André Alexandre de Sousa Brandão

Robô Autónomo Guia e para Vigilância Automática

Dissertação de Mestrado

Setembro de 2013

· U C ·

Universidade de Coimbra

UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Robô Autónomo Guia e para Vigilância Automática

André Alexandre de Sousa Brandão

Dissertação submetida para obtenção do Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Orientadores: Prof. Doutor António Paulo Mendes Breda Dias Coimbra
Prof. Doutor Manuel Marques Crisóstomo

Co-Orientador: Prof. Doutor Mateus Daniel Almeida Mendes

Júri:

Professor Doutor Urbano José Carreira Nunes (Presidente)
Professor Doutor António Paulo Mendes Breda Dias Coimbra (Orientador)
Professor Doutor Manuel Marques Crisóstomo (Orientador)
Professor Doutor Jorge Nuno de Almeida e Sousa Almada Lobo (Vogal)

Trabalho desenvolvido no Instituto de Sistemas e Robótica da Universidade de Coimbra

Coimbra, Setembro de 2013

AGRADECIMENTOS

Começo por agradecer aos meus pais e ao meu irmão por todo o apoio e compreensão prestados ao longo da minha formação académica.

Em segundo lugar, gostaria de agradecer aos meus orientadores Paulo Coimbra, Manuel Crisóstomo e Mateus Mendes pela sua orientação, conhecimento transmitido e pelas sugestões e correcções apresentadas ao longo do desenvolvimento deste trabalho. Ao meu colega de laboratório André Rodrigues pela ajuda e troca de ideias, assim como a todos os amigos que me acompanharam ao longo destes últimos anos.

Agradeço por fim à Faculdade de Ciências e Tecnologia da Universidade de Coimbra, bem como ao Instituto de Sistemas e Robótica, por toda a logística necessária para a minha formação académica e desenvolvimento desta dissertação.

RESUMO

Nos últimos anos, com o aumento da capacidade de processamento e a diminuição do custo das memórias, a navegação robótica com base em informações visuais tem vindo a tornar-se alvo de intensa pesquisa. Dentre as várias abordagens que surgiram com o decréscimo do custo do *hardware*, a técnica *teach and follow* foi a usada neste trabalho para a navegação de um robô móvel. Este método necessita de uma fase inicial de aprendizagem, onde as informações aprendidas, imagens e dados adicionais, ficam guardadas na Memória Esparsa Distribuída (SDM). A SDM é uma memória associativa, com capacidade de lidar com dados com bastante ruído, ou mesmo incompletos, e reter grandes quantidades de informação. De modo a aumentar a sua velocidade de processamento, a memória pode ser implementada em paralelo numa GPU (Unidade de Processamento Gráfico).

Nesta tese é feita a implementação da navegação de um robô autónomo com o método descrito acima, para a qual se usou uma SDM implementada numa GPU, que guarda as imagens capturadas ao longo de um percurso ensinado ao robô por um operador humano. De seguida, o robô, através da consulta das informações aprendidas guardadas na memória, tem a capacidade de percorrer autonomamente o caminho previamente aprendido. Durante a navegação autónoma, a localização do robô é feita comparando a vista actual com as vistas aprendidas. Assim, usando a vista mais próxima, ou seja, mais semelhante, o robô tem a capacidade de se localizar ao longo de um trajecto. Quando o robô navega por caminhos diferentes com segmentos muito parecidos entre si, a pesquisa pela imagem mais próxima pode não ser suficiente para a estimação correcta da posição actual do robô, sendo necessário nestes casos limitar a pesquisa apenas às imagens aprendidas mais próximas da última predição. Durante a navegação num ambiente real, o robô pode ainda encontrar obstáculos ao longo do seu trajecto. Então, a implementação de um método de desvio de obstáculos permite ao robô continuar o seu percurso mesmo quando encontra barreiras físicas ou pessoas durante a navegação autónoma.

A aplicação desenvolvida, e testada nos corredores do edifício do DEEC, nesta tese permite que o robô percorra autonomamente trajectos anteriormente aprendidos dentro de edifícios.

Palavras-chave: Navegação Robótica Baseada em Visão, SDM, Robótica Móvel, GPU.

ABSTRACT

In the latest years, the increasing processing capacities and the decreasing of the cost of memories has turned the robotic navigation based on visual information into a subject of intense research. Among the several approaches that emerged from the declining cost of *hardware*, the "teach and follow" technique was used in this work for the navigation of a mobile robot.

This method requires an initial learning phase, where the learned information, images and additional data, are stored in the Sparse Distributed Memory (SDM). The SDM is an associative memory with the capacity to process noisy data, or even incomplete data, and retain large amounts of information. The memory may be implemented on a GPU (Graphics Processing Unit) to increase its processing speed.

In this thesis, the implementation of the autonomous robot navigation is made using the method described above, for which we used a SDM implemented on the GPU, which stores the images captured over a route taught to the robot by a human operator. Then, the robot using previously learned information stored in memory has the ability to autonomously navigate the path previously learned. During autonomous navigation, the location of the robot is done by comparing the current view with all the learned views. Thus, using the closest view, (the most similar), the robot is able to locate itself in the route. However, when the robot navigates through paths with different but similar segments, the search for the closest image may not be sufficient to correctly predict the current position of the robot, so it is necessary in these cases to limit the search only to the learned images which are closer to the last prediction. During navigation in a real environment, the robot can still find obstacles along its path. Therefore, the implementation of a method for obstacle avoidance enables the robot to continue its way when it runs into physical barriers or people during the autonomous navigation.

The application developed and tested in the corridors of the DEEC building allows the robot to navigate autonomously through previously learned routes within buildings.

Keywords: View Based Navigation, SDM, Mobile Robotics, GPU.

Índice

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÓNIMOS

BMP	Bitmap Image File
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DSP	Digital Signal Processor
GPGPU	General-purpose Graphics Processing Unit
GPU	Graphics Processing Unit
LCD	Liquid Crystal Display
MCU	Multipoint Control Unit
MFC	Microsoft Foundation Classes
MIPS	Millions of Instructions per Second
PGM	Portable Graymap
RAM	Random Access Memory
SDM	Sparse Distributed Memory
SRAM	Static Random Access Memory

1. INTRODUÇÃO

A pesquisa para o desenvolvimento de robôs inteligentes tem sido alvo de esforços por parte dos cientistas ao longo dos últimos anos [35]. Contudo, os resultados obtidos neste âmbito não têm sido tão positivos quanto o esperado, havendo ainda um longo caminho a percorrer nesta área. Torna-se então necessário utilizar novas abordagens nesta área. Neste trabalho, foi implementada uma Memória Esparsa Distribuída (SDM) num robô móvel, permitindo-lhe navegar autonomamente com base em memórias visuais. Foi também desenvolvido um módulo com algoritmos inovadores para desvio de obstáculos, que dota o robô da capacidade de contornar obstáculos durante a navegação autónoma.

A área da robótica móvel tem sido alvo de investigação intensa, apresentando-se portanto em franco crescimento [36], uma vez que têm sido desenvolvidos sistemas robóticos autónomos com as mais diversas finalidades, como a vigilância, o suporte a serviços médicos, a ajuda em tarefas agrícolas e industriais, a pesquisa em lugares perigosos ou inacessíveis ao homem e o entretenimento.

1.1 OBJECTIVOS

O objectivo principal desta tese foi desenvolver *software* para dotar um robô móvel com a capacidade de percorrer diversos trajectos de forma autónoma. A navegação é baseada principalmente em visão, informação odométrica e no uso de uma memória associativa (SDM – *Sparse Distributed Memory*). Pretendia-se integrar neste trabalho a implementação da SDM numa GPU [3], para melhoria da performance da SDM em navegação robótica.

De modo a que a navegação fosse o mais robusta possível, pretendia-se também testar vários métodos de navegação, bem como um método para o desvio de obstáculos. Desta forma, esperava-se que o robô ficasse apto para tarefas como vigilância, guia ou a entrega de mensagens ou objectos.

1.2 MEMÓRIA ESPARSA DISTRIBUÍDA E NAVEGAÇÃO ROBÓTICA

A SDM é um tipo de memória associativa, proposta por Pentti Kanerva [2], adequada para lidar com vectores de elevada dimensionalidade. Para além de ser bastante tolerante ao ruído e conseguir reconhecer padrões mesmo quando os dados recebidos estão incompletos, a SDM tem a capacidade de aprender e mais tarde esquecer os dados aprendidos de forma natural, em condições semelhantes ao que acontece no cérebro humano, exibindo ainda outras propriedades semelhantes à memória humana de longa duração. A memória é esparsa, uma vez que apenas uma pequena parte de todas as localizações possíveis de endereçar vão ser realmente usadas. Outra característica é o facto de a memória ser distribuída, uma vez que o mesmo dado pode ser guardado em vários pontos da memória. A informação guardada na memória consiste num vector de dados e num vector de endereços. O vector de endereços de um dado activa um conjunto de localizações físicas, chamadas *hard locations*, que vão ser usadas para guardar os dados. A informação aí guardada pode ser posteriormente acedida, através das operações de leitura da memória.

Esta memória é adequada para guardar todo o tipo de informação. Permite também, tal como o cérebro humano, que as informações sejam processadas em paralelo.

A capacidade de processar dados em paralelo é muito útil em aplicações de tempo real, como a navegação autónoma de um robô, na qual é necessária uma resposta rápida aos estímulos do meio recebidos pelo robô. Uma forma simples de se tirar vantagem desta propriedade da SDM é a implementação da SDM numa GPU, usando por exemplo a tecnologia CUDA [17]. Esta tecnologia oferece o suporte necessário para o desenvolvimento de algoritmos de computação paralela que aceleram o processamento de grandes volumes de dados.

Abaixo encontram-se sumariados alguns dos trabalhos publicados na área da navegação robótica usando SDMs.

- Em 1996, Rao e Fuentes [4] simularam a navegação robótica com base numa SDM. Nesta simulação, um robô era capaz de descrever um trajecto autonomamente. A SDM foi implementada numa rede neuronal, onde era armazenada a informação obtida durante o processo de aprendizagem de um determinado percurso. De seguida, o robô tinha capacidade de fazer o percurso previamente aprendido com base nas informações guardadas na SDM.

- Em 2001, Watanabe, Furukawa e Kakazu [5] executaram a simulação de um modelo para auxiliar, com base numa SDM, veículos autónomos em rotas pré-definidas em ambientes industriais. A SDM foi implementada numa rede neuronal e permitia aos veículos percorrerem várias vezes a mesma rota, para transporte de materiais. Além disto, nesta simulação, os robôs tinham a capacidade de se desviar de obstáculos que surgissem ao longo dos percursos. Na fase de aprendizagem, as informações sensoriais de um percurso eram guardadas na SDM. Posteriormente, a SDM era usada para reconhecer padrões com base na informação aprendida, de modo a que os veículos fossem capazes de navegar autonomamente.

1.3 CONTRIBUIÇÕES E TRABALHO DESENVOLVIDO

Este trabalho surge na sequência do trabalho de doutoramento de Mateus Mendes, *Intelligent Robot Navigation Using a Sparse Distributed Memory* [37] e do trabalho de dissertação de mestrado de André Rodrigues [38].

Para o desenvolvimento deste trabalho foi necessário realizar um estudo prévio sobre o funcionamento da SDM, assim como os meios de comunicação e controlo do robô adquirido. Além disso, foi importante compreender o modo de implementação de uma SDM, tanto em CPU como em GPU. Foram desenvolvidos ou melhorados vários algoritmos de navegação. Resumidamente, o trabalho realizado foi o seguinte:

- Aquisição e operacionalização do robô X80Pro. Foi necessário estudar e escolher a plataforma, adquirir cabos de comunicação, contactar o fabricante para obter *drivers* e bibliotecas de comunicação, adaptar essas bibliotecas e integra-las na aplicação desenvolvida neste trabalho.
- Estudo de uma SDM implementada na CPU [1] e integração na aplicação para o X80Pro. Isso incluiu a escrita do algoritmo básico de navegação, que requereu processos novos de aquisição e manipulação de imagens.
- Estudo da SDM implementada na GPU [3] e integração na aplicação desenvolvida neste trabalho.
- Realizados testes preliminares de navegação baseada em visão.
- Implementação do algoritmo de navegação com janela deslizante de largura fixa. Este passo requereu alterações significativas ao código desenvolvido por Rodrigues et al [3], designadamente para conseguir truncar o espaço de pesquisa da SDM e aceder aos dados na memória da GPU.

- Implementados algoritmos para fazer a pesquisa fora da janela deslizante para resolver o problema do "robô raptado".
- Implementado o modo vigilante, que consiste numa navegação contínua, em circuito fechado.
- Por fim, foram desenvolvidos e testados os algoritmos de desvio de obstáculos.

No final do trabalho desenvolvido, chegou-se um algoritmo para navegação autónoma baseada em visão robusto, capaz de conduzir autonomamente o robô ao longo de percursos previamente ensinados, dentro de edifícios, mesmo quando surgem obstáculos imprevistos. Esta é a principal contribuição desta tese para o estado da arte, tendo sido escrito um artigo científico, que se encontra no anexo C e que foi submetido ao ICRA 2014.

1.4 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está dividida em 7 capítulos.

No primeiro capítulo é apresentada uma breve introdução ao tema proposto para a tese.

A descrição da SDM, explicando as suas principais características e as suas diversas formas de implementação, é apresentada no segundo capítulo. Além disso, é explicado o modo de implementação da memória tanto numa CPU como numa GPU para o processo de navegação autónoma de um robô móvel.

No capítulo três faz-se um breve resumo do estado da arte ao nível da navegação de robôs baseada em visão.

No capítulo quatro é apresentado o *hardware* utilizado nos testes. Além disso, é apresentada a interface desenvolvida e explicada de forma sucinta a arquitectura do *software*.

No capítulo cinco é apresentada a implementação prática desenvolvida no presente trabalho para a navegação de um robô.

O capítulo seis apresenta o módulo desenvolvido para o desvio de obstáculos em tempo real durante a navegação autónoma. São apresentadas duas soluções complementares de modo a que quando o robô se depare com uma obstrução no trajecto seja capaz de a ultrapassar e chegar ao destino.

As conclusões da dissertação, assim como as sugestões para trabalho futuro, são expostas no sétimo capítulo.

2. MEMÓRIA ESPARSA DISTRIBUÍDA

Neste capítulo aborda-se a Memória Esparsa Distribuída (SDM), proposta por Pentti Kanerva, e nele serão descritas as principais características, os problemas que se pretendem resolver com este tipo de memória e as suas principais vantagens.

Explicar-se-á também a implementação desta memória numa CPU, que serviu como base para a navegação de um robô autónomo, assim como a sua implementação numa GPU, de modo a aumentar a performance da memória e então permitir ao robô fazer uma estimativa mais rápida e precisa da sua localização.

Assim, este capítulo apresenta uma breve descrição dos vários modelos da SDM, implementados por Mendes et al [1], e da sua implementação em paralelo numa GPU, realizada por Rodrigues et al [3]. O leitor já familiarizado com o funcionamento da SDM pode avançar para o capítulo seguinte.

2.1 INTRODUÇÃO À SDM

Nos finais dos anos 80, Pentti Kanerva propôs a Memória Esparsa Distribuída, um tipo de memória associativa, com semelhanças com a memória de acesso aleatório (RAM) [2,6], que apresenta também semelhanças à memória humana de longa duração. Este tipo de memória consegue armazenar grandes quantidades de informação e recuperá-las sem ser necessário conhecer a localização exacta dos dados na memória. A SDM assenta nas propriedades dos espaços booleanos de grandes dimensões. A ideia por detrás da SDM consiste no mapeamento de uma grande memória binária num conjunto mais pequeno de localizações físicas, chamadas *hard locations*, que devem estar distribuídas uniformemente no espaço virtual, para simular um vasto espaço virtual da forma mais precisa possível. Assim, esta memória tem em conta que, tal como na memória humana, as distâncias entre conceitos presentes na memória correspondem a distâncias entre pontos num espaço booleano de elevada dimensionalidade, onde um ponto de interesse está, em geral, bastante distante de outros pontos do mesmo espaço.

A possibilidade de descobrir ligações entre duas ideias que aparentemente não estavam relacionadas, lembrar-se de um facto a partir de pouca informação sobre o mesmo, ou inferir uma resposta a partir da associação de um conjunto de conceitos, que separados não constituem uma resposta, são exemplos de como esta memória apresenta comportamentos semelhantes ao do

cerebelo humano. Doya [7] propõe que o cerebelo pode ser entendido como um dispositivo de aprendizagem supervisionada. O cerebelo pode ser interpretado como um sistema de armazenamento que, de forma extremamente rápida, associa acções adequadas para dar resposta a estímulos recebidos das entradas sensoriais.

Algoritmos com *lookup tables* assim como redes neuronais foram propostos para simular memórias com características semelhantes às descritas acima. No entanto, ambas as soluções só funcionavam bem para problemas de pequena dimensão, apresentando várias dificuldades quando a dimensão do problema começava a aumentar [1,8]. A SDM permitiu resolver problemas na aprendizagem de grandes quantidades de dados, que trazia dificuldades tanto nas *lookup tables* como nas redes neuronais [1,2].

A tentativa de implementação de modelos de memória humana em máquinas, como uma forma de inteligência artificial, tem sido desenvolvida com pouco sucesso desde há cerca de 40 anos, no entanto a SDM veio trazer progressos nesta área permitindo agora que estes tipos de memórias sejam implementados em sistemas robóticos [9].

Na SDM, as entradas sensoriais são vectores de bits de grandes dimensões, mapeados num conjunto mais pequeno de localizações físicas, as *hard locations*, sendo que cada um destes vectores pode ser guardado em várias *hard locations* para que a memória fique distribuída. Para se recuperar um vector guardado, Kanerva propôs fazer-se a média dos valores nas *hard locations* onde o vector foi guardado, o que implica que a informação recuperada possa não ser exactamente igual à informação guardada, estando a precisão da informação recuperada dependente do nível de saturação da memória. Para além da média, existem outras técnicas para a recuperação dos dados, como a mediana, a média ponderada ou a moda [1].

A SDM pode ser implementada numa rede neuronal ou como uma lista ligada. Para aumentar a performance da SDM, esta também já foi implementada numa GPU que, através da sua capacidade de processamento paralelo, torna a memória mais rápida.

As quatro ideias base na SDM de Kanerva são:

1. O espaço 2^n, para $100 < n < 10^5$, onde a informação é guardada, exibe propriedades como a noção humana de intuição na relação entre conceitos.

2. Neurónios com *n* entradas podem ser usados como descodificadores de endereços de uma memória de acesso aleatório, se a memória, como Kanerva propôs, for implementada numa rede neuronal.

3. O princípio unificador: os dados guardados na memória podem ser usados como endereço para a mesma memória, ou seja, a informação guardada numa *hard location* pode ser, ao mesmo tempo, o seu próprio endereço na memória.

4. Se os dados estiverem organizados como uma sequência de eventos, pode ser determinada a ordem a sequencial do seu armazenamento na SDM.

2.2 O ESPAÇO BOOLEANO

A SDM proposta por Kanerva baseia-se nas propriedades do espaço booleano, 2^n, também referido por N, onde n representa o número de dimensões do espaço e 2^n o número de pontos no espaço. Um ponto do espaço 2^n é representado por *n* dígitos binários. Estes números binários podem não estar ordenados, pois 001 pode vir antes ou depois de 000. Em teoria, a memória poderia guardar até 2^n itens, um item por cada ponto no espaço, com tamanho *n*.

As propriedades do espaço booleano permitem efectuar várias operações com pontos no espaço, como a diferença, distância, *betweenness*, ortogonalidade ou círculo.

A diferença entre dois pontos é expressa pelo conjunto de bits onde os pontos diferem, ou seja, o conceito de OR exclusivo. Por exemplo, a diferença entre 1010 e 1101 é 0111.

Na implementação da SDM, Kanerva usa, para o cálculo da distância entre dois pontos, a distância de Hamming, que é definida como o número de bits em que dois pontos diferem. Por exemplo, a distância de Hamming entre 1101 e 1010 são 3 bits. A distância entre dois pontos permite ver o quanto eles estão relacionados, ou seja, quanto mais pequena for a distância, mais semelhantes são os dois vectores.

A noção de *betweenness* aplica-se quando um ponto y está entre dois pontos x e z. Um ponto y está entre x e z (x:y:z) quando a distância de x a z é igual à soma das distâncias de x a y e de y a z.

Um dos conceitos mais relevantes no espaço booleano é o conceito de ortogonalidade. Dois pontos são ortogonais, perpendiculares, ou indiferentes, se a distância entre eles for igual ao número de dimensões a dividir por 2. Uma propriedade importante é que se x for indiferente a y, então é também indiferente ao seu complemento y'[1].

O círculo é um objecto importante na teoria da SDM, e tem a mesma interpretação que o círculo no espaço euclidiano. Na implementação da SDM, o conceito de círculo é usado para verificar se uma determinada *hard location* está dentro do raio de activação de um vector de

[1] O complemento de y, y', é a negação do vector y.

referência. O centro do círculo é o endereço do vector de referência e é utilizada a distância de Hamming para medir a distância entre endereços.

2.3 A MEMÓRIA ESPARSA E DISTRIBUÍDA

Como já foi visto, o modelo da SDM de Kanerva baseia-se no espaço 2^n, que exibe propriedades e características semelhantes, em muitos aspectos, às do cérebro humano. Desta forma, é um modelo interessante para ser usado na construção de uma memória associativa. No entanto, para a implementação de uma memória baseada nas propriedades do espaço 2^n, é necessário resolver alguns problemas práticos.

O primeiro problema a resolver, que surge antes da implementação da SDM, é a dimensão do espaço. Isto pode tornar-se um grave problema, pois para um N extremamente largo torna-se muito difícil implementar a SDM. Este problema poderia condicionar o trabalho apresentado, pois o robô irá usar a SDM para armazenar as imagens que capturou, o que vai obrigar a SDM a lidar com vectores de estado com um tamanho bastante elevado, o que fará cair drasticamente a performance em tempo real.

Para resolver este problema, Kanerva propôs que a memória fosse esparsa, ou seja, a memória contém apenas uma pequena parte dos endereços físicos. Os endereços físicos são chamados *hard locations*. São apenas uma pequena parte do espaço de endereços, que conseguem representar todo espaço com precisão, através da amostragem. Estas localizações físicas são criadas logo no início, ou seja, antes de se guardar algo na memória, e são distribuídas aleatoriamente no espaço *N*, não podendo ser modificadas quando a memória está a ser usada, excepto os seus conteúdos (em operações de leitura e escrita). Embora este seja o método original para a criação das *hard locations*, normalmente e para que não haja desperdício de memória, as *hard locations* são alocadas dinamicamente quando necessário. Kanerva mostra que um conjunto de 2^{20} *hard locations* e raio de acesso de 425 bits são suficientes para vectores de tamanho de 1000 bits, o que evidencia como a memória é bastante esparsa. Apesar de demonstrar que a capacidade da memória na implementação da SDM deixou de ser um problema, este exemplo já está ultrapassado. Neste trabalho foram usados vectores com tamanho muito superior a 1000 bits guardados num conjunto muito inferior a 2^{20} *hard locations*.

Esta memória permite assim que se consiga recuperar uma informação quando se tem apenas parte dessa informação na SDM. Kanerva afirma que apenas com 200 bits correctos em 1000, há uma probabilidade de 0,6 (apenas 600 em 1000 bits estarem correctos) de recuperar o vector correcto. Os restantes 800 bits desconhecidos são preenchidos aleatoriamente, pelo que há uma probabilidade de 0,5 de acertar no valor correcto.

$$0{,}2 + \ 0{,}8 \ \times 0{,}5 = 0{,}6 \tag{2.1}$$

Com esta propriedade, segundo Kanerva, é possível imitar a memória humana ao conseguir reconhecer um objecto em diferentes ambientes.

Escrita: na SDM os dados são guardados em *hard locations* que constituem apenas uma pequena parte do espaço. Assim, Kanerva propôs que a escrita fosse feita em mais que uma *hard location*, isto é, um dado deve ser escrito em todas as *hard locations* dentro de um raio definido. Desta forma, a memória diz-se distribuída. Os novos dados podem substituir os anteriores ou serem-lhes somados.

Leitura: para recuperar um vector na memória são usadas todas as *hard locations* dentro do raio de acesso, tal como na escrita. Para recuperar o vector guardado em todas as *hard locations* activas, o método utilizado por Kanerva passa por fazer a média[2] de cada elemento e, em seguida, aplicar um limiar ao vector resultante.

2.4 PROPRIEDADES DA SDM

A SDM, implementada tendo em conta as propriedades do espaço booleano, exibe comportamentos idênticos aos da memória humana de longa duração, tais como:

Grande dimensão – A SDM é adequada para lidar com sistemas de grande dimensão. Kanerva estudou a SDM com base em vectores de 1000 bits, mas neste trabalho são usados vectores muito maiores Neste caso foram usados vectores com um tamanho superior a 25000 mil bits.

Associatividade – A SDM é uma memória associativa, consegue associar padrões através da semelhança de vectores, assim como consegue trabalhar com informação incompleta.

Tolerância a erros – Devido ao armazenamento distribuído, o uso de limiares e ao facto de ser uma memória associativa, a SDM apresenta uma boa tolerância no tratamento de dados com erros.

Aprendizagem natural – Dependendo do nível de saturação, a SDM deve ser capaz de aprender apenas numa passagem. Além disso, quantas mais vezes aprender determinada informação, mais difícil será esquecê-la.

Esquecimento natural – A memória na SDM não pode ser apagada. Deve ser esquecida com a passagem do tempo e quando a memória começa a ficar cheia.

[2] Para além da média, existem outras técnicas para a recuperação dos dados, como a mediana, a média ponderada ou a moda.

Degradação gradual – No caso de uma falha, a SDM não deixa de funcionar subitamente. Começa por ficar cada vez menos exacta até que a sua saída se torna totalmente lixo.

Dados incompletos – Com apenas 20% dos bits correctos e com os 80% restantes preenchidos aleatoriamente, a SDM mostra ainda uma boa probabilidade de apresentar uma resposta correcta.

Processamento paralelo – A SDM pode ser processada em paralelo, tal como o cérebro humano, o que permite atingir velocidades de computação mais rápidas ou a introdução de vários dados em simultâneo.

2.5 IMPLEMENTAÇÃO DE UMA SDM USANDO LISTAS LIGADAS

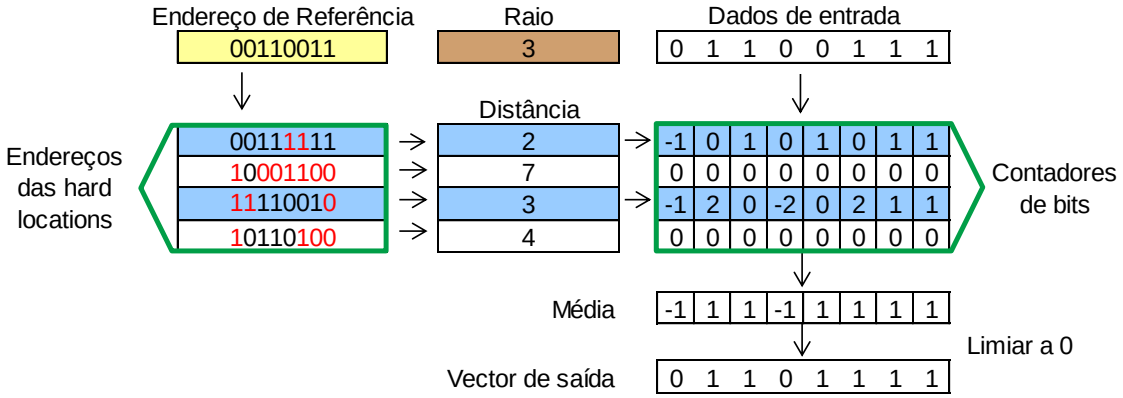


Figura 1 - Modelo da SDM de Kanerva. Retirado de [1,10]

A figura 1 ilustra a implementação do modelo da SDM de Kanerva. A operação de escrita começa por seleccionar, através da distância de Hamming, os endereços dentro do raio de activação definido, neste caso 3. Os endereços dentro do raio de acesso são seleccionados e os seus valores actualizados. Para actualizar os valores das *hard locations*, são usados contadores, um por bit, que são incrementados se o valor a guardar é 1 ou decrementados se o valor a guardar é 0. Desta forma, os dados escritos mais vezes são mais difíceis de serem esquecidos. No início, quando a memória ainda não tem nenhuma informação guardada, todos os contadores de bits estão a 0.

Para guardar um bit com o valor 1, é somado 1 ao elemento correspondente na *hard location* e para guardar um bit com o valor 0 é subtraído 1 ao respectivo elemento na *hard location*. Na prática, nesta implementação, são usados contadores de 8 bits mas, em teoria, os valores a guardar variam de -40 a +40, o que permite que contadores de 6 bits sejam suficientes.

A operação de leitura consiste em somar coluna a coluna todos os bits das *hard locations* activas, fazer a média de cada elemento e aplicar-lhe um limiar, neste caso de 0. Como se verifica, não há a garantia de se obter dados exactamente iguais aos guardados.

Existe também uma versão auto-associativa da SDM, em que o vector de endereço é igual ao vector de dados. Nesta versão da SDM não há necessidade de se trabalhar com dois vectores diferentes, uma vez que o mesmo vector contém os dados e seu próprio endereço.

2.6 A SDM Implementada numa Rede Neuronal

A implementação da SDM como uma rede neuronal é mais fácil de visualizar e mais idêntica aos sistemas biológicos. A SDM pode ser modelada como uma rede neuronal, na qual existem três camadas (figura 2).

