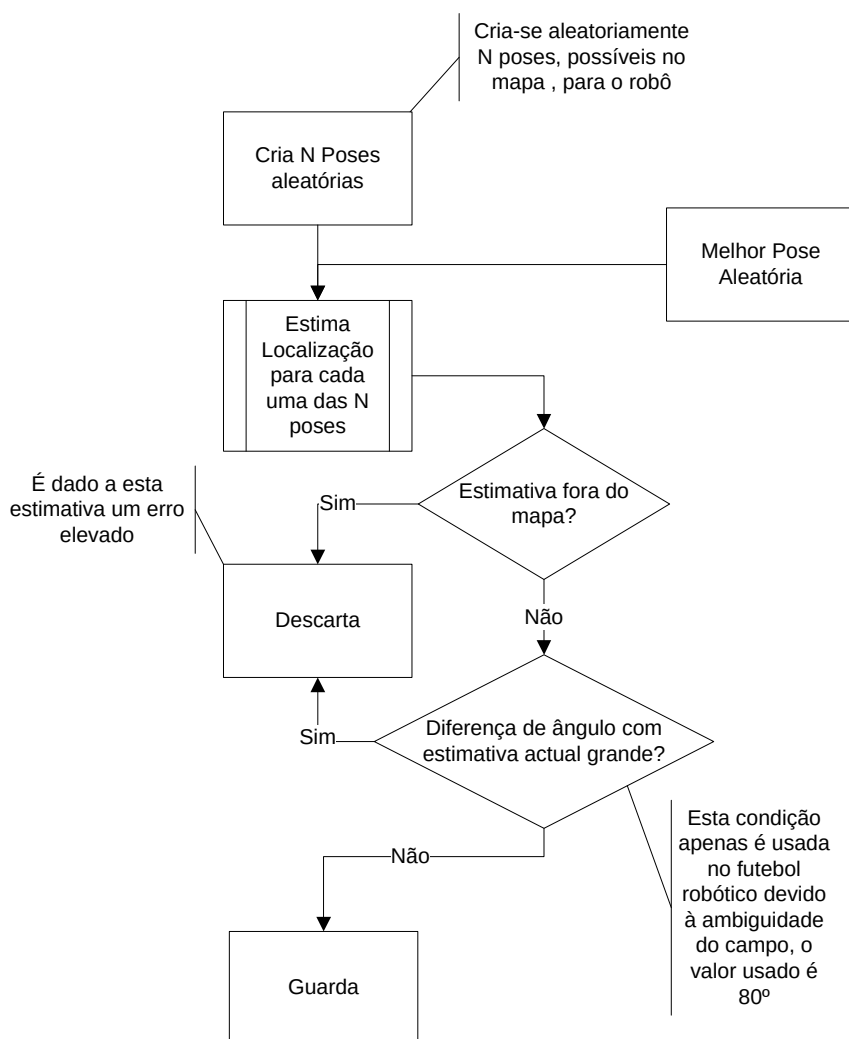


Figura 5.6: Localização global, estimativa da localização para n posições aleatórias

A Figura 5.6 representa o algoritmo implementado para a criação e estimação das poses aleatórias. Assim, em cada iteração temporal são lançadas N poses aleatórias onde o robô poderá estar. Para cada uma das poses é usada a informação sensorial actual para calcular a melhor estimativa da localização do robô a partir dessa pose. A estimativa da melhor pose aleatória, calculada no ciclo anterior, é também actualizada, usando a informação sensorial actual. Para cada uma das estimativas é calculado o correspondente custo, ou seja, a minimização da função custo, equação (2.4). Uma vez que são poses aleatórias pode acontecer que o resultado da estimativa calculada esteja fora do mapa. Ora como isso não é possível estas estimativas são descartadas, dando-

lhes um custo muito elevado. Por outro lado, e no caso do futebol robótico, não nos interessa que o robô inverta instantaneamente a sua orientação, que devido à estrutura do campo pode acontecer facilmente devido à simetria deste. Assim o ângulo da estimativa aleatória é verificado e se a diferença deste para o ângulo da localização actual for grande esta estimativa aleatória é também descartada. Se realmente o robô rodou mais que os 80° (ângulo limite usado), devido a rapto, por exemplo, a localização global consegue relocalizar o robô, pois os robôs de futebol robótico têm uma bússola para garantir que o robô sabe sempre minimamente a sua orientação, devido também à simetria do campo. O código desenvolvido pode ser visto no Anexo A.5.

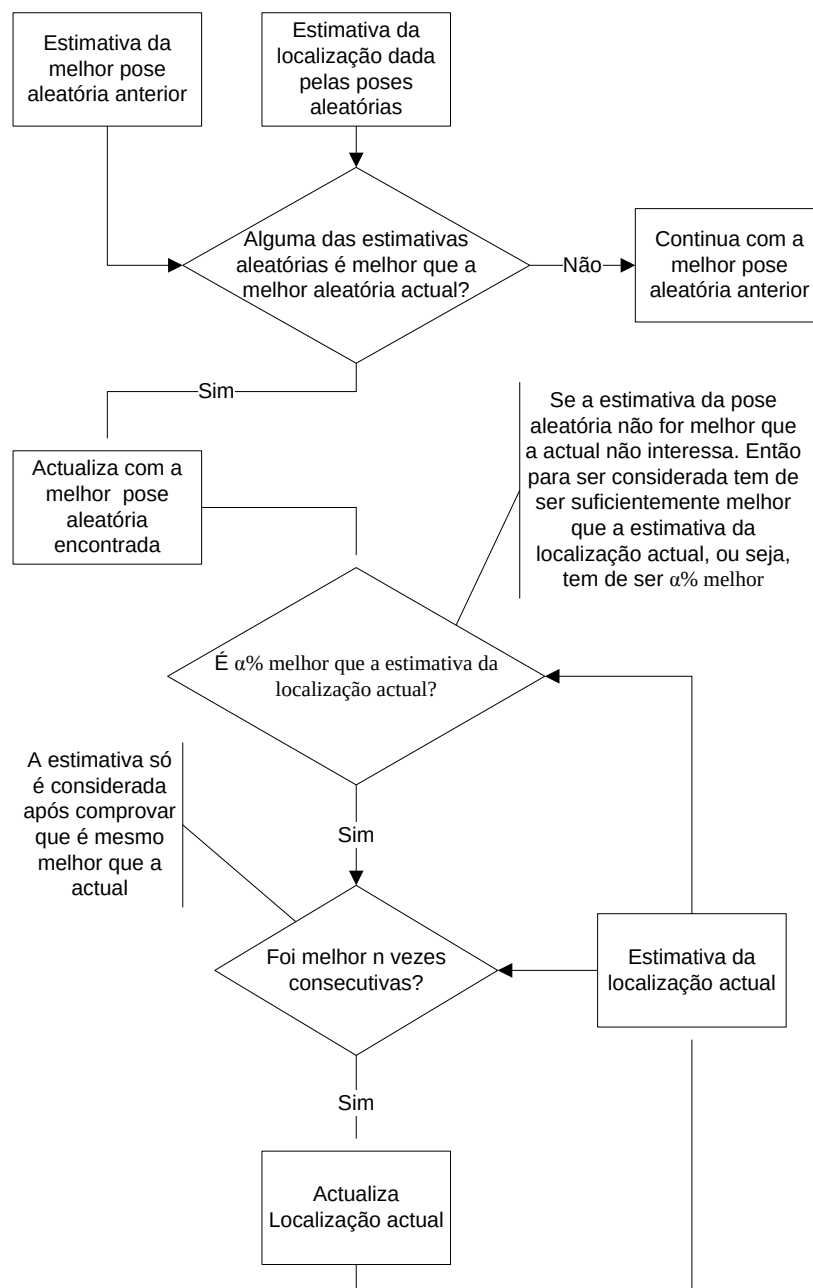


Figura 5.7: Localização global, decisão sobre as poses aleatórias

A Figura 5.7 representa a implementação do tratamento das estimativas aleatórias calculadas para determinar a possível relocalização do robô. Conhecido o custo de cada

uma das estimativas aleatórias, inclusive o da melhor anterior, estas são comparadas entre e si, e é escolhida a estimativa com menor custo para ser “proposta” a substituir a localização actual do robô. A melhor estimativa aleatória é então comparada com a estimativa actual, em termos de custo, sendo verificado se a primeira é melhor que a segunda por, no mínimo um factor α , isto para garantir que é uma estimativa confiável. Além disso esta estimativa tem ainda de conseguir ser melhor que a estimativa actual durante algum tempo para ser considerada, ou seja, durante n ciclos consecutivos. Passados os testes apresentados a estimativa aleatória substitui a estimativa actual.

Na implementação considerou-se $\alpha=5\%$ e $n=5$, ou seja, a estimativa da pose aleatória tem de ser no mínimo 5% melhor que a actual localização e tem de sê-lo 5 vezes consecutivas.

Neste capítulo foi apresentado a implementação do algoritmo de localização estudado. No capítulo seguinte serão apresentados resultados dos testes efectuados aplicando o algoritmo para a localização de robôs móveis omnidireccionais.

Capítulo 6

Resultados

Para comprovar a eficiência, robustez, bem como a possibilidade de se generalizar a utilização do algoritmo, este foi utilizado em situações distintas, robôs diferentes, bem como usando sensores diferentes. No Capítulo 1 foram já apresentados os sensores utilizados para testar o algoritmo implementado. Nas figuras seguintes pode-se observar os dois tipos de robôs utilizados: um robô de futebol robótico omnidireccional usando como sensor a visão (Figura 6.1); o *medical robot* [26], também omnidireccional, baseado na estrutura do anterior mas usando sonares, sensores de infra-vermelhos e/ou *laser range finder* (Figura 6.3)



Figura 6.1: Robô de futebol robótico da equipa 5DPO



Figura 6.2: *Medical Robot*

Nas secções seguintes será apresentada a interface criada para o algoritmo, para se poder visualizar o comportamento do algoritmo ao longo do tempo, bem como outros dados relevantes. Serão ainda apresentados os resultados de testes efectuados num robô de futebol robótico, no *medical robot* e com um *laser range finder*.

6.1 Interface implementada

Embora o algoritmo de localização se resuma a cálculos numéricos, mostrou-se necessário criar uma interface visual que permitisse observar o processamento e funcionamento do algoritmo. Assim implementou-se a referida interface que possui as seguintes funcionalidades:

- Criação do mapa, Figura 4.7.
- Visualização das leituras dos sensores.
- Representação, em forma de imagem, do mapa.
- Localização do robô nesse mapa, Figura 6.4.
- Representação da informação fornecida pelos sensores, relativamente ao robô, em forma de imagem, bem como dos dados adquiridos pelo sensor laser em específico (Figura 6.4).

Deste modo, esta aproximação permitiu fazer um *debugging* mais completo do algoritmo, encontrar possíveis falhas e corrigi-las, imediatamente, testar se os sensores estão a funcionar correctamente, bem como representar a informação relativa à localização que permita uma visualização, em tempo real, da estimativa da localização do robô. Permitiu ainda fazer testes *offline* do algoritmo, ou seja, através de logs, com dados reais, obtidos com o robô em movimento, e registando toda a informação necessária ao algoritmo num ficheiro, o qual é usado para os referidos testes e, dessa forma, não estar dependente da disponibilidade do robô, bem como garantir que tudo funciona correctamente antes de ser testado experimentalmente.

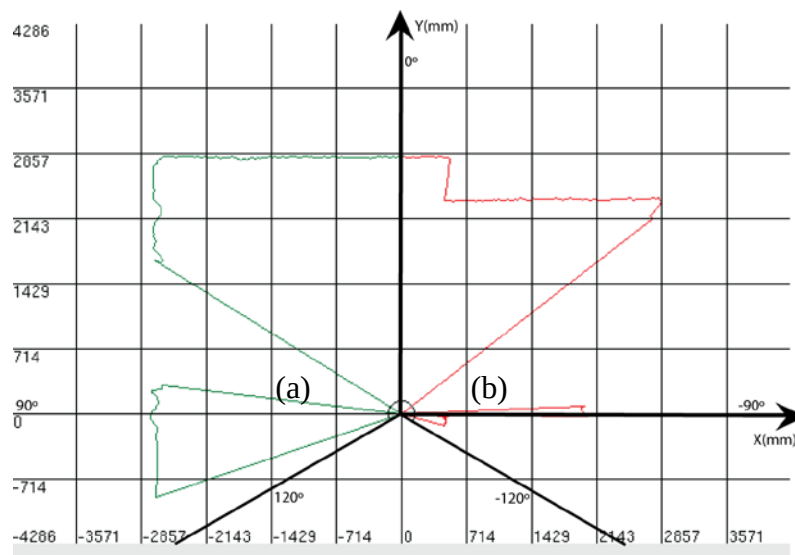


Figura 6.3: Representação da imagem do ambiente obtida pelo sensor laser

A Figura 6.3 apresenta uma imagem, em duas dimensões, associada a medidas do *laser range finder*. As linhas verde e vermelha, permitem diferenciar as medidas com ângulo menor que zero (a verde) e ângulo maior que zero (a vermelho). A informação visualizada diz respeito a coordenadas dos vários passos medidos. Assim, esta imagem permite-nos ter uma percepção da “estrutura do(s) objecto(s)”, em termos de distância, que o laser está a medir bem como se os está a medir correctamente. As zonas (a) e (b),

representadas na Figura 6.4, não correspondem a obstáculos mas sim a zonas onde o sensor laser não consegue medir por se encontrarem fora do seu alcance.

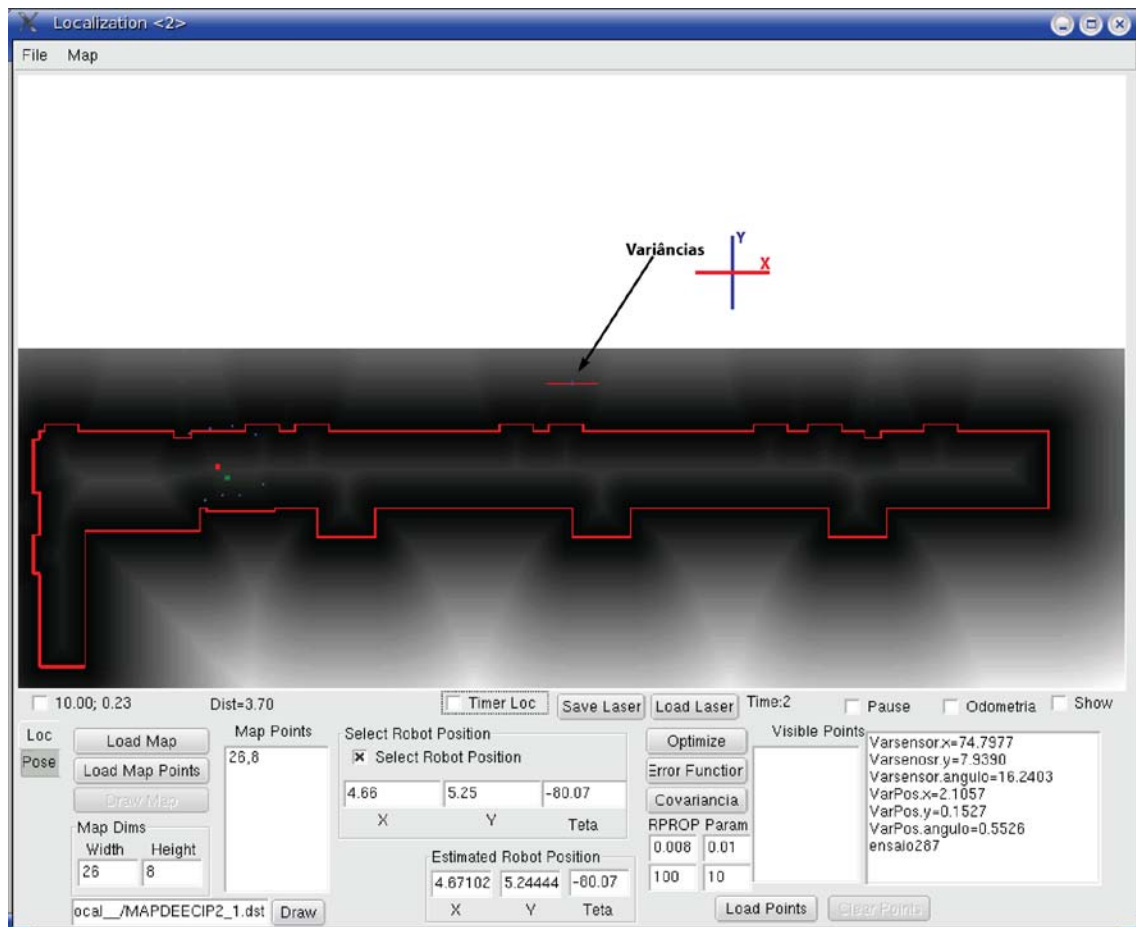


Figura 6.4: Interface do algoritmo de localização

A Figura 6.4 ilustra a interface de localização. Esta apresenta informação variada relativa à localização do robô. Em forma de imagem: o mapa onde o robô está a actuar; a localização actual do robô (ponto vermelho); a localização estimada pelo algoritmo antes de se efectuar a fusão de estimativas, representada pelo ponto verde; a informação relativa aos sensores, pontos azuis. A figura mostra ainda, a informação relativa à variância global da posição (x,y) do robô, no local indicado no mapa. Esta tem a forma de duas linhas, uma para o eixo x, linha vermelha, e outra para o eixo y, linha azul, que variam o seu tamanho em função da variância, a partir do ponto indicado no mapa, Figura 6.4. A figura permite também observar esta mesma informação em forma numérica, por baixo da imagem, havendo ainda botões e caixas de parâmetros que permitem executar situações específicas do programa.

6.2 Futebol robótico

No futebol robótico o tipo de ambiente onde o robô se insere é devidamente tratado, ou seja, o campo com medidas definidas nas regras do futebol robótico, [27]. O chão é

uma carpete verde (simulando o relvado de um campo real), e é delimitado por linhas brancas, tal como num campo de futebol real.

A criação do mapa do campo de futebol é, por si só, simples, pois todas as medidas são estritamente definidas pelas regras do futebol robótico, [27]. Na Figura 6.6 é possível observar uma representação gráfica do mapa de um campo de futebol robótico, gerado pelo programa. A representação gráfica é feita em forma de gradiente de cinzentos, que é função da distância. As zonas mais escuras representam as distâncias menores e as zonas mais claras representam as distâncias maiores.

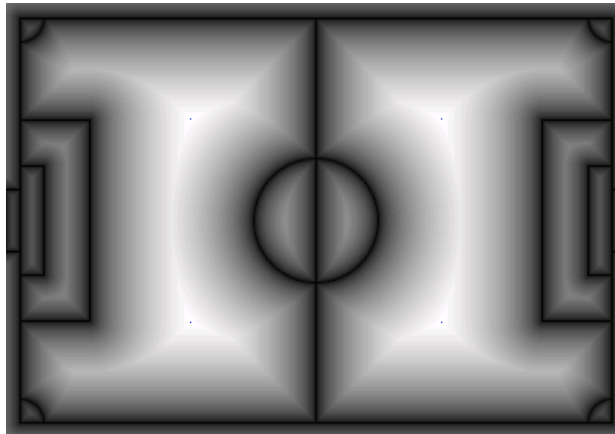


Figura 6.5: Mapa de distâncias do campo de futebol

Devido à grande quantidade de zonas definidas, isto é, linhas brancas que definem o campo, e por se utilizar uma câmara de vídeo, que obtém uma imagem omnidireccional do campo de futebol, é possível adquirir uma grande quantidade de informação utilizável da qual o algoritmo pode tirar usufruto. Assim a estimativa da localização obtida usando essa informação mostrou-se ser de muito boa qualidade, sendo mesmo possível usar apenas o algoritmo, sem fusão de estimativa com a odometria para localizar o robô no campo, embora se tenha mostrado mais susceptível a ruído e menos robusto.

Como se sabe, o campo de futebol é geometricamente simétrico, o que gera ambiguidade. Uma vez que a câmara do robô é omnidireccional, a forma como este vê o campo não fornece informação sobre a sua orientação, pelo que existem sempre dois locais possíveis no campo onde este se pode localizar, apenas diferenciáveis pela orientação do robô. Por esse motivo o conhecimento da orientação do robô é imprescindível para que este, usando apenas o algoritmo não se localize na posição ambígua da sua localização actual. Deste modo, e como na estrutura do robô já existe uma bússola que indica a orientação, esta é usada pelo algoritmo com este fim.

Devido à data em que foi possível testar o algoritmo no futebol robótico, a fusão da estimativa, apresentada anteriormente não estava ainda implementada, pelo que não foi ainda testada, neste caso. Para estes robôs, a fusão de estimativa, usada durante os testes, usa uma pesagem linear entre a informação da odometria do robô e a estimativa do algoritmo, ou seja:

$$0,8 \times (RoboPos(x, y, \phi) + \text{deltaOdom}(x, y, \phi)) + 0,2 \times EstimatedLoc(x, y, \phi) \quad (6.1)$$

Embora esta fusão linear dê uma maior confiança à odometria, permite que o robô não se perca, pois só com odometria, e após alguns metros de deslocamento o robô perde a sua localização devido ao erro inerente. Assim, a estimativa calculada pelo algoritmo garante a localização global do robô, mesmo em caso de “rapto”, pois, devido à informação disponível pelo processamento de imagem da câmara, o robô relocaliza-se no mapa, mesmo que a odometria indique que o robô pouco se moveu.

A Figura 6.7 apresenta a informação visual obtida pela câmara do robô. Esta é ilustra uma frame, obtida pela câmara omnidireccional, onde se pode observar a representação gráfica do processamento de imagem efectuado, isto é, para cada uma das linhas a vermelho é detectada a transição de verde para branco, pequena cruz desenhada na transição de cor. Essa transição permite saber as coordenadas desse ponto relativamente ao eixo de coordenadas do robô. Deste modo é criada uma lista das coordenadas dos pontos visíveis, que, depois, é usada para calcular a estimativa da localização do robô pelo algoritmo.

Na Figura 6.8 é possível observar o mapa geral do campo, pertencente ao programa de controlo do robô. Este fornece informação sobre a localização de todos os robôs no campo, informação obtida usando o algoritmo de localização. É possível observar nesta a localização do robô 3, robô que estava a ser utilizado para a demonstração.

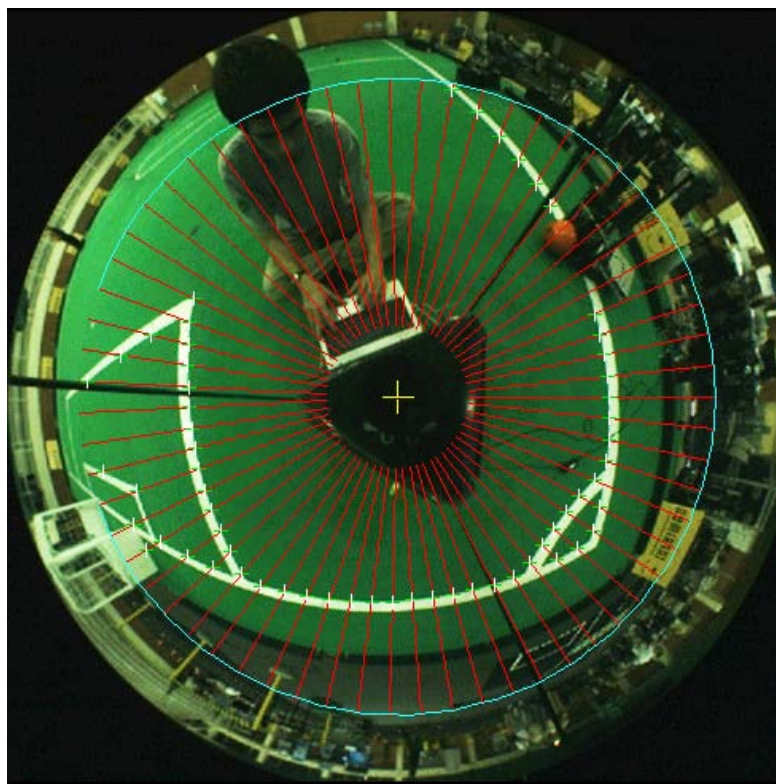


Figura 6.6: Imagem obtida pela câmara do robô, demonstrando também o processamento de imagem efectuado

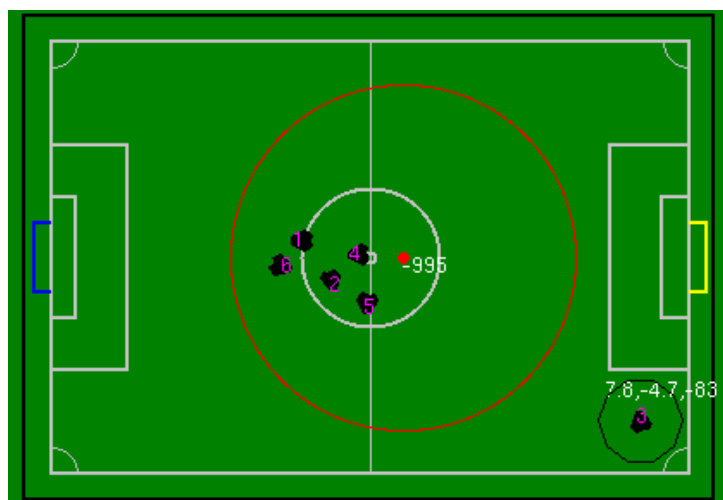


Figura 6.7: Mapa do campo de futebol, onde é possível fazer visualização do estado dos robôs da equipa durante o jogo.

A Figura 6.8 ilustra uma representação do mapa usado pelo algoritmo. Nesta pode-se observar a localização do robô três, dada pelo ponto vermelho localizado no canto inferior direito do mapa, as coordenadas dos sensores lidos pela câmara identificadas pelos pontos azuis, relativamente ao robô, a estimativa das poses aleatórias (a verde) no canto inferior esquerdo, por exemplo e quatro pequenos pontos azuis que indicam a distancia máxima da matriz de distancias. Como se pode verificar pelas três figuras, Figura 6.7, Figura 6.8 e Figura 6.9, o robô encontra-se correctamente localizado.

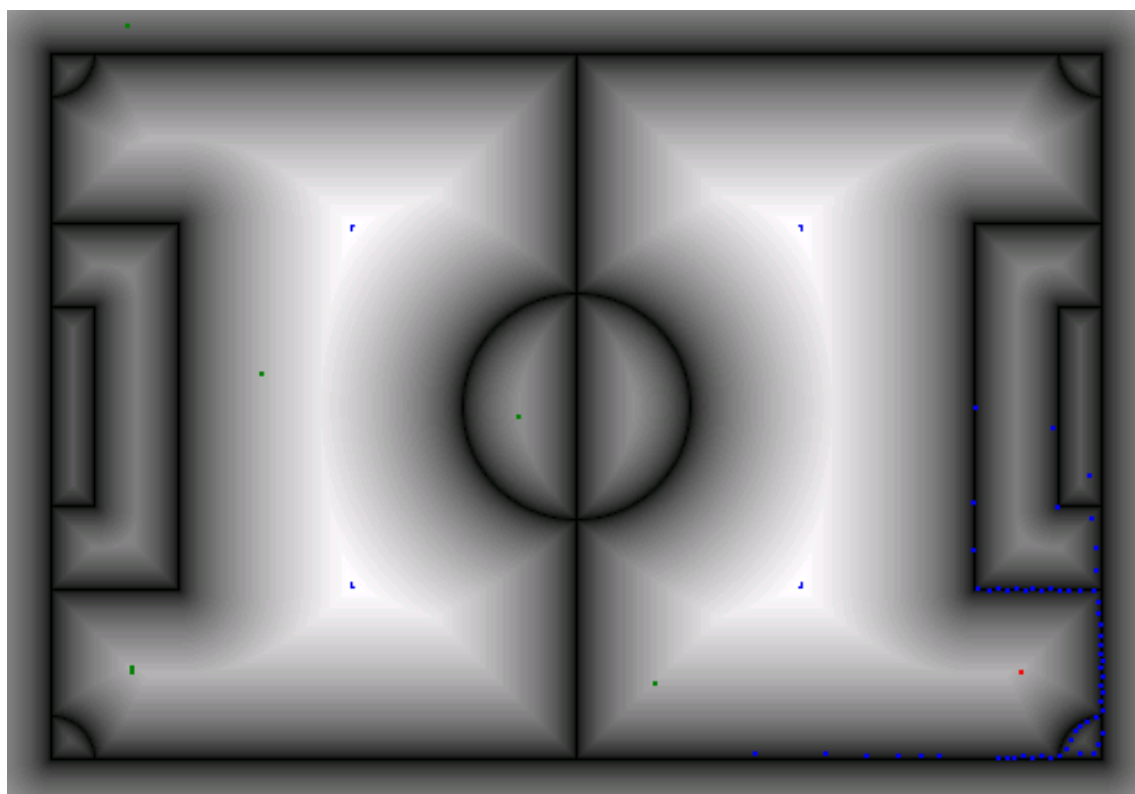


Figura 6.8: Mapa do campo de futebol robótico, com a representação da localização do robô (ponto vermelho), dos pontos da linha visíveis (pontos azuis) e das posições aleatórias para a localização global (pontos verdes)

Nas zonas laterais, junto à linha, como a câmara apenas consegue identificar pontos numa só linha, verificou-se que o algoritmo tem maior dificuldade em localizar correctamente o robô. Isto deve-se ao facto da estrutura de dados ser má nesta situação, pelo que, embora o robô possa ser correctamente localizado num dos eixos no outro eixo ele não o é de todo. Daí que a fusão da estimativa com a odometria seja realmente importante para evitar que o robô se perca, durante o tempo em que obtém penas informação deste tipo.

Em termos computacionais e temporais, tendo em conta o computador disponibilizado para fazer os testes, um portátil com um processador “Intel Centrino M 1,6Ghz”, da primeira geração deste tipo de processadores, e 1Gb de memória Ram, verificou-se que todo o processamento do algoritmo é efectuado em cerca de 4ms, o que é realmente bom quando comparado com os tempos obtidos com outros algoritmos, [11]. Pelo que, tendo em conta a evolução do hardware dos computadores, em especial dos processadores, que actualmente já vão na quinta geração dos processadores Centrino, com uma capacidade muito superior ao processador usado, os 4ms conseguidos podem ser reduzidos facilmente, o que permite, por outro lado, aumentar a quantidade de poses aleatórias geradas aumentando assim a eficiência da localização global do algoritmo.

6.3 Medical Robot

O *Medical Robot* [26], Figura 6.3, é um robô construído com base num robô de futebol robótico, pelo que a seu controlo é idêntico a um robô deste tipo. A diferença principal consiste na sua estrutura, e nos sensores usados que são adaptados para o ambiente onde se pretende que este opere: um hospital. Assim, neste robô estão montados, circularmente, vários sensores de infra-vermelhos (Sharp) e sonares [26]. Para testes específicos da localização foi ainda usado o *laser range finder*. Este foi colocado na parte superior do robô.

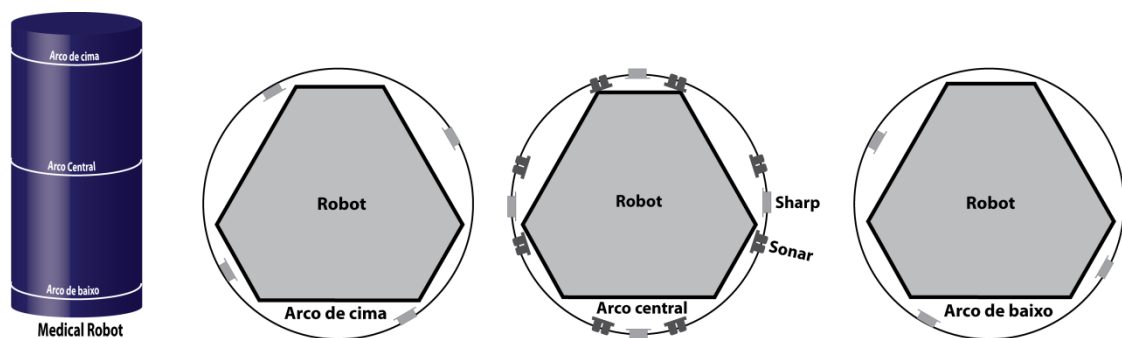


Figura 6.9: Posicionamento dos sensores, sonares e sharps no Medical Robot

A Figura 6.9 apresenta a disposição dos sensores (sonares e sensores IR) no *medical robot*.

Devido à maior disponibilidade deste robô, e a ser possível utiliza-lo no corredor do Departamento de Engenharia Electrotécnica e de Computadores (DEEC) da Faculdade de Engenharia da Universidade do Porto (FEUP), este foi o robô que foi mais usado e aquele que gerou mais resultados.

6.3.1 Sonares

Após alguns testes verificou-se que a informação resultante dos sonares não é de boa qualidade para poder ser tratável pelo algoritmo, já que, devido ao grande cone de abertura, cerca de 60° , são obtidas, constantemente, medidas erradas. Por este facto o sonar acabou por não ser usado com o algoritmo. Por outro lado, as características que são más para a sua utilização no algoritmo de localização, são boas para um algoritmo de controlo de desvio de obstáculos.

6.3.2 Infra-vermelhos

O cone de abertura dos sensores de infra-vermelhos é muito estreito pelo que a medida obtida contém menos imprecisão na medida, podendo existir um dos sensores que, devido à sua orientação, esteja a medir uma distância que na realidade não devia medir. Foi assim usada esta informação obtendo-se resultados satisfatórios no que toca à localização do robô. Neste caso o uso da fusão de estimativa é essencial pois o número de sensores de infra-vermelhos presentes no robô é pequeno, onze, não sendo essa informação suficiente para garantir a localização correcta do robô, em especial num corredor, onde as medidas obtidas são, na sua generalidade idênticas, isto sem contar com algum erro inerente a essa medida.

6.3.3 Localização

Considerando o mapa da Figura 6.11 e o movimento do robô, representado pela linha azul, e usando a fusão das estimativas, a Figura 6.12 apresenta o percurso do robô usando apenas odometria. Na Figura 6.14 podemos ver o percurso do robô dado pelo algoritmo de localização, e, na Figura 6.13 o percurso do robô após se efectuar a fusão da estimativa entre o algoritmo e a odometria. Note-se que, na representação presente na Figura 6.14, a estimativa da localização actual teve influência da fusão da estimativa no instante anterior, isto é, a primeira figura representa a estimativa da localização antes da fusão e a segunda representa a estimativa da localização depois da fusão.