A primeira camada, de endereçamento, é constituída por perceptrões com pesos fixos. Os perceptrões têm a função de mapear $\{0,1\}^n \rightarrow \{0,1\}$, isto é, recebe como entrada um vector com *n* bits e tem como saída apenas um bit. Com isto, a rede neuronal activa as entradas que tiverem o número de uns acima de um dado limiar, este procedimento é semelhante ao cálculo da distância de Hamming para verificar quais as localizações activas. Kanerva propôs que os pesos fossem fixos no início para os endereços ficarem distribuídos aleatoriamente pelo espaço de endereços, uma vez que uma distribuição aleatória produz bons resultados. A saída de cada neurónio liga-se a uma coluna de neurónios da camada seguinte.

Na segunda camada, camada escondida, cada linha de neurónios liga-se a apenas um neurónio da camada de saída, ou seja, activa apenas um endereço de memória. Essa ligação é feita através de sinapses onde os pesos das sinapses funcionam como os contadores de bits.

Na terceira camada, de saída, cada neurónio aplica um limiar ao resultado do somatório de uma linha de neurónios da segunda camada, que produz o valor um se o resultado for superior ao limiar ou zero se for inferior.

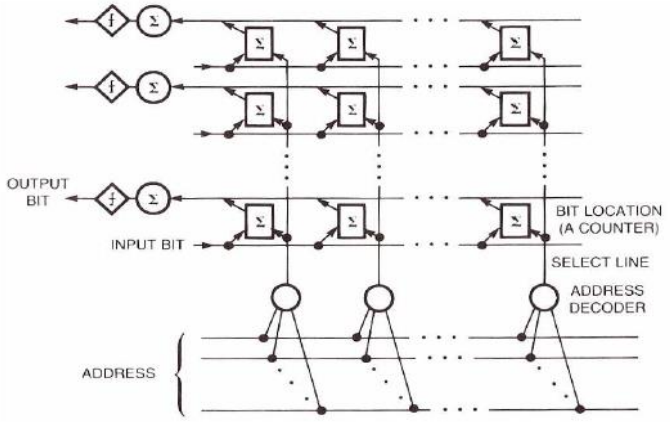


Figura 2 - Modelo de uma SDM implementada numa rede neuronal. Retirado de [1].

2.7 A SDM NUMA CPU PARA A NAVEGAÇÃO ROBÓTICA AUTÓNOMA

A SDM já foi implementada e testada, na forma de listas ligadas, figura 3, por Mendes et al [1], como base para a navegação autónoma de um robô. A SDM era usada durante o processo de aprendizagem para guardar as vistas aprendidas e informação odométrica. Durante a navegação autónoma, as imagens presentes na SDM eram comparadas com a vista corrente do robô.

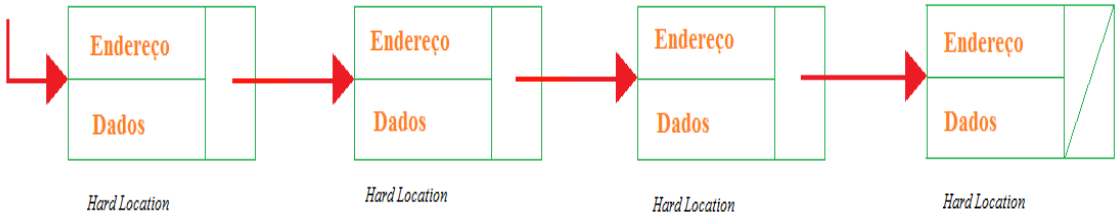


Figura 3 - SDM implementada como uma lista ligada. Retirado de [3].

2.7.1 APRENDIZAGEM

Na aprendizagem, o robô era conduzido ao longo de um percurso durante o qual ia capturando várias imagens. Isto permitia ao robô localizar-se nos percursos aprendidos, quando as imagens correntes fossem idênticas às aprendidas.

Cada imagem aprendida podia ser guardada em várias *hard locations*. Em cada *hard location* eram guardados os píxeis de cada imagem no vector de endereço e um vector de dados x_i dado por:

$$x_i = < img, img_{id}, seq_{id}, timestamp, mov > \quad (2.2)$$

Para além dos píxeis de cada imagem (img), era guardada informação adicional de modo a facilitar o processo de controlo dos motores em tempo real, além de poder ser usada também para desambiguação quando existiam imagens muito parecidas mas de sequências diferentes. (img_{id}) era o número de imagem que, facilmente, permitia saber a localização de uma imagem dentro da sua sequência. (seq_{id}) representava o número de sequência que identificava, de forma rápida, qual a sequência onde o robô estava a navegar. O $timestamp$, que era um inteiro lido do sistema operativo, era usado como informação adicional para fins de *debugging*. mov era um caractere que representava o movimento realizado pelo robô quando a imagem foi capturada. Este caractere podia tomar quatro valores: "f" se o robô estava a andar em frente, "l" quando o robô virava para a esquerda, "r" se o robô estava a virar para a direita e "b" se o robô estava a andar para trás.

2.7.2 NAVEGAÇÃO AUTÓNOMA

Durante a navegação autónoma, para se localizar a cada instante, o robô capturava a imagem da vista corrente e de seguida essa imagem era comparada com todas as imagens presentes na SDM, obtendo a imagem aprendida mais próxima, através da diferença de píxeis, e de seguida reproduzia o comando referente a essa imagem.

Para o cálculo de distâncias entre *hard locations* foi testada a distância de Hamming e a distância aritmética, tendo a última apresentado melhor desempenho computacional.

De modo a obter-se a melhor performance possível da SDM durante a navegação autónoma, foram implementadas várias versões da SDM.

A versão original da SDM proposta por Kanerva apresentou um fraco desempenho quando simulada em *software*.

A versão aritmética, que incluía a aprendizagem por reforço, onde os bits eram agrupados como inteiros e a distância entre *hard locations* era calculada usando a distância aritmética em vez da distância de Hamming, foi a versão que apresentou melhores resultados nos testes efectuados.

Nos testes realizados, a SDM apresentou bons resultados para a navegação autónoma, pois mostrou ter um bom desempenho.

Nesta dissertação usou-se a SDM implementada numa GPU, de modo a aumentar a performance.

2.8 A SDM NUMA GPU PARA A NAVEGAÇÃO ROBÓTICA AUTÓNOMA

As primeiras placas gráficas surgiram para servir apenas como aceleradores gráficos. A sua arquitectura não era programável e apenas permitia realizar cálculos de programação gráfica.

No final dos anos 90, o *hardware* era já computacionalmente muito poderoso e além disso tinha-se tornado também muito mais programável, tendo a NVIDIA [11] lançado a sua primeira GPU em 1999 [11].

Apesar de os utilizadores estarem a ter bons resultados, as GPGPU [13] ainda apresentavam alguns problemas para os seus programadores. Assim, motivados pelos bons desempenhos obtidos, empresas como a ATI [14,15] e NVIDIA [16,17] fizeram um sério investimento para tornar as suas GPUs totalmente programáveis, como é o caso da CUDA [17] da NVIDIA, na qual a programação é feita numa linguagem familiar aos programadores de C, C++ e Fortran.

A versão da SDM implementada na GPU [3] foi a versão aritmética com aprendizagem por reforço. A distância entre a imagem corrente e as imagens guardadas na SDM é dada pela soma das diferenças absolutas entre os píxeis das imagens. A implementação da SDM numa GPU consiste em transformar a lista ligada num conjunto de vectores (Endereços e Dados), que posteriormente são transferidos para a memória da GPU. No processo de aprendizagem, neste caso o armazenamento de imagens na SDM, a lista ligada é actualizada na CPU com os dados referentes a cada nova *hard location* criada. Quando o processo de aprendizagem termina, o conteúdo de cada nó da lista é copiado para um vector global endereços e para um vector global de dados que terá todos os dados aprendidos. Uma vez preenchidos os vectores globais, estes são transferidos do *host* (CPU) para o *device* (GPU), figura 4.

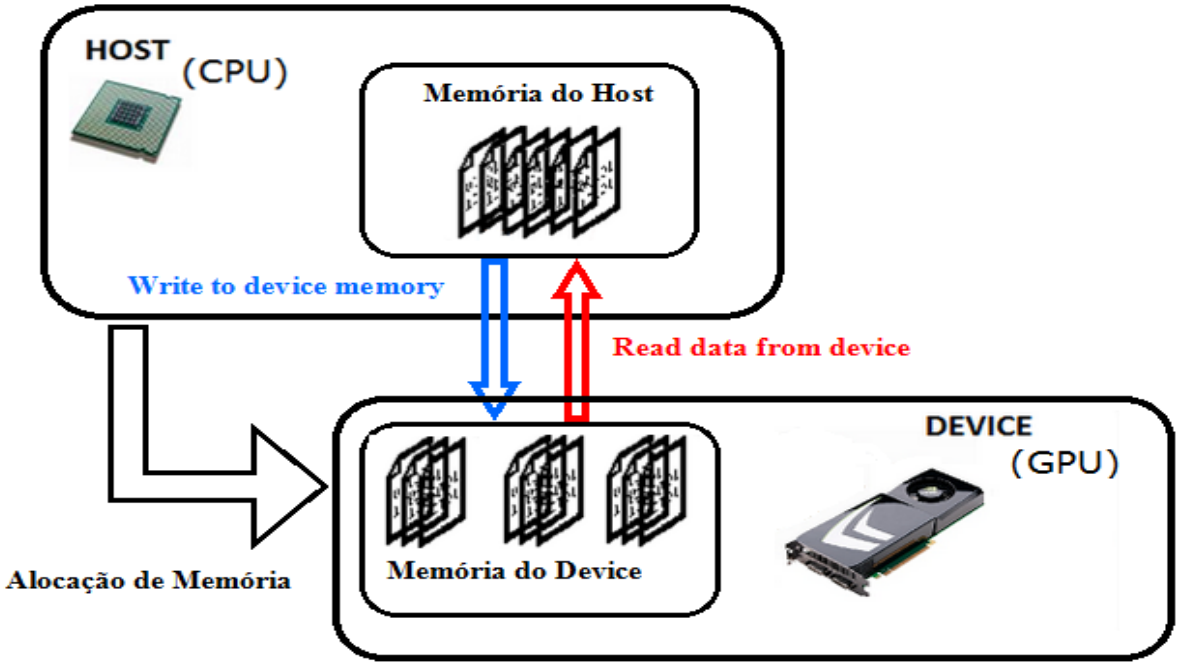


Figura 4 - Transferência de dados entre o *host* e o *device*.

Agora, tendo todos os dados aprendidos na memória da GPU, é preciso obter a imagem mais próxima da vista actual. Para a execução desta tarefa foram implementados seis *kernels*[3].

I. Cálculo da diferença entre os píxeis da imagem corrente e das imagens armazenadas, usando vários blocos em paralelo.

[3] Um *kernel* é uma função definida pelo utilizador, chamada no *host* (CPU) e executada no *device* (GPU) numa grelha de blocos de *threads* .

II. Cálculo das diferenças totais de cada imagem para a imagem corrente.

III. Procura do valor da menor diferença.

IV. Procura do índice da imagem mais próxima.

V. Verificação da existência de localizações activas.

VI. Apresentação do vector de dados recuperado e do número de localizações activas.

Esta implementação apresentou bons resultados, ou seja, a predição da imagem mais próxima devolve realmente a imagem aprendida que mais se assemelha com a imagem actual. Para além disso, apresentou um aumento considerável de performance, obtendo-se *speedups* [18] (medida do grau de desempenho de um algoritmo de programação paralela em relação a um algoritmo de programação sequencial) de 5.31× e 2.51× em relação à CPU sem e com compilador optimizado no PC portátil que controla o robô.

3. NAVEGAÇÃO ROBÓTICA BASEADA EM VISÃO

Este capítulo apresenta de forma sucinta o estado da arte da navegação robótica baseada em visão, bem como o trabalho anteriormente desenvolvido por Mendes et al [1] na navegação de um robô baseada em visão. O leitor familiarizado com os temas pode passar ao capítulo 4.

A visão é o sentido principal no qual os humanos se baseiam para se orientarem e navegarem. No entanto, a visão por computador requer grande poder de processamento e capacidade de memória. Por isso, outro tipo de sensores têm sido usados para realizar a navegação robótica autónoma em tempo real. Contudo, com o aumento da capacidade de processamento dos computadores, associado ao declínio do custo das memórias, torna-se interessante abordar algumas técnicas baseadas em visão e memória, que, para além de serem biologicamente inspiradas, são poderosas e cada vez mais fáceis de implementar, uma vez o *hardware* necessário está mais acessível. Muitas abordagens na navegação baseada em visão são do tipo *teach and follow*. Neste tipo de navegação, um humano começa por ensinar um determinado caminho ao robô. Durante este processo, o robô captura imagens que ficam guardadas em disco assim como outras informações importantes para a navegação, como o número da sequência, o número da imagem na sequência e o movimento que o robô executava quando a imagem foi capturada. Durante a navegação autónoma, o robô é capaz de percorrer o percurso anteriormente ensinado. Para isso, o robô captura uma imagem e de seguida procura na sua memória a imagem mais próxima e executa o comando relativo a essa imagem, guardado aquando da aprendizagem. Durante a navegação são também aplicados métodos de correcção de desvios, de modo a que o robô percorra o caminho de forma mais idêntica possível ao que lhe foi ensinado. Este processo é repetido desde o início ao fim do percurso de modo a que o robô consiga efectuar todo o caminho aprendido.

Quando comparada com outras técnicas que fazem uso de outro tipo de sensores para a construção de mapas, a navegação baseada em visão apresenta algumas vantagens bem como algumas desvantagens. Uma câmara simples pode ser obtida por um preço razoável e consegue fornecer informação detalhada, como uma matriz de píxeis, de uma ampla parte do cenário. Ao contrário das câmaras, outros tipos de sensores, como os sonares ou os sensores de infravermelhos, apenas conseguem fornecer informação de uma faixa estreita do cenário ou de um ponto específico, na forma de um número ou de um vector.

3.0.1 A TÉCNICA "*TEACH AND FOLLOW*"

A técnica *teach and follow* baseada em visão, além de ser biologicamente inspirada, é também mais intuitiva que grande parte das técnicas de navegação.

A principal desvantagem desta técnica encontra-se na necessidade de requerer a supervisão humana ou de uma máquina ensinada, na fase de aprendizagem. Além disto, numa implementação baseada em visão, as imagens têm de ser gravadas durante a aprendizagem, o que pode fazer com que na navegação autónoma se tenha que fazer uma pesquisa numa base de dados de imagens com um tamanho considerável. As abordagens baseadas em visão têm também que ter em conta o problema da variação da iluminação. Imagens com boa resolução requerem muito espaço de armazenamento assim como grande poder de processamento. Para ultrapassar este problema, alguns autores extraem apenas algumas *features* das imagens e é feito apenas o processamento dessas *features* durante a navegação em tempo real.

Outra possível abordagem para a navegação baseada em visão é recorrendo ao uso de *landmarks*. Estas *landmarks* podem ser colocadas na paisagem e podem facilitar a navegação. C. Rasmussen e G. Hager [19] seguiram esta abordagem. Guardavam imagens panorâmicas, ou seja, uma vista inteira de uma área circunvizinha, em intervalos de tempo regulares, na aprendizagem e seleccionavam marcas. *Landmarks* são características consideradas invariantes ao longo do tempo como a cor, a forma ou a textura. Seguindo estas marcas, o sistema consegue reduzir significativamente a necessidade de processamento, pois não é necessário processar toda a imagem para detectar a melhor correspondência.

3.1 NAVEGAÇÃO BASEADA EM SEQUÊNCIA DE IMAGENS

Y. Matsumoto et al [20] propuseram uma abordagem baseada em visão para a navegação robótica que consistia numa vista de sequências e numa *look-up table* que continha os comandos necessários para o controlo do robô.

Esta abordagem requer uma etapa inicial de aprendizagem, durante a qual o robô é guiado por um humano e enquanto isso memoriza sequências de vistas. As vistas são imagens capturadas em intervalos de tempo regulares, formando assim uma sequência ao longo do tempo. Durante a navegação autónoma, o robô faz a localização automática e detecção de obstáculos em tempo real.

Para se localizar, o robô calcula a similaridade entre duas vistas, uma guardada durante a aprendizagem e outra capturada em tempo real. O robô tenta encontrar semelhanças entre as duas imagens e calcula a distância entre elas para saber o quão longe está do caminho correcto. O

controlo dos motores, previamente guardado numa tabela, é obtido com base na distância calculada.

Deste modo é reproduzido o comando relativo à imagem aprendida mais próxima da vista corrente e assim se consegue que o robô percorra o caminho previamente ensinado.

Para o cálculo da distância entre duas vistas *X* e *Y* foi usada a distância Euclidiana, definida como (sendo *n* o número de píxeis):

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{3.1}$$

No entanto, para reduzir o custo computacional, foi adoptada a norma L-1 simplificada, equação 3.2, para o cálculo da distância.

$$e = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n| \tag{3.2}$$

De modo a manter o robô o mais perto possível do caminho aprendido, é calculado o deslocamento lateral entre as vistas actual e aprendida, sendo esse deslocamento usado para corrigir a trajectória. Matsumoto utilizou um processo de correspondência por bloco, como mostrado na figura 5.

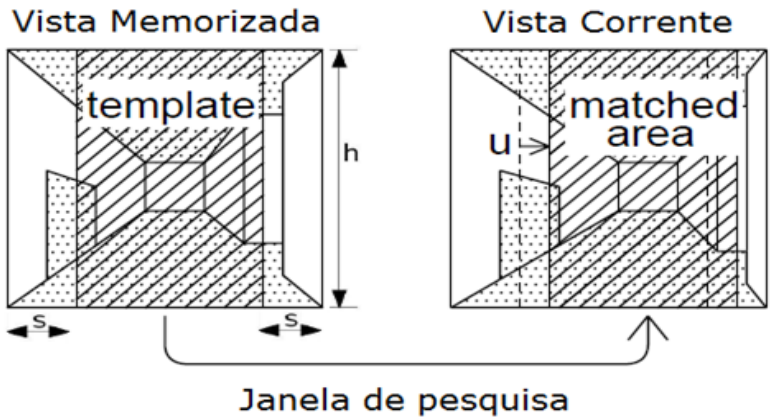


Figura 5 - Correspondência de duas imagens usando a janela de pesquisa. Retirado de [20].

A janela de pesquisa é obtida a partir da vista corrente e, de seguida, é comparada com uma janela equivalente de uma imagem guardada.

Considerando duas imagens I_1 e I_2, Matsumoto definiu, como mostra a equação 3.3, o erro entre as duas imagens:

$$e(u) = \sum_{x=s}^{w-s} \sum_{y=0}^{h} |I_1(x, y) - I_2(x + u, y)| \tag{3.3}$$

Na fórmula, w e h são, respectivamente, a largura e altura da imagem, em píxeis. A letra u representa o deslocamento horizontal da janela de pesquisa. A janela de pesquisa tem uma largura de $w - 2s$ e altura h. O valor de $I_i(x, y)$ corresponde ao valor do píxel (x, y) da imagem i.

A medida do deslizamento horizontal é o valor de *u* que minimiza $e(u)$.

Matsumoto usa imagens com 32 x 32 píxeis, uma janela de pesquisa de 16 x 32 píxeis e define uma gama máxima de 8 píxeis para pesquisa do deslocamento horizontal, $-8 \leq u < 8$. O robô usado captura imagens com uma resolução de 480 x 480 píxeis, mas o autor afirma que não há perda de informação essencial quando as imagens são convertidas para 32 x 32 píxeis.

Num estado mais avançado do trabalho [21], Matsumoto equipou o robô com uma câmara omnidireccional. As imagens omnidireccionais contêm informação de todo o ambiente circundante ao robô. As vistas omnidireccionais não são fáceis de reconhecer para os humanos nem fáceis de processar pelos algoritmos convencionais de processamento de imagem. No entanto, permitem aos robôs ultrapassar a limitação imposta pelas câmaras convencionais que apenas capturam uma faixa do ambiente circundante, dependente da orientação do robô.

Assim, os movimentos para a frente, para trás e a detecção de obstáculos podem ser planeados à custa de um único sensor, com a mesma precisão em todas as direcções ao mesmo tempo.

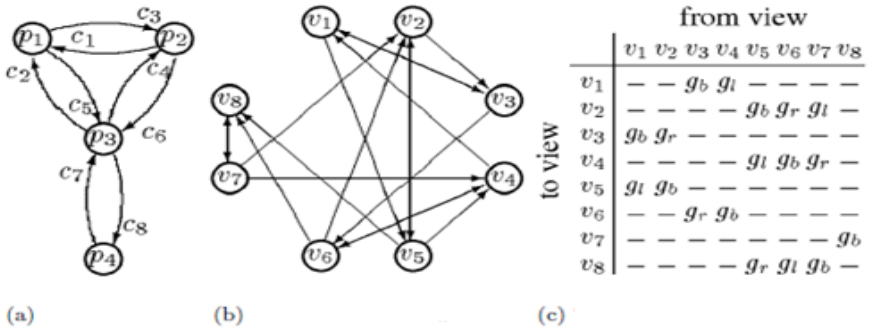
Matsumoto melhorou os algoritmos de navegação e a representação de sequências. Junções, como a intersecção de corredores, eram detectadas com base nas diferenças de fluxo óptico. Assumindo que o fluxo óptico se deve apenas ao movimento do robô, objectos mais próximos geram maiores fluxos ópticos na imagem. Desta forma, simplifica-se a detecção de junções assim como a construção de mapas topológicos. As imagens guardadas continham informação adicional das imagens vizinhas, o que possibilitava seguir um corredor e, na união certa, trocar de corredor. O planeamento do caminho foi feito usando o algoritmo de Dijkstra, para que fosse percorrido sempre o caminho mais curto. No entanto, esta abordagem tem a desvantagem de ser computacionalmente muito pesada. Além disso, este método apenas permite a navegação em ambientes *indoor*, onde, em circunstâncias normais, a iluminação não sofre grandes alterações.

3.2 LOCALIZAÇÃO USANDO INFRAVERMELHOS E REDE NEURONAL

Mallot et al. [22] usam vistas de infravermelhos do ambiente para localizarem o robô num labirinto e implementar o planeamento do caminho, através de uma representação

topológica. O robô explora o ambiente através da detecção de objectos próximos, usando os sensores de infravermelhos. Quando um objecto é detectado na direcção do movimento do robô, este pára para evitar a colisão e guarda a sequência de leituras para referência futura.

As sequências de vistas são guardadas numa rede neuronal. A rede neuronal é treinada para associar a vista e movimento correntes com a próxima vista. Desta forma, o sistema consegue saber a partir da rede neuronal quais os locais que pode aceder a partir de uma dada localização. Os autores usaram este comportamento para fazer planeamento de percursos. Apresentando à rede neuronal imagens com os respectivos movimentos, é possível simular um caminho completo sem realmente o percorrer. Isto é muito parecido com a capacidade humana de pensar, planear e seguir mentalmente um certo percurso. Este último comportamento é algo muito semelhante ao que é feito com a SDM por Mendes et al [1].



to view \ from view	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	–	–	g_b	g_l	–	–	–	–
v_2	–	–	–	–	g_b	g_r	g_l	–
v_3	g_b	g_r	–	–	–	–	–	–
v_4	–	–	–	–	g_l	g_b	g_r	–
v_5	g_l	g_b	–	–	–	–	–	–
v_6	–	–	g_r	g_b	–	–	–	–
v_7	–	–	–	–	–	–	–	g_b
v_8	–	–	–	–	g_r	g_l	g_b	–

Figura 6 - (a) Representação Topológica. (b) Esquema correspondente de vistas. (c) Tabela de transição dos movimentos do robô. Retirado de [22].

Das sequências de vistas, Mallot constrói uma representação topológica para tarefas de planeamento de mais alto nível. A figura 6(a) exemplifica esta técnica. O primeiro esquema é a representação topológica de mais alto nível do labirinto. Os pontos p_i são os lugares do labirinto. Esses locais estão ligados pelos corredores c_i. Como os corredores podem ser percorridos em ambas as direcções, cada corredor está representado por dois arcos. A figura 6(b) representa o esquema de vistas correspondente. Este esquema é muito mais complexo, uma vez que cada local p_i na representação topológica pode originar vários nós no esquema de vistas. Esses nós correspondem todos ao mesmo local, mas de diferentes pontos de vista. A figura 6(c) representa a tabela de transição correspondente. A tabela sumariza os movimentos que foram aprendidos para se mover de uma vista para outra. A transição g_b significa andar para trás, g_f andar para a frente, g_r andar para a direita e g_l andar para a esquerda.

3.3 NAVEGAÇÃO BASEADA NA TRANSFORMADA DE FOURIER DAS IMAGENS

Outra implementação para a navegação robótica autónoma baseada em visão foi proposta por Ishiguro e Tsuji [23]. Ishiguro usa vistas omnidireccionais, mas, para aumentar a velocidade do processamento, as imagens são substituídas pelas suas transformadas de Fourier. As vistas omnidireccionais são periódicas ao longo do eixo de azimute (eixo das abcissas). Assim, Ishiguro aplica a transformada de Fourier a cada linha da imagem e então a imagem é substituída pela magnitude e fase dos coeficientes de Fourier, reduzindo consideravelmente tanto a memória como o processamento requeridos.

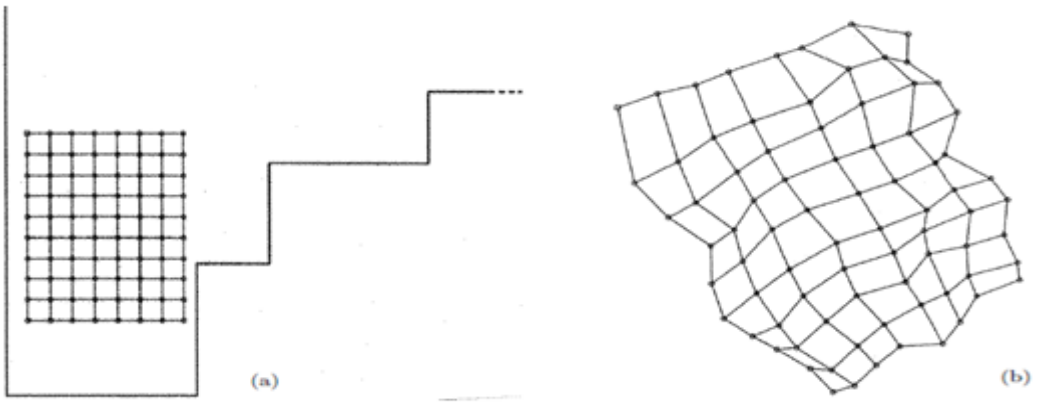


Figura 7 - (a) Pontos de observação. (b) Reconstituição da geometria do ambiente. Retirado de [23].

Para modelar a geometria do ambiente, Ishiguro define um conjunto de pontos de observação. Para cada ponto de observação, o robô captura uma vista I_i do ambiente circundante. Para cada nova imagem I_i aprendida é calculada a similaridade em relação a todas as imagens I_j já aprendidas. As imagens obtidas são organizadas através da similaridade entre si, dando assim uma ideia da geometria do ambiente (figura 7). A similaridade é dada por:

$$Si\left(I_i, I_j\right) = \sum_{y=0}^{l-1} \sum_{k=0}^{m-1} |F_{iy}(k) - F_{jy}(k)| \tag{3.4}$$

Na fórmula acima, $F_{iy}(k)$ e $F_{jy}(k)$ são os coeficientes de Fourier de frequência k da linha y das imagens I_i e I_j. A similaridade é interpretada como a medida da distância, portanto quanto maior a medida da similaridade, mais afastadas as imagens devem estar. Esta medida tem um valor máximo e, acima do limiar definido, a medida da similaridade é descartada, ou seja, assume-se que as imagens não têm qualquer relação.

Para iniciar o processo, o agente explora um ambiente desconhecido, capturando e guardando imagens nos locais de observação. De seguida, essas imagens são processadas e organizadas de acordo com as similaridades calculadas, levando assim a uma representação geométrica, como mostra a figura 8, que deve descrever correctamente o ambiente em termos de caminhos que podem ser percorridos. Assim, o robô fica apto para navegar de um ponto de referência para outro, escolhendo o caminho mais próximo de acordo com a distância entre imagens.

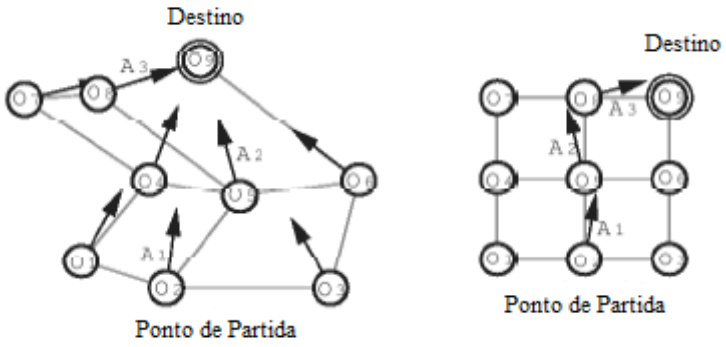


Figura 8 - Geometria recuperada e geometria real do ambiente. Retirado de [23].

3.4 LOCALIZAÇÃO BASEADA EM T-NET

A Target Network (T-Net), também implementada por Hiroshi Ishiguro [24], é, em muitos aspectos, parecida com a navegação descrita na secção anterior. A T-Net é semelhante ao modelo de mapeamento topológico, embora esta abordagem dependa de alguns pontos-chave, caminhos e intersecções entre caminhos, como mostra a figura 9.