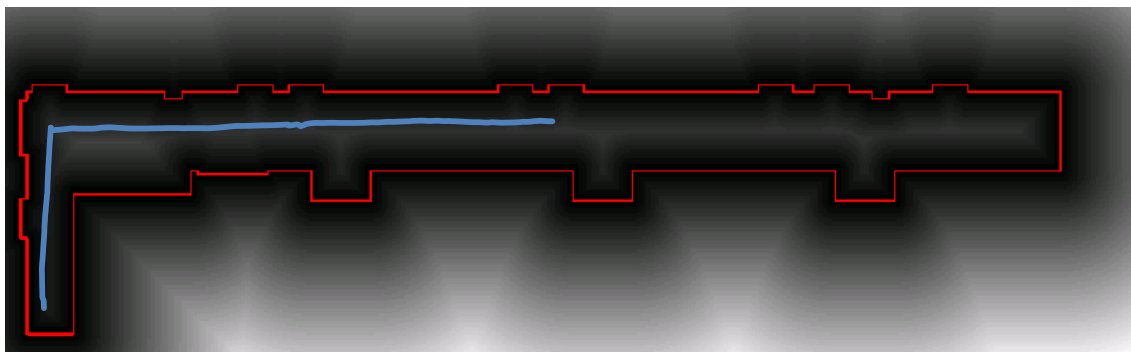


Figura 6.10: Mapa do corredor onde o robô circulou, a azul encontra-se o percurso efectuado pelo robô.

Como se pode verificar pela Figura 6.12 o robô parte de uma posição desconhecida, em concreto da origem do mapa, mas sem saber se essa localização é a real. Assim, para a odometria parte da origem mas, para o robô as coordenadas são de cerca de 0,5m, em x, relativamente à origem, ou seja no centro do corredor. Como se pode concluir a partir da Figura 6.13 este facto é, desde logo, corrigido pela estimativa do algoritmo.

Após o movimento do robô e a rotação sobre si próprio pode-se observar, na Figura 6.12, que a odometria fornece, ao robô, uma localização errada, pelo menos num dos eixos, pois na realidade o robô está a andar em frente e não para o lado, ao contrário do que a odometria está a indicar. Isto porque o movimento do robô, principalmente, a rotação, aumentam o erro da odometria [6].

A informação dos sensores permite ao robô saber onde este se encontra no mapa, ou seja, visto que estes medem as paredes laterais do corredor, é possível ao algoritmo localizar o robô correctamente segundo o eixo perpendicular ao deslocamento do mesmo. Note-se que o mesmo não acontece no eixo paralelo, por este não ter informação dos sensores que medem para a frente, ou para trás, do robô, caso esse que só acontece quando o robô chega ao canto superior esquerdo, zona onde roda sobre si próprio 90°. Embora não seja perceptível na Figura 6.14 a estimativa do algoritmo, segundo o eixo paralelo ao deslocamento do robô, é muito imprecisa pois esta não tem informação suficiente para determinar correctamente a localização segundo este eixo. Assim esta estimativa tem a tendência de puxar o robô para trás, ou para a frente. No entanto tem uma variância grande e não afecta muito a estimativa depois da fusão. Neste caso a estimativa da odometria é que permite ao algoritmo de localização ter noção do movimento do robô segundo este eixo.

A fusão das estimativas, apresentada na Figura 6.13, demonstra como é essencial o seu uso, isto é, a odometria, por si só, levaria o robô a perder-se rapidamente, mas, por outro lado o algoritmo de localização, devido à estrutura do mapa, não seria capaz de identificar se o robô se está a mover. Assim, a odometria fornece a informação de movimento do robô, enquanto o algoritmo de localização garante que o robô se mantém sempre localizado. Como se pode ver, Figura 6.13, a fusão da estimativa suaviza a estimativa da localização do robô.

Na Figura 6.15, é possível observar como evoluiu o movimento do robô ao longo do percurso em termos de odometria. Como se pode constatar, inicialmente o robô moveu-se no eixo dos yy, a uma velocidade de cerca de 0,35m/s por tempo de amostragem (100 ms), até uma distância de cerca de 5m, ver Figura 6.13. Durante este percurso, do robô, é possível observar algum ruído na variação do ângulo medido, que se deve apenas à instabilidade da odometria. Note-se que cerca dos 5m o robô rodou, como se pode observar pela curva da variação do ângulo, tendo, a partir deste ponto, mantido o ângulo o mais constante possível, notando-se ainda uma pequena rotação, um pouco antes da amostragem 300, e, também, um pouco depois da amostragem 400.

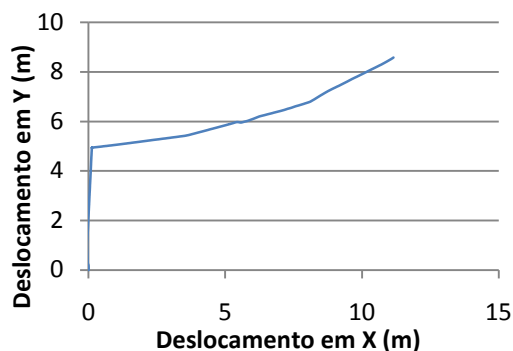


Figura 6.11: Percurso do robô usando apenas a odometria

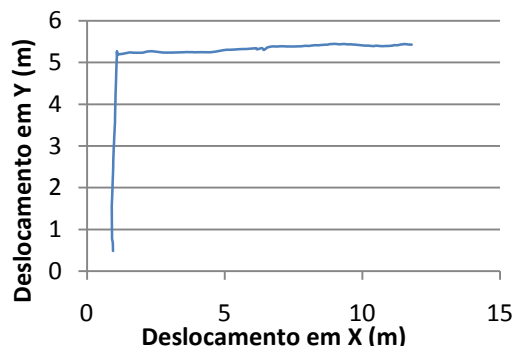


Figura 6.12: Percurso do robô, resultante da localização usando o fusão da estimativa da odometria com a do algoritmo

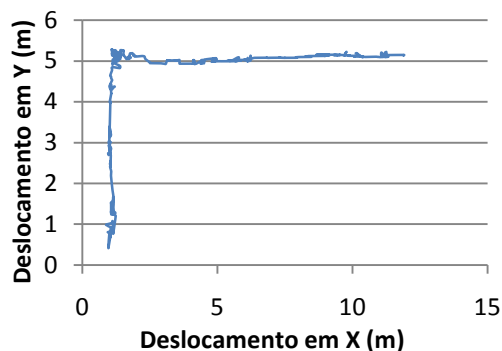


Figura 6.13: Percurso do robô resultante da estimativa da posição obtida pelo algoritmo

Após rodar, aos 5m, o robô apenas se deslocou no eixo dos xx durante o restante percurso. Como se pode observar, pela curva de variação da odometria segundo o eixo xx, o robô, inicialmente, não se deslocou segundo este eixo, sendo a sua velocidade após a rotação, aos 5m, de cerca de 0,35m/s. Da observação pela variação da odometria segundo o eixo dos yy, as rotações originaram a inserção de erro na mesma segundo este eixo, pelo que, se nota que após a primeira vez, que o robô rodou, à amostragem 170, existe variação da odometria, embora pequena, segundo este eixo, sendo que esta aumenta na rotação seguinte do robô, continuando a aumentar gradualmente, nas pequenas rotações seguintes.

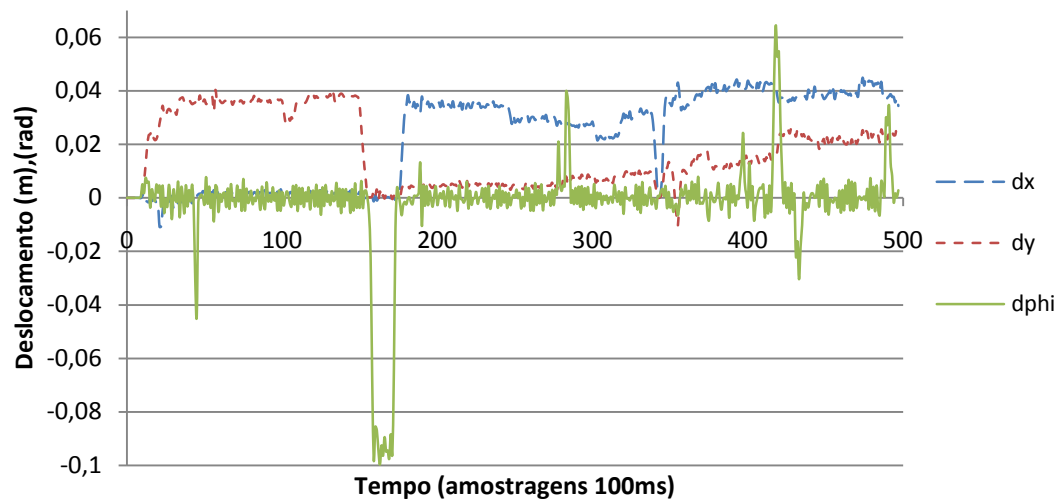


Figura 6.14: Evolução da variação da odometria ao longo do deslocamento do robô, segundo o eixo x, o eixo y e a orientação ϕ do robô.

Nas figuras seguintes, Figura 6.16, Figura 6.17 e Figura 6.18, são apresentados os gráficos da evolução do movimento segundo x,y e phi, respectivamente.

Comparando as três figuras apresentadas é possível observar, uma vez mais, que embora a odometria indique que o robô se desloca segundo o eixo yy, a partir da amostragem 200, o algoritmo de localização efectua a devida compensação, mantendo o robô na sua localização correcta. Por outro lado, a odometria garante que o robô se desloca segundo o eixo xx. A comparação entre a Figura 6.17 e a Figura 6.18 demonstra como a fusão das estimativas garante a suavização da estimativa da localização do robô, pois o algoritmo tem tendência a variar constantemente a sua estimativa da localização, devido à fraca qualidade e quantidade da informação sensorial.

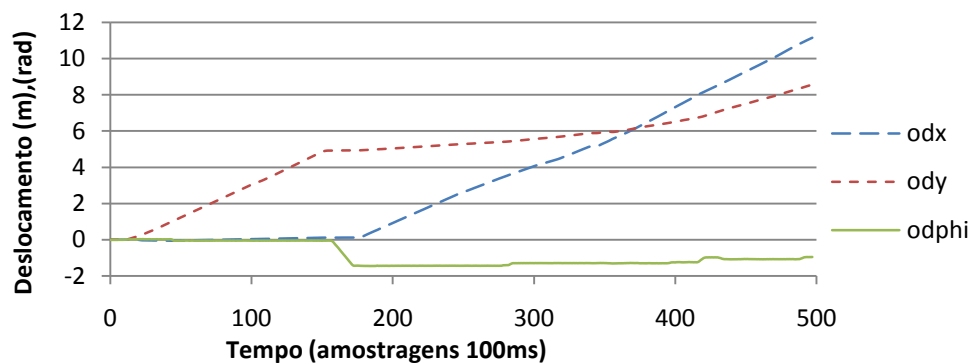


Figura 6.15: Evolução do deslocamento do robô segundo a odometria

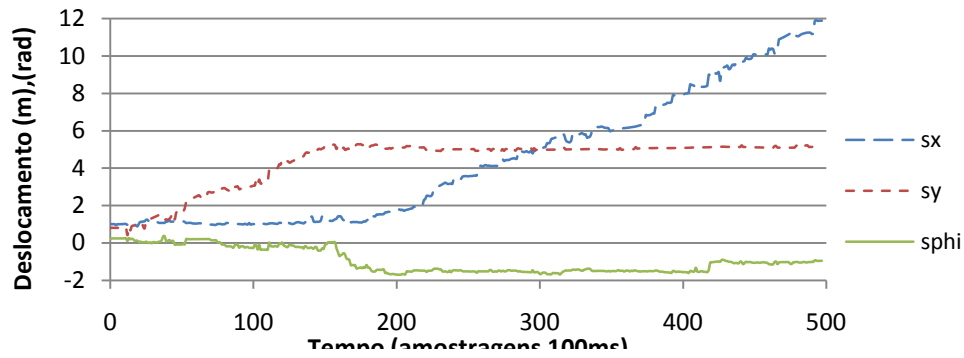


Figura 6.16: Evolução do deslocamento do robô segundo o algoritmo de localização, antes da fusão de estimativas

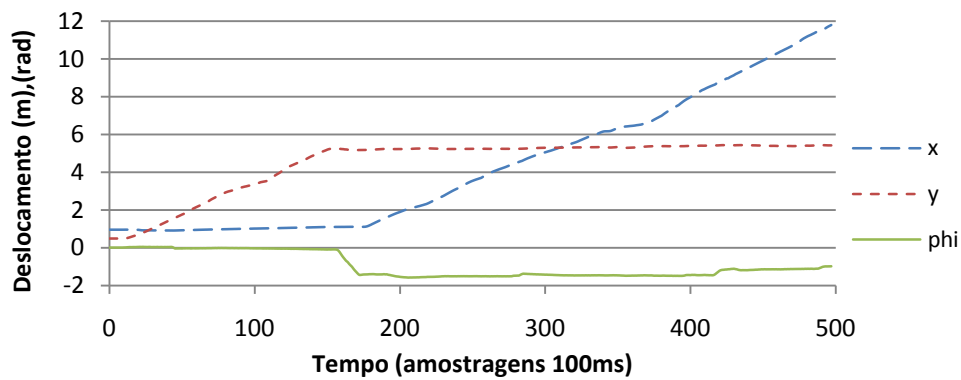


Figura 6.17: Evolução do deslocamento do robô com a localização efectuada, depois da fusão de estimativas

6.3.4 Variâncias e fusão das estimativas

As figuras seguintes, Figura 6.19, Figura 6.20, Figura 6.21, Figura 6.22 e Figura 6.23 apresentam os gráficos da evolução da variância das estimativas envolvidas na localização para o percurso efectuado pelo robô apresentado na Figura 6.10.

Assim, a Figura 6.19 mostra como evolui a variância da odometria, sem a influência da variância da estimativa de localização, ou seja, sem que seja efectuada a fusão de estimativas. Os cálculos associados são efectuados de acordo com as equações (2.28), (2.29) e (2.30). Na mesma pode ver-se a influência do movimento do robô, segundo o correspondente eixo, na variação da variância.

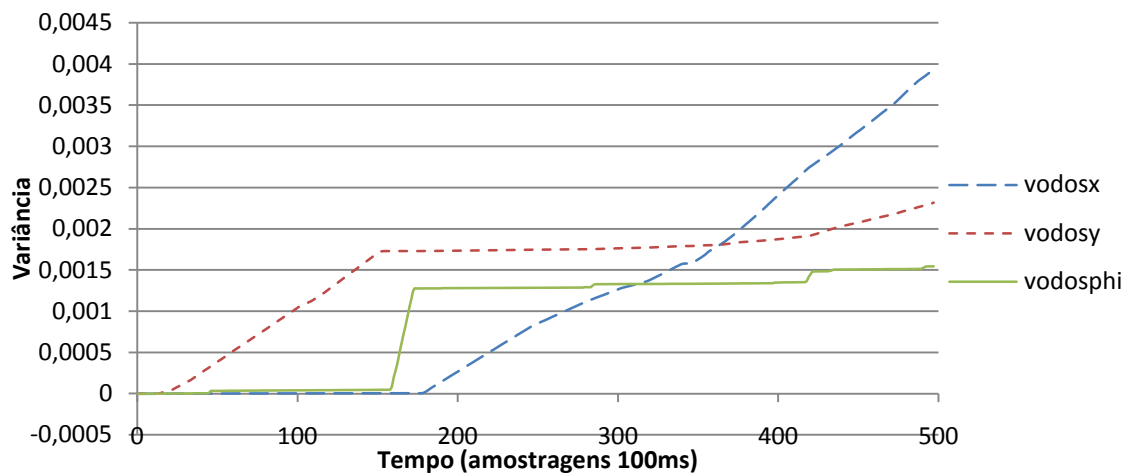


Figura 6.18: Evolução da variância da odometria ao longo do deslocamento do robô

A Figura 6.19, apresenta a variância segundo o eixo xx. Na mesma pode observar-se:

- Inicialmente, como a informação dos sensores do robô é dada segundo o eixo xx, a sua variância é baixa. Isto porque o algoritmo consegue determinar, com confiança, a estimativa da posição segundo este eixo.
- A partir do momento em que existe movimento no mesmo, amostragem 150, Figura 6.19, deixa de haver informação sensorial segundo este eixo, é difícil ao algoritmo determinar, com exactidão a posição do robô. Deste modo há um aumento da variância. Tal é mais acentuado nas zonas com paredes lisas, ver Figura 6.10, pois, nestas zonas, a incerteza da posição do robô é muito maior. Note-se as amostragens entre 230 e 330, e entre 430 e 500, Figura 6.19.

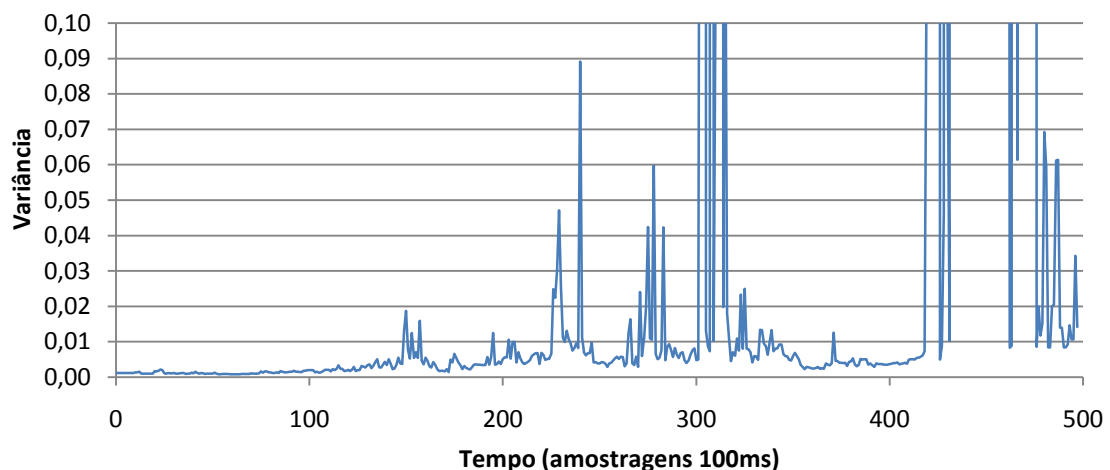


Figura 6.19: Evolução da variância do algoritmo de localização para o movimento no eixo x

A Figura 6.20, que apresenta a variância segundo o eixo yy, permite perceber a elevada incerteza da posição do robô, segundo este eixo, logo no início do seu deslocamento (até à zona, amostragem 110, Figura 6.20). A após o robô rodar este desloca-se segundo o eixo xx, tendo ao seu dispor, constantemente, a informação dos sensores segundo o eixo yy. Assim, a variância permanece baixa até ao fim do percurso do robô, pois, neste eixo, o algoritmo localiza o robô com elevada exactidão.

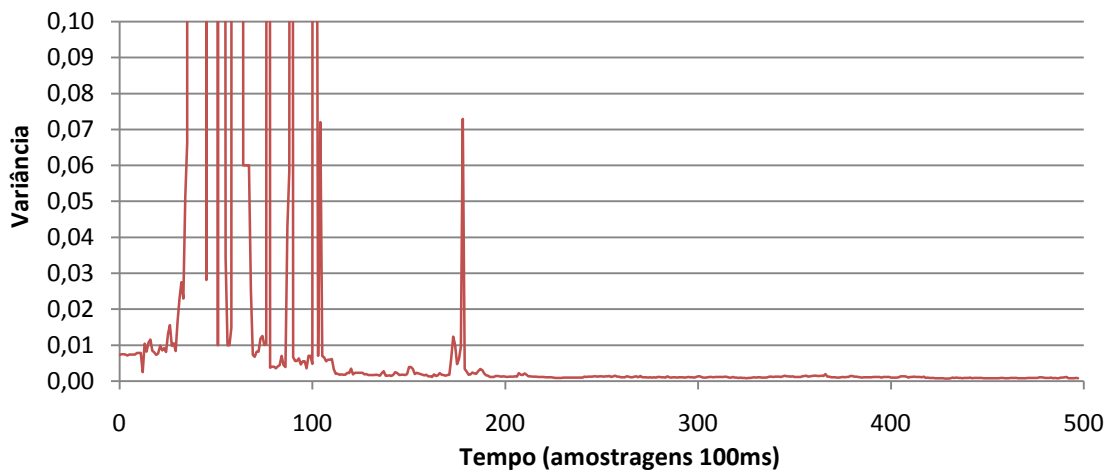


Figura 6.20: Evolução da variância do algoritmo de localização para o movimento no eixo y

A Figura 6.21, ilustra a variância associada à orientação. Denota-se que, inicialmente, a incerteza, sobre a orientação do robô é elevada, principalmente até à altura em que o robô roda, a partir da qual essa incerteza se mantém baixa. A incerteza inicial deve-se ao facto de o robô não estar num local onde é possível obter informação distinta, que permita estimar, exactamente, a sua orientação, bem como ao facto de os sensores do arco superior nada medirem, por nesta zona haver uma parede baixa. Assim, quando este começa a andar, visto que a estrutura do mapa não muda muito, a estimativa mantém uma elevada incerteza quanto à orientação do robô. Note-se que a partir do momento que este roda e como o robô se encontra junto a um canto, a informação dos sensores é suficientemente variada, e bem estruturada, para que se estime com exactidão, a sua orientação. Deste modo a variância estabiliza.

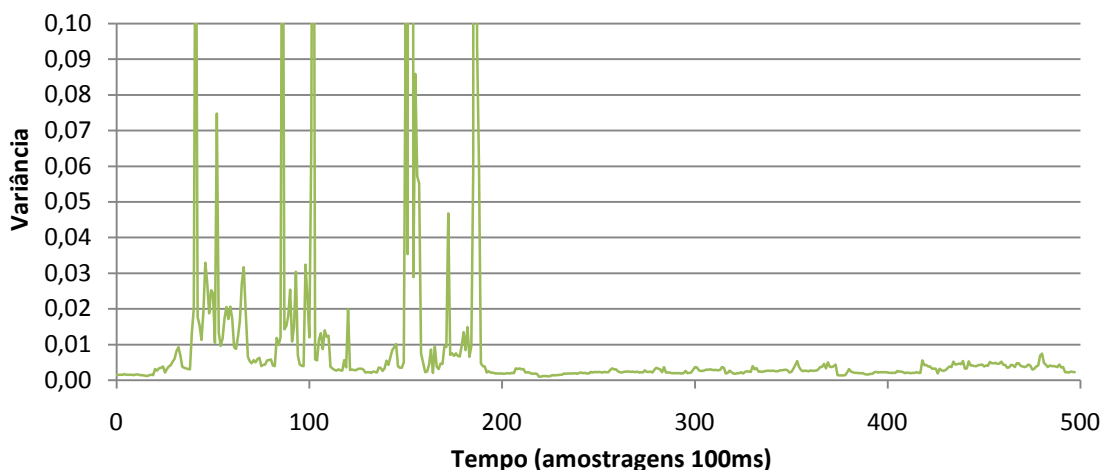


Figura 6.21: Evolução da variância do algoritmo de localização para a orientação do robô

Na Figura 6.22 é possível observar a evolução da variância global do robô, após a fusão das estimativas. Os cálculos efectuados estão de acordo com as equações (2.35), (2.37) e (2.39), para x , y e ϕ , respectivamente. Assim, como se pode observar pelos gráficos, inicialmente a incerteza, segundo o eixo yy , aumenta gradualmente, pois o robô desloca-se neste sentido, e a informação sensorial não é suficiente para o algoritmo a diminuir. A partir do momento em que os sensores conseguem medir a parede frontal, amostragem 70, Figura 6.22, o algoritmo diminui a variância global. Na amostragem 170, Figura 6.22, quando o robô roda, a variância da orientação aumenta, puxada pela odometria, diminuindo assim que o robô termina a rotação, pois a informação dos sensores é agora suficientemente boa para determinar a orientação. Neste instante começa também o movimento segundo o eixo xx , e, por esse motivo, a variância segundo este eixo aumenta, levada pelo erro inerente à odometria. Esta, porém, é limitada pelo algoritmo, pois no mapa existem pontos, como as portas ou esquinas do corredor, que permitem identificar a posição do robô segundo este eixo, embora com pouca exactidão. A partir da amostragem 400, Figura 6.22, surge uma parede lisa. Assim, o algoritmo não consegue estimar, com precisão, a posição do robô e, por esse motivo, a odometria, leva este valor a aumentar, até que na amostragem 480, Figura 6.22, surgem duas portas.

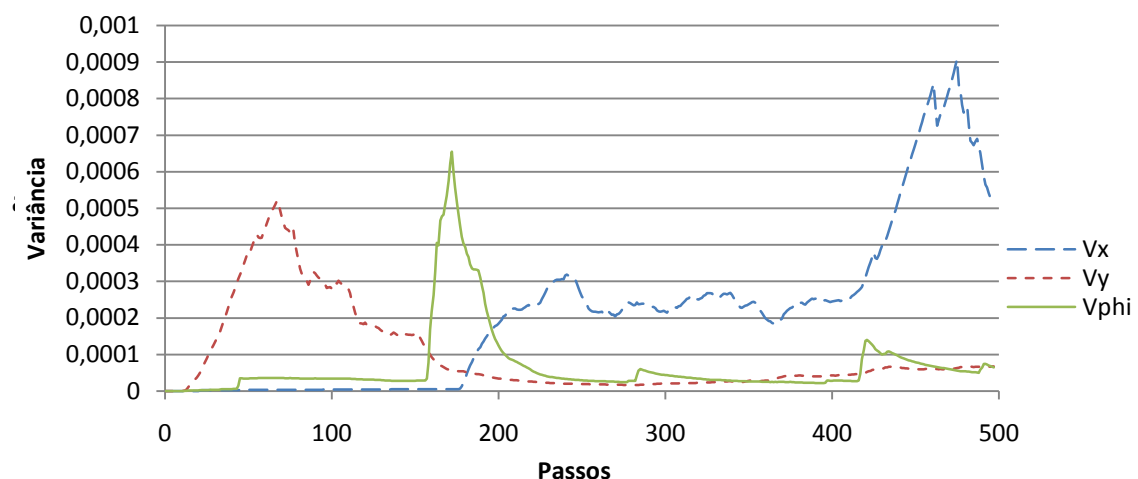


Figura 6.22: Evolução das variâncias globais do robô, para o movimento x , y e orientação ϕ

6.4 Laser Range Finder

Tendo sido apresentado na secção anterior o desempenho do algoritmo com um robô de futebol robótico e com o *medical robot* com uma câmara de vídeo e sensores IR respectivamente, apresenta-se neste ponto o desempenho na situação de um corredor mas usando um *laser range finder*.

A utilização do *laser range finder* permite demonstrar a eficiência do algoritmo quando a informação é de qualidade. Com o *laser range finder* utilizado, Figura 3.9, é possível obter medidas com uma resolução de milímetros, e em grande quantidade, o que permite o cálculo da estimativa da localização pelo algoritmo com grande robustez e exactidão. Os testes efectuados demonstram que usando apenas o *laser range finder* é possível estimar, robustamente, a localização global do robô, sem odometria. Assim,

com a fusão da estimativa do algoritmo com a odometria, a localização do robô é eficiente, robusta e exacta.

Tendo em conta as condições disponíveis, não foi possível testar o uso do *laser range finder* em conjunto com a odometria do robô, pelo que, apenas se obtiveram dados relativos à localização do robô apenas com o *laser range finder*. Foram efectuados três testes com a diferença no número de pontos medidos pelo *laser range finder* no seu raio de leitura de 240°. Como referido, no Capítulo 2, este sensor é capaz de medir 681 pontos distintos em cada ciclo de medida.

As características visíveis do mapa, bem como a precisão das leituras do laser, permitem que o robô seja capaz de se localizar usando apenas o laser, desde que se disponha de uma quantidade aceitável de pontos. Tal é comprovado em três testes que se apresentam em seguida: o primeiro teste usa 115 pontos distintos, cobrindo a amplitude total de medida; o segundo teste usa 34 pontos, e o terceiro, e último, teste usa 12 pontos.

6.4.1 Resultados usando 115 pontos de medida

O teste do *laser range finder* é feito usando uma amostra de 115 pontos dos 681 possíveis. Note-se que a utilização de todos os pontos originaria um aumento do tempo de processamento e com consequente diminuição da eficiência. A linha azul da Figura 6.23 representa no mapa, o percurso de ida e volta do robô. Da sua análise verifica-se que, com este número de pontos, a localização do robô é exacta, não deixando que este se perca durante o seu movimento, mostrando capacidade de distinguir o movimento do robô.

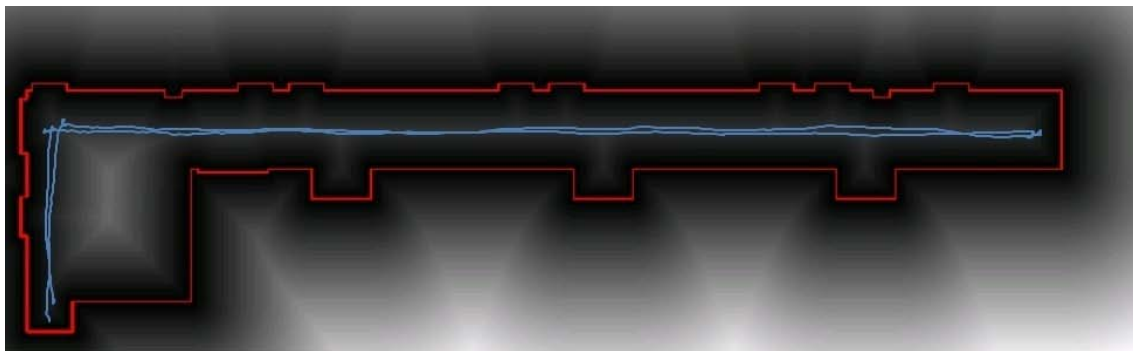


Figura 6.23: Mapa e percurso efectuado pelo robô (linha a azul), usando 115 medidas do laser

Para se ter uma melhor percepção do movimento efectuado, a Figura 6.25 e a Figura 6.26 representam o movimento do robô apenas num dos sentidos de deslocamento. Como se pode observar o movimento do robô não é linear já que, este foi, neste caso, conduzido manualmente. Os percursos apresentados são calculados aplicando o algoritmo com a informação obtida do laser. Assim, pode, desde já, verificar-se que o algoritmo é capaz de estimar a correcta localização do robô com alguma precisão, sendo

mesmo possível, devido à resolução do mapa criado, 4cm (tamanho da célula), ter uma percepção dos mínimos desvios associados ao movimento

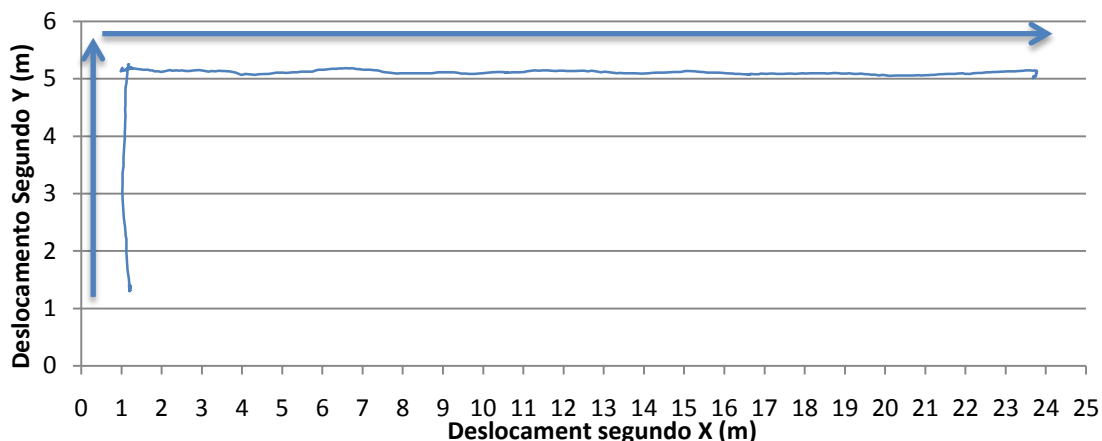


Figura 6.24: Movimento do robô em pormenor na direcção positiva

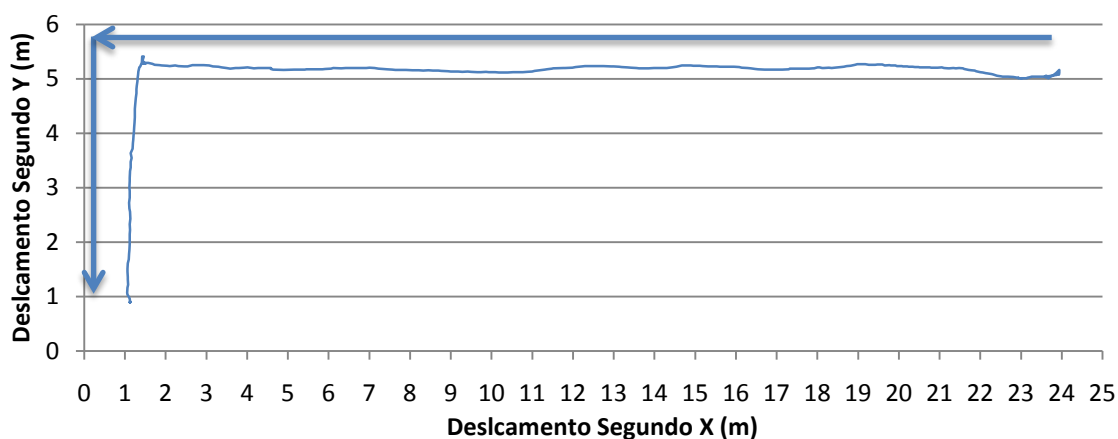


Figura 6.25: Movimento do robô em pormenor na direcção negativa

Como se pode observar, na Figura 6.26, o robô inicia o seu deslocamento aproximadamente na posição (1.2,1.2) com uma orientação de 90° , Figura 6.26. Inicialmente há um deslocamento segundo o eixo yy até aos 5 metros de distância. Aqui, o robô roda cerca de 90° , no sentido horário, e a partir deste ponto o robô desloca-se segundo o eixo xx até chegar ao fim do corredor. Aproximadamente na amostragem 400, o robô inverte o sentido do movimento, e refaz o percurso, no sentido inverso, até ao ponto de partida. Como se pode ver todo o percurso é estável e é possível perceber que a localização foi estimada com sucesso.