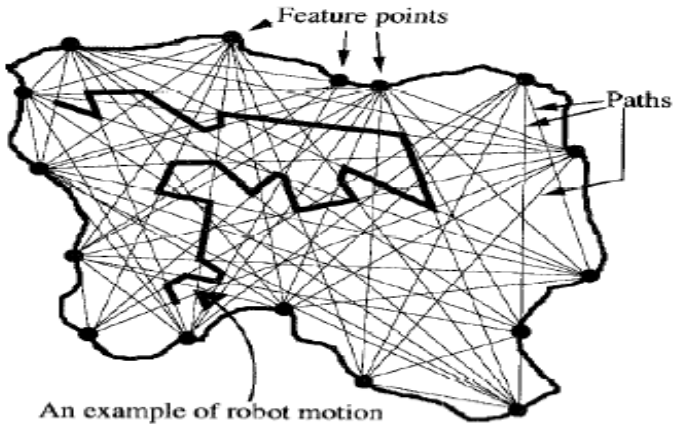


Figura 9 - Pontos característicos e caminhos de uma T-Net. Retirado de [24].

Nesta técnica, Ishiguro usa um robô com um sistema de visão catadióptrico (sistema que utiliza espelhos para permitir a captura de imagens com campo de visão de 360º graus do ambiente [25]). O robô navega pelos caminhos reconhecendo os pontos característicos dos caminhos. A relação angular entre os caminhos é calculada com base nas vistas omnidireccionais capturadas. Enquanto se movimenta, o robô escolhe pontos candidatos para a intersecção entre caminhos e explora o ambiente para verificar se as intersecções escolhidas são possíveis.

Se as intersecções são possíveis, são guardadas como pontos de passagem entre caminhos. A T-Net é constituída por um conjunto de pontos característicos, caminhos e intersecções, o que fornece uma base sólida para um planeamento rápido e eficiente planeamento do percurso e navegação.

A similaridade entre imagens dos pontos característicos é calculada por um algoritmo especial, conhecido como *Sequential Similarity Detection Algorithm* (SSDA). Porém, para fazer a correspondência entre possíveis pontos característicos, é necessário que as imagens que entram no algoritmo SSDA não tenham grande diferença de tamanho. Este problema foi resolvido por Ishiguro usando uma câmara com zoom. O factor de ampliação é decrementado à medida que o robô se aproxima dos pontos de interesse e é aumentado quando o robô se afasta deles.

Assim, pode dizer-se que a T-Net é um método interessante e robusto para uma navegação baseada em visão. No entanto, é algo confusa, além de que é também dispendiosa, visto que é necessário um robô móvel com boa capacidade de processamento e pelo menos duas câmaras com controlo de zoom. Um problema ainda em aberto é a definição dos pontos de interesse. Resumidamente, apesar desta abordagem parecer apelativa e robusta, traz um custo elevado de implementação e de funcionamento.

3.5 OUTRAS ABORDAGENS DA NAVEGAÇÃO BASEADA EM VISÃO

Uma abordagem muito parecida com a ideia de Matsumoto, descrita na secção 3.1, foi realizada por Stephen Jones et al. [26]. As imagens usadas neste sistema não são panorâmicas, mas de baixa resolução e com grande angular. A correlação entre imagens é feita usando o método *Zero mean Normalised Cross Correlation* (ZNCC). Para corrigir pequenos desvios durante o percurso, a trajectória do robô é ajustada de modo a maximizar-se a correlação entre a vista actual e as imagens guardadas pelo robô. É também usada informação odométrica para corrigir e estimar a posição do robô, tornando o sistema mais robusto.

Uma pesquisa similar a esta foi feita por Mao-Hai Li et al. [27]. Li usava partições de imagens e gerava uma árvore de características de *k* dimensões durante o estágio de

aprendizagem. A estimação era melhorada usando um filtro de partículas. Durante a navegação autónoma, as características eram comparadas usando a técnica do vizinho mais próximo.

Uma abordagem diferente foi feita por Niall Winters e Santos-Victor [28]. Estes autores propuseram um método de navegação baseado em visão usando imagens omnidireccionais. As imagens eram comprimidas usando *Principal Component Analysis* (PCA), produzindo assim um espaço de valores de pequena dimensão. Os valores próprios da imagem eram projectados no espaço dos valores próprios e ligados por um modelo de interpolação linear. Isto forma um *manifold*, no espaço dos valores próprios, que representa o caminho que o robô deve percorrer. O *manifold* é uma espécie de mapa topológico. O robô é então guiado com o objectivo de minimizar o ângulo entre o caminho seguido e o caminho desejado.

A abordagem de Winters e Santos-Victor é em muitos aspectos semelhante à abordagem seguida por Mathias O. Franz et al. [29]. No entanto, estes últimos não comprimiam as imagens. O *manifold* era constituído pelas próprias imagens. Para poupar processamento, apenas as imagens relevantes para a navegação eram guardadas, o que significa que apenas as imagens tiradas nos pontos de decisão eram preservadas. Os autores também consideraram o problema das vistas similares, que resolveram usando o histórico do robô, assumindo que o robô se move continuamente para a frente no espaço de navegação.

4. *HARDWARE* E *SOFTWARE*

Neste capítulo é apresentado o *hardware* e o *software* necessários para a realização desta tese.

São descritas as principais características do *hardware* utilizado, que consiste num robô X80Pro[4] controlado com um portátil. Na descrição do *software* começa-se por explicar a fase inicial do desenvolvimento do projecto, que começou com a ligação do código para controlo do robô e o código da SDM. É também mostrada a interface apresentada ao utilizador e feita uma breve descrição das suas principais características.

Por fim, dá-se uma ideia geral e uma breve apresentação do modo de funcionamento do *software* desenvolvido ao longo deste trabalho.

4.1 PLATAFORMA EXPERIMENTAL

Para a realização dos testes foi usado o robô X80Pro da Dr. Robot [30], mostrado na figura 10. Esta plataforma robótica é um veículo de condução diferencial, equipado com duas rodas motrizes de 18 cm de diâmetro, cada uma com um motor DC de 12 Volt, e uma roda Castor traseira, para dar estabilidade. O robô pesa 3,5 kg, podendo suportar uma carga adicional de 15 kg, e apresenta um diâmetro de 38 cm e uma altura de 25,5 cm. O X80Pro está equipado com um *encoder* com 800 pulsos por volta em cada roda e uma bateria de 7,2 Volt com capacidade de 3700 mAh, que permite uma autonomia de 3 horas.

Ao nível da comunicação, o robô tem integrado um sistema WiFi (802.11g) e apresenta a possibilidade de comunicação através de uma porta série.

Em relação aos sensores para a detecção de obstáculos, o robô tem 6 sensores de ultra-sons [31] e 7 sensores de infravermelhos [32] com um alcance máximo de 2,55 m e 0,80 m, respectivamente.

O X80Pro tem acoplada uma câmara de vídeo, localizada na frente do robô, que permite capturar imagens, até um máximo de 15 por segundo, com resolução 176×144 no formato BMP.

Além disto, o X80Pro está também equipado com outros tipos de sensores e acessórios, como um sensor de inclinação-aceleração, um sensor de temperatura, dois sensores de presença

[4] http://www.drrobot.com

de humanos e um LCD de 128×64 píxeis. Tem ainda um controlador de multimédia com um DSP de 120 MIPS de 16-bit com 256k×16 bits de SRAM e um controlador para a parte sensorial e de movimento com um DSP/MCU de 40 MIPS de 16-bit com 2.5k×16 bits de SRAM.

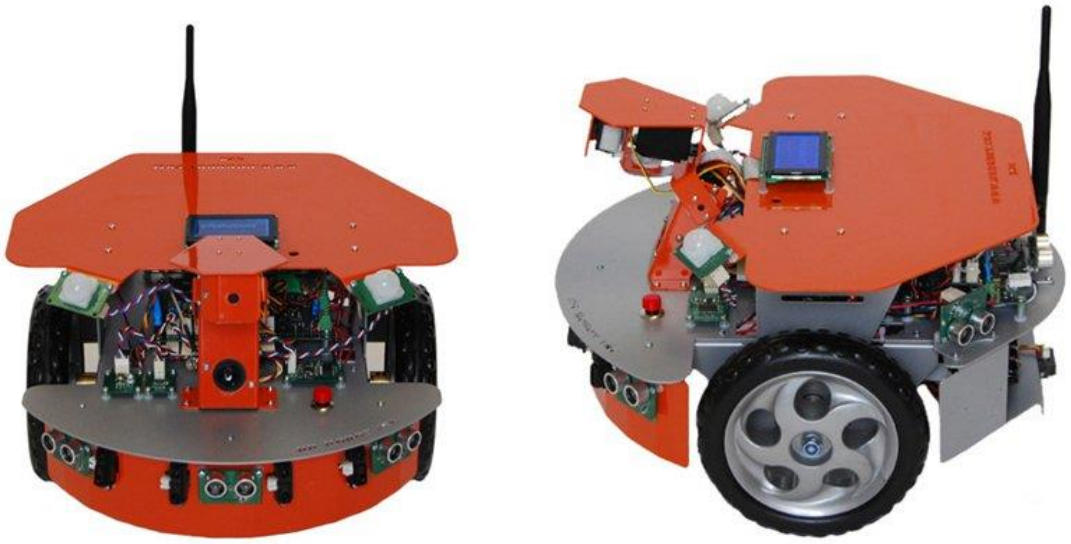


Figura 10 – Robô usado, X80Pro. Imagem retirada de [30].

O robô transporta um portátil ASUS K55VJ, com um processador Intel Core i7 a 2.40 Ghz, 6 Gb de memória RAM e uma GPU NVIDIA GeForce GT 635M de 2 Gb com 96 CUDA cores, que irá realizar todo o processamento necessário para a navegação bem como o interface com o utilizador (figura 11). O computador corre o sistema operativo Windows 7 e o *software* para o desenvolvimento deste projecto foi escrito em C++. A comunicação entre o robô e o computador é feita através de uma porta série (RS 232).

Figura 11 - Robô X80Pro e o seu PC portátil, durante um teste nos corredores do DEEC.

4.2 *SOFTWARE*

Antes de se começar a desenvolver o *software* necessário para atingir os objectivos propostos para esta tese, o primeiro passo foi integrar o código existente, SDM implementada numa CPU, explicada na secção 2.7, com o *sample code* fornecido pela Dr. Robot para o controlo do robô. O passo seguinte foi a escolha de uma biblioteca que permitisse fazer a conversão das imagens captadas pelo robô no formato BMP para o formato PGM, uma vez que este é o formato usado na SDM implementada anteriormente. Foi escolhida a biblioteca FreeImage [33], uma biblioteca *open source* que, além de simples de usar, não trouxe problemas ao ser integrada no projecto.

Neste ponto do trabalho, era já possível usar a SDM adaptada ao robô X80Pro.

De seguida, incorporou-se o código da SDM para a GPU, desenvolvido em [3], explicado na secção 2.8, de modo a que todas as partes do código mantivessem as suas funcionalidades.

A integração deste novo desenvolvimento veio trazer uma maior velocidade no processamento dos dados em tempo real, melhorando deste modo a navegação autónoma. No entanto, este método, SDM implementada numa GPU, ainda não tinha sido testado para navegação robótica, pelo que este trabalho é também neste aspecto uma inovação.

O trabalho foi desenvolvido no ambiente de programação Microsoft Visual Studio 2010, na linguagem C++, usando as Classes Fundamentais da Microsoft (MFC) [34], de modo a permitir ao utilizador interagir com o programa através de uma interface gráfica. Este ambiente de programação permite que seja feita a ligação do código desenvolvido em CUDA para a GPU com o código para o controlo do robô e navegação do robô.

Neste trabalho foi desenvolvido todo o *software* necessário para o processo de aprendizagem, que passa por conduzir o robô e enviar posteriormente a informação relevante para a SDM, bem como todo o *software* que permite a navegação autónoma, através de informação obtida da SDM, algoritmos de correcção da trajectória, desvio de obstáculos e melhorias ao método de predição da SDM (figura 12).

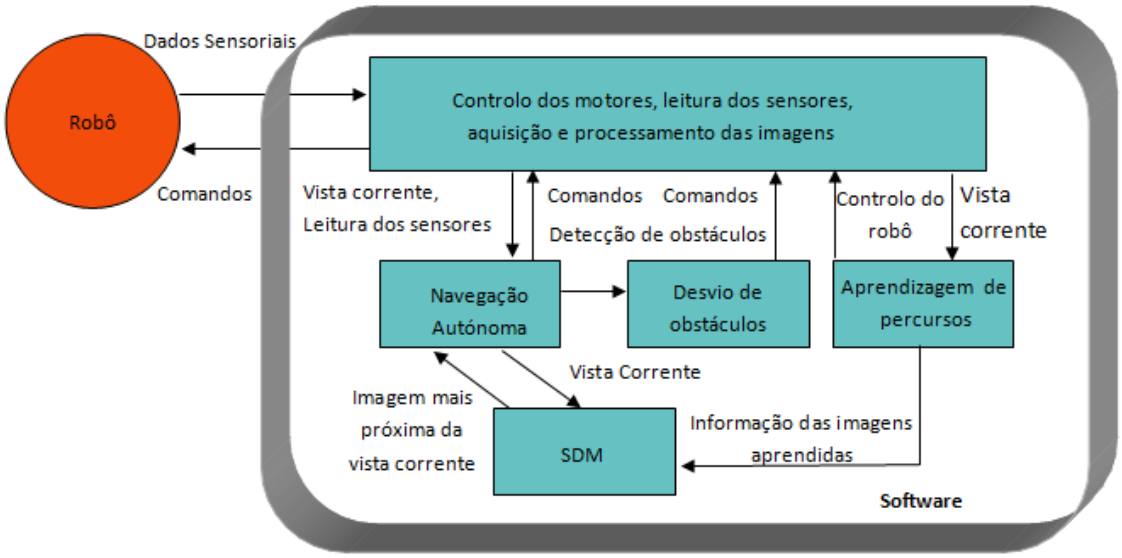


Figura 12 – Arquitectura do *software* desenvolvido.

A interface implementada (figura 13) permite ao utilizador ensinar o robô ao longo de diferentes percursos, escolher os percursos a carregar para a SDM, bem como iniciar e parar a navegação autónoma ou a vigilância automática (modo vigilante). Durante as várias fases de execução, o programa apresenta ao utilizador diversa informação útil, como por exemplo se o robô se encontra a desviar-se de um obstáculo, o caminho actual que está a percorrer, o vector de dados da imagem predita pela SDM, o número de imagens carregadas para a memória, o raio de activação, a velocidade do robô em cada instante em cm/s, a imagem corrente, a leitura de diversos sensores ou se chegou ao fim do percurso. Para ajudar ainda o utilizador enquanto este ensina o caminho ao robô, a interface mostra também se o robô se encontra ou não com velocidade angular, assim como a velocidade em cada roda durante a aprendizagem.

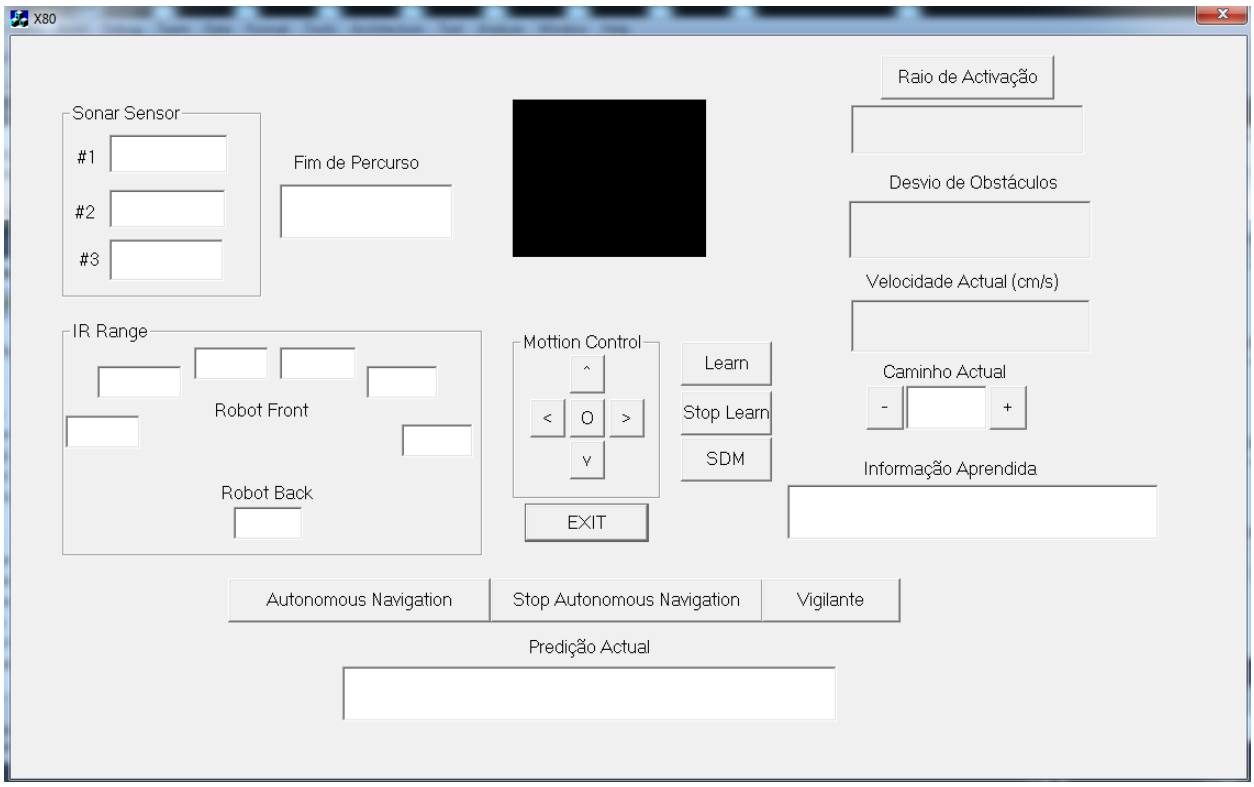


Figura 13 - Interface gráfica desenvolvida para o robô.

4.3 NAVEGAÇÃO

A navegação implementada pode ser dividida em duas partes distintas, a aprendizagem e a navegação autónoma. A aprendizagem é supervisionada e, durante este processo, o utilizador conduz manualmente o robô que guarda automaticamente a informação necessária para a navegação autónoma posterior. Na navegação autónoma, o robô segue o caminho previamente ensinado usando a informação guardada na SDM. Foi também implementado um módulo de desvio de obstáculos, para que o robô, durante a navegação autónoma, seja capaz de detectar e ultrapassar objectos imprevistos que surjam no seu percurso.

5. IMPLEMENTAÇÃO E RESULTADOS DA NAVEGAÇÃO AUTÓNOMA

Neste capítulo são descritas as várias implementações dos algoritmos desenvolvidos e apresentados alguns resultados dos testes de navegação autónoma. Explica-se o modo de condução durante a aprendizagem, o formato dos dados a ser guardados na SDM e é dada uma breve explicação da interface apresentada ao utilizador.

De modo a resolver o problema principal da navegação baseada em visão, no qual o robô se perde em percursos com zonas idênticas, foram desenvolvidos dois novos algoritmos de navegação baseada em visão e no uso da SDM numa GPU. O primeiro foi o algoritmo com janela deslizante de largura fixa, que apenas permite que o robô procure a imagem mais próxima seguinte num número limitado de imagens em torno da imagem predita anteriormente. Apesar de resolver o problema anterior, este algoritmo não resolve o problema do robô raptado. Assim, foi ainda implantado um algoritmo baseado em janela deslizante que permite que a pesquisa seja feita em toda a memória quando existem várias predições sucessivas que indicam que o robô está noutra zona do percurso. Estes dois algoritmos apresentaram resultados bastante positivos na navegação deste robô.

Por fim, é apresentado o modo vigilante, que permite ao robô percorrer um trajecto de forma cíclica vezes sem conta por chegar ao fim do percurso, reconhecer novamente as imagens iniciais.

Por falta de espaço, é apresentada no anexo A a identificação dos percursos relativos aos testes apresentados, bem como outros testes realizados.

5.1 APRENDIZAGEM – CONDUÇÃO

Nesta etapa, o utilizador conduz o robô ao longo de um percurso. Para o utilizador iniciar este processo tem de actuar no botão “Learn” na interface, mostrada na imagem 13. A condução do robô é feita através dos botões do rato da seguinte forma: para o robô se mover em frente, pressiona-se o botão esquerdo, ou seja, é aumentada, de igual modo, a velocidade de ambas as rodas. Do mesmo modo, para se reduzir a velocidade, pressiona-se o botão direito do rato, o que fará reduzir a velocidade em ambas as rodas. Nestas duas situações, o robô tem apenas velocidade linear, a velocidade angular tem valor 0.

Para alterar a orientação do robô, pressiona-se o botão do meio do rato, que faz alternar entre o modo de condução linear e o modo de condução angular. A partir deste momento, o botão esquerdo do rato fará com que a velocidade da roda esquerda diminua, ou seja, o robô vira para a esquerda. De igual modo, o botão direito faz com que a velocidade da roda direita diminua de modo a que o robô vire para a direita. Neste modo de condução o robô tem velocidade angular. Para voltar ao modo anterior, basta premir novamente o botão do meio do rato e o robô irá deslocar-se com velocidade igual em ambas as rodas. Esta velocidade é igual à menor velocidade positiva entre as velocidades de cada roda.

Durante todo o processo de aprendizagem, o robô adquire imagens no formato BMP com resolução 176×144, que são convertidas para o formato PGM. Cada imagem (figura 14) é guardada em disco com o nome S#Img#d#e#.pgm:

- S# (número de Sequência) – # que define a sequência a que a imagem pertence.

- Img# (número de imagem) – #identifica o número da imagem dentro da sequência.

- d#e# (velocidade) – #corresponde ao valor da velocidade da roda direita e esquerda, respectivamente, lidas do robô no momento em que a imagem foi capturada.

O nome da imagem contém, assim informação necessária para a navegação autónoma e corresponde à informação a guardar na SDM no vector de dados.

Figura 14 – Exemplo de uma imagem capturada pelo robô durante a aprendizagem.

Para o utilizador terminar a aprendizagem, isto é, definir o fim do caminho, actua no botão “Stop Learn” na interface, o que fará parar o processo de aprendizagem assim como o robô. Para garantir que o robô reconhece correctamente o fim do percurso, é boa prática reduzir gradualmente a velocidade na parte final do percurso.

5.2 APRENDIZAGEM – CARREGAMENTO DE DADOS PARA A SDM

Neste processo, todas as imagens aprendidas são passadas para a SDM, para que seja possível, na navegação autónoma, percorrer os caminhos previamente ensinados. Para isso, durante a aprendizagem, as localizações de todas as imagens aprendidas são guardadas num ficheiro de texto, sendo deste modo possível ler os valores de todos os píxeis de todas as imagens para a SDM, que correspondem ao vector de endereços, assim como o nome de cada imagem, que contem informação relevante, correspondente ao vector de dados.

Se o utilizador ensinou mais que um caminho ao robô, então pode ir adicionando os caminhos que pretende carregar para a SDM. Como as imagens e a sua respectiva informação adicional estão guardadas em disco, é possível ensinar um caminho e percorrê-lo mais tarde. Uma vez aprendidos um ou vários caminhos, o utilizador escolhe o caminho que pretende e, de seguida, actua no botão "SDM" que irá carregar os dados aprendidos, do caminho seleccionado, para a SDM.

5.3 NAVEGAÇÃO AUTÓNOMA – ALGORITMO BÁSICO

Esta fase do algoritmo (figura 15) é responsável por conduzir autonomamente o robô desde o início até ao fim de um percurso, previamente ensinado, com base nas imagens capturadas durante a aprendizagem.

Para iniciar a navegação autónoma, o utilizador actua sobre o botão "Autonomous Navigation", iniciando-se assim o reconhecimento do percurso.

O robô começa por capturar uma imagem, no formato BMP, (ponto 1 do algoritmo da figura 15) que, de seguida, é convertida para o formato PGM (ponto 2). Depois, esta imagem é enviada para a SDM para ser comparada com todas as imagens aprendidas, existentes na memória, e a SDM devolve a imagem mais próxima, ou seja, a imagem classificada como a mais semelhante à imagem capturada, (ponto 3). O processo de comparação de imagens é executado na GPU [3] utilizando técnicas de processamento paralelo de modo a torná-lo o mais rápido possível, visto que é nesta fase que o processamento é mais exigente.

Algoritmo Básico de Navegação

1. Tira Foto
2. Converte Foto
3. SDM_Predict (clue) - Retorna vector de dados da imagem mais próxima
3.1. Se a sequência da imagem predita é diferente da sequência actual
    Volta para o passo 1 (ignora a informação vinda da SDM)
    Se durante 30 predições consecutivas a sequência da imagem predita for diferente da actual
    Então o robô está a navegar noutra sequência que passa a ser reconhecida como actual
3.2. Se o número da imagem predita está fora da janela
    Volta para o passo 1 (ignora a informação vinda da SDM)
    Se durante 30 predições consecutivas imagem predita estiver fora da janela
    Então o robô está a navegar numa zona diferente do percurso
4. Adquire velocidade das rodas a partir do vector de dados da imagem mais próxima
5. Calcula e corrige o deslocamento horizontal
6. Se está no fim do percurso
    Termina a execução do algoritmo
7. Impõe nas rodas as velocidade adequadas
    Volta para o passo 1

Figura 15 - Algoritmo básico de navegação.

De modo a reduzir-se o mais possível os erros na trajectória e na orientação do robô, a informação guardada no vector de dados é usada para se fazer alguma filtragem e contextualização antes de se passar para a actuação nas rodas do robô. Para isso, começa por verificar-se se a sequência a que a imagem pertence é a mesma que foi seleccionada pelo utilizador (ponto 3.1), ou seja, no caso de o utilizador ter ensinado vários caminhos, ao seleccionar o caminho pretendido é esperado que o robô se mantenha nesse caminho até ao fim. Contudo, em determinadas situações, o robô pode detectar imagens de outras sequências se as imagens aprendidas de uma determinada zona da sequência forem muito idênticas às de outra sequência ou no caso de o robô ser "raptado" para uma zona de uma sequência diferente.

Se a sequência da imagem devolvida pela SDM for diferente da sequência que o utilizador escolheu, então a informação dessa imagem é ignorada e é adquirida uma nova imagem, realizando-se o mesmo teste. Se o teste for negativo trinta vezes consecutivas para uma mesma sequência diferente da actual, assume-se então que o robô está a navegar noutra sequência, a qual passa a ser reconhecida como a sequência actual. Este comportamento permite

filtrar erros de predição da SDM, sem comprometer a capacidade do robô de resolver o problema do "robô raptado".

Após a verificação de que o robô está a navegar na sequência correcta, atesta-se se a imagem mais próxima dada pela SDM está relativamente perto, na sequência, da imagem anterior, o que é normal, pois não se espera que o robô salte partes do percurso mas sim que o vá seguindo gradualmente do início ao fim (ponto 3.2). Para isso, o número da última imagem predita pela SDM é comparado com o número da imagem da predição actual. Para isso, começa por se criar uma janela com dimensão de um terço do tamanho total da sequência e, de seguida, verifica-se se a distância entre o número imagem predita anteriormente e o número da imagem predita actual, através da diferença do número de cada imagem, é superior ao tamanho da janela. Se isto se confirmar, assume-se que esta predição está errada e portanto o comando relativo a essa imagem é ignorado. No entanto, se durante 30 predições consecutivas a imagem mais próxima estiver fora da janela, assume-se que o robô está noutra parte do percurso, e então começa a seguir novamente as imagens sugeridas pela SDM. Este processo pode ser visto como uma memória de "curta duração", uma vez que o robô sabe em que parte do percurso estava no passado recente e não é esperado que em pouco tempo tenha passado para uma zona muito mais à frente ou atrás no percurso. Se tal acontecer, isto é conhecido como o "robô raptado", pois pode ter sido movido por alguém para outra parte do percurso. No entanto, o problema de reconhecer imagens distantes, na sequência, da imagem predita anteriormente também é de grande importância quando o robô navega por locais muito semelhantes, como por exemplo os corredores de grandes edifícios que são muito parecidos em locais diferentes. Desta forma, o robô pode assumir que foi raptado, quando na realidade está a detectar como imagem mais próxima, imagens de outra zona do percurso idêntica à zona actual. Apesar de ser parecido com o robô raptado, uma vez que imagens preditas sucessivamente se encontram afastadas entre si, este é um problema relevante, uma vez que se um percurso tiver locais parecidos em diferentes zonas há uma grande probabilidade de o robô se perder. Uma forma de se resolver este problema é recorrer-se à memória de "curta duração".

Depois de efectuados estes dois testes, adquire-se a velocidade das rodas que o robô tinha naquela zona do percurso durante a aprendizagem, a partir do vector de dados, obtido da SDM, da imagem mais próxima em relação à vista corrente (ponto 4).

De modo a que o robô pare quando está na zona de fim de percurso, foi implementado um método para garantir a paragem, do robô e do algoritmo de navegação, e informar o utilizar de que a navegação autónoma parou uma vez que o percurso ensinado acabou ali (ponto 5). De facto, o robô considera que está em fim de percurso se a imagem predita estiver entre as últimas 10% aprendidas e a velocidade correspondente a essa imagem for uma velocidade reduzida. Os

resultados obtidos com este método mostraram-se bastante favoráveis, uma vez que o robô reconhece facilmente o fim o percurso.

Para corrigir pequenos desvios na trajectória de modo a manter o robô o mais próximo possível do percurso aprendido, é calculado o deslocamento horizontal do robô relativamente ao percurso ensinado. O método para o cálculo do deslocamento horizontal é baseado no método usado por Y. Matsumoto et al [20] descrito em 3.1 (ponto 6). No nosso caso, foi usada uma janela com largura de 16 píxeis por usarmos uma resolução de imagens superior.