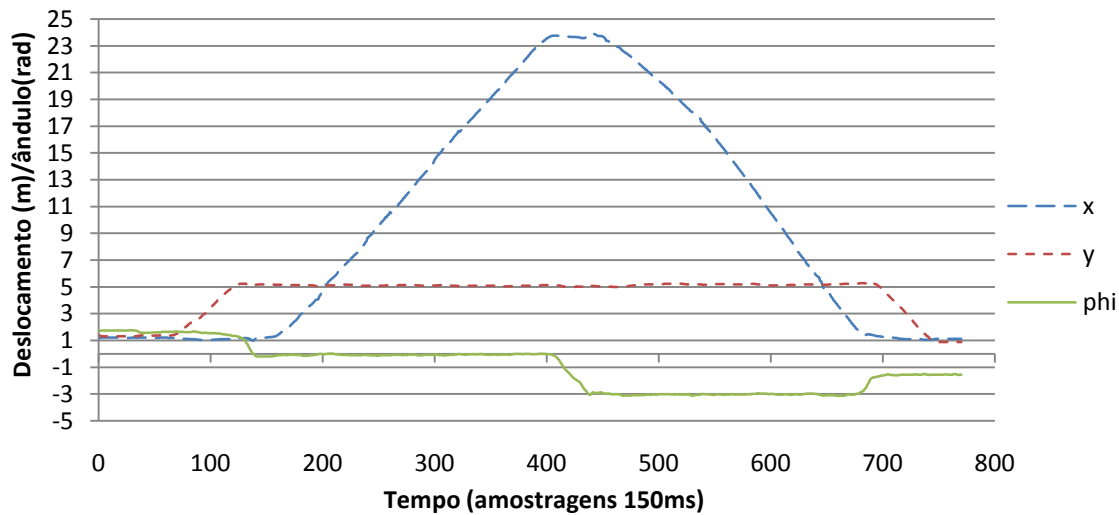


Figura 6.26 Variação da posição (m) e orientação do robot (radianos) ao longo do tempo para todo o percurso efectuado

A análise das variâncias das estimativas obtidas pelo algoritmo permitem ter uma melhor percepção do que se passou durante o percurso do robô. Assim, na Figura 6.28, Figura 6.29 e Figura 6.30, são apresentadas as variâncias das três componente da localização do robô. Tal como se pode observar, pelos seus valores numéricos quando comparadas com os resultados obtidos para o uso dos sensores infra-vermelhos, Figura 6.20, Figura 6.21e Figura 6.22, a confiança na estimativa calculada é elevada, já que, o valor da variância é pequeno.

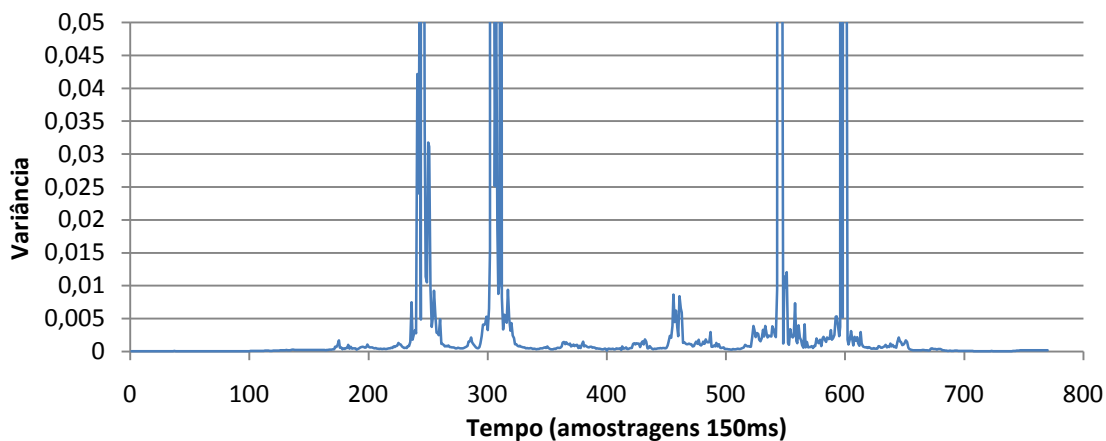


Figura 6.27: Variância da estimativa da posição segundo o eixo x

A variância segundo o eixo xx, Figura 6.27, ilustra o comportamento do algoritmo no cálculo da estimativa segundo este eixo. Inicialmente, o robô desloca-se segundo o eixo yy. Devido às características do mapa, a variância é muito baixa, durante maior parte do percurso, havendo apenas picos na sua variação quando o robô passa junto às paredes lisas, ver Figura 6.23. Estas zonas provocam um aumento da incerteza da posição, devido à pouca variação das medidas, originando dois picos na variância durante a ida (amostra 235 e 305), e outros dois picos na volta (amostra 540 e 590, Figura 6.27). Pelo meio, no fim do corredor surge um pequeno pico da variância, amostra 450, Figura 6.27. Tal acontece quando o robô roda sobre si próprio nesta

posição, o que é normal devido à variação das medidas dos sensores, durante o movimento rotativo.

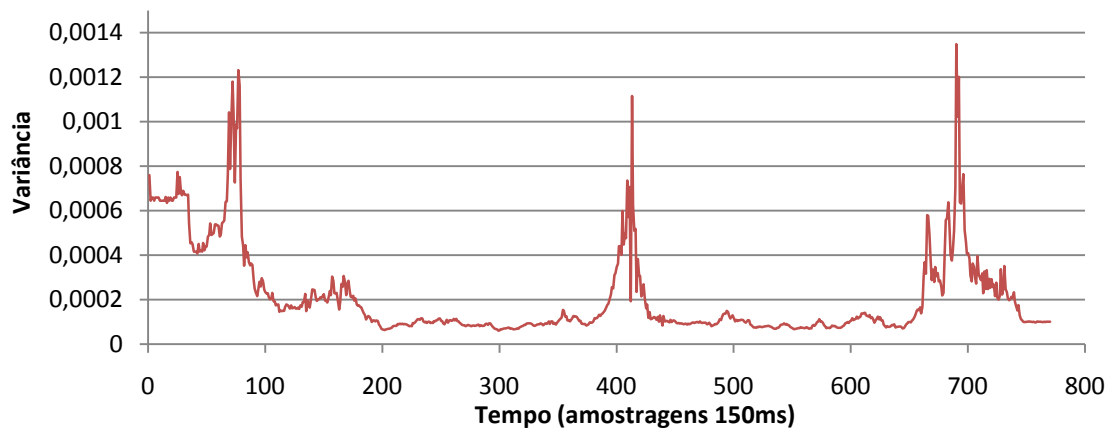


Figura 6.28: Variância da estimativa da posição segundo o eixo y

A variância segundo o eixo yy, Figura 6.28, não varia tão bruscamente como no caso anterior, havendo apenas um pequeno aumento ao início do deslocamento. Isto surge devido às características do mapa. Pode observar-se ainda dois outros picos: um quando o robô roda e outro no fim do percurso, na zona em que o robô se desloca de novo segundo este eixo, Figura 6.28. Note-se que os valores da variância, em média, são baixos, quando comparado com o caso anterior.

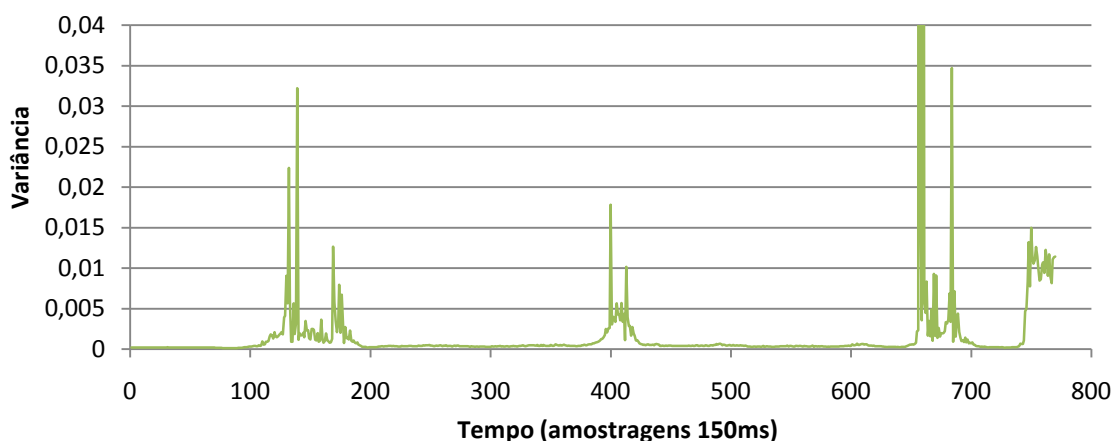


Figura 6.29: Variância da estimativa da orientação do robô

No que toca à variância da orientação do robô, Figura 6.30, vemos que esta é sempre baixa, durante todo o percurso do robô. Assim este, sabe sempre a sua orientação. Esta exhibe valores mais elevados quando o robô roda, tal como era esperado. Como se pode verificar este aumento é, ainda assim baixo.

A Figura 6.31, apresenta um gráfico da variação do valor da função custo da estimativa calculada pelo algoritmo que ilustra a sua exactidão. Como se pode observar, apesar do seu baixo valor, este aumenta, tanto no início como no fim do percurso e, ainda, quando o robô roda. No primeiro caso deve-se claramente à estrutura de dados

ser mais fraca que no restante percurso, ou seja, maiores distâncias para medir, o que, por vezes, devido ao alcance do *laser range finder* não é feito. Quando o robô roda, a variação da orientação, e o ajuste da sua estimativa provoca um aumento do erro que se deve ao facto do algoritmo de convergência da estimativa (RPROP) efectuar uma convergência mais fina para a orientação.

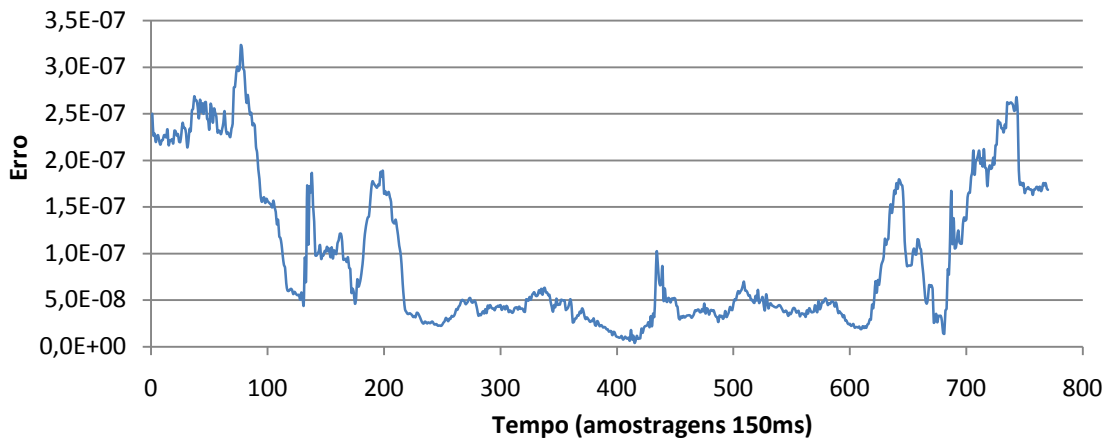


Figura 6.30: Variação do valor da função custo da estimativa ao longo do percurso efectuado pelo robô

6.4.2 Resultados usando 34 pontos de medida

Esta secção será descrita mais resumidamente sendo efectuada, principalmente, uma análise comparativa com a anterior.

Para este teste a quantidade de medidas foi reduzida para menos de metade das efectuadas no teste anterior. Pretende-se analisar a influência da quantidade de medidas na estimativa da localização calculada.

A Figura 6.32, apresenta o percurso efectuado pelo robô no corredor. Foi tentado o mesmo percurso anterior. Também aqui são perceptíveis as oscilações da posição. O algoritmo foi capaz de, mesmo com poucos pontos efectuar correctamente a sua localização.

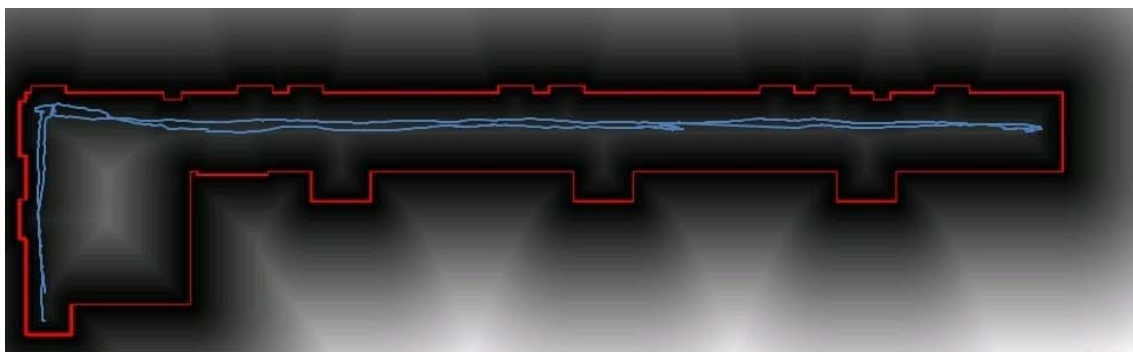


Figura 6.31: Mapa e percurso efectuado pelo robô (linha a azul), usando 34 medidas do laser

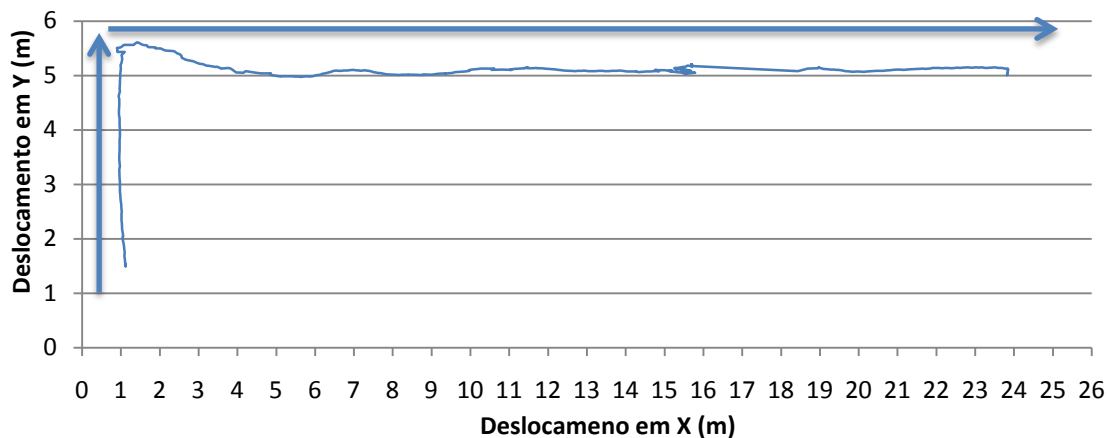


Figura 6.32: Movimento do robô em pormenor na direcção positiva

A Figura 6.32, apresenta o sentido positivo do deslocamento,. Neste sentido o único ponto a apontar verifica-se entre a posição (6, 15) e (6,17), Figura 6.32. Local onde o robô “perde” a sua localização. Note-se que, no entanto, este foi capaz de se relocalizar um pouco depois. Já durante o deslocamento negativo, no regresso à posição de origem, Figura 6.33, nada há a apontar pois o algoritmo de localização comportou-se bem tendo localizado correctamente o robô.

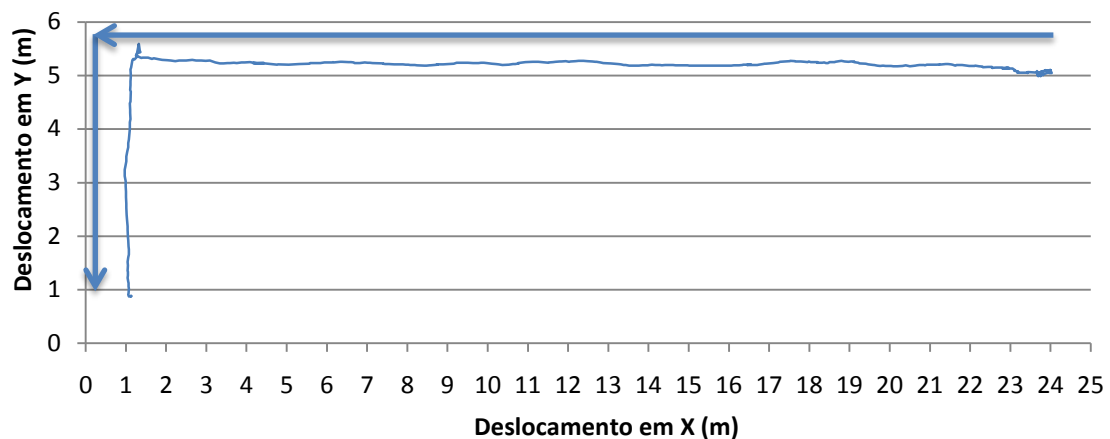


Figura 6.33 Movimento do robô em pormenor na direcção negativa

Na Figura 6.34, pode-se observar o deslocamento do robô segundo os eixos xx e yy bem como a variação da sua orientação. Uma análise da curva a azul, deslocamento segundo xx, permite, desde logo, perceber que algo anormal se passou. A partir da amostragem 270, na posição (15,5), Figura 6.34, o robô “parou”. De acordo com a localização, uma verificação do mapa, Figura 6.32, permite perceber que esta posição se situa num local onde o *laser range finder* apenas mede paredes lisas. Assim, e dado que, durante um certo espaço de tempo, não há informação “variada”, a localização não foi capaz de manter o robô em movimento. Note-se que com a fusão de estimativa isto não aconteceria. Para verificar se este conseguiria relocalizar-se a velocidade do robô foi também reduzida. Deste modo na posição (19,5), o laser foi capaz de medir as portas aí existentes, o que permitiu à localização global relocalizar o robô correctamente. A partir

deste local o algoritmo não teve problemas efectuando sempre a localização correcta do robô.