Este método permite corrigir, em cada predição, pequenos desvios na trajectória do robô. Se a vista corrente estiver deslocada para a esquerda relativamente à imagem mais próxima, a velocidade da roda direita é reduzida. Do mesmo modo, se estiver deslocada para a direita, então é diminuída a velocidade da roda esquerda. Esta diminuição é tanto maior quanto maior for a diferença de píxeis na horizontal entre as imagens, até um máximo de 16 píxeis para um dos lados. Neste caso, apenas é considerada a distância horizontal, uma vez que não há deslocamento na vertical, visto que a câmara é mantida numa posição fixa de modo a ficar paralela ao chão e direccionada para a frente.

Após todas as verificações descritas acima, actua-se nas rodas do robô, com as velocidades adequadas para que este percorra o trajecto de forma mais idêntica possível ao trajecto aprendido (ponto 7). Este procedimento vai sendo repetido até ao fim do percurso, a cerca de 10 vezes por segundo.

Os primeiros testes revelaram que, em alguns percursos, o robô perdia-se antes de chegar ao destino. O principal problema durante a navegação básica prendia-se com o facto de a localização obtida através da SDM ser errada, devido às várias zonas idênticas do percurso. Muitas vezes, a imagem mais próxima em relação à vista corrente obtida da SDM referia-se a uma zona diferente do percurso, embora idêntica, fazendo com que o robô não chegasse ao destino.

Para a análise dos resultados da navegação, e construção dos gráficos das predições apresentados abaixo, o número de cada imagem predita pela SDM é guardado num ficheiro de texto.Com esta informação guardada durante a navegação autónoma, torna-se fácil traçar um gráfico elucidativo da progressão do robô ao longo do trajecto, no qual o número da imagem predita corresponde à localização estimada pela SDM, (depois de aplicados os filtros descritos) e o número da iteração corresponde ao tempo. Nestes gráficos, é esperado que o número da imagem predita vá aumentando gradualmente até se atingir o fim do percurso.

O primeiro teste foi efectuado no piso 1 do DEEC, entre o laboratório 1.18 e o gabinete 1 (figura 34 do anexo A). A planta do piso 1 bem como o traçado do percurso são apresentados no anexo A por falta de espaço.

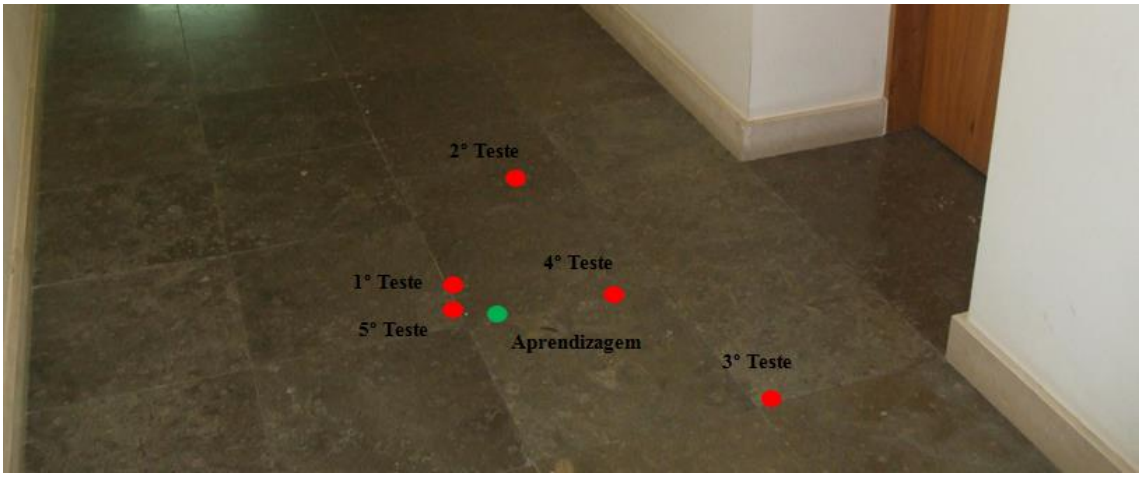


Figura 16 – Localização do robô no fim da navegação autónoma relativa ao percurso entre o laboratório 1.18 e o gabinete 1.

Para se avaliar a performance do algoritmo básico de navegação, o percurso referido acima foi percorrido 5 vezes consecutivas, tendo o robô em todas as tentativas alcançado o objectivo (figura 16). Como pode ser verificado na figura 17, para este percurso, o algoritmo básico para a navegação autónoma foi suficiente para orientar o robô ao longo do trajecto, uma vez que à medida que aumenta o número da iteração, correspondente ao tempo, aumenta também o número da imagem predita, até à imagem final, comprovando assim o avanço do robô no percurso.

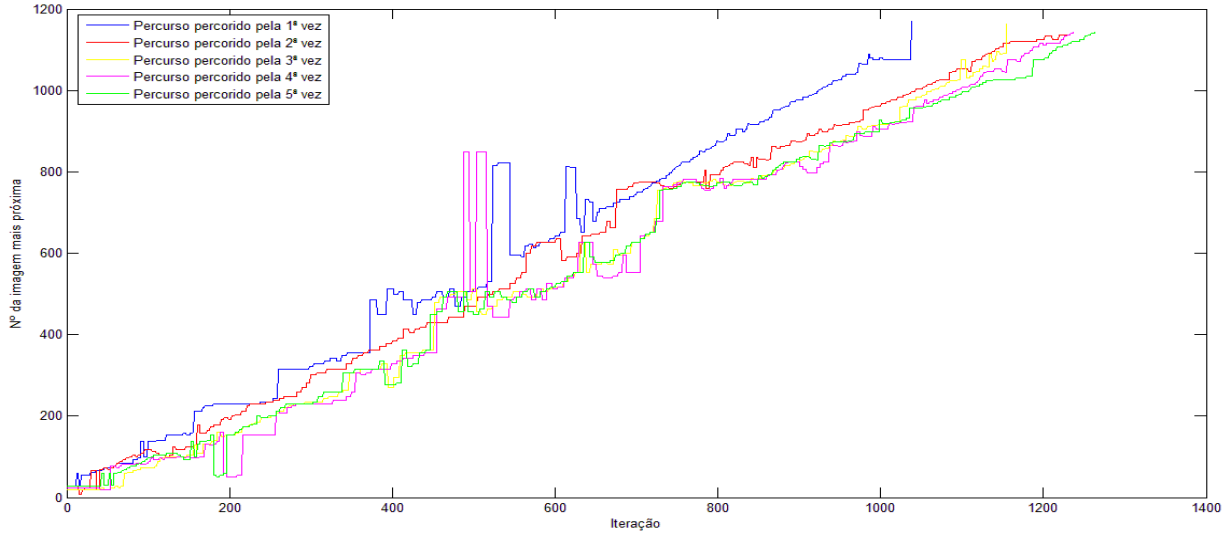


Figura 17 - Gráfico das predições relativas ao percurso entre o laboratório 1.18 e o gabinete 1.

No gráfico, ao longo do trajecto, isto é, com o aumento do número de iterações, observam-se vários picos, que surgem porque a imagem na SDM considerada mais próxima da vista corrente está afastada da imagem predita anteriormente. Isto ocorre em percursos que

apresentam várias zonas idênticas, tendo em conta que o algoritmo base faz a pesquisa em toda a memória.

Para além deste teste, foram realizados vários outros, de modo testar a robustez deste algoritmo. Outro dos testes realizados foi efectuado entre os elevadores das zonas R e S (figura 18 e 35 do anexo A). Neste ensaio, o trajecto foi percorrido apenas duas vezes, tendo o robô alcançado em ambos o destino (figura 36 do anexo A), como se observa no gráfico seguinte.

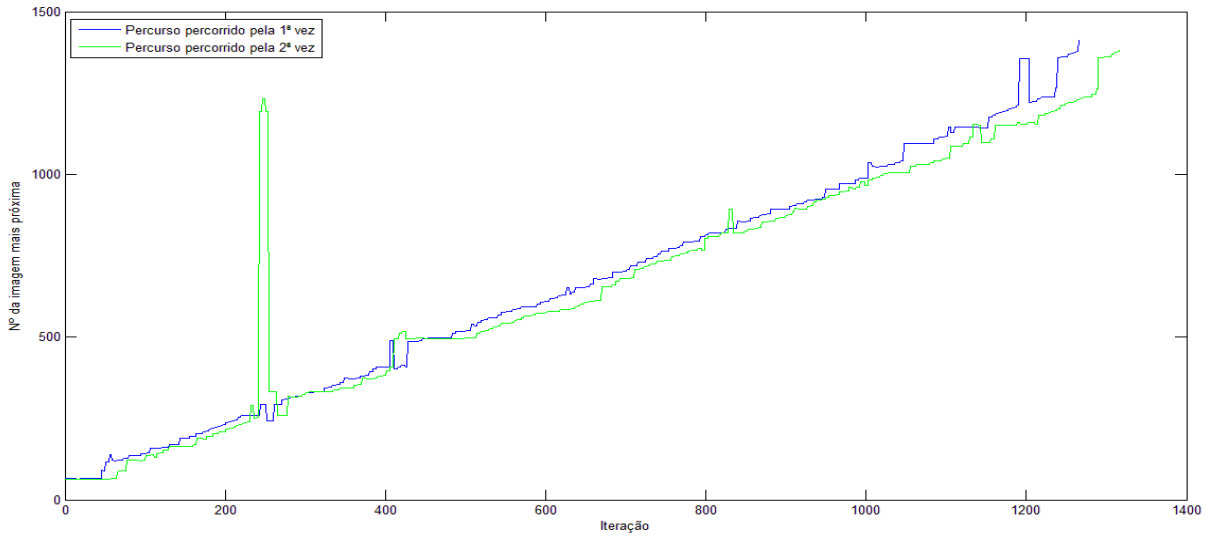


Figura 18 - Gráfico das predições relativas ao percurso entre os elevadores as zonas R e S.

O percurso relativo ao gráfico abaixo (figura 19 e 37 do anexo A) corresponde à navegação entre o elevador da zona S e o laboratório 1.18. Aqui, o robô não atingiu o fim do percurso, perdendo-se na primeira vez junto ao gabinete 1.3 e na segunda junto ao gabinete 1.5. Em ambas as situações, o robô aproximou-se demasiado da parede.

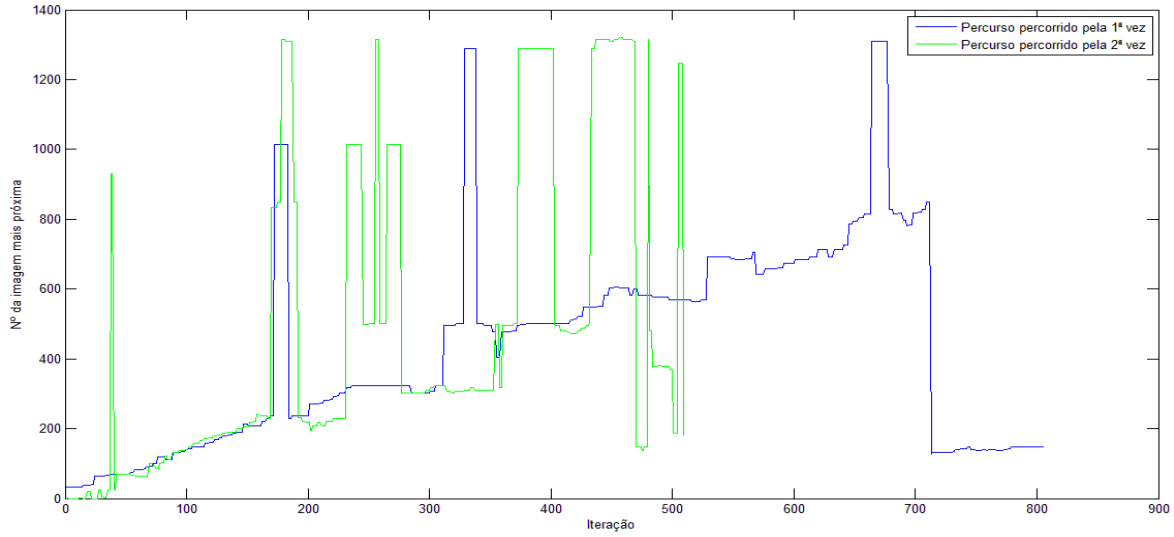


Figura 19 - Gráfico das predições relativas ao percurso entre o elevador da zona S e o laboratório 1.18.

Como se verifica nos resultados acima, a imagem predita pela SDM está frequentemente muito afastada da imagem da predição anterior. Então, de modo a ultrapassar este problema, passou a fazer-se um pré-processamento da informação a ser guardada na SDM e foi implementado o algoritmo da janela deslizante de largura fixa.

5.3.1 PRÉ-PROCESSAMENTO DOS DADOS ENVIADOS PARA A SDM

Uma forma de melhorar a navegação do robô foi através da melhoria da qualidade das imagens por equalização. Para isso, calcula-se o histograma da imagem e obtém-se depois o histograma cumulativo, que, por fim, é equalizado. Este processo melhora a qualidade da imagem através de um aumento do contraste da imagem original.

Outra melhoria introduzida consiste em carregar para a SDM apenas as imagens aprendidas relevantes para a navegação. Considera-se que uma imagem contém informação relevante quando é suficientemente diferente da imagem anterior, ou seja, a diferença entre duas imagens consecutivas é superior ao raio de activação da SDM. O raio de activação é calculado actuando-se no botão "Raio de Activação" da interface. Quando este botão é pressionado o robô captura 3 imagens consecutivas, faz a equalização das três e calcula então a diferença média entre elas. Este valor será usado como raio de activação. Apesar de todas as imagens aprendidas continuarem guardadas em disco, apenas as imagens relevantes são guardadas na SDM. As imagens enviadas para a SDM recebem ainda um novo número, de modo a que todas as imagens na memória fiquem numeradas sequencialmente.

Este processo de melhoria e filtragem das imagens revelou-se bastante benéfico, tanto no processo de carregamento de informação para a SDM, reduzindo substancialmente o tempo de carregamento das imagens, como durante a navegação, uma vez que o número de imagens a serem comparadas com a vista corrente é menor, o que se traduz num menor tempo entre cada predição e numa navegação mais precisa. Isto tem importância tanto maior quanto maior for o percurso, pois se o número de imagens sem informação relevante for, por exemplo, 10 numa sequência, ao fim de 2000 predições foram feitas menos 20000 comparações.

Contudo, e apesar de apresentar melhorias durante o processo de localização do robô, este aperfeiçoamento não resolve o principal problema que surgia quando o robô navegava por locais muito idênticos em fases distintas do percurso. No entanto, aumenta a performance da SDM.

5.4 NAVEGAÇÃO AUTÓNOMA COM BASE EM JANELA DESLIZANTE DE LARGURA FIXA

A principal fonte de erros no processo de localização do robô deve-se ao facto de, ao fazer-se a pesquisa em toda a memória, quando existem zonas muito idênticas no percurso, haver uma grande possibilidade da localização do robô prevista pela SDM ser errada.

Para se resolver este problema, implementou-se um método que apenas usa as imagens mais próximas da última predição para a localização do robô. Assim é eliminado o problema de o robô se confundir quando o percurso tem várias zonas idênticas, uma vez que apenas as imagens da zona onde o robô se encontra são usadas para a navegação.

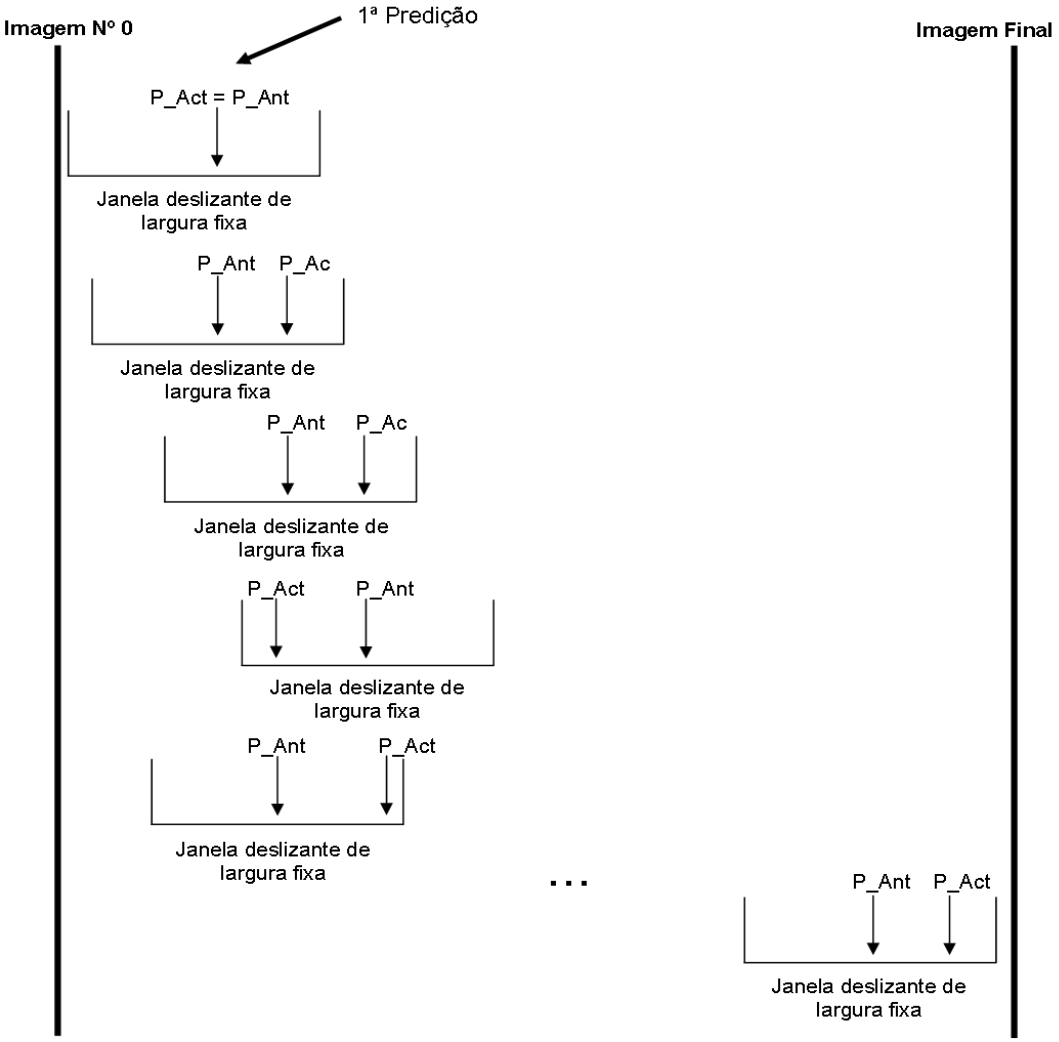


Figura 20 – Esquema de navegação autónoma com janela deslizante de largura fixa.

Neste método, é criada uma janela deslizante de largura fixa de modo a que a próxima imagem predita só possa estar, no máximo, 20 imagens à frente ou atrás da última imagem obtida da SDM (figura 20). Deste modo é resolvido o problema de o robô confundir a localização corrente com outras parecidas, mais afastadas, do mesmo percurso. Se, por exemplo,

a última imagem predita pela SDM for a imagem número 50, a próxima pesquisa é feita apenas entre as imagens número 30 e número 70, resolvendo assim o problema de, por exemplo, as imagens de 1000 a 1010 serem muito parecidas com as da zona actual. Esta filtragem faz todo o sentido nesta implementação, uma vez que é esperado que o robô vá percorrendo os percursos gradualmente do início até ao fim, resolvendo-se assim o principal problema da navegação com o algoritmo básico.

Este algoritmo apresenta bons resultados, uma vez que o problema mais importante, relacionado com o facto de o robô assumir que está noutra parte do percurso em percursos muito idênticos, está resolvido com a implementação da janela deslizante de largura fixa, pois esta obriga a que a imagem mais próxima esteja relativamente perto da imagem mais próxima anterior. Porém, esta abordagem torna-se muito rígida, dado que obriga o robô a manter-se sempre dentro da zona confinada pela janela em todas as predições, o que não permite tratar o problema do robô raptado, por exemplo. Com este algoritmo, se o robô for mudado para um local mais atrás ou à frente do percurso, este irá provavelmente perder-se, pois a janela deslizante de largura fixa não permite saltos no percurso.

De modo a comprovar-se que este novo algoritmo consegue ultrapassar os problemas que surgiram com a navegação básica, foi efectuado novamente o percurso entre o elevador da zona S e o laboratório 1.18. Com este algoritmo, o robô foi capaz de percorrer todo o percurso aprendido, como se pode verificar na figura 21 (e figura 37 do anexo A).

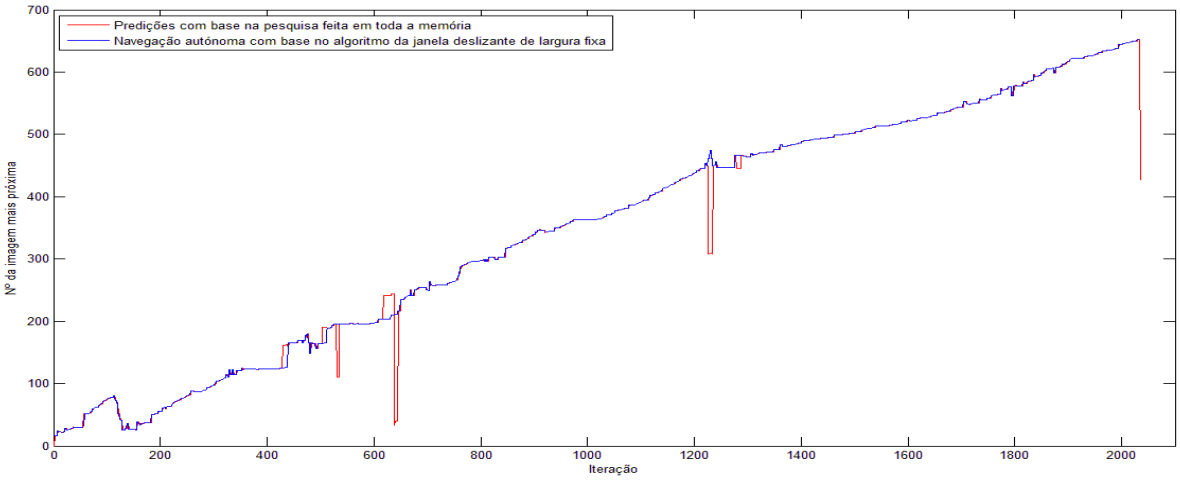


Figura 21 - Gráfico das predições relativas ao percurso entre o elevador a zona S e o laboratório 1.18, usando a janela deslizante de largura fixa.

Agora, para a construção do gráfico das predições, são apresentados 2 traçados distintos: o azul corresponde à navegação autónoma com base nas predições do algoritmo da janela deslizante de largura fixa, enquanto o traçado vermelho é referente às predições fazendo a pesquisa em toda a memória.

De seguida, foi efectuado outro teste, mas agora no sentido contrário, ou seja, do laboratório 1.18 até ao elevador da zona S, mostrado na figura 22 (e figura 37 do anexo A).

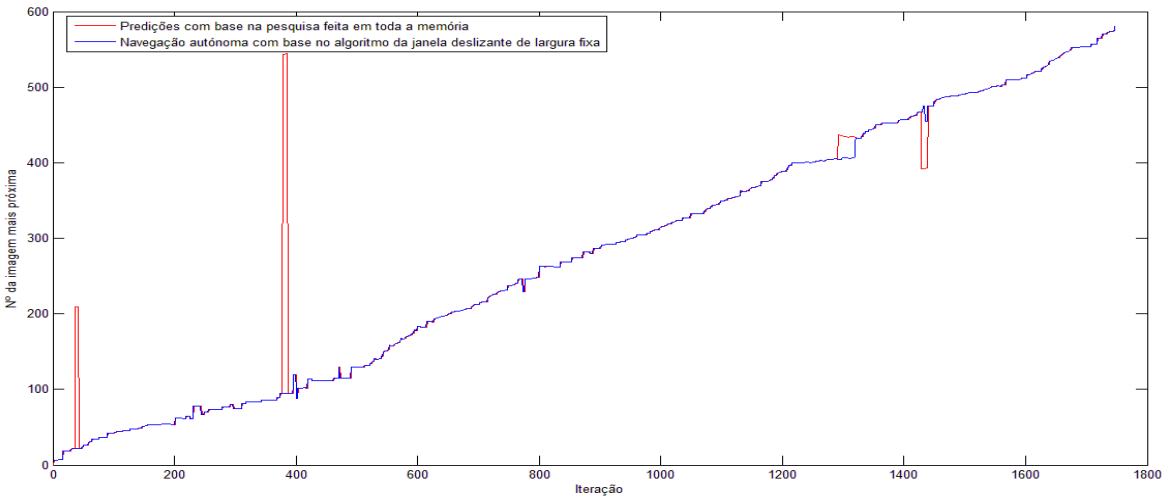


Figura 22 - Gráfico das predições relativas ao percurso entre o laboratório 1.18 e o elevador da zona S, usando a janela deslizante de largura fixa.

Como se comprova pelos resultados acima, este algoritmo consegue eliminar o principal problema da navegação autónoma ao não permitir que a imagem predita esteja muito afastada da imagem predita anteriormente.

Apesar desta última implementação apresentar resultados favoráveis, ao resolver o principal problema, além de ser um algoritmo robusto, trata-se de uma abordagem muito rígida que obriga o robô a restringir-se à informação das imagens que existem dentro da janela. Assim, de modo a contornar o problema da rigidez da janela deslizante de largura fixa, foi implementado um novo método para a navegação autónoma.

5.5 NAVEGAÇÃO AUTÓNOMA COM BASE NA JANELA DESLIZANTE

A ideia base deste método é manter a janela deslizante, que apresentou melhorias significativas na navegação autónoma e, ao mesmo tempo, permitir saltos durante o percurso, quando estes realmente acontecem, como no caso do "robô raptado". O novo método implementado permite então que a pesquisa seja efectuada em toda a memória, de maneira a que em determinadas situações (por exemplo "robô raptado") a própria janela deslizante se mova para zonas distintas do percurso (figura 23).

Neste novo método, a pesquisa pela imagem mais próxima é feita usando toda a informação presente na memória. Verifica-se depois se a imagem predita está dentro da janela, que é o esperado. Se a imagem mais próxima se encontrar dentro da janela, actua-se nos motores

usando a informação da imagem predita. Se a imagem predita estiver fora da janela, é feita uma nova pesquisa, apenas nas imagens dentro da janela, e é o resultado desta pesquisa que irá ser usado para a navegação do robô. Por exemplo, se a imagem obtida da predição anterior é a número 50 e se a predição actual reconheceu a imagem mais próxima como sendo a número 1000, então é inicialmente feita uma nova pesquisa apenas entre as imagens 30 a 70. No entanto, se durante 50 predições consecutivas a imagem mais próxima, em toda a memória, estiver fora da janela, assume-se que a informação obtida da memória global está correcta, e então a janela passa a estar centrada na imagem obtida da pesquisa em toda a memória.

A dimensão da janela foi mantida da implementação anterior, que permitia uma pesquisa nas 40 imagens mais próximas. Assim, tal como na implementação anterior, obtiveram-se bons resultados com esta nova abordagem. Este algoritmo combina o objectivo da janela deslizante de largura fixa com a capacidade do robô se localizar depois de ter sido movido (raptado) para um local diferente do percurso, sendo deste modo um método muito mais flexível.

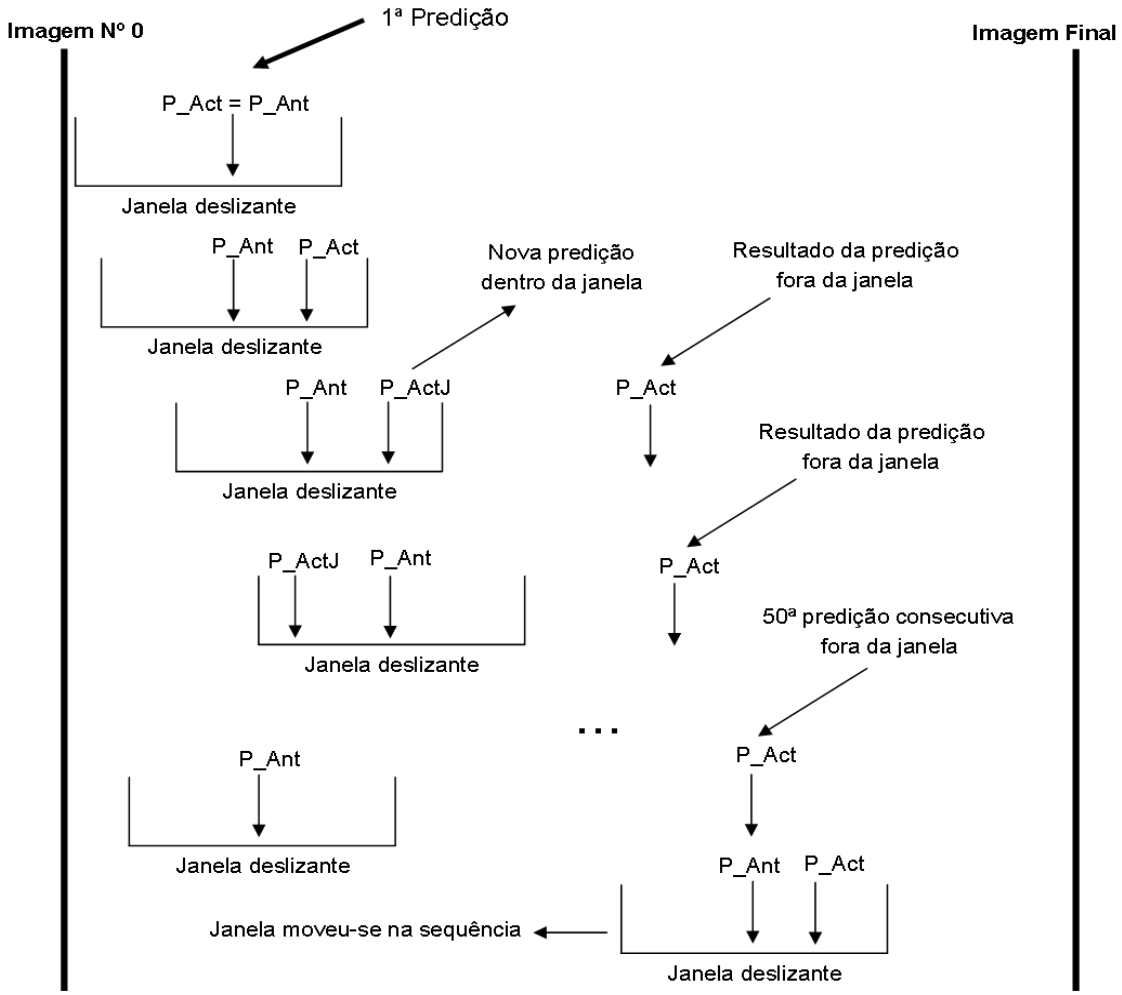


Figura 23 – Esquema de navegação autónoma com janela deslizante que se move quando o robô é raptado.

Para testar a performance deste algoritmo, usou-se o percurso ensinado e percorrido anteriormente com o algoritmo da janela deslizante de largura fixa, entre o laboratório 1.18 e o levador a zona S (figura 24 e figura 37 do anexo A). De seguida, testou-se novamente a performance da navegação autónoma seguindo o caminho ensinado em sentido contrário, ou seja, do elevador da zona S até ao laboratório 1.18 (figura 25 e figura 37 do anexo A). Em ambos os testes o robô chegou, com sucesso, ao fim do trajecto estabelecido .

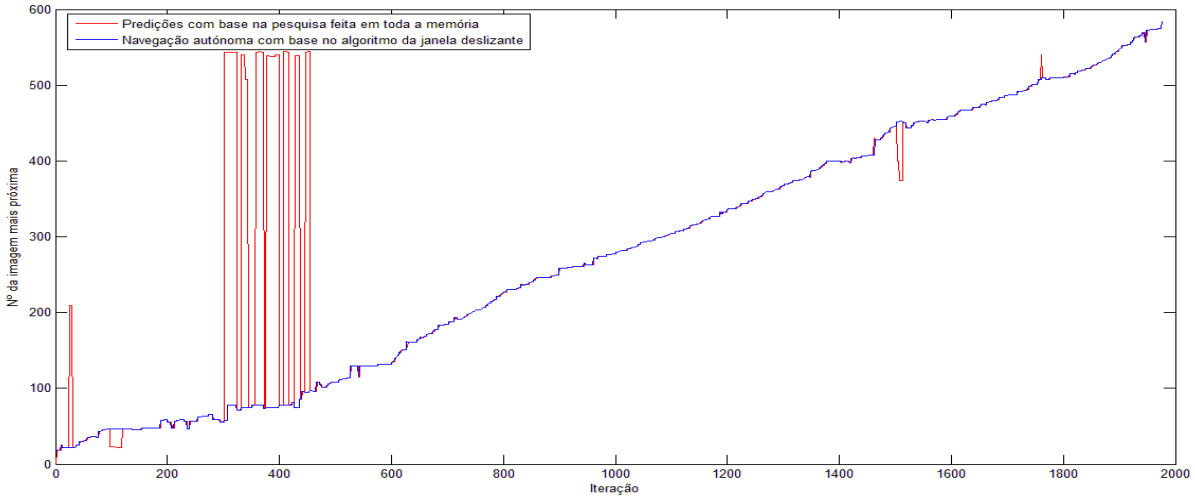


Figura 24 - Gráfico das predições relativas ao percurso entre o laboratório 1.18 e o elevador a zona S, usando a janela deslizante.

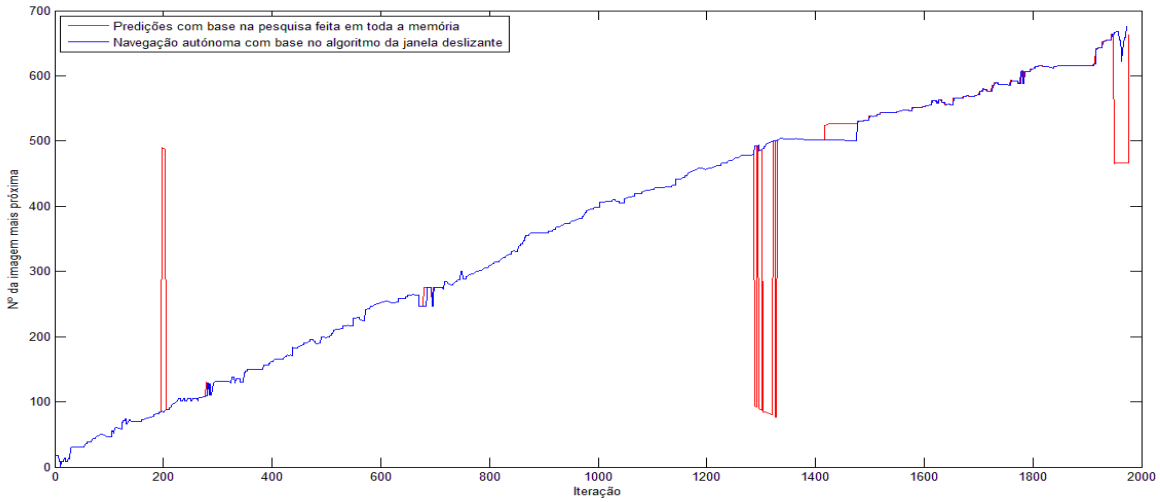


Figura 25 - Gráfico das predições relativas ao percurso entre o elevador a zona S e o laboratório 1.18, usando a janela deslizante.

Estes resultados mostram que, para além de ser mais flexível, este último algoritmo apresenta os mesmos resultados positivos

Assim se chegou ao algoritmo final proposto para a navegação autónoma, o qual apresenta fiabilidade e robustez elevadas, uma vez que torna possível a navegação do robô

mesmo quando há vistas semelhantes ao longo do percurso, conseguindo ainda responder a situações especiais, como no caso do "robô raptado".

5.6 NAVEGAÇÃO AUTÓNOMA – O MODO VIGILANTE

Para tornar o robô capaz de efectuar missões de vigilância, foi desenvolvido um novo modo para a navegação autónoma, o modo vigilante.

Para que este algoritmo funcione correctamente, o percurso ensinado ao robô tem que ser um percurso cíclico, ou seja, a parte final do percurso deve coincidir com o início, de modo a que as vistas iniciais e finais sejam idênticas. Durante a navegação autónoma o robô faz uso do algoritmo da janela deslizante para se localizar ao longo do trajecto no entanto; neste modo, o robô não faz uso do algoritmo de fim de trajecto. Quando se encontra no fim do caminho aprendido, começa a detectar novamente o início do percurso, e assim encontra as imagens iniciais e torna a percorrer o mesmo trajecto, até ser parado pelo utilizador.

De modo a testar-se a capacidade do modo vigilante, foram ensinados dois percursos cíclicos. O primeiro percurso foi realizado em frente ao laboratório 1.18 (figura 26 e figura 48 do anexo A), e o segundo entre os corredores das zonas R e S do piso 2 (figura 27 e figura 49 do anexo A).

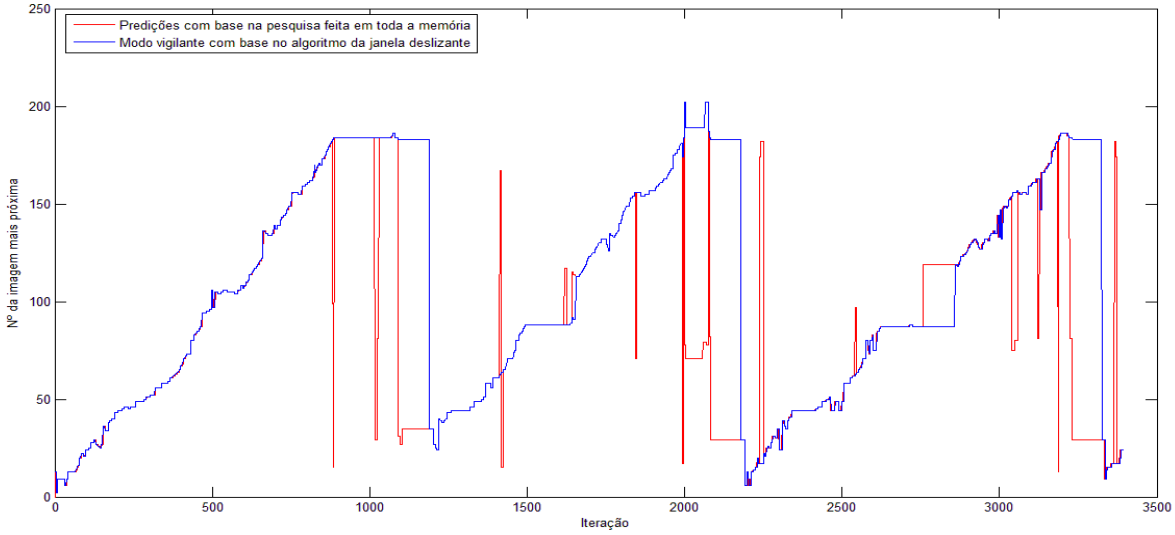


Figura 26 - Gráfico das predições relativas ao percurso cíclico em frente ao laboratório 1.18.

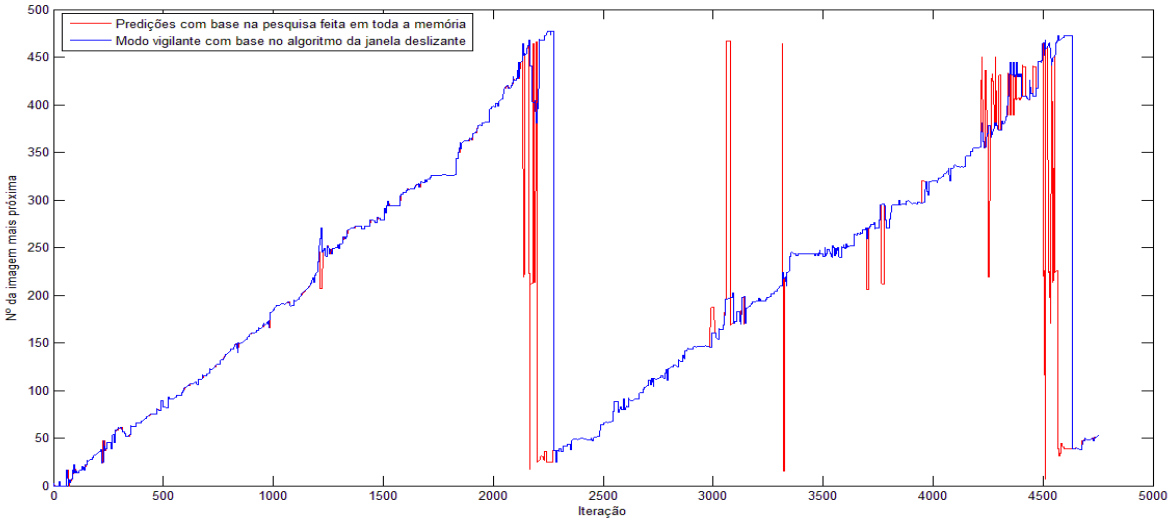


Figura 27 - Gráfico das predições relativas ao percurso cíclico entre os corredores das zonas R e S do piso 2 no modo vigilante.

Como se verifica nas duas figuras acima, o modo vigilante permite ao robô fazer a vigilância de um espaço, percorrendo-o vezes sem conta.

5.7 Resumo dos Testes de Validação dos Algoritmos de Navegação Autónoma

Nesta secção são sumarizados, na tabela 1, os vários testes efectuados com os algoritmos de Navegação Autónoma.

	Descrição do teste	Resultado Esperado	Sim	Não
1	Navegação em linha recta. Testados 3percursos até aproximadamente 30 metros em corredores.	O robô chega e pára no fim do caminho aprendido.	X	
2	Navegação com curvas. Testados mais de 10 percursos com uma e duas curvas até ao comprimento total de 50 metros em corredores.	O robô faz as curvas correctamente e chega ao objectivo.	X	
3	Navegação num ambiente dinâmico. Testados percursos em que passaram pessoas pelos dois lados do robô.	O robô chega ao objectivo mesmo quando há pequenas alterações no ambiente.	X	
4	Teste do modo vigilante. Testados 2 percursos até 55 metros.	O robô chega ao fim e passa para o início da sequência.	X	

Tabela 1 - Testes aos algoritmos de Navegação Autónoma.

6. DESVIO DE OBSTÁCULOS

De modo a que o robô possa navegar em ambientes reais, é necessário que o processo de navegação em modo autónomo seja robusto o suficiente para que possíveis obstáculos que surjam durante o percurso sejam ultrapassados. Para isso, durante a navegação autónoma, o robô tem de ter a capacidade de se desviar de obstáculos que surjam no seu trajecto. Neste capítulo, por questões de espaço disponível, apenas se descrevem dois testes para exemplificar o processo do desvio de obstáculos. Outros testes são apresentados no anexo B.

O robô considera como obstáculo um objecto que se encontre relativamente próximo do seu trajecto durante a navegação autónoma (normalmente esse objecto não estaria lá durante a aprendizagem). Quando o robô detecta um obstáculo, deixa de navegar com base na informação visual até que o obstáculo esteja ultrapassado. A navegação com base na informação visual aprendida é apenas retomada quando o caminho se encontrar novamente livre de obstáculos.

Para o desvio de obstáculos é usada a informação dos três sensores de ultra-sons e quatro sensores de infravermelhos, localizados na frente do robô. Foi implementado um filtro de mediana, para cada sensor, que consiste em usar a mediana das três últimas leituras, para que as medidas destes sensores tenham uma maior robustez e fiabilidade.

Um objecto é considerado obstáculo se estiver a uma distância inferior a 1 m do sonar da frente ou a uma distância inferior a 0,40 m de qualquer outro dos sensores laterais da frente.

Em relação aos obstáculos, o mais frequente é tratar-se de um obstáculo dinâmico, como por exemplo uma pessoa, que apenas se apresenta ao robô como obstáculo durante um pequeno período de tempo. Assim, nestas situações o robô pode não ter que fazer um desvio da sua rota. Por isso, se for detectado um obstáculo pelo sonar da frente a uma distância inferior a 1 m mas superior a 0,50 m, o robô começa por reduzir a sua velocidade, o que pode ser suficiente para o obstáculo deixar de ser detectado. Contudo, se o obstáculo persistir, o robô começa a desviar-se para o lado com mais espaço livre, recorrendo às leituras dos sensores de distância situados em ambos os lados do sonar da frente. Para que o robô se desvie para o lado mais livre, é reduzida a velocidade da roda do lado contrário ao sensor cuja distância ao obstáculo é menor. Ou seja, se o sensor que está a fornecer a menor distância ao obstáculo estiver do lado direito em relação ao sonar do meio, então a velocidade da roda esquerda começa a diminuir.

Assim, se o sonar frontal esquerdo ou os infravermelhos frontais esquerdos detectarem um objecto a uma distância inferior a 0,40 m, então o robô desvia-se para o lado direito, e se o

sonar frontal direito ou os infravermelhos frontais direitos detectarem um objecto a uma distância inferior a 0,40 m, então o robô desvia-se para o lado esquerdo.

Com este comportamento, quando se encontra um obstáculo, garante-se que o robô nunca embate contra objectos no seu percurso.

No entanto, após o desvio de um obstáculo, o robô não pode retomar imediatamente a navegação com base em visão, pois, como está com uma orientação diferente à que tinha naquele local durante a aprendizagem, há uma grande probabilidade de se perder, uma vez que a vista corrente não tem, provavelmente, nenhuma correspondência com as imagens aprendidas.

Para resolver este problema, foi criada uma pilha de desvios (figura 28), onde são guardados todos os desvios efectuados. Assim, depois de contornado o obstáculo, quando o robô se encontra novamente num espaço livre, a pilha é esvaziada de modo a repor a orientação do robô antes do desvio de obstáculos.

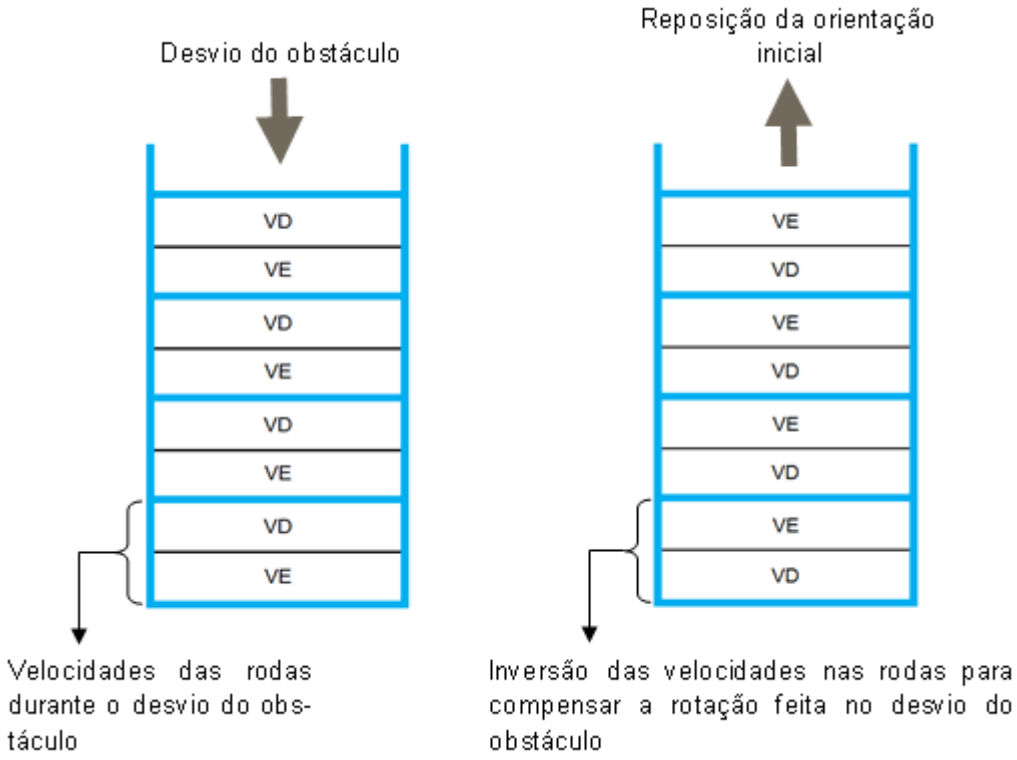


Figura 28 – Funcionamento da pilha de desvios.

Este processo funciona da seguinte forma: enquanto o robô se está a desviar, sempre que a velocidade das rodas é alterada, estes valores são guardados na pilha. Portanto, quando acaba o desvio, é usada a informação guardada na pilha de desvios para repor a orientação que o robô tinha antes do contorno do obstáculo. Para tal, a pilha é esvaziada de modo a que as velocidades que foram impostas para a roda esquerda sejam agora aplicadas na roda direita e as impostas na roda esquerda sejam agora aplicadas na direita, durante o mesmo período de tempo. Deste modo, depois de contornado o ou os obstáculos, o robô está pronto para retomar a navegação baseada

em visão para chegar ao fim do percurso com sucesso. Este método apresentou bons resultados, sendo o robô capaz de ultrapassar vários obstáculos consecutivos.

As linhas de trajectórias das figuras seguintes marcam a trajectória da traseira do robô, não do centro de massa. Note-se que quando o robô roda para a esquerda traça um arco para a direita. Isto acontece por estas terem sido traçadas no solo com um marcador preso à traseira do robô. As linhas depois reforçadas à mão no "Paint".

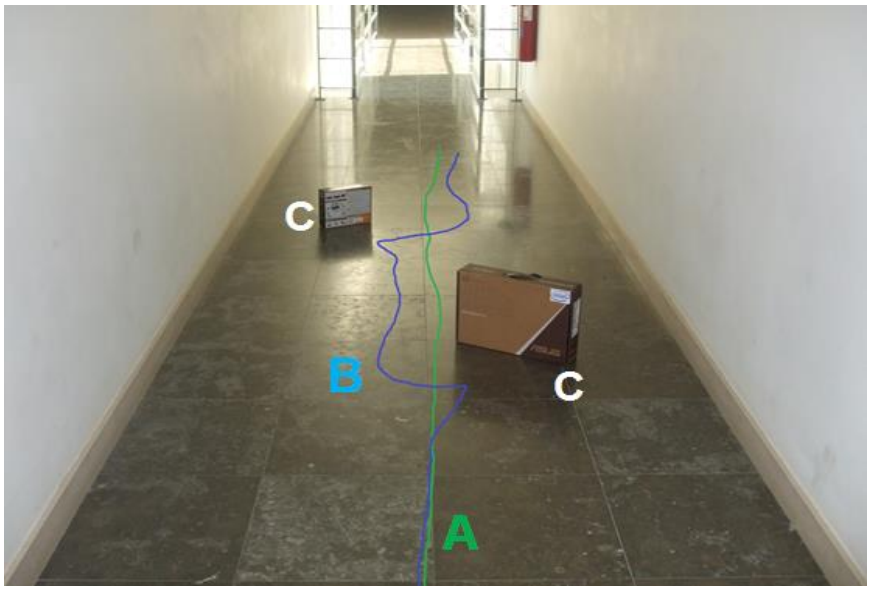


Figura 29 – Exemplo de desvio de obstáculos. A) Percurso sem obstáculos. B) Percurso com obstáculos. C) Obstáculo.

Apesar dos resultados favoráveis obtidos (exemplo da figura 29), este método falha nalguns dos casos em que o robô encontra um obstáculo numa curva. No caso particular em que, numa curva, o robô se aproxima demasiado de uma parede, com o algoritmo descrito acima, o robô não será capaz de continuar o seu caminho, pois, quando acaba de se afastar da parede, é recuperada a orientação original, que fará com que o robô se vire novamente contra a parede, afastando-se repetidamente da parede para voltar a orientar-se (figura 30). Desta forma, não será capaz de continuar a percorrer o percurso aprendido por estar sempre no modo de desvio de obstáculos.

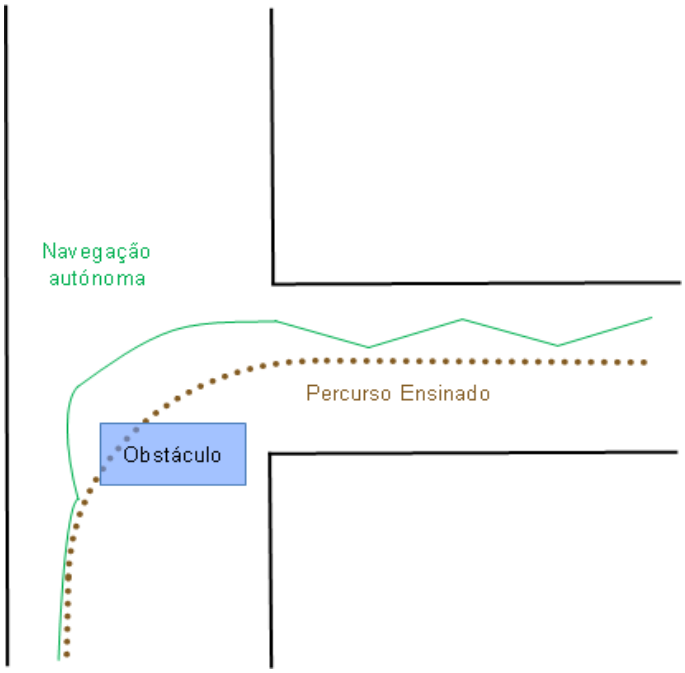


Figura 30 – Representação do problema do desvio de obstáculos em curva usando o método da pilha de desvios.

Para resolver este problema foi implementado um novo método para o desvio de obstáculos quando o robô, ao descrever uma curva, detecta um obstáculo. Então, ao entrar no modo de desvio, é inicialmente verificado se o robô estava a descrever uma curva na aprendizagem referente à zona do percurso em que se encontra o obstáculo, comparando as velocidades das duas rodas obtidas a partir da informação relativa à imagem mais próxima. Se estas forem diferentes, o robô descreve uma curva, uma vez que tinha velocidade angular na aprendizagem.

Neste novo método, sempre que o robô descreve uma curva, as distâncias a que se considera um obstáculo são reduzidas, sendo considerados obstáculos todos os objectos que estiverem a menos de 0,30 m de qualquer sensor lateral da frente do robô ou a 0,80 m do sonar frontal. Se um objecto se encontrar entre 0,40 m e 0,80 m do sonar frontal é considerado um obstáculo temporário.

Além disso, nestes casos, não é usada a pilha de desvios, para evitar a situação acima referida e referida figura 30. Assim, numa curva quando o robô deixa de ver o obstáculo, volta de imediato a utilizar a SDM para navegar, e como se encontra em curva, ou no final desta, as imagens que o robô começa a capturar são suficientes para que este acabe de contornar o obstáculo, desta vez enquanto vai navegando com as imagens da SDM.

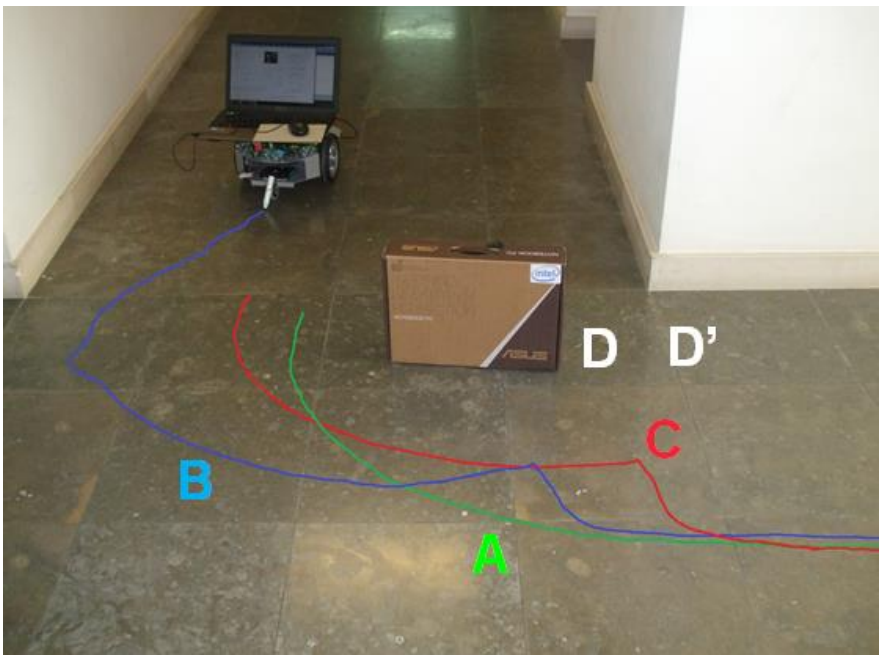


Figura 31 – Exemplo de desvio de um obstáculo numa curva com o novo método. A) Percurso aprendido. B) Percurso com dois obstáculos (caixa e canto do parede). C) Percurso seguido pelo robô quando o obstáculo é movido para D'. D) Obstáculo.

Este método apresentou resultados bastante positivos, uma vez que o robô consegue ultrapassar, quando tem caminho livre, todos os obstáculos colocados ao longo de uma curva (figura 31). De facto, este algoritmo, combinado com o primeiro (aplicado fora das curvas), permite ao robô desviar-se de qualquer obstáculo que surja no seu trajecto (quando existe espaço para se desviar), e retomar o seu caminho.

São sumarizados, na tabela 2, os vários testes efectuados com os algoritmos de Desvio de obstáculos.

	Descrição do teste	**Resultado Esperado**	**Sim**	**Não**
1	Robô detecta e contorna obstáculos, pela direita e pela esquerda, ao navegar em linha recta.	Robô desvia-se dos obstáculos e retoma navegação baseada em visão.	X	
2	Robô detecta e contorna obstáculos, pela direita e pela esquerda, que apareçam em curvas.	Robô desvia-se do obstáculo numa curva e retoma navegação baseada em visão.	X	

Tabela 2 - Testes aos algoritmos de Desvio de Obstáculos.

7. CONCLUSÕES E SUGESTÕES PARA TRABALHO FUTURO

Este último capítulo pretende apresentar ao leitor as conclusões sobre o trabalho realizado na dissertação, indicando também algumas sugestões para trabalho futuro.

7.1 CONCLUSÕES

No final do trabalho desenvolvido, verifica-se que todos os objectivos propostos foram alcançados. Este trabalho está relacionado com o trabalho desenvolvido por Rodrigues et al [3], que consistiu na paralelização da SDM numa GPU e que foi tema para a publicação conjunta.

O desenvolvimento do *software* necessário para dar ao robô a capacidade de percorrer autonomamente caminhos, previamente ensinados, através do reconhecimento de imagens guardadas numa SDM implementada numa GPU, teve bons resultados. O primeiro passo foi o desenvolvimento de uma interface através da qual o utilizador pudesse ensinar ao robô um ou mais trajectos. A implementação de uma SDM numa GPU para suporte à navegação autónoma baseada em visão foi uma inovação integrada neste trabalho. De modo a desenvolver uma navegação autónoma robusta, foram testados vários algoritmos, sendo a janela deslizante o algoritmo final, uma vez que foi o que apresentou melhores resultados.