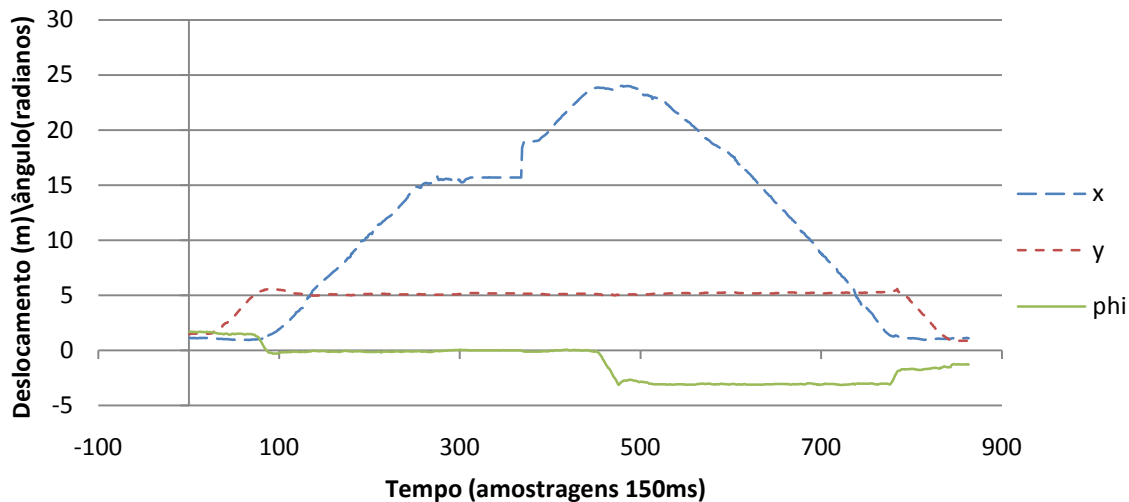


Figura 6.34: Variação da posição (m) e orientação do robot (radianos) ao longo do tempo para todo o percurso efectuado

A comparação entre os gráficos das variâncias para este caso, Figura 6.36, Figura 6.37 e Figura 6.38, e para o caso anterior, Figura 6.28, Figura 6.29 e Figura 6.30, permite verificar a existência de um aumento na incerteza do algoritmo de localização. Ainda assim os valores obtidos são baixos.

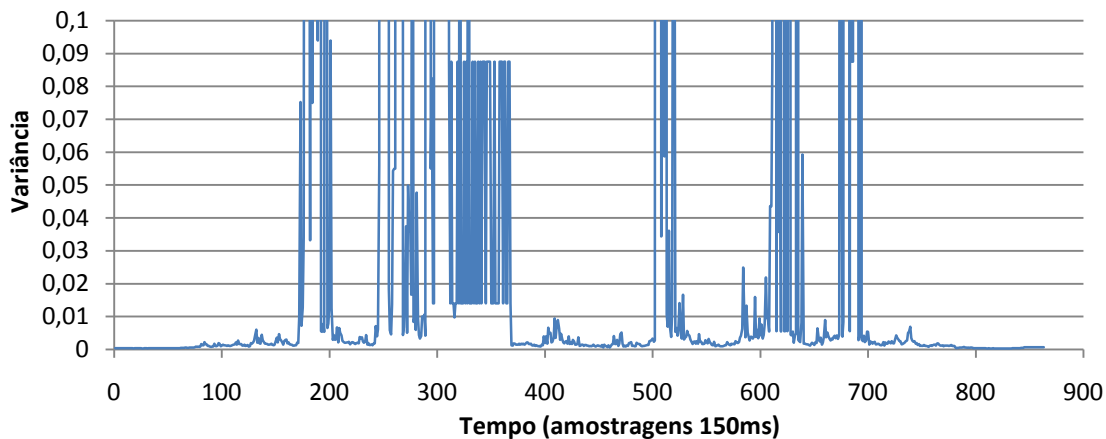


Figura 6.35 Variância da estimativa da posição segundo o eixo x

A variância da estimativa segundo o eixo xx, Figura 6.36, é em tudo idêntica à obtida anteriormente, com a salvaguarda que, nas zonas onde o *laser range finder* apenas consegue medir paredes lisas, o aumento da variância é mais significativo. Veja-se as amostragens 180, 235, 600 e 670, bem como quando o robô roda sobre si próprio no fim do corredor, amostragem 500. Note-se agora que, entre a amostragem 235 e a 370, a variância é muito elevada, durante este tempo, sendo reduzida repentinamente neste último. Isto deve-se à localização do robô se ter perdido nesta zona, mantendo sempre uma elevada incerteza na sua posição. Assim que encontrou uma localização o algoritmo relocalizou o robô, nessa posição, o que se vê pela menor variância obtida.

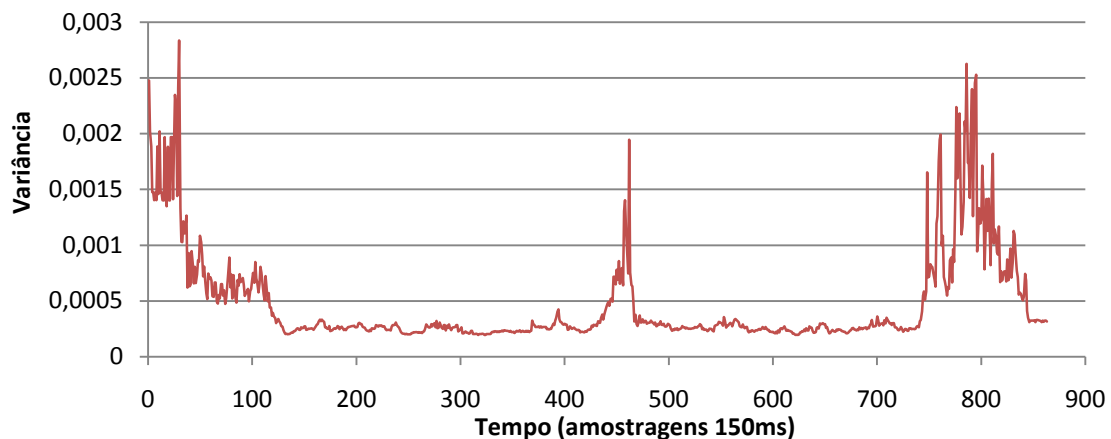


Figura 6.36: Variância da estimativa da posição segundo o eixo y

Para a variância da estimativa da posição segundo o eixo yy, Figura 6.37, e para a variância da estimativa da orientação, Figura 6.38, o gráfico mantém-se idêntico ao obtido para o caso anterior, salvo o pequeno aumento dos valores em jogo. Assim, demonstra-se que, mesmo com poucos pontos, o algoritmo se comportou bem na estimação da posição e da orientação do robô, em especial esta última, Note-se que é pois mais importante uma boa estrutura dos pontos medidos do que a sua quantidade.

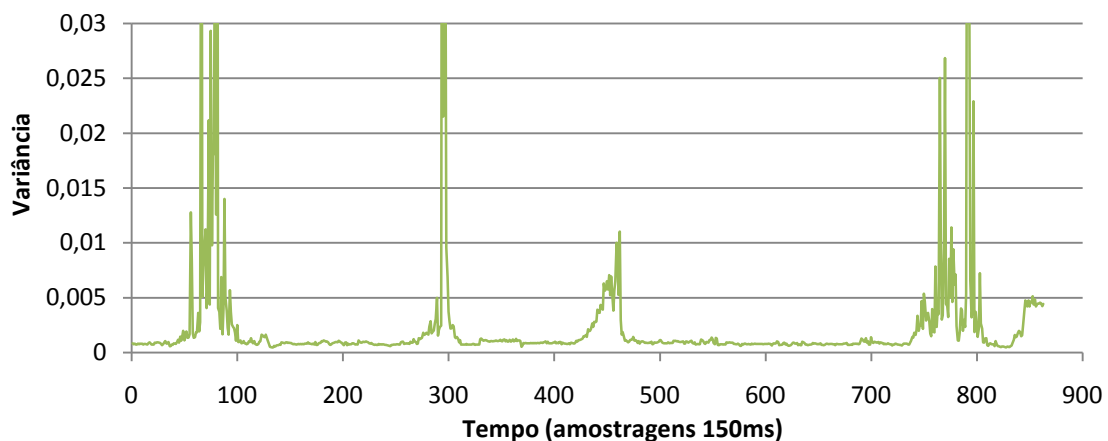


Figura 6.37: Variância da estimativa da orientação do robô

Por fim o erro, Figura 6.39, demonstra, de novo, o bom comportamento do algoritmo de localização. Tal como no caso anterior, Figura 6.31, o erro mantém-se baixo e, de certa forma, constante, com exceção da falha de localização na amostragem 300. Nesta, como se pode verificar pelo gráfico da Figura 6.39, o erro aumentou em relação ao seu valor normal. Este aumento foi o suficiente para que permitisse à localização global inserir a sua estimativa, amostragem 370.

Como o valor do erro se mantém aproximadamente constante ao longo de todo o percurso, e a localização global usa esse mesmo erro para decidir entre relocar, ou não, o robô noutra local, quando o erro aumenta muito, em relação ao normal, as hipóteses da relocação também aumentam, pois, assim que se entra numa zona

característica, o algoritmo é capaz de obter uma estimativa melhor que a actual, com erro menor que o actual, e assim relocalizar o robô.

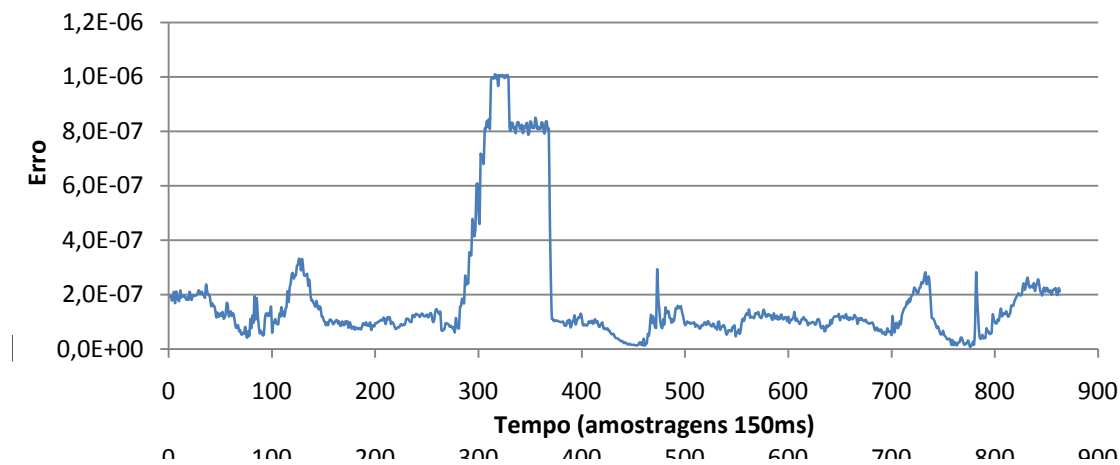


Figura 6.38: Variação do erro da estimativa ao longo do percurso efectuado pelo robô

Figura 6.39: Variação do valor da função custo da estimativa ao longo do percurso efectuado pelo robô

6.4.3 Resultados usando 12 pontos de medida

Para este teste não são apresentados resultados pois, efectuando o mesmo percurso, dos dois casos anteriores, o algoritmo não foi capaz de localizar o robô, tendo um comportamento semelhante ao que acontece com a utilização dos sensores de infravermelhos. Isto prova, uma vez mais, que a quantidade de medidas é essencial, desde que bem estruturadas. Com tão pequena quantidade de pontos, o algoritmo não consegue ter a percepção do movimento do robô, pelo que, para este caso, seria necessário a odometria, para inserção no cálculo da estimativa.

Uma outra situação identificada foi que, se as medidas forem feitas apenas numa direcção (apenas num raio de 90° por exemplo), embora em quantidade, não têm uma estrutura geral boa, e o algoritmo tem dificuldade em localizar o robô usando apenas esta informação. Seria um caso comparável a um corredor em que num dos lados teríamos uma parede lisa, e no outro uma parede com obstáculos/objectos (portas, janelas). Usando apenas o laser, em toda a sua amplitude (240°), e mesmo que a quantidade de pontos fosse grande, a localização não seria suficientemente exacta, sendo necessário integrar a odometria do robô para que se tivesse uma noção do deslocamento segundo o eixo paralelo à parede.

Capítulo 7

Conclusões e Trabalhos Futuros

Nesta dissertação estudou-se e implementou-se o algoritmo proposto por [M. Lauer \[11\]](#). Este é um algoritmo do tipo de correspondência de mapas o qual demonstrou já ter uma boa precisão, ser robusto e também eficiente em termos computacionais.

No futebol robótico o algoritmo mostrou ser robusto e eficiente, dentro do pretendido, principalmente em termos computacionais, pois, neste sistema, os tempos de processamento são críticos, já que a frequência de amostragem é de 25Hz, e o facto de se conseguir efectuar todo o processamento em cerca de 4ms demonstra uma grande eficiência, quando comparado com algoritmos concorrentes em que no mínimo, se consegue efectuar todo o processamento em cerca de 14ms. Tal, é o caso do Filtro de Partículas, com 400 partículas. O facto de o campo de futebol robótico ter uma estrutura bem definida e do sensor usado permitir uma grande quantidade de medidas, permitiu ao algoritmo localizar correctamente o robô, de certa forma exacta e precisa.

Demonstrou-se que quando a qualidade da informação é boa, ou seja, as medidas contêm pouco erro é possível localizar correctamente o robô mesmo quando a quantidade da informação é pouca ou é adquirida pouca informação útil. Por outro lado, quando a quantidade da informação é elevada, mesmo que nessa, uma certa percentagem da informação contenha elevado número de *outliers*, o algoritmo consegue compensar isso e localizar correctamente o robô.

Para os robôs usando sensores de infra-vermelhos e sonares, demonstrou-se que é inevitável usar a informação da odometria para que a localização do robô seja robusta e suave. A utilização do sensor laser, devido às suas qualidades, demonstrou que se o número de pontos medidos for razoável, o algoritmo por si só é capaz de localizar continuamente o robô sem ser necessária a informação do movimento dada pela odometria, embora esta venha garantir que o robô nunca se perde e melhorar a resolução da localização que assim pode ser inferior ao tamanho da célula utilizado, 4cm. Por outro lado, o laser desmontou também como uma baixa quantidade de pontos é crítica para o algoritmo de localização, não sendo possível a este localizar correctamente o robô sem ter a informação do movimento do robô dada pela odometria.

Os resultados demonstraram ainda que o algoritmo em questão pode ser aplicado aos mais variados tipos de robôs, desde que estes tenham sensores capazes de fornecer a informação necessária para o seu bom funcionamento, ou seja, medidas de distâncias, aos objectos identificados no mapa que este usa.

No entanto, existem vários pormenores que podem melhorar o funcionamento do algoritmo que podem ainda ser implementados. Estes não têm directamente a ver com o funcionamento do algoritmo, mas, sim, com o tratamento e selecção da informação sensorial. Seria útil limitar as distâncias medidas erradamente, os *outliers*, considerando, por exemplo, uma faixa de medida aceitável na zona onde é espectável existir um objecto visível pelos sensores, devidamente identificado no mapa. Isto apenas para o cálculo da estimativa da localização do robô. Por outro lado, é útil usar essa informação para a localização global, pois o facto de se estar a desconsiderar medidas num local pode dever-se ao robô se encontrar mal localizado. Assim, essas medidas podem ser verdadeiras, noutra localização, devendo essa decisão ser tomada em função da variância da estimativa actual.

O teste do uso de sonares é também essencial. Embora o algoritmo funcione usando a informação dos sonares, é necessário proceder a uma colocação correcta destes no robô, bem como trabalhar no seu ciclo de leitura, para evitar interferências nas medidas uns dos outros, devido ao seu largo cone de medida. Só assim é possível obter uma boa estrutura de medidas, o que é essencial para o algoritmo.

Será também interessante efectuar todos os testes, não realizados ainda, com outros robôs e usando a globalidade dos sensores disponíveis. Por exemplo, integrar o algoritmo no software de controlo da cadeira de rodas autónoma, por esta ter um tipo de tracção diferente dos robôs utilizados, bem como por a localização dos sensores nesta ser diferente dos do *Medical Robot*. Estes não foram efectuados durante este trabalho, por indisponibilidade dos robôs e por limitações em termos de tempo.

Será ainda importante testar o funcionamento do algoritmo nos robôs de forma autónoma, teste que não foi possível fazer, ou seja, definir um percurso para o robô e verificar se este, usando o algoritmo para se localizar, é capaz de efectuar o percurso suave e correctamente.