Na sequência do trabalho realizado nesta tese, verifica-se que o robô é capaz de percorrer caminhos autonomamente, através do reconhecimento de imagens guardadas numa SDM implementada numa GPU, e, durante este processo o robô é também capaz de detectar e desviar-se de obstáculos, eficazmente e em tempo real, o que permite a sua aplicação em tarefas como guiar pessoas dentro de edifícios ou fazer a vigilância de espaços desocupados.

Desta forma, conclui-se que a navegação robótica baseada em visão, através da memória de sequências de vistas guardadas na SDM, é uma boa solução para o desenvolvimento de novas aplicações que requeiram a movimentação autónoma de um robô móvel ao longo de ambientes *indoor*.

Os algoritmos da janela deslizante e desvio de obstáculos aplicados à navegação baseada em visão, sendo inovadores e tendo apresentado bons resultados, deram origem a um artigo (anexo C), a ser submetido à *International Conference on Robotics and Automation* (ICRA) 2014.

7.2 SUGESTÕES PARA TRABALHO FUTURO

Os objectivos iniciais da tese foram cumpridos, mas há ainda espaço para melhoramentos e inovações posteriores.

Sugere-se o desenvolvimento de um novo módulo, explorador (*wandering*), no qual o robô vai explorando sozinho o ambiente e vai construindo um mapa virtual em tempo real da vizinhança através da leitura dos sensores de infravermelhos e sonares, minimizando, assim, o tempo de aprendizagem.

Sugere-se ainda o desenvolvimento de algoritmos de desvio de obstáculos que recorram ao modo explorador, referido acima, de forma a que seja possível ultrapassar obstáculos em situações em que o caminho ensinado esteja totalmente impedido e seja necessário explorar caminhos alternativos.

Para uma fase posterior sugere-se uma abordagem cognitiva para a localização do robô, implementando algoritmos de detecção e comparação de *features*. As *features* poderiam ser guardadas no vector de dados de cada *hard location*, juntamente com a informação já lá guardada (número de imagem, número de sequência e movimento do robô), ou pode ser implementada uma hierarquia de memória, em que as *features* estarão a um nível mais alto do que as imagens reais, como se pensa que acontece também no cérebro humano.

REFERÊNCIAS

[1] Mendes, M., Coimbra, A.P. and Crisóstomo, M.M. (2012). Vision-based Robot Navigation: Quest for Intelligent Approaches using a Sparse Distributed Memory. Universal Publishers, Boca Raton, Florida, 2012, ISBN 978-1-61233-104-1.

[2] Flynn, M.J., Kanerva, P. and Bhadkamkar, N. (1989). Sparse distributed memory: Principles and operation. In *Technical Report CSL-TR-89-400,* Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, California 94305-4055.

[3] André Rodrigues, André Brandão, Mateus Mendes, A. Paulo Coimbra, Fernando Barros e Manuel Crisóstomo, "Parallel Implementation of a SDM for Vision-based Robot Navigation," 13th Spanish-Portuguese Conference on Electrical Engineering (13CHLIE), Politechnical University of Valencia (UPV), Spain, July 2013.

[4] Rao, R.P.N. and Fuentes, O. (1996). Learning Navigational Behaviors using a Predictive Sparse Distributed Memory. In *From Animals to Animats: The Fourth International Conference on Simulation of Adapative Behavior*. MIT Press.

[5] Watanabe, M., Furukawa, M. and Kakazu, Y. (2000). Intelligent AGV driving toward an autonomous decentralized manufacturing system.

[6] Sparse Distributed Memory. http://en.wikipedia.org/wiki/Sparse_distributed_memory (visto em 4-9-13)

[7] Kenji Doya (2000). Complementary roles of basal ganglia and cerebellum in learning and motor control. In *Current Opinion in Neurobiology*, Volume 10, Issue 6, pages 732-739.

[8] Rogers, D. (1989). Predicting weather using a genetic memory: A combination of Kanerva's sparse distributed memory with holland's genetic algorithms. In *NIPS*.

[9] Denning, P.J. (1989). Sparse Distributed Memory. In *American Scientist 77*, pages 333-335.

[10] Mateus Mendes, Manuel Crisóstomo, e A. Paulo Coimbra. "Robot navigation using a sparse distributed memory". In *Proc. of the 2008 IEEE International Conference on Robotics and Automation*, Pasadena, Califórnia, USA, Maio 2008.

[11] Nvidia. http://www.nvidia.com.br/page/home.html (visto em 4-9-13)

[12] NVIDIA GeForce 256. http://www.nvidia.com/page/geforce256.html (visto em 4-9-13)

[13] Definição de GPGPU. http://pt.wikipedia.org/wiki/GPGPU (visto em 4-9-13)

[14] Programação de GPUs da ATI. http://robotzeitgeist.com/2006/10/ati-announces-stream-computing-platform.html (visto em 4-9-13)

[15] ATI. http://www.amd.com/br/Pages/AMDHomePage.aspx (visto em 4-9-13)

[16] História da programação em GPUs com CUDA. http://www.nvidia.com.br/object/cuda_home_new_br.html (visto em 4-9-13)

[17] CUDA.http://en.wikipedia.org/wiki/CUDA (visto em 4-9-13)

[18] O conceito de *speedup*. http://en.wikipedia.org/wiki/Speedup (visto em 4-9-13)

[19] Christopher Rasmussen and Gregory D. Hager (1996). Robot navigation using image sequences. In *In Proc*. AAAI, pages 938–943.

[20] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue (2000). View-based approach to robot navigation.In *Proceedings of* 2000 IEEE/RSJ *International Conference on Intelligent Robots and Systems* (IROS 2000).

[21] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue(2003). View-based navigation using an omniview sequence in a corridor environment. In *Machine Vision and Applications*.

[22] Hanspeter A. Mallot, Heinrich H. B¨ulthoff, Philipp Georg, Bernhard Sch¨olkpf, and Ken Yasuhara(1995). View-based cognitive map learning by an autonomous robot. *Adaptive Behavior*, 3(3):311–348.

[23] Hiroshi Ishiguro and Saburo Tsuji. Image-based memory of environment (1996). In *in Proc*. IEEE/RSJ *Int. Conf. Intelligent Robots and Systems*, pages 634–639.

[24] Hiroshi Ishiguro, Takahiro Miyashita, and Saburo Tsuji (1995). T-net for navigating a vision-guided robot in real world. In ICRA, pages 1068–1074.

[25] Ribeiro, M. M. (2013). Ferramenta de Análise e Simulação Computacional de Sistema Catadióptrico Omnidireccional Hiperbólico de Lobo Duplo. Tese de mestrado.

[26] Stephen D. Jones, Claus Andresen, and James L. Crawley (1997). Appearance based processes for visual navigation. In *Proceedings of the* IEEE *International Conference on Intelligent Robots and Systems*, Grenoble, France.

[27] Mao-Hai Li, Bing-Rong Honga, Ze-Su Caia, Song-Hao Piaoa, and Qing-Cheng Huanga (2007). Novel indoor mobile robot navigation using monocular vision. *Engineering Applications of Artificial Intelligence*, 21(3).

[28] Niall Winters and José Santos-Victor(1999). Mobile robot navigation using omni-directional vision. In I*n Proc. 3rd Irish Machine Vision and Image Processing Conference* (IMVIP'99), pages 151–166.

[29] Matthias O. Franz, Bernhard Schölkopf, Hanspeter A. Mallot, and Heinrich H. B¨ulthoff (1998). Learning view graphs for robot navigation. *Autonomous Robots*, 5(1):111–125.

[30] Dr. Robot- X80Pro. http://www.drrobot.com/products_item.asp?itemNumber=X80Pro (visto em 4-9-13)

[31] DUR5200 Ultrasonic Range Sensor Module User Manual. http://www.drrobot.com/products_item.asp?itemNumber=DUR5200 (visto em 4-9-13)

[32] GP2Y0A21YK Sharp IR range. http://www.drrobot.com/products_item.asp?itemNumber=GP2Y0A21YK (visto em 4-9-13)

[33] The FreeImage Project. http://freeimage.sourceforge.net/ (visto em 4-9-13)

[34] Microsoft Foundation Class Library**.** http://en.wikipedia.org/wiki/Microsoft_Foundation_Class_Library (visto em 4-9-13)

[35] Denning, P.J. (1989). Sparse Distributed Memory. In *American Scientist 77*, pages 333-335.

[36] Baptista, J. (2011). Apontamentos de Sistemas Robóticos e Robótica Móvel – Introdução. Universidade de Coimbra (Portugal).

[37] Mendes, M. (2010) *Intelligent Robot Navigation Using a Sparse Distributed Memory*. Tese de Doutoramento, Universidade de Coimbra.

[38] Rodrigues, A (2013*) Implementação de uma SDM com GPUs para Robô autónomo*.Tese de Mestrado, Universidade de Coimbra.

ANEXO A

Neste anexo são apresentados os locais onde foram realizados os testes apresentados na tese, assim como outros resultados relativos à navegação autónoma (sem obstáculos), não mostrados no corpo da tese por falta de espaço. Todos os testes foram realizados nos pisos 1 e 2 do DEEC (figuras 32 e 33).

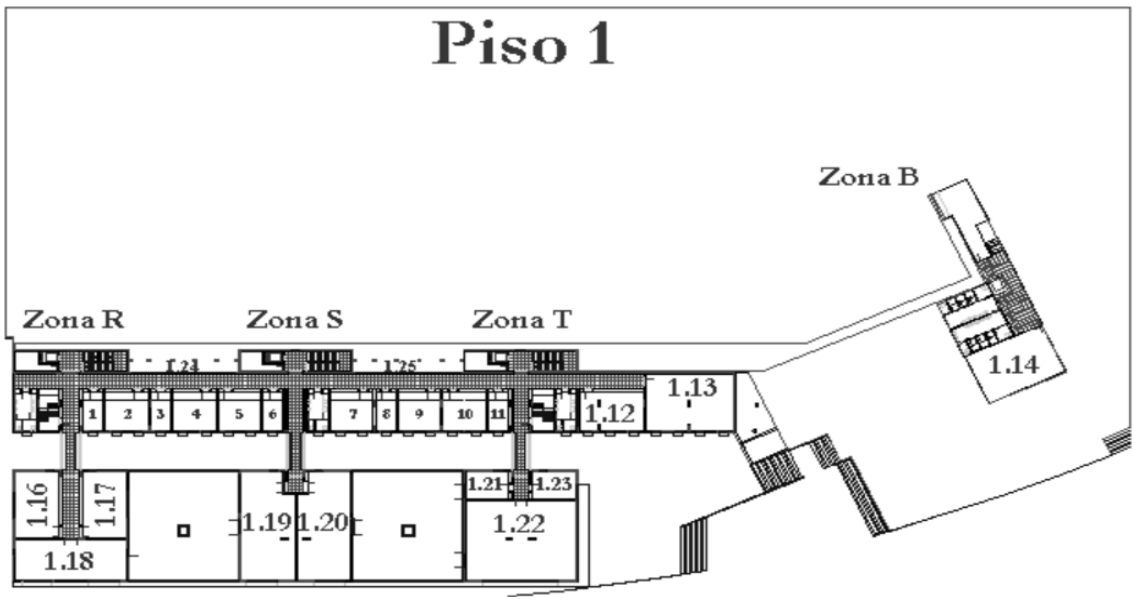


Figura 32 - Planta do piso 1 do DEEC.

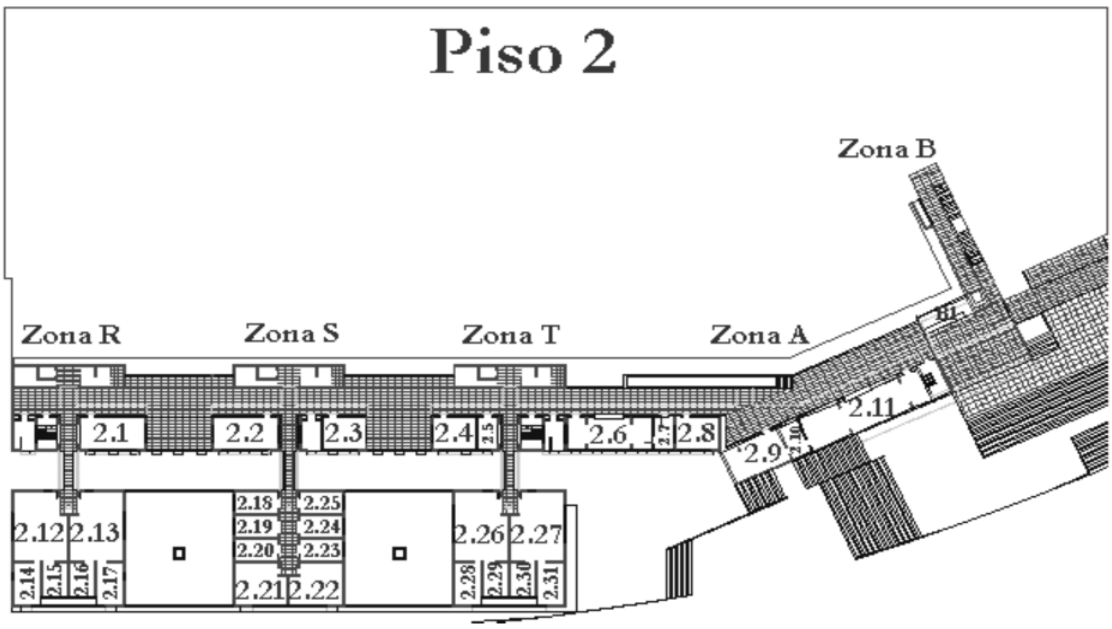


Figura 33 - Planta do piso 2 do DEEC.

As figuras abaixo mostram os percursos dos testes apresentados no capítulo 5 (figuras 34, 35, 37, 48 e 49).

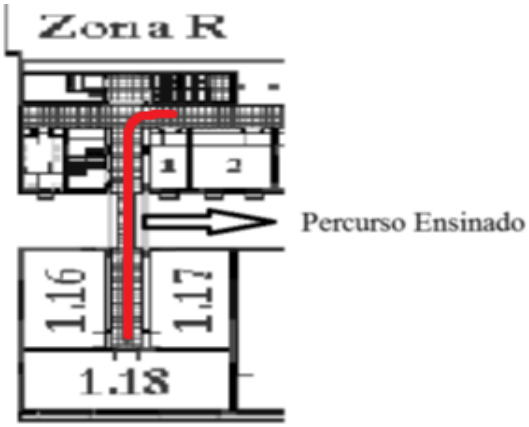


Figura 34 - Percurso ensinado entre o laboratório 1.18 e o gabinete 1, relativo aos testes apresentados na figura 17.

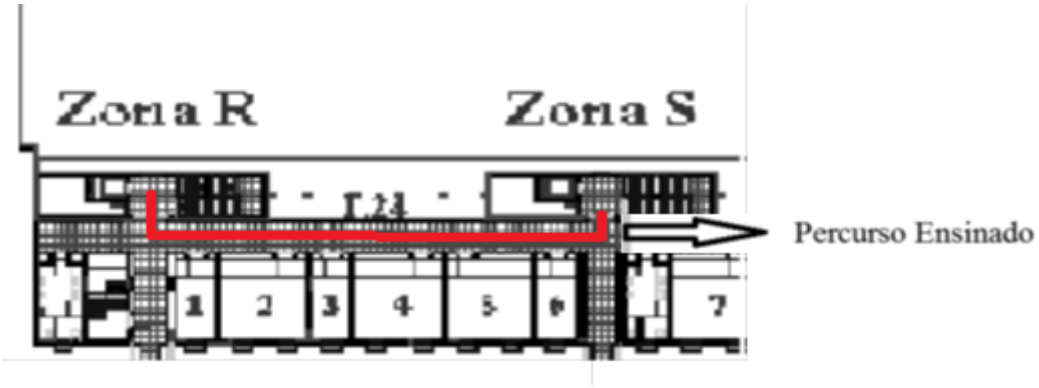


Figura 35 - Percurso ensinado entre os elevadores das zonas R e S, relativo aos testes apresentados na figura 18.

A localização do robô no final da navegação autónoma ao longo do percurso representado na figura 35 pode ser vista na figura abaixo.

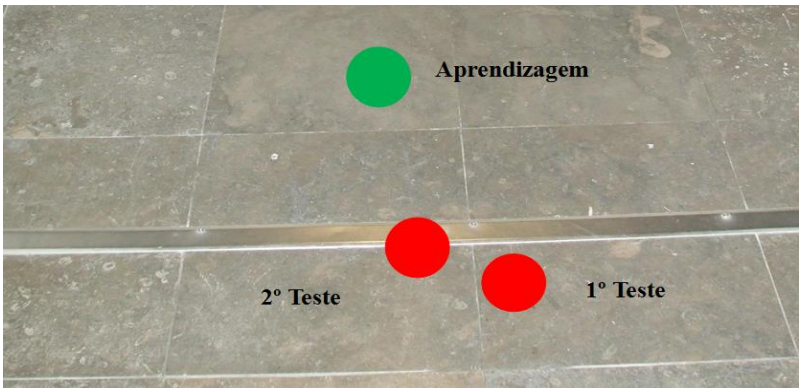


Figura 36 – Localização do robô no fim da navegação autónoma relativa aos testes apresentados na figura 18.

Figura 37 - Percurso ensinado entre o elevador da zona S e o laboratório 1.18, relativo aos testes apresentados nas figuras 19, 21, 22, 24 e 25. (Os testes 22 e 24 foram realizados no sentido inverso, ou seja, nestes testes o percurso começava em frente ao laboratório 1.18 e acabava na zona S).

Para o percurso acima assinalado foram realizados três testes com a janela deslizante de largura fixa. O gráfico das predições relativas ao primeiro teste encontra-se na figura 21 e os restantes nas figuras 38 e 39. A localização do robô no fim da navegação autónoma é mostrada na figura 40.

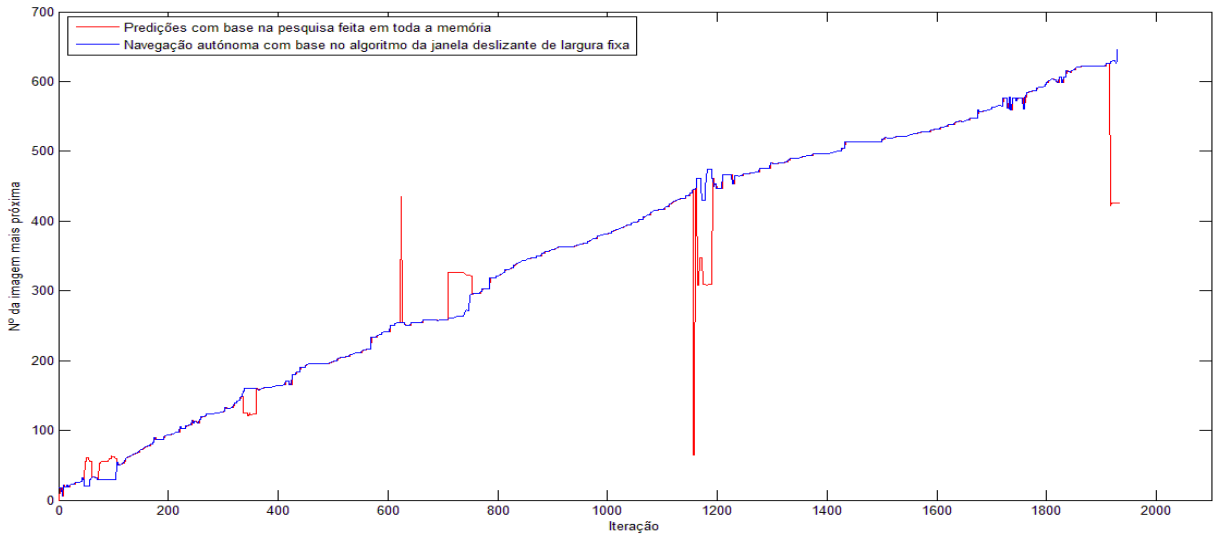


Figura 38 - Gráfico das predições relativas ao percurso entre o elevador da zona S e o laboratório 1.18, percorrido pela segunda vez, usando a janela deslizante de largura fixa.

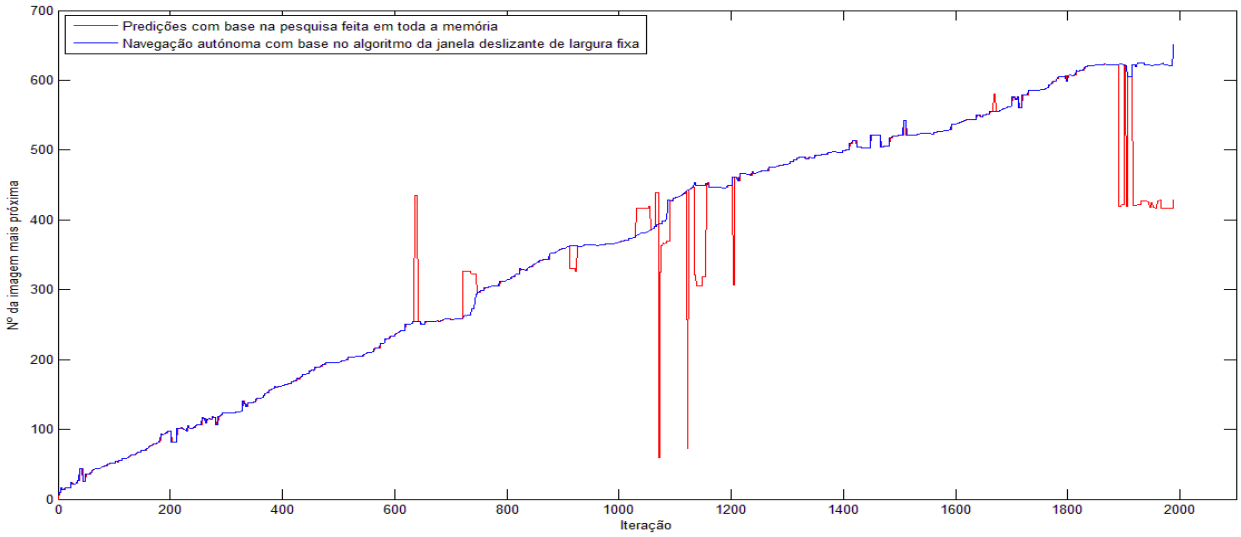


Figura 39 - Gráfico das predições relativas ao percurso entre o elevador da zona S e o laboratório 1.18, percorrido pela terceira vez, usando a janela deslizante de largura fixa.

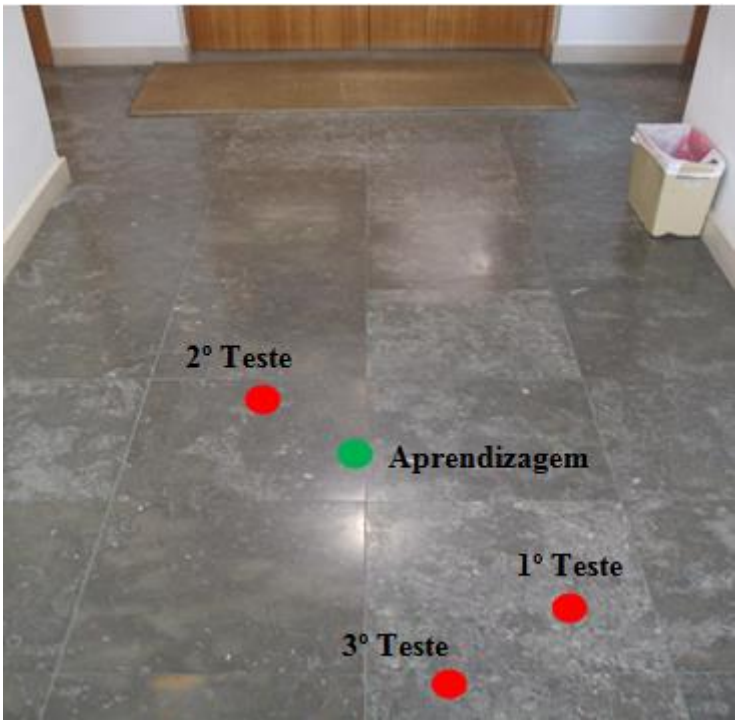


Figura 40 – Localização do robô no fim da navegação autónoma relativa aos testes apresentados nas figuras 21, 38 e 39.

Este percurso foi percorrido também três vezes no sentido inverso, com a janela deslizante de largura fixa, estando a primeira vez representada na figura 22 e as restantes nas duas figuras 41 e 42.

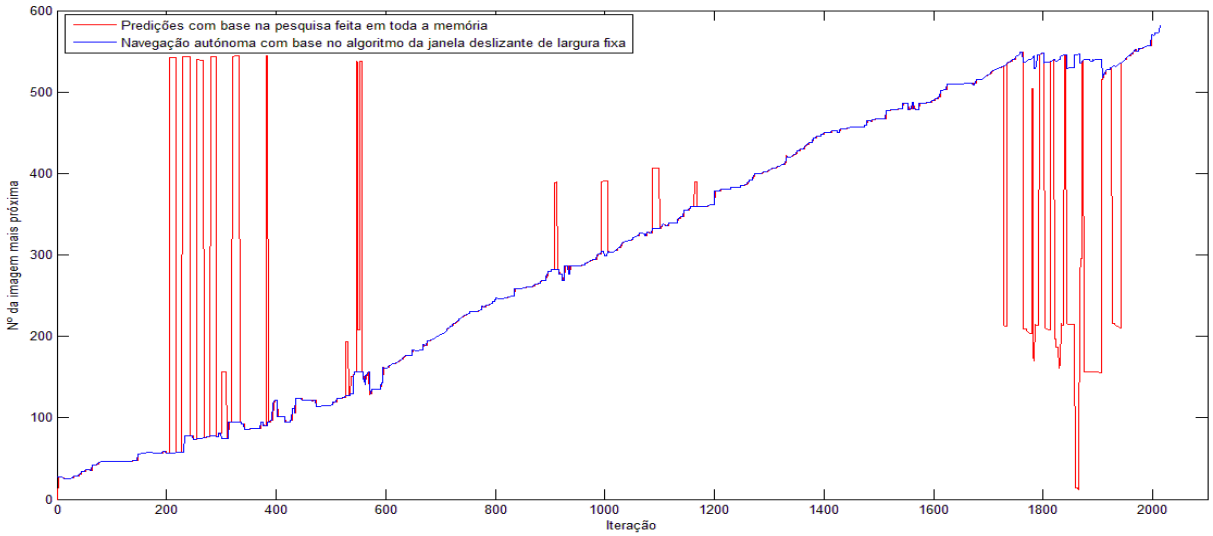


Figura 41 - Gráfico das predições relativas ao percurso entre o laboratório 1.18 a o elevador da zona S, percorrido pela segunda vez, usando a janela deslizante de largura fixa.

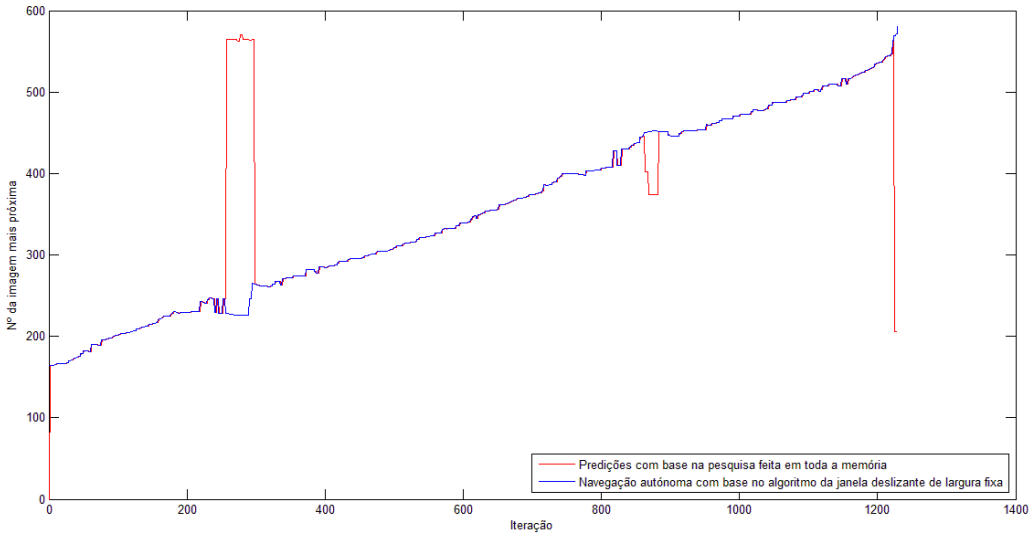


Figura 42 - Gráfico das predições relativas ao percurso entre o laboratório 1.18 e o elevador da zona S, percorrido pela terceira vez, usando a janela deslizante de largura fixa

No terceiro teste, cujo gráfico de predições está apresentado na figura acima, o robô começou a navegação autónoma a meio do percurso e, como é possível observar, chegou com sucesso ao destino (figura 43), mostrando assim que o algoritmo da janela deslizante de largura fixa permite que o robô inicie a navegação em qualquer parte do percurso aprendido.