No algoritmo seria interessante verificar a resposta da localização global tendo em conta outros factores de decisão além da função custo.

Por fim implementar comunicação por UDP na interface do algoritmo para não ser necessário integrar o código no controlador de robô.

Referências

- [1] J., S. Esteves., "Metodologia de auto-localização absoluta em ambientes quase-estruturados." *Tese de doutoramento em Engenharia Electrónica Industrial, Universidade do Minho*. 2005.
- [2] C., Cohen e F., Kross., "A comprehensive Study oh three object Triangulation." *Mobile Robots VII, SPIE*. 1992. Vol. 1831.
- [3] M., Betke e L., Gurvits., "Mobile robot localization using landmarks." *IEEE Transactions on Robotics and Automation*. 1997. Vol. 13.
- [4] J., S. Esteves, A., Carvalho e C., Couto., "Generalized geometric triangulation algorithm for mobile robot absolute self-localization." *In IEEE International Symposium On Industrial Electronics (ISIE)*. Rio de JAneiro : s.n., 2003.
- [5] A., Sousa, et al., "Self localization of an Autonomous Robot: Using an EKF to merge Odometry and vision based landmarks." *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation*. 2005. pp. 227-234.
- [6] S., Thrun, W., Burgard e D., Fox., "Probabilistic Robotics." *The MIT Press*. 2005.
- [7] J., J. Leonard e H., F. Durrant-Whyte., "Directed Sonar Sensing for Mobile Robot Navigation." *Kluwer Academic*. Boston : s.n., 1992.
- [8] R., Negenborn., "Robot localization and Kalman filters on finding your position in a noisy world." *Master's Thesis, UTRECHT UNIVERSITY, Institute of Information and Computing Sciences*. 2003.
- [9] D., Cabecinhas, et al., "Self-Localization based on Kalman Filter and Monte Carlo Fusion of Odometry and Visual Information." *Robótica*. 2006.
- [10] D., Fox, et al., "Bayesian Filters for location estimation." *Pervasive Computing, IEEE*. 2003.
- [11] M., Lauer, S., Lange and M., Riedmiller., "Calculating the perfect match: an efficient and accurate approach for robot self-localization." *Robocup 2005*. 2005.

[12] I., Rekleitis., "A particle filter tutorial for mobile robot localization." *Technical Report TR-CIM-04-02*. Quebec, Montreal, Canada : Centre for Intelligent Machines, McGill University, 2004.

[13] F., G. Bravo, A., Vale e M., I. Ribeiro., "Navigation Strategies for Cooperative Localization based on Particle Filters." [ed.] IOS Press. *Integrated Computer-Aided Engineering*. 2007. Vol. 14:3, pp. 263-279.

[14] C., Penedo, et al., "Markov Localization in the Robocup Simulation League." *Robótica, 4º trimestre*. 2003. pp. 16-21.

[15] N., Karlsson, et al., "The vSLAM Algorithm for Robust Localization and Mapping." *In international Conference on Robotics and Automation*. ICRA 2005.

[16] X., Yun, K., Latt e J., S. Glennon., "Mobile robot localization using the Hough Transform and neural networks." *Proceedings of the 1998 IEEE International Symposium*. 1998.

[17] M., Riedmiller e Braun, H., "A direct adaptative method for faster backpropagation learning: the RPROP algorithm." *In Proceedings of the IEEE International Conference on Neural Networks*. 1993. pp. 586-591.

[18] Tribots, Brainstormers., "Homepage da Equipa de Futebol Robótico Brainstormers Tribots." [Online] [Citação: 15 de Julho de 2008.] <http://www.ni.uos.de/index.php?id=2>.

[19] A., Merke, S., Welker and M., Riedmiller., "Line based robot localization under natural light conditons." *In ECAI 2004 Workshop on Agents in Dynamic and Real Time Environments*. 2004.

[20] F., Dellaert, W., Burgard e S., Thrun., "Monte Carlo localisation for mobile robots." *In IEEE internacional Conference on Robotics and Automation (ICRA99)*. 1999.

[21] , Robot Sonars. *Society of Robots*. [Online] [Citação: 20 de Junho de 2008.] http://www.societyofrobots.com/sensors_sonar.shtml.

[22] IR Range Finder. *Society of Robots*. [Online] [Citação: 20 de Junho de 2008.] http://www.societyofrobots.com/sensors_sharpirrange.shtml.

[23] Visão geral sobre o laser range finder Hokuyo URG-04LX. [Online] [Citação: 20 de Junho de 2008.] <http://www.acroname.com/robotics/info/articles/laser/laser.html>.

[24] Hokuyo., Informação Hokuyo sobre o laser range finder Hokuyo URG-04LX. [Online] [Citação: 20 de Junho de 2008.] <http://www.hokuyo-aut.jp/02sensor/07scanner/urg.html>.

[25] , Sobel Operator. *Wikipédia*. [Online] [Citação: 15 de Julho de 2008.] http://en.wikipedia.org/wiki/Sobel_operator.

[26] T., Costa., "Medical Robot." *Tese de mestrado em Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto*. 2008.

[27] Committee, MSL Technical., "Middle Size Robot League Rules and Regulations for 2008." 2007.

[28] D., Fox, et al., "Particle Filters for mobile robot localization." *In Sequential Monte Carlo Methods in Practice*. 2000. pp. 470-498.

[29] , Visão geral sobre o Devantech Sonic Range Finder. [Online] [Citação: 20 de Junho de 2008.] <http://www.acroname.com/robotics/articles/devantech/srf.html>.

[30] Sharp IR rangers information. [Online] [Citação: 20 de Junho de 2008.] <http://www.acroname.com/robotics/info/articles/sharp/sharp.html>.

Anexo A

Funções do Algoritmo

A.1 Minimização da função custo

```
//Minimização do erro: sabendo a localização anterior estimada do robô
//(x,y,phi) e a lista de pontos medidos pelos sensores
// calcula o erro da nova estimativa e as primeiras derivadas da estimativa da
//localização

procedure MinimizeError(out PointErr:double; out dx:double; out dy:double; out
dphi:double; x, y, phi:double; vis:PointList; max_points:longword);
var
  nPoints:longword ;
  vis_iterator: longword;
  vp, ddistdpos: TPos;
  dist, derrddist: double;
begin
  //nº de pontos adquiridos
  if (round(Length(vis)) < max_points) then
    nPoints:=Length(vis)
  else
    nPoints:=max_points;

  PointErr:=0;
  dx:=0;
  dy:=0;
  dphi:=0;

  //para todos os pontos visiveis
  for vis_iterator:=low(vis) to nPoints -1 do
    begin
      //calcula a posição do ponto no eixo de coordenadas global
      vp.x:=x + cos(phi)*vis[vis_iterator].x - sin(phi)*vis[vis_iterator].y;
      vp.y:=y + sin(phi)*vis[vis_iterator].x + cos(phi)*vis[vis_iterator].y;

      //verifica qual a distância correspondente no mapa para esse ponto
      if (vp.x >=0) and (vp.y>=0) then
        dist:=Distance(vp, MapDist);

      //calcula a função custo, somatório do custo de todos os pontos
      PointErr:= (PointErr + (1 - (Lc*Lc/(Lc*Lc+dist*dist))));

      //calculo da derivada da função custo em ordem à distância
      derrddist:=(2*Lc*Lc*dist)/((Lc*Lc+dist*dist)*(Lc*Lc+dist*dist));
```

```

//calculo da derivada distância em ordem á posição, usando o gradiente do
mapa
ddistdpos:=Gradient(vp, MapGrad);

//calculo do somatório das derivadas de todos os pontos
dx:=dx+derrddist*ddistdpos.x;
dy:=dy+derrddist*ddistdpos.y;
dphi:=dphi+derrddist*(ddistdpos.x*(-sin(phi)*vis[vis_iterator].x-
cos(phi)*vis[vis_iterator].y)+ddistdpos.y*(cos(phi)*vis[vis_iterator].x-
sin(phi)*vis[vis_iterator].y));
end;
end;

```

A.2 Optimização da estimativa da localização

```

//Optimização da estimativa da localização: para a lista de pontos medidos
pelos sensores
//calcula a convergência da estimativa da localização do robô
function Optimize(out xy:TPos;out phi:double; vis:PointList; niter:longword;
max_points: longword;stepwidthxy:double; stepwidthphi:double):double;
var
  parami: array [0..2] of double;
  gradi: array [0..2] of double;
  latest_grad: array [0..2] of double;
  PointErr:double;
  stepwidth: array [0..2] of double;
  i, j:longword;
begin
  //inicialização de parâmetros
  parami[0]:=xy.x;
  parami[1]:=xy.y;
  parami[2]:=phi;
  latest_grad[0]:=0;
  latest_grad[1]:=0;
  latest_grad[2]:=0;
  stepwidth[0]:=stepwidthxy;
  stepwidth[1]:=stepwidthxy;
  stepwidth[2]:=stepwidthphi;

  //para todos os pontos visíveis efectua as niter(n iterações) do algoritmo
  RPROP
  for i := 0 to niter do
    begin
      //calcula o erro o gradiente para a estimativa actual, ou seja, usando a
      posição estimada e a informação dos sensores adquirida
      MinimizeError(PointErr,gradi[0],gradi[1], gradi[2],
      parami[0],parami[1],parami[2], vis, max_points);

      //aplica o algoritmo RPROP para cada componente da posição, x, y e
      orientação phi
      for j := 0 to 2 do
        begin
          //Se a derivada é nula não actualize os passos, caso contrário
          actualiza-os
          if (gradi[j]<>0) then begin
            //derivada manteve o sinal, aumenta o passo
            if (gradi[j]*latest_grad[j]>0) then begin
              stepwidth[j]:= stepwidth[j]*1.2;
            //derivada mudou de sinal, diminui o peso
            end else if (gradi[j]*latest_grad[j]<0) then begin
              stepwidth[j]:= stepwidth[j]*0.5;
            end;

            //derivada positiva, o erro aumentou, está a ir no sentido inverso,
            if (gradi[j] > 0) then begin
              //inverte o sentido, ou seja, subtrai actualização

```

```

        parami[j]:=parami[j]-stepwidth[j];
        //derivada negative, erro diminuiu, está a ir no sentido correcto
    end else if (gradi[j]<0) then begin
        //continua, soma actualização
        parami[j]:=parami[j]+stepwidth[j];
    end;
end;
//actualize a derivada
latest_grad[j]:=gradi[j];
end;
end;
//actualiza a estimativa actual
xy.x:=parami[0];
xy.y:=parami[1];
phi:=NormalizeAngle(parami[2]);

result:=PointErr/max_points;
end;

```

A.3 Cálculo da segunda derivada e da variância a estimativa da localização

```

//Calculo da segunda derivada e variância da estimativa da localização
procedure SecondDerivate(out d2xy: TPos; out d2phi:double; xy:TPos;
phi:double; vis:PointList; max_points: longword);
var
    PointErr, dErr, d2Err, PointDist:double;
    n_points: longword;
    dDist:TPos;
    vp, dposdphi, d2posdphi: TPos;
    i: longword;
begin
    PointErr:=0;
    d2xy.x:=0;
    d2xy.y:=0;
    dp2hi:=0;
    //verifica o numero de pontos
    if (round(max_points)> Length(vis)) then begin
        n_points:=Length(vis);
    end else begin
        n_points:=max_points;
    end;
    //para todos os pontos medidos pelos sensores
    for i:=low(vis) to n_points - 1 do begin
        //calcula a posição dos pontos no eixo global de coordenadas
        vp.x:=xy.x + cos(phi)*vis[i].x-sin(phi)*vis[i].y;
        vp.y:=xy.y + sin(phi)*vis[i].x+cos(phi)*vis[i].y;
        //obtem a distância correspondente ao ponto
        PointDist:=Distance(vp, MapDist);
        //calcula o erro
        PointErr:= (PointErr + ( 1 - (Lc*Lc/(Lc*Lc+PointDist*PointDist))));
        //se o erro for aceitável, calcula a segunda derivada da estimativa da
        localização
        if (PointDist<2*Lc) then
            begin
                dErr:=PointDist/(Lc*Lc);
                d2err:=1/(Lc*Lc);
                ddist:=Gradient(vp, MapGrad);
                dposdphi.x:=-sin(phi)*vis[i].x-cos(phi)*vis[i].y;
                dposdphi.y:= cos(phi)*vis[i].x-sin(phi)*vis[i].y;
                d2posdphi.x:=-cos(phi)*vis[i].x+sin(phi)*vis[i].y;
                d2posdphi.y:=-sin(phi)*vis[i].x-cos(phi)*vis[i].y;

                //cálculo das segundas derivadas da função custo em ordem à posição
                d2xy.x:=hd2xy.x + d2err*ddist.x*ddist.x;

```

```

        d2xy.y:=hd2xy.y + d2err*ddist.y*ddist.y;
        d2phi:=d2phi+d2err*(ddist.x*dposdphi.x+ddist.y*dposdphi.y)*
(ddist.x*dposdphi.x+ddist.y*dposdphi.y)+
derr*(ddist.x*d2posdphi.x+ddist.y*d2posdphi.y);
    end;
end;

//cálculo das variâncias
//VAR_CONST=1.4e-10;
//calcula a covariância
if d2xy.x <> 0 then begin
    d2xy.x:=abs((1/d2xy.x)*VAR_CONST);
    if d2xy.x > MAX_VAR then
        d2xy.x:=MAX_VAR;
    end else begin
        d2xy.x:=MAX_VAR;
    end;
end;

if d2xy.y <> 0 then begin
    d2xy.y:=abs((1/d2xy.y)*VAR_CONST);
    if d2xy.y > MAX_VAR then
        d2xy.y:=MAX_VAR;
    end else begin
        d2xy.y:=MAX_VAR;
    end;
end;

if d2phi <>0 then begin
    d2phi:=abs((1/d2phi)*VAR_CONST*10);
    if d2phi > MAX_VAR then
        d2phi:=MAX_VAR;
    end else begin
        d2phi:=MAX_VAR;
    end;
end;
end;

```

A.4 Fusão da estimativa da localização com a odometria

```

//Fusão das estimativas
procedure EstimateFusion(var RobotPos: TPos; OdosDeltaPos:TPos;
SensorPos:TPos; var VarPos:TPos; VarSensor :TPos; SensoresON:boolean);
var
    odoserror: TPos;
begin
    //faz um rotação da variação da odometria para que esta fique em relação ao
    eixo global de coordenadas e não em relação ao eixo do robô
    rotatepos(OdosDeltaPos,RobotPos.phi);

    //calcula a imprecisão do movimento do robô, dado pela odometria
    odoserror.x:=min(0.1*abs(OdosDeltaPos.x),0.1);
    odoserror.y:=min(0.1*abs(OdosDeltaPos.y),0.1);
    odoserror.phi:=min(0.1*abs(OdosDeltaPos.phi),0.1);

    //actualiza a variância global do robô com a imprecisão da odometria
    VarPos.x:=VarPos.x + odoserror.x* odoserror.x;
    VarPos.y:=VarPos.y + odoserror.y* odoserror.y;
    VarPos.phi:=VarPos.phi + odoserror.phi* odoserror.phi;

    //actualiza a localização do robô e a variância das estimativas pesando a
    estimativa da localização com a odometria

    RobotPos.x:=(VarSensorP.x*RobotPos.x+VarPos.x*SensorPos.x)/(VarSensor.x+VarPos
.x);
    VarPos.x:=(VarPos.x*VarSensor.x)/(VarPos.x+VarSensor.x);

    RobotPos.y:=(VarSensor.y*RobotPos.y+VarPos.y*SensorPos.y)/(VarSensorP.y+VarPos
.y);

```

```

VarPos.y:=(VarPos.y*VarSensor.y)/(VarPos.y+VarSensor.y);

RobotPos.phi:=NormalizeAngle(VarSensor.phi*RobotPos.phi+VarPos.phi*SensorPos.p
hi)/(VarSensor.phi+VarPos.phi);
VarPos.phi:=(VarPos.phi*VarSensor.phi)/(VarPos.phi+VarSensor.phi);
end;
end;

```

A.5 Localização global

```

function EstimatePosition(LocalForm:TFLocalization;var rxy:TPos; var
rphi:double; visList:PointList; contador:integer; nposes:integer): double;
var
  xy: TPos;
  i: Integer;
  phi,err_opt, min_err, rerr: double;
  step1,step2,step3,step4:double;
begin
  Randomize;

  //localização actual
  //calcula a estimativa da localização actual
  err_opt:=Optimize(rxy,rphi,visList,10,contador,step1,stp2);
  rerr:=err_opt;

  //loalização global
  //cria lista de posições candidatas aleatória
  for i:=0 to nposes-2 do begin
    cand_pose[i].x:=Random(MapAreaW)*cell_size;
    cand_pose[i].y:=Random(MapAreaH)*cell_size;
    cand_pose[i].phi:=Random(round(pi*2));
  end;

  //para todas as posições incluindo a bestpose calcula a respectiva
  estimativa da localização
  for i:=0 to nposes-1 do begin
    xy.x:=cand_pose[i].x;
    xy.y:=cand_pose[i].y;
    phi:=cand_pose[i].phi;
    err_opt:=Optimize(xy,phi,visList,10,contador,t3,t4);

    //se a optimização colocar o robô fora do mapa a pose é descartada
    if ((xy.x > MapAreaW*cell_size-1) or (xy.y > MapAreaH*cell_size-1) or
(xy.x < 0) or (xy.y<0)) then begin
      cand_pose[i].PointError:=1;

      //guarda a optimização das posições e o respectivo custo
    end else begin
      cand_pose[i].x:=xy.x;
      cand_pose[i].y:=xy.y;
      cand_pose[i].phi:=phi;
      //se o angulo variar mais de 80 graus a posição é descartada
      if Abs(DiffAngle(rphi,cand_pose[i].phi)) > degtorad(80) then begin
        cand_pose[i].PointError:=err_opt+1;
      end else begin
        cand_pose[i].PointError:=err_opt;
      end;
    end;
  end;

  //considera-se que no fim do vector fica sempre a posição com menor custo ,
  a bestpose(este valor nunca é criado aleatoriamente)
  min_err:=cand_pose[nposes-1].PointError;
  best_pose_index:=nposes-1;

  //verifica se o erro das novas posições é menor que o da bestpose

```

```

for i:=0 to nposes-2 do begin
  if (cand_pose[i].PointError <= min_err*1) then begin
    min_err:=cand_pose[i].PointError;
    best_pose_index:=i;
    best_cand_pose_counter:=0;
  end;
end;

//actualiza a bestpose com a melhor posição aleatória
cand_pose[nposes-1].x:=cand_pose[best_pose_index].x;
cand_pose[nposes-1].y:=cand_pose[best_pose_index].y;
cand_pose[nposes-1].phi:=cand_pose[best_pose_index].phi;
cand_pose[nposes-1].PointError:=cand_pose[best_pose_index].PointError;

//se a bestpose for 5% melhor, 5 vezes consecutivas, que a posição actual o
algoritmo auto-localiza o robô nessa posição
if cand_pose[nposes-1].PointError <= rerr*0.95 then begin
  best_cand_pose_counter:=best_cand_pose_counter+1;
end else begin
  best_cand_pose_counter:=0;
end;

//actualiza a localização do robô para a bestpose
if best_cand_pose_counter > 5 then begin
  best_cand_pose_counter:=0;
  rxy.x:=cand_pose[nposes-1].x;
  rxy.y:=cand_pose[nposes-1].y;
  rphi:=cand_pose[nposes-1].phi;
  rerr:=cand_pose[nposes-1].PointError;
end;
result:=rerr;
end;

```

A.6 Leitura de distância no mapa

```

function Distance(p:TPos; var Dist:TMapDist):double;
var
  xi, yi:integer;
begin
  xi:=round(p.x/cell_size);
  yi:=round(p.y/cell_size);
  //se o ponto se encontra dentro do mapa lê a distância no limite do mapa
  if (xi<0) then xi:=0;
  if (yi<0) then yi:=0;
  if (xi>=MapAreaW-1) then xi:=MapAreaW-1;
  if (yi>=MapAreaH-1) then yi:=MapAreaH-1;
  Result:=Dist[yi,xi];
end;

```

A.7 Leitura do gradiente

```

function Gradient(p:TPos; var Grad:TMapGrad):TPos;
var
  xi, yi:integer;
begin
  xi:=round(p.x/cell_size);
  yi:=round(p.y/cell_size);
  //se o ponto estiver fora do mapa lê o gradiente nos limites do mapa
  if xi<0 then xi:=0;
  if yi<0 then yi:=0;
  if xi>=MapAreaW-1 then xi:=MapAreaW-1;
  if yi>=MapAreaH-1 then yi:=MapAreaH-1;
  Result:=Grad[yi,xi];
end;

```

Anexo B

Funções de Mapeamento

B.1 Tratamento das Linhas

```
procedure DrawLineSegment(linepoints:TPointsLine;LocalForm: TFMapping);
var
  xi,yi: longword;
  p1,p2,p:TPos;
  tv, dist_p:double;
  ang: double;
begin
  p1.x:=linepoints.x1;
  p1.y:=linepoints.y1;
  p2.x:=linepoints.x2;
  p2.y:=linepoints.y2;
  ang:=NormalizeAngle(arctan2(p2.y-p1.y, p2.x-p1.x));
  //rotação dos pontos da linha
  p1:=rotatepos(p1,ang);
  p2:=rotatepos(p2,ang);

  //para todos as células do mapa calcula a distância á linha a "desenhar"
  for yi := 0 to MapAreaH-1 do begin
    for xi := 0 to MapAreaW-1 do begin
      p.x:=xi*cell_size;
      p.y:=yi*cell_size;

      //rotação do ponto em estudo
      p:=rotatepos(pp,ang);

      //calcula da distância do ponto à linha
      if(p1.x<p2.x) then begin
        if (p.x>p1.x) and p.x<p2.x then
          dist_p:=abs(p.y-p1.y)
        else if (p.x<p1.x) and (p.x<p2.x) then
          dist_p:=sqrt((p1.x-pp.x)*(p1.x-pp.x)+(p1.y-pp.y)*(p1.y-pp.y))
        else if (p.x>p1.x) and (p.x>p2.x) then
          dist_p:=sqrt((p2.x-pp.x)*(p2.x-pp.x)+(p2.y-pp.y)*(p2.y-pp.y));
      end else begin
        if (p.x<p1.x) and p.x>p2.x then
          dist_p:=abs(p.y-p1.y)
        else if (p.x<p1.x) and (p.x<p2.x) then
          dist_p:=sqrt((p2.x-pp.x)*(p2.x-pp.x)+(p2.y-pp.y)*(p2.y-pp.y))
        else if (p.x>p1.x) and (p.x>p2.x) then
          dist_p:=sqrt((p1.x-pp.x)*(p1.x-pp.x)+(p1.y-pp.y)*(p1.y-pp.y));
      end;
    end;
  end;
```

```

        //actualiza o mapa de distâncias mínimas "d"
        UpdateDist(xi,yi,dist_p);
    end;
end;
end;

```

B.2 Tratamento dos Arcos

```

procedure DrawArc(points:TPointsLine;LocalForm: TFMMapping);
var
    xi,yi: longword;
    pp, v:Tpos;
    angulo, dist_pp, d1, d2:double;
    radius:integer;
    center:TPos;
    tetastart, tetaend:double;
begin
    tetastart:=normalizeangle(points.tetastart);
    tetaend:=tetastart+points.tetaend;
    //percorre todas as células e calcula as distâncias em relação ao arco
    for yi := 0 to MapAreaH-1 do begin
        for xi := 0 to MapAreaW-1 do begin
            pp.x:=xi*cell_size;
            pp.y:=yi*cell_size;

            //calcula o ângulo ao centro do arco
            v.x:=pp.x-points.center.x;
            v.y:=pp.y-points.center.y;
            angulo:=radtodeg(arctan2(v.y,v.x));
            angulo:=normalizeangle(angulo);

            //se o angulo estiver dentro dos limites do arco
            if ((angulo>tetastart) and (angulo<(tetaend))) then begin
                dist_pp:=abs(sqrt(v.x*v.x+v.y*v.y)-points.radius);
            end else begin

                //se estiver fora aproxima á extremidade do arco mais próxima
                d1:=sqrt((pp.x-points.center.x-
                    points.radius*cos(degtorad(tetastart)))*(pp.x-points.center.x-
                    points.radius*cos(degtorad(tetastart)))+(pp.y-points.center.y-
                    points.radius*sin(degtorad(tetastart)))*(pp.y-points.center.y-
                    points.radius*sin(degtorad(tetastart))));
                d2:=sqrt((pp.x-points.center.x-
                    points.radius*cos(degtorad(tetaend)))*(pp.x-points.center.x-
                    points.radius*cos(degtorad(tetaend)))+(pp.y-points.center.y-
                    points.radius*sin(degtorad(tetaend)))*(pp.y-points.center.y-
                    points.radius*sin(degtorad(tetaend))));
                if (d1<d2) then
                    dist_pp:=d1
                else
                    dist_pp:=d2;
            end;

            //actualiza o mapa de distâncias
            UpdateDist(xi,yi,dist_pp);
        end;
    end;
end;

```

B.3 Atualização das distâncias no mapa

```

procedure UpdateDist(xi, yi:longword; v: double);
begin
  //se o ponto se encontra dentro do mapa, atualiza valor
  if ((yi<=MapAreaH-1) and (xi<=MapAreaW-1)) then
    begin
      if(v<MapDist[yi,xi]) then
        MapDist[yi,xi]:=v;
      end;
    end;
end;

```

B.4 Cálculo do Gradiente

```

procedure CalcGradient(var Dist: TMapDist;var Grad: TMapGrad);
var
  xi, yi:longword;
  lo, lm, lu, ro, rm, ru, mo, mu:double;
begin
  for yi:= 1 to MapAreaH-2 do begin
    for xi := 1 to MapAreaW-2 do begin
      Grad[yi,xi].x:=0;
      Grad[yi,xi].y:=0;
    end;
  end;

  //aplicação do filtro de Sobel
  for yi:= 1 to MapAreaH-2 do begin
    for xi := 1 to MapAreaW-2 do begin
      //Cálculos preliminares dos pontos da matriz 3x3 que envolve o ponto
      lo:=Dist[yi-1,xi-1];
      lm:=Dist[yi,xi-1];
      lu:=Dist[yi+1,xi-1];

      ro:=Dist[yi-1,xi+1];
      rm:=Dist[yi,xi+1];
      ru:=Dist[yi+1,xi+1];

      mo:=Dist[yi-1,xi];
      mu:=Dist[yi+1,xi];

      //cálculo do gradiente de x e de y dentro do campo usando o filtro de
      Sobel
      Grad[yi,xi].x:=0.125*(-lo+ro-2*lm+2*rm-lu+ru);
      Grad[yi,xi].y:=0.125*(-lo+lu-2*mo+2*mu-ro+ru);
    end;
  end;

  //como nas linhas laterais não é possível aplicar o filtro de Sobel (por não
  existir uma matriz 3x3 em torno do ponto)

  //cálculo do gradiente para as linhas laterais
  for yi := 1 to MapAreaH-2 do begin
    Grad[yi,0].x:=Grad[yi,1].x;
    Grad[yi,0].y:=Grad[yi,1].y;
    Grad[yi,MapAreaW-1].x:=Grad[yi,MapAreaW-2].x;
    Grad[yi,MapAreaW-1].y:=Grad[yi,MapAreaW-2].y;
  end;

  //cálculo do gradiente para as linhas de fundo
  for xi := 1 to MapAreaW-2 do begin
    Grad[0,xi].x:=Grad[1,xi].x;
    Grad[0,xi].y:=Grad[1,xi].y;
    Grad[MapAreaH-1,xi].x:=Grad[MapAreaH-2,xi].x;
    Grad[MapAreaH-1,xi].y:=Grad[MapAreaH-2,xi].y;
  end;
end;

```

```

end;

//cálculo do gradiente para os cantos
Grad[0,0].x:=0.5*(Grad[0,1].x+Grad[1,0].x);
Grad[0,0].y:=0.5*(Grad[0,1].y+Grad[1,0].y);
Grad[MapAreaH-1,0].x:=0.5*(Grad[MapAreaH-1,1].x+Grad[MapAreaH-2,0].x);
Grad[MapAreaH-1,0].y:=0.5*(Grad[MapAreaH-1,1].y+Grad[MapAreaH-2,0].y);
Grad[0,MapAreaW-1].x:=0.5*(Grad[0,MapAreaW-2].x+Grad[1,MapAreaW-2].x);
Grad[0,MapAreaW-1].y:=0.5*(Grad[0,MapAreaW-2].y+Grad[1,MapAreaW-1].y);
Grad[MapAreaH-1,MapAreaW-1].x:=0.5*(Grad[MapAreaH-1,MapAreaW-
2].x+Grad[MapAreaH-2,MapAreaW-1].x);
Grad[MapAreaH-1,MapAreaW-1].y:=0.5*(Grad[MapAreaH-1,MapAreaW-
2].y+Grad[MapAreaH-2,MapAreaW-1].y);
end;

```

B.5 Desenho do mapa em forma de imagem

```

procedure DrawArrayDist(var TheArray: TMapDist; SizeW,
SizeH:integer;Image:TImage);
var
  SrcIntfImg: TLazIntfImage;
  ImgHandle,ImgMaskHandle: HBitmap;
  SrcBitmap: TBitmap;
  vi, ui, gray: integer;
  yi,xi:integer;
  fpcol: TFPcolor;
  min_val:double;
begin
  TempBitmap:=TBitmap.Create;
  SrcBitmap:=TBitmap.Create;
  SrcIntfImg:=TLazIntfImage.Create(0,0);

  //cálculo do factor de imagem com o mapa para redimensionar a imagem
  if SizeW/Image.Width > SizeH/Image.Height then
    imfactor:=SizeW/Image.Width
  else
    imfactor:=SizeH/Image.Height;

  SrcBitmap.Width:=Image.Width;
  SrcBitmap.Height:=Image.Height;
  SrcIntfImg.LoadFromBitmap(SrcBitmap.Handle, SrcBitmap.MaskHandle);

  //cálculo do valor máximo de distância do mapa
  max_val:=0;
  for yi := 0 to SizeH-1 do begin
    for xi := 0 to SizeW-1 do begin
      if (TheArray[yi,xi]>max_val) then
        max_val:=TheArray[yi,xi];
      end;
    end;
  end;
  min_val:=0.022;

  SrcIntfImg.FillPixels(colWhite);
  fpcol.alpha:=0;

  //criação da imagem
  for vi :=0 to Image.Height-1 do begin
    for ui :=0 to Image.Width-1 do begin
      //ler apenas valores do mapa
      if (vi*imfactor<SizeH-1) and (ui*imfactor<SizeW-1) then begin

        //cálculo da relação entre distância da célula cor do pixel da imagem
        gray:=round(MapDist[round(SizeH-1-imfactor*vi),round(imfactor*ui)]*
65536/max_val);
        if (gray > 65535 )then begin
          gray:=255*255;

```

```

        fpcol.red := gray;
        fpcol.green := gray;
        fpcol.blue := gray;
    end else if gray<=min_val*65536/max_val then begin
        gray:=255*255;
        fpcol.red := gray;
        fpcol.green := 0;
        fpcol.blue := 0;
    end else begin
        fpcol.red := gray;
        fpcol.green := gray;
        fpcol.blue := gray;
    end;
    SrcIntfImg.Colors[ui,round((Image.Height-SizeH)*imfactor+5+vi)] :=
fpcol;
    end;
end;
end;
SrcIntfImg.CreateBitmaps(ImgHandle,ImgMaskHandle,true);
TempBitmap.Handle:=ImgHandle;
Image.Canvas.Clear;
Image.Canvas.Draw(0,0,TempBitmap);
SrcIntfImg.Free;
end;

```


Anexo C

Interface comunicação laser

C.1 Leitura do buffer de dados do laser

```
procedure TFLaser.SdpoSerial1RxData(Sender: TObject);
var
  val:double;
  i:integer;
  cab:string;
  numpoints:integer;
  data:string;
begin
  //lê buffer da porta série
  buffertemp:=buffertemp+ SdpoSerial1.ReadData;
  //procura pelos caracteres de finalização
  for i:=0 to Length(buffertemp) do begin
    if(buffertemp[i]=chr(10)) and (buffertemp[i+1]=chr(10)) then begin
      buffer:='';
      buffer:=buffertemp;
      //verifica se o início do cabeçalho é válido
      if buffer[1]='G' then
        //extraí os dados do buffer
        extractmessage(buffer);
      end;
    end;
  end;
end;
```

C.2 Extracção dos dados do buffer

```
procedure extractmessage(var buf: string);
var
  dist, ang, px,py, pxl,pyl:double;
  i:integer;
  cab:string;
  astart, aend, acluster:integer;
  data:string;
  contador:integer;
begin
  numpoints:=0;
  contador:=0;
  //verifica os dados do cabeçalho
```

```

cab:=Copy(buf,0,12);
if (cab[1]='G') and (cab[10]=chr(10)) and (cab[12]=chr(10)) then begin
  astart:=StrToInt(cab[2]+cab[3]+cab[4]);
  aend:=StrToInt(cab[5]+cab[6]+cab[7]);
  acluster:=StrToInt(cab[8]+cab[9]);
  status.s:=astart;
  status.e:=aend;
  status.c:=acluster;
  numpoints:=round((aend-astart+1)/acluster);
  data:=Copy(buf, 13, 2*numpoints+round(numpoints/32));

  //extraí as distancias e guarda numa lista de pontos
  for i:=0 to numpoints-1 do begin
    dist:=((ord(data[1])-48)*64+(ord(data[2])-48));
    if dist>100 then begin
      if contador < 100 then begin
        ang:=degto rad((astart+(i+1)*acluster-acluster*0.5)*RANG-135);
        px:=dist*cos(ang);
        py:=dist*sin(ang);
        LaserPointList.PList[contador].x:=px/1000;
        LaserPointList.PList[contador].y:=py/1000;
        contador:=contador+1;
      end;
    end;
    //quando recebe tramas maiores que 64bytes tem de extrair o character
    que as separa
    data:=Copy(data,3,Length(data));
    if Length(data) > 1 then begin
      if data[1]=chr(10) then
        data:=Copy(data,2,Length(data));
      if data[2]=chr(10) then
        data:=data[1]+Copy(data,3,Length(data));
    end;
  end;
end;
LaserPointList.PCount:=contador;
end;

```