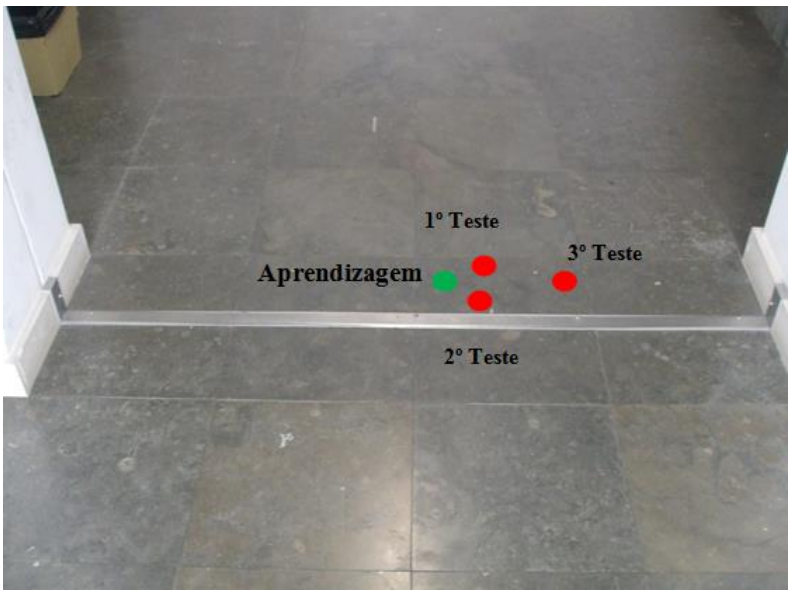


Figura 43 – Localização do robô no fim da navegação autónoma relativa aos testes apresentados nas figuras 22, 40 e 41.

Após a apresentação dos resultados obtidos com a janela deslizante de largura fixa, os resultados da navegação com o algoritmo da janela deslizante são expostos abaixo.

O primeiro percurso testado com o algoritmo da janela deslizante estende-se entre o laboratório 1.18 e o elevador da zona S, estando o 1º teste realizado neste caminho representado na figura 24 e os resultados do 2º teste são apresentados de seguida nas figuras 44 e 45. Neste 2º teste a janela move-se ao fim de apenas 30 predições consecutivas fora da janela (nos restantes testes a janela move-se ao fim de 50 predições fora da janela).

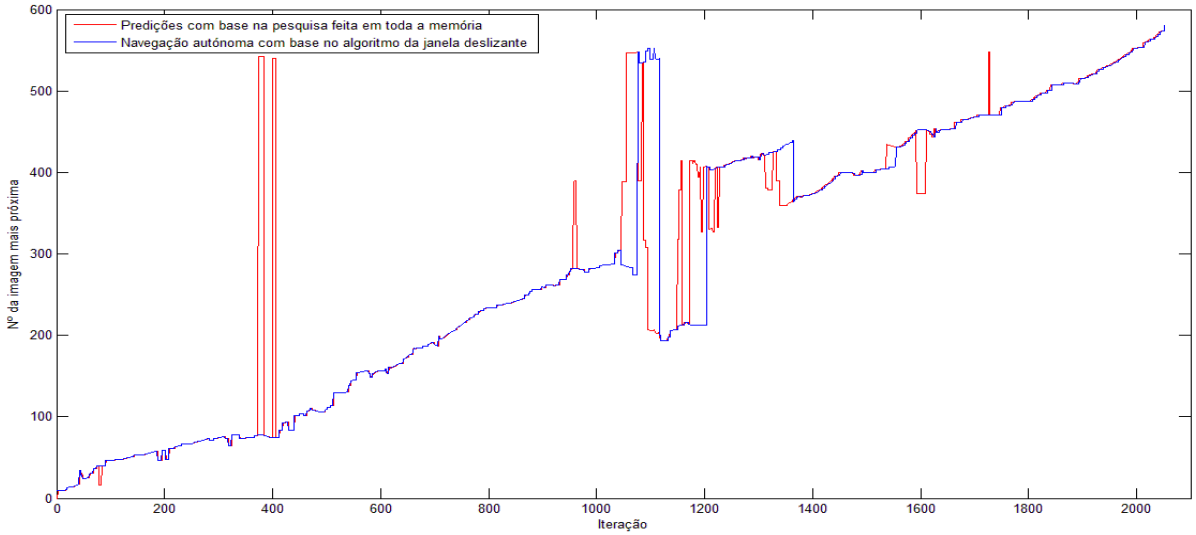


Figura 44 - Gráfico das predições relativas ao percurso entre o laboratório 1.18 e o elevador da zona S, percorrido pela segunda vez, usando a janela deslizante (neste caso a janela move-se ao fim de 30 predições consecutivas fora da janela).

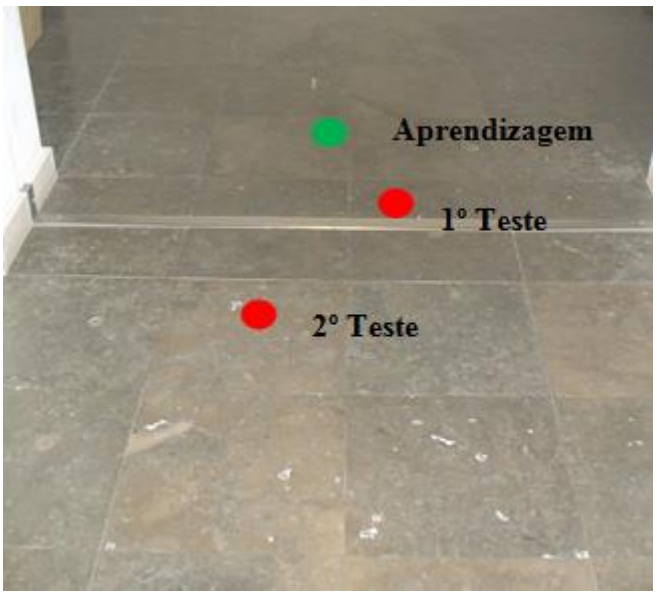


Figura 45 – Localização do robô no fim da navegação autónoma relativa aos testes apresentados nas figuras 24 e 43.

De seguida apresenta-se o resultado do 2º teste do caminho inverso do teste anterior, entre o elevador da zona S e o laboratório 1.18 (figuras 46 e 47).

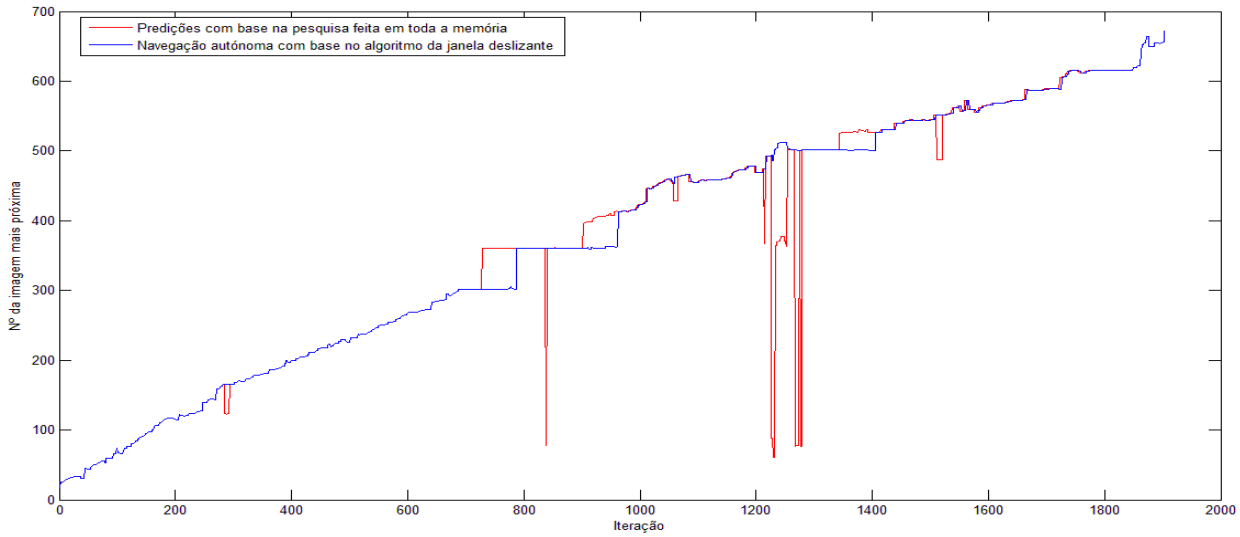


Figura 46 - Gráfico das predições relativas ao percurso entre a torre S e o laboratório 1.18, percorrido pela segunda vez, usando a janela deslizante.

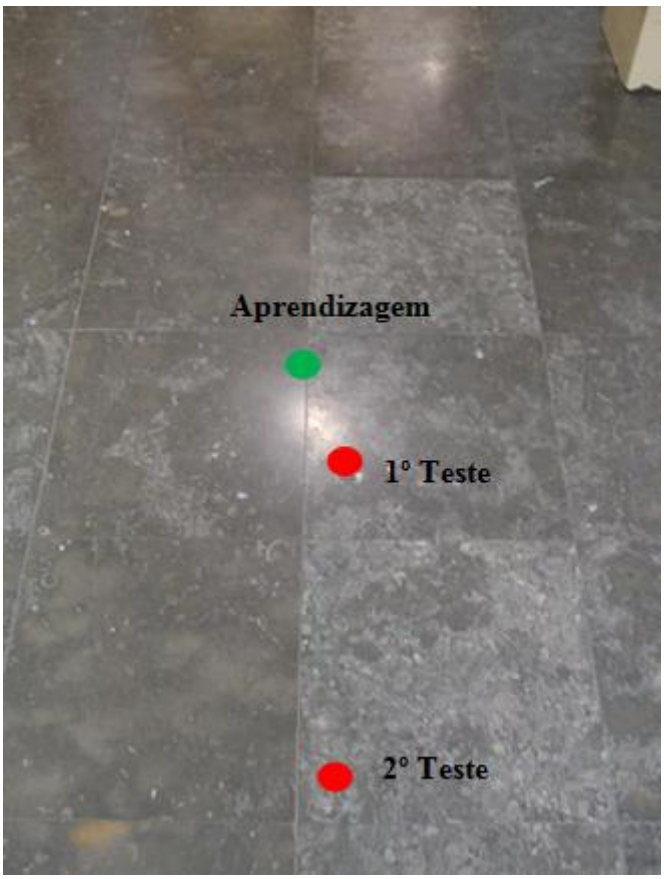


Figura 47 – Localização do robô no fim da navegação autónoma relativa aos testes apresentados nas figuras 25 e 46.

Apresentam-se agora as zonas nas quais foram efectuados os testes relativos ao modo vigilante (figuras 48 e 49).

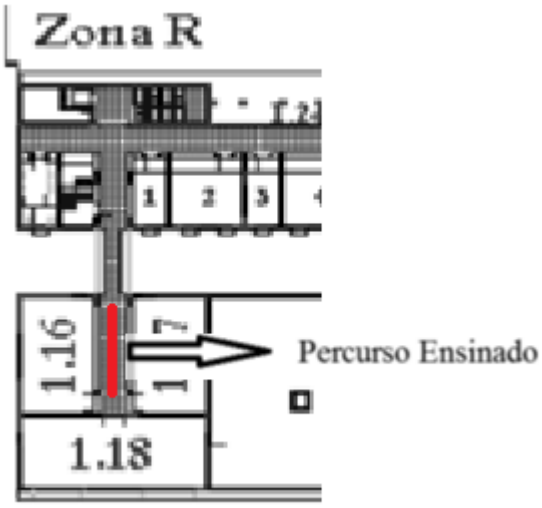


Figura 48 – Percurso, no modo vigilante, em frente ao laboratório 1.18, relativo ao teste apresentado na figura 26.

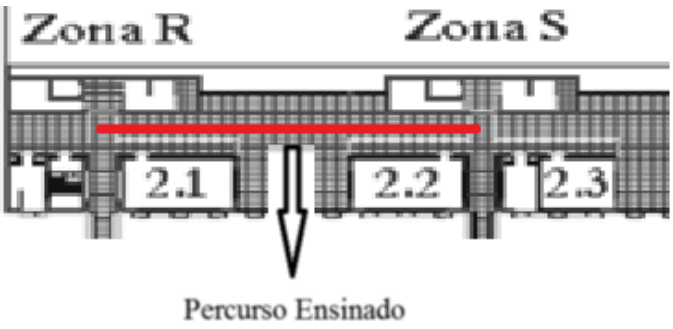


Figura 49 - Percurso, no modo vigilante, entre os corredores das zonas R e S do piso 2, relativo aos testes apresentados na figura 27.

Para além dos testes apresentados acima e descritos na tese, foram ainda realizados mais testes (figuras 50 e 51). Um desses testes foi efectuado entre a zona R e a secretaria (sala 2.11 na planta da figura 33), no piso 2, usando o algoritmo da janela deslizante fixa e o algoritmo da janela deslizante, tendo o robô alcançado o destino em ambos os casos.

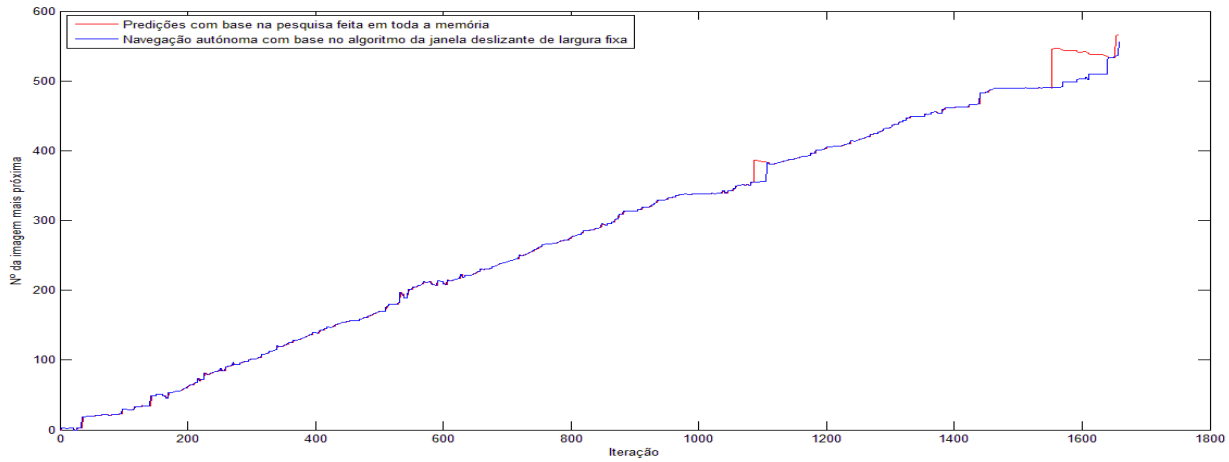


Figura 50 - Gráfico das predições relativas ao percurso entre a torre R e a secretaria, usando a janela deslizante de largura fixa.

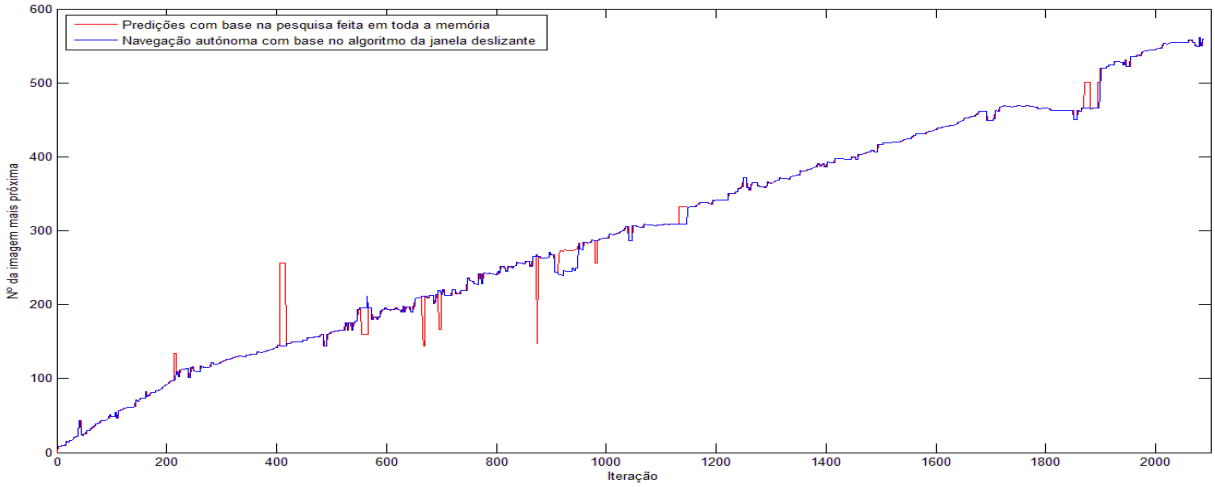


Figura 51 - Gráfico das predições relativas ao percurso entre a torre R e a secretaria, usando a janela deslizante.

ANEXO B

Este anexo apresenta os testes de desvio de obstáculos efectuados durante a navegação autónoma que, por falta de espaço, não puderam ser expostos no capítulo 6 (figuras 52 a 58). As linhas marcam a trajectória da traseira do robô, não do centro de massa. Note-se que quando o robô roda para a esquerda traça um arco para a direita.

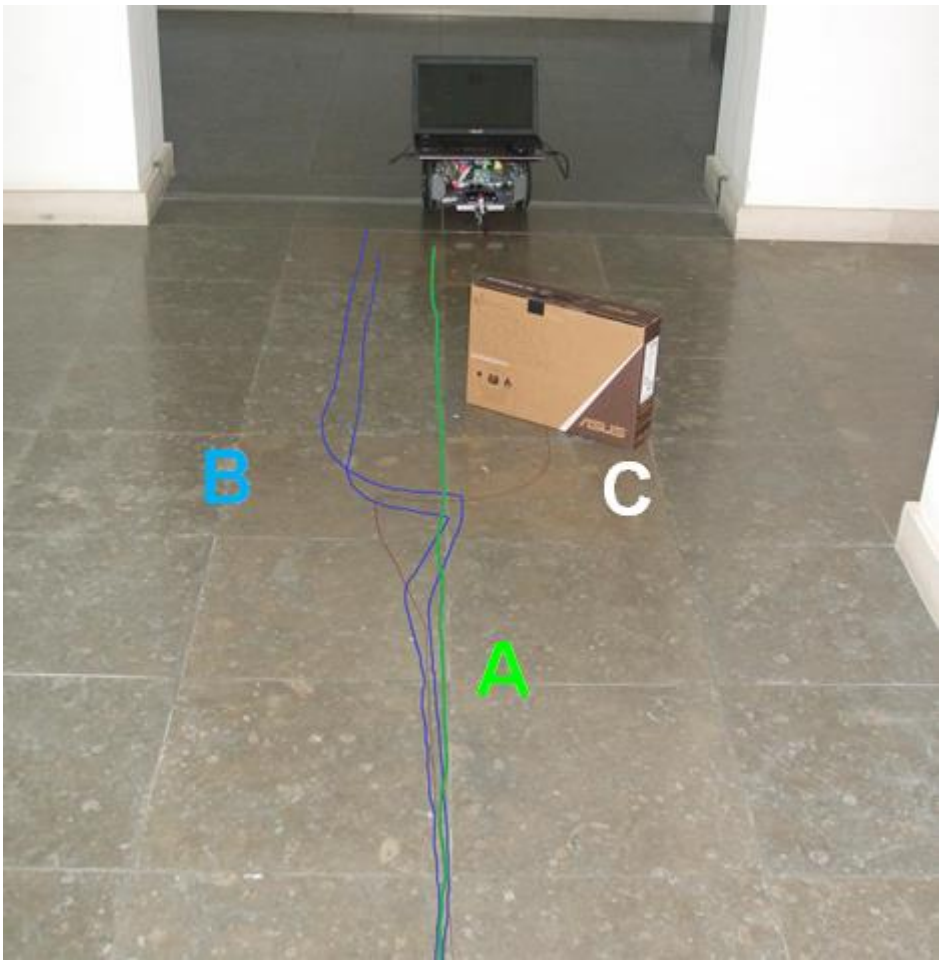


Figura 52 – Exemplo de desvio de um obstáculo pela esquerda. A). Percurso sem obstáculo. B) Percursos com obstáculo C) Obstáculo.

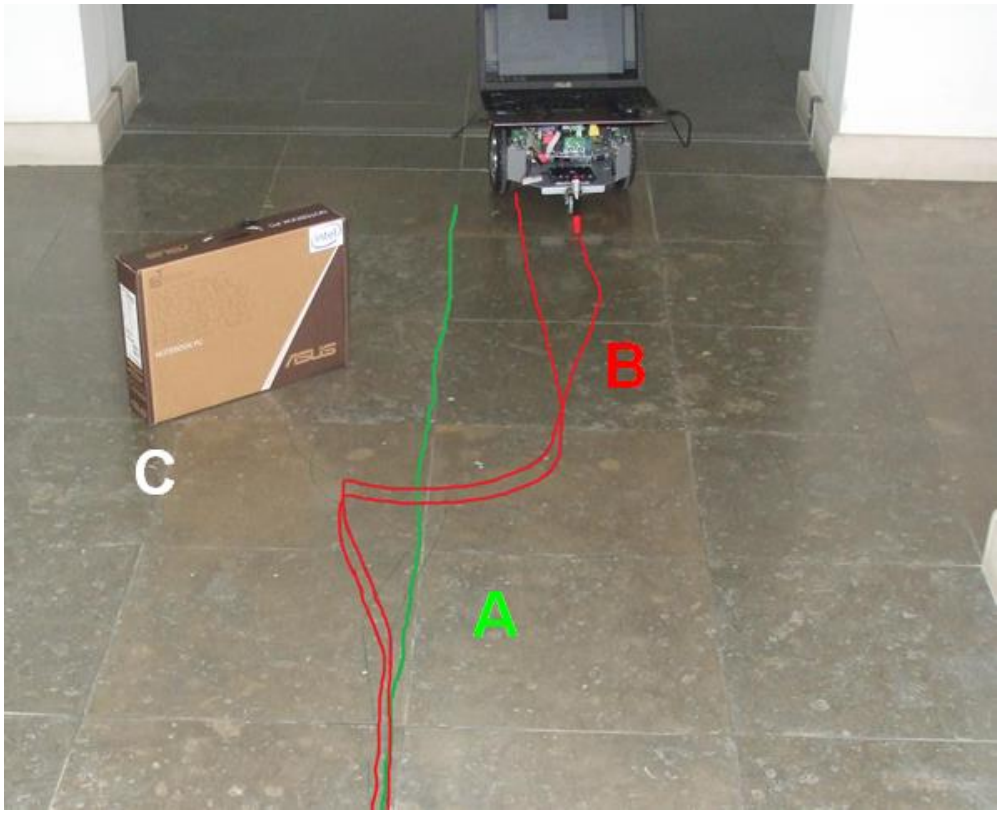


Figura 53 – Exemplo de desvio de um obstáculo pela direita. A) Percursos sem obstáculo. B) Percurso com obstáculo. C) Obstáculo.

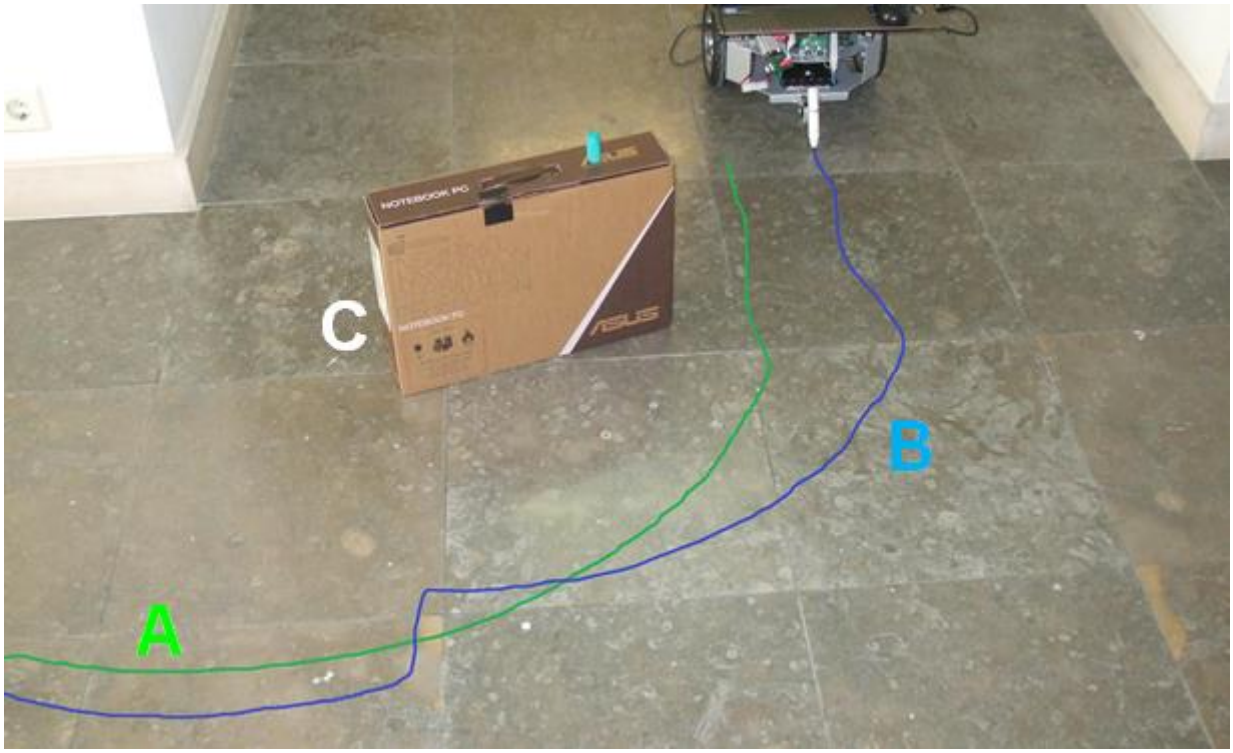


Figura 54 – Exemplo de desvio de um obstáculo numa curva. A) Percurso sem obstáculo. B) Percurso com obstáculo. C) Obstáculo.

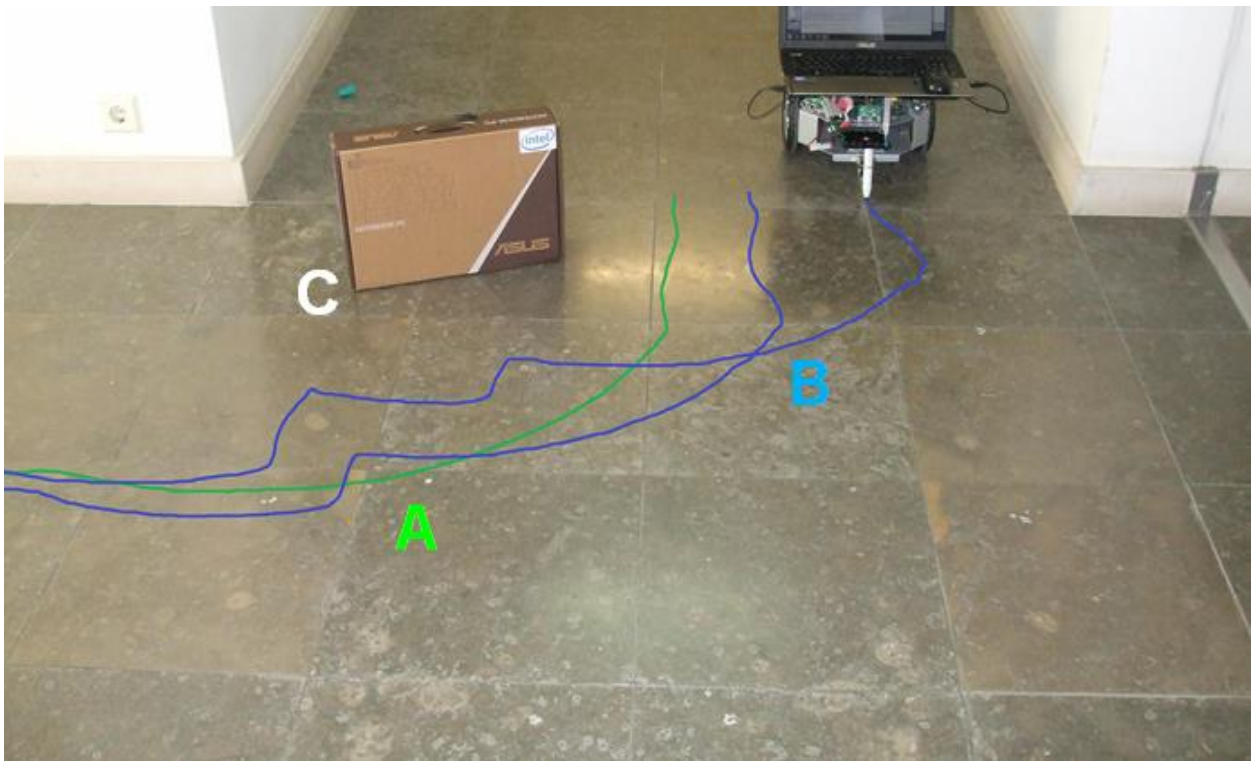


Figura 55 – Exemplo de um obstáculo numa curva em que o robô teve de fazer dois desvios. A) Percurso sem obstáculo. B) Percursos com obstáculo. C) Obstáculo.

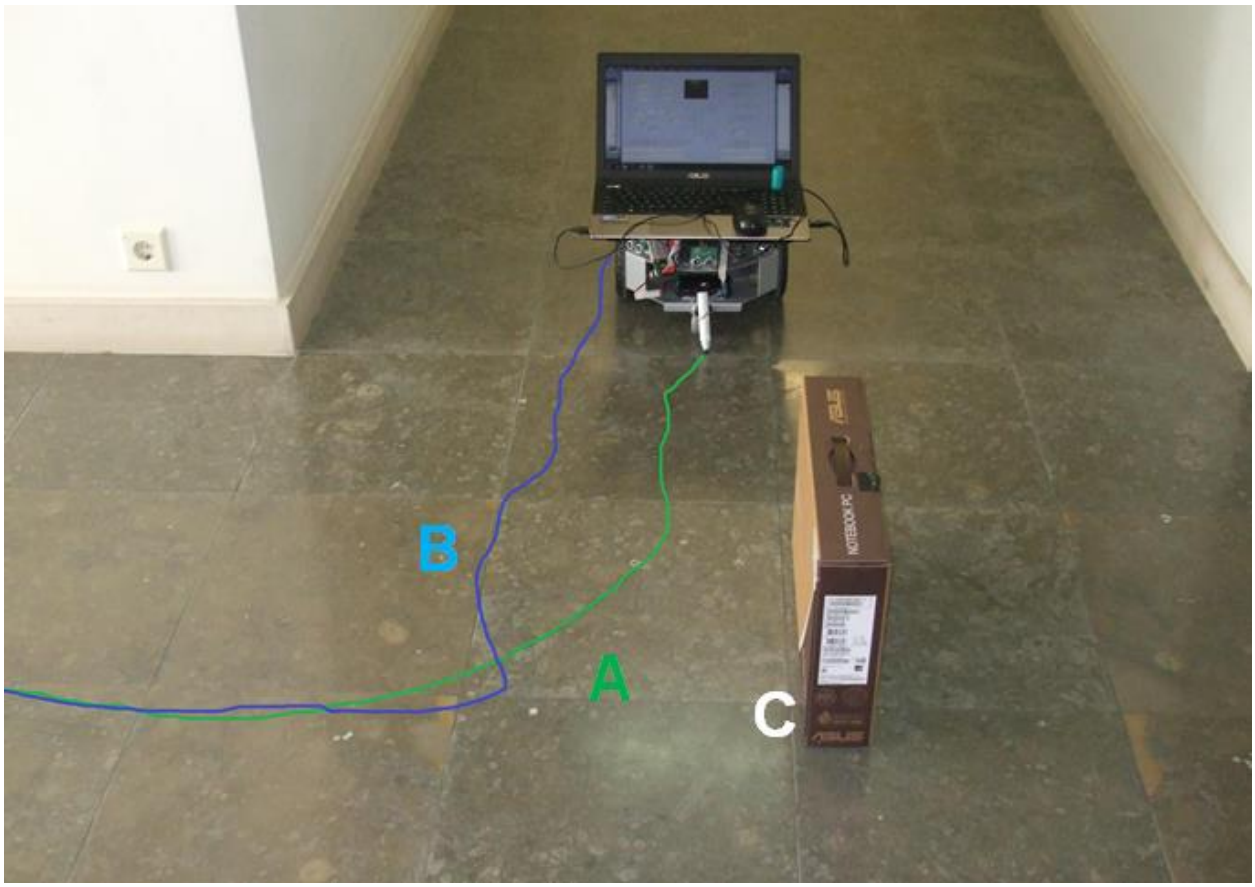


Figura 56 – Exemplo de desvio de um obstáculo numa curva, pela esquerda (pela parte de dentro da curva). A) Percurso sem obstáculo. B) Percurso com obstáculo. C) Obstáculo.

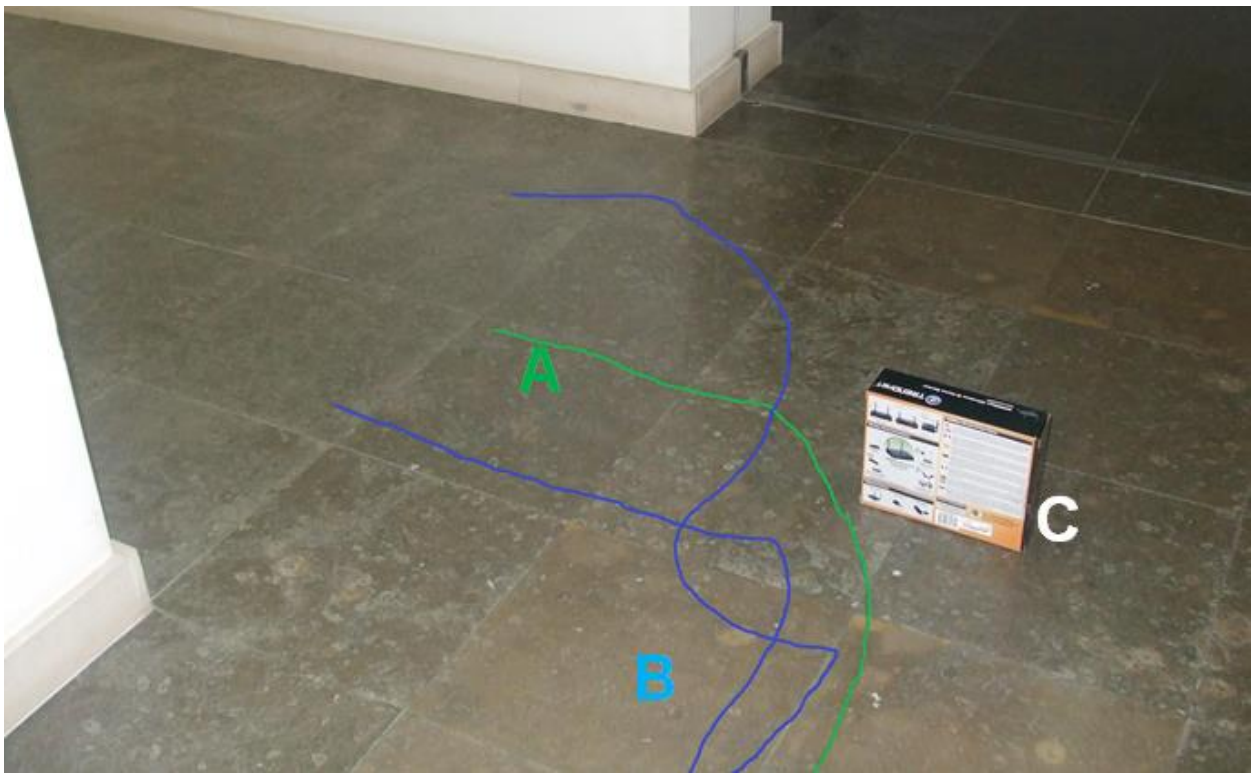


Figura 57 – Exemplo de desvio de um obstáculo numa curva pela parte de dentro da curva. A) Percurso sem obstáculo. B) Percursos com obstáculo. C) Obstáculo.

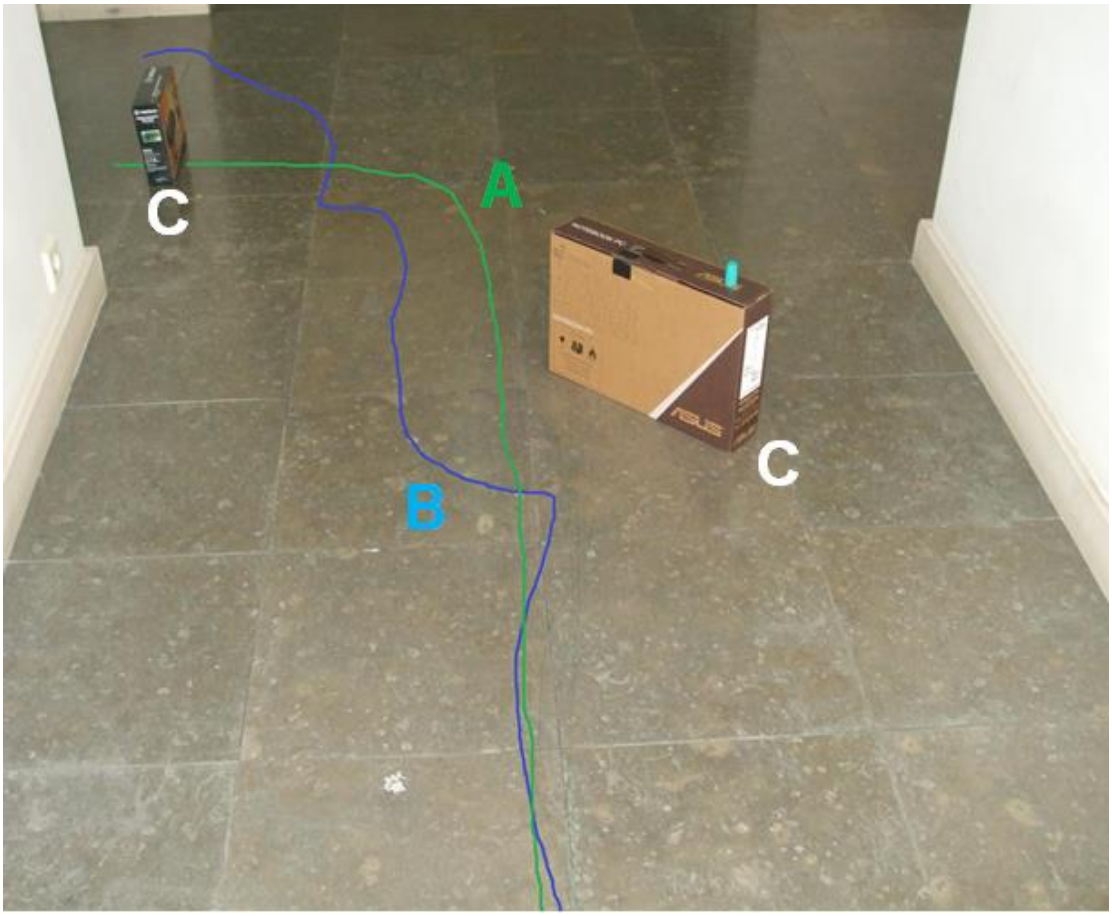


Figura 58 – Exemplo de desvio de obstáculos numa curva com dois obstáculos. A) Percurso sem obstáculos. B) Percurso com obstáculo. C) Obstáculo.

ANEXO C

Artigo a ser submetido à *International Conference on Robotics and Automation* (ICRA) 2014 – Versão à data de entrega da tese.

View-based robot navigation using a SDM implemented in a GPU*

André Brandão, Mateus Mendes, A. Paulo Coimbra and Manuel Crisóstomo

***Abstract*— Constant increase of processing power and constant cost decrease of computer memory, along with evidence that the source of intelligent behavior is possibly strongly supported on the use of a large sophisticated memory, have encouraged research in vision-based methods for robot navigation. The present approach uses images stored into a Sparse Distributed Memory, implemented in parallel in a Graphics Processing Unit, as a method for robot localization and navigation. Algorithms for following previously learnt paths and avoiding random obstacles are described.**

Keywords: Autonomous Navigation, View-Based Navigation, View-based Localization, SDM.

I. INTRODUCTION

Development of intelligent robots is an area of intense research. Nonetheless, the results achieved so far are still humble. Thus, it is important to try new models. The present approach uses a Sparse Distributed Memory (SDM) as the grounds for vision and memory-based robot localization and navigation.

The SDM is a type of associative memory suitable to work with high-dimensional boolean vectors. It was proposed in the 1980s by P. Kanerva [1] and has successfully been used before for vision-based robot navigation [2, 3]. Simple vision-based methods, such as implemented by Matsumoto [4], although sufficient for many environments, in monotonous environments such as corridors may present a large number of errors.

The robot learns new paths during a supervised learning stage. While learning, the robot captures and stores views of the surrounding environment, and stores them in the SDM, with some odometric and additional information. In autonomous navigation, the robot captures updated views and uses the memory to search for the closest image, using the image's additional information as basis for localization and navigation. Memory search is performed in parallel in a GPU. Still during the autonomous navigation mode, the environment is scanned using infra-red sensors (IR) and sonar sensors. If obstacles are detected in robot's path, an obstacle-avoidance algorithm takes control of navigation until the obstacle is overcome, and then vision-based navigation is resumed.

Section II briefly describes the SDM. Section III presents the key features of the experimental platform used. The principles for vision–based navigation is explained in Section IV. Section V describes two of the navigation algorithms implemented in the robot. It also presents the results of the tests performed with those algorithms. In Section VI, the obstacle avoidance algorithms are described, along with the validation tests and results. Conclusions and future work are presented in Section VII.

*Research supported in part by FCT (PEst programme).

André Brandão, A. Paulo Coimbra and Manuel Crisóstomo are with Institute of Robotics – University of Coimbra, Electrical and Computer Engineering Department, Pólo 2, 3030-290 Coimbra Portugal. andre_alexandre_sb@hotmail.com, acoimbra@deec.uc.pt, mcris@isr.uc.pt

Mateus Mendes is with Polytechnic Institute of Coimbra – ESTGOH Rua General Santos Costa, 3400-124 Oliveira do Hospital, Portugal. mmendes@estgoh.ipc.pt

II. SPARSE DISTRIBUTED MEMORY

The properties of the SDM are inherited from the properties of high-dimensional binary spaces, as originally described by P. Kanerva [1]. Kanerva proves that high-dimensional binary spaces exhibit properties in many aspects related to those of the human brain, such as naturally learning (one-short learning, reinforcement of an idea), naturally forgetting over time, ability to work with incomplete information and large tolerance to noise.

A. Example of the original SDM model

Figure 1 shows a minimalist example of the original SDM model. The main structures are an array of addresses and an array of data counters. The memory is sparse, in the sense that it contains only a fraction of the locations of the addressable space. The locations which physically exist are called hard locations. Each input address activates all the hard locations which are within a predefined access radius (3 bits in the example). The distance between the input address and each SDM location is computed using the Hamming distance, which is the number of bits in which two binary numbers are different.

Data are stored in the bit counters. Each location contains one bit counter for each bit of the input datum. To write a datum in the memory, the bit counters of the selected locations will be decremented where the input datum is zero and incremented where the input datum is one.

Reading is performed by sampling the active locations. The average of the values of the bit counters is computed column-wise for each bit, and if the value is above a given threshold, a one is retrieved. Otherwise, a zero is retrieved. Therefore, the retrieved vector may not be exactly equal to the stored vector, but Kanerva proves that most of the times it is.

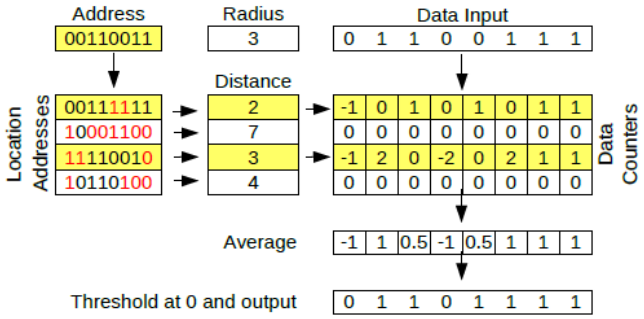


Figure 1 – Model of a SDM using bit counters.

B. *A simplified SDM model*

The original SDM model, using bit counters, has some disadvantages. One problem is that it has a low storage rate, of about 0.1 data bits per bit of physical memory. Another problem is that the counters slow down the system in real time. Yet another problem is that data encoded using the natural binary code are sensitive to the positional value of the bits [5], and that negatively affects the performance of the system.

In order to overcome some of the drawbacks described, other SDM models have been proposed [5, 6, 7]. Figure 2 shows a model based on Ratitch et al.'s approach [6], which groups data bits as integers and uses the sum of absolute differences instead of the Hamming distance to compute the distance between an input address and location addresses in the memory. That has been the model used in the present work. It was named "arithmetic SDM," and its performance has been superior to the original model [2].

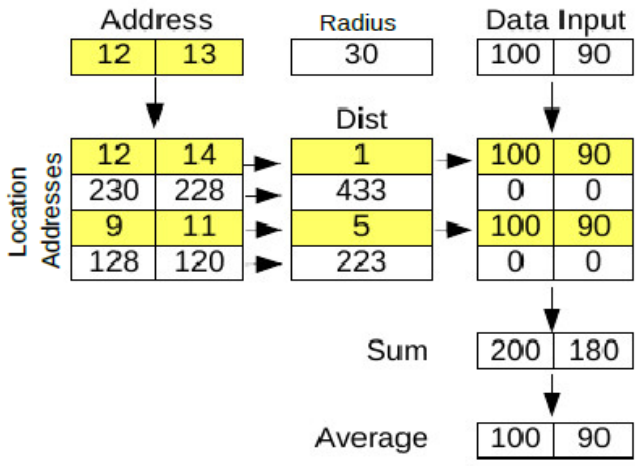


Figure 2 – Arithmetic SDM model

C. *Parallel implementation in a GPU*

The SDM model was implemented first in a CPU, using linked lists as depicted in the upper part of Figure 3. As the volume of data available to process in real time increased, it became clear that the system could benefit from a parallel implementation of the search procedure. The SDM was then implemented in parallel, in a GPU, using CUDA architecture, as shown in the lower part of Figure 3.

During the learning stage of the robot, the linked list is built, only using the CPU and central RAM memory. When learning is over, the list contents are copied to an array of addresses and an array of data in GPU's memory. Later, when necessary to retrieve any information from the SDM, multiple GPU kernels are launched in parallel to check all memory locations and get a quick prediction. The parallel implementation is described in more detail in [8].

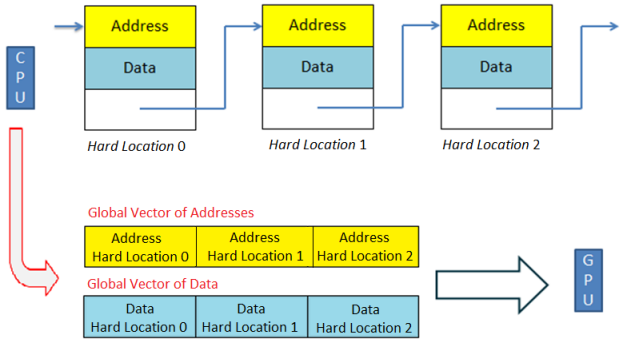


Figure 3 – Data structures of the SDM implemented in the CPU and in the GPU.

III. EXPERIMENTAL PLATFORM

A. *Hardware*

The robot used was an X80Pro[1], as shown in Figure 4. It is a differential drive vehicle equipped with two driven wheels, each with a DC motor, and a rear swivel caster wheel, for stability. The robot has a built-in digital video camera, an integrated WiFi system (802.11g), and offers the possibility of communication by USB or serial port. For obstacle avoidance, it has six ultrasound sensors and seven infra-red sensors with a sensing range of respectively 2.55 m and 0.80 m.

Figure 4 – Pictures of the robot and laptop used.

The robot was controlled in real time from a laptop with a 2.40 GHz Intel Core i7 processor, 6 Gb RAM and a NVIDIA GPU with 2Gb memory and 96 CUDA cores.

B. *Software modules*

Figure 5 shows the interactions between the different software modules developed and the robot. The laptop was carried on board and linked to the robot through the serial port.

[1] http://www.drrobot.com (last checked 2013-09-14).

Besides the SDM module, where the navigation information was stored, different modules were developed to interact with the robot, control the supervised learning process and the autonomous run mode, which required the obstacle avoidance module.

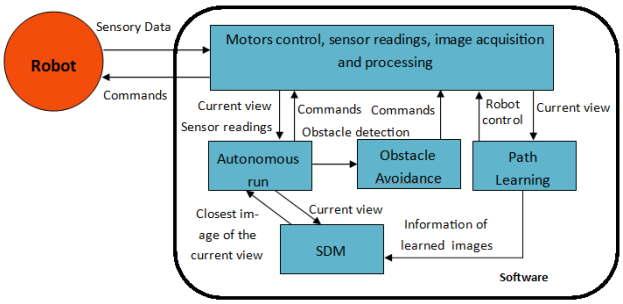


Figure 5 – Interactions between software modules and the robot.

IV. Principles of Vision-Based Navigation

Robot localization and navigation are based on visual memories, which are stored into the SDM during a learning stage and later used in the autonomous run mode.

A. Supervised learning

In the supervised learning stage, the user drives the robot along a path using the mouse. During this process, the robot acquires pictures in BMP format with 176×144 resolution, which are converted to PGM format (Figure 6). Each recorded image is saved in the disk, along with additional data, and later stored into the SDM. Each path has a unique sequence number and is described by a sequence of views, where each view is also assigned a unique view number. Hence, the images are used as addresses for the SDM and the data vectors stored into the SDM are in the following format:

$$< seq\#, im\#, vr, vl > \quad (1)$$

The sequence number is *seq#*. The number of the image in the sequence is *im#*. The angular velocity of the right and left wheels is, respectively, *vr* and *vl*.

Figure 6- Picture acquired during supervised learning.

B. Autonomous run mode

For the autonomous run mode, the user first chooses the destination among the set of previously learnt paths.

The robot starts by capturing an image that is converted to PGM. Then, it is sent to the SDM to be compared with the images in the memory, and the SDM returns the image ranked as the most similar to the captured image.

After obtaining the closest image, the information associated with it is used to reproduce the motion performed at the time the image was captured and stored. In order to reduce possible drifts and errors in the trajectory and orientation of the robot, the retrieved information is monitored as explained in subsection D below.

C. Image filtering and pre-processing

To improve the tolerance of the system to illumination changes, all images are equalized once they are acquired from the camera. That leads to a small performance improvement under different illumination conditions [2].

To improve memory usage and minimize chances of confusion between stored images, only images which are considered relevant are stored into the SDM. An image is considered *irrelevant* if the difference to the previous stored image, computed using the SDM's similarity calculation method, is less than the SDM's access radius. This procedure leads to filtering many images of corridors and other long monotonous scenarios, which improves the performance of the system.

D. Lateral drift correction

Since the robot is following the paths by following the same commands executed during the learning stage, small drifts inevitably occur. In order to prevent those drifts from accumulating a large error, a correction algorithm was also implemented, following Matsumoto et al.'s approach [8], as described in more detail in [2]. Once an image has been predicted, a block-matching method is used to determine the horizontal displacement between the robot's current view and its view during the learning stage. If a difference is found, the robot's heading is adjusted by proportionally decreasing *vr* or *vl*, in order to compensate the lateral drift.

E. Sequence disambiguation

When following a sequence, the data associated with each image that is retrieved from the memory is checked to determine if the image belongs to the sequence (path) that is being followed. Under normal circumstances, the robot is not expected to skip from one sequence to another. Nonetheless, under exceptional circumstances it may happen that the robot is actually moved from one path to another, a problem commonly known as the "kidnapped robot." Thus, if the robot consecutively retrieves a number of images (thirty was used) of a different sequence, then it switches to that new sequence and updates its assumed location.

F. Use of a sliding window

When the robot is following a path, it is also expected to retrieve images only within a limited range. For example, if it is at the middle of a long path, it is not expected to get back to the beginning or right to the end of the path. Therefore, the search space can be truncated to a moving "sliding window," within the sequence that is being followed, so that the search space for the robot at image *n* is limited to images

{ max{*0, n-w/2*}, min{*n+w/2, t*} }, for a sequence containing *t* images using a sliding window of width *w*.

The sliding window may prevent the robot from solving the kidnapped robot problem. To overcome the limitation, a all-memory search is performed first and a short-term memory retains whether the last *x* images were predicted within the sliding window or not (in the same sequence). When a prediction is out of the sliding window, but the count (short-term memory) is still under *x*, a search is made in the sliding window and this prediction is used for navigation. A number of consecutive predictions out of the sliding window is interpreted as if the robot had been kidnapped. In that case, the last prediction is assumed as being the most correct prediction for the robot's location, and the sliding window is moved to that location. In the present work fifty was used for *x* (smaller values did not work so well).

V. Navigation Algorithms Tested

Different navigation algorithms were implemented, and two of them are described in the following subsections: the simplest one and the most robust.

A. *Basic algorithm*

In the basic navigation algorithm implemented the search is performed in all memory. If the predicted image number jumps more than a third of the number of images of the whole sequence describing that path, that prediction is ignored and the robot maintains its movement.

The performance of this basic algorithm was tested indoors in the corridors of the Institute of Systems and Robotics of the University of Coimbra, Portugal. The robot was taught a path about 22 meters long, from a laboratory to an office, and then instructed to follow that path 5 consecutive times.

Figure 7 shows the numbers of the images that were taught and retrieved each time. The robot did not get lost and always reached a point very close to the goal point.

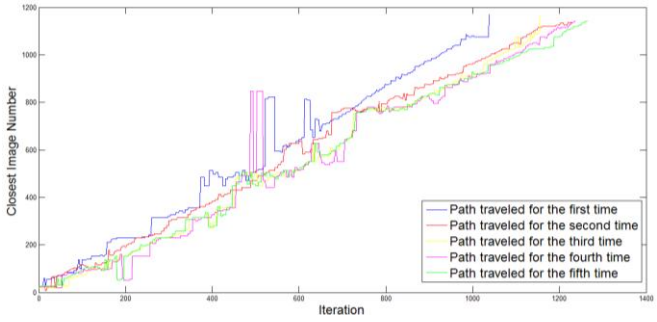


Figure 7- Images predicted by the SDM following the path from a laboratory to an office. The graph shows the number of the images that are retrieved from the memory as the robot progresses towards the goal.

In a second test, the robot was taught a path about 47 meters long. The results are shown in Figure 8. As the figure shows, the robot was not able to reach the goal, it got lost at about the 520th prediction in the first run and at the 810th in the second run.

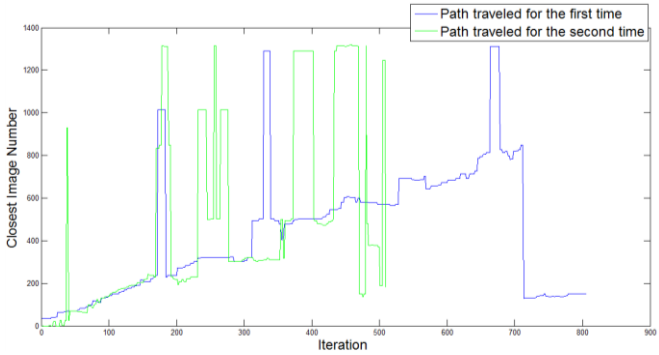


Figure 8 – Images predicted by the SDM following the second test path with the basic algorithm.

B. *Autonomous navigation with sliding window*

Most of the prediction errors happen where the images are not very rich and there are many similar views in the same path or other paths also stored in the memory. The use of a sliding window to narrow the acceptable predictions improves the results, but it is not enough. The improved sliding window algorithm with the other principles described in section IV worked in all situations.

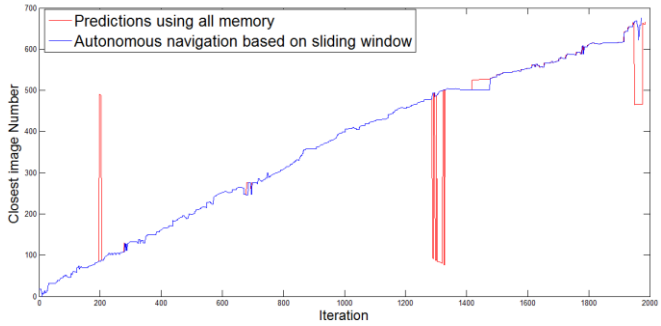


Figure 9- Images predicted by the SDM following the second test path using the sliding window algorithm.

Figure 9 shows the result obtained with this navigation algorithm when the robot was made to follow the same path used for Figure 8 (second test path). A sliding window 40 images wide was used. As the graph shows, the sliding window filters out all the spurious predictions which could otherwise compromise the ability of the robot to reach the goal. About iterations 200 and 1300 we can verify that some of the most similar images are out of the sliding window but those images were not used.

C. *The patrol mode*

To make the robot able to carry out patrol missions, a special navigation mode was developed, which is called "the patrol mode."

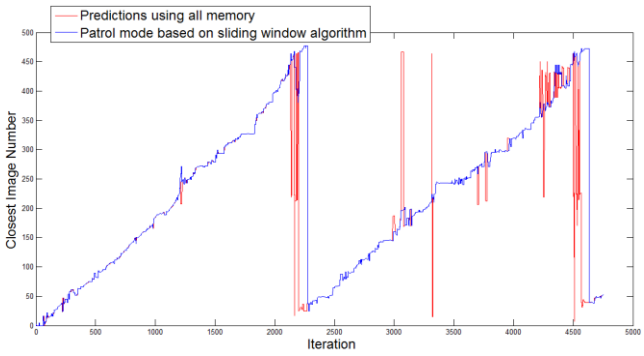


Figure 10 –Images predicted by the SDM following a path, using the sliding window in the patrol mode. Close to image 480 the robot gets back to the beginning of the sequence.

The patrol mode uses a closed path in which its end overlaps with its beginning. During the learning stage, the robot recognizes that the path ends at a point where the views coincide with those of the beginning of the same path. If the patrol mode is turned on, then the robot is able to jump directly from the end of a sequence to its beginning, thus following the same path continuously, until stopped by the operator. Figure 10 shows an example of the predictions made while following a path continuously twice. The path is 55 meters long and described by about 480 images in the SDM.

VI. Obstacle Avoidance

In order for the robot to navigate in real environments, it is necessary that the navigation process in autonomous mode is robust enough to detect and avoid possible obstacles in the way. Two algorithms were implemented, one for obstacles which appear in straight line paths and another for obstacles that appear in curves.

A. Obstacles in straight paths

In the autonomous navigation mode, the sonar sensors are activated. All objects that are detected at less than 1 m from the robot are considered obstacles and trigger the obstacle avoidance algorithm. A median of 3 filter was implemented to filter out possible outliers in the sonar readings.

When an obstacle is detected, the robot suspends memory-based navigation and changes direction to the side of the obstacle that seems free. The robot tries to circumvent the obstacle and logs the wheel movements in a stack. When the obstacle is no longer detected, the wheel movements logged are then performed in reverse order, thus returning the robot to its original heading. When the stack is emptied, the robot tries to localize itself again based on visual memories and resume memory-based navigation.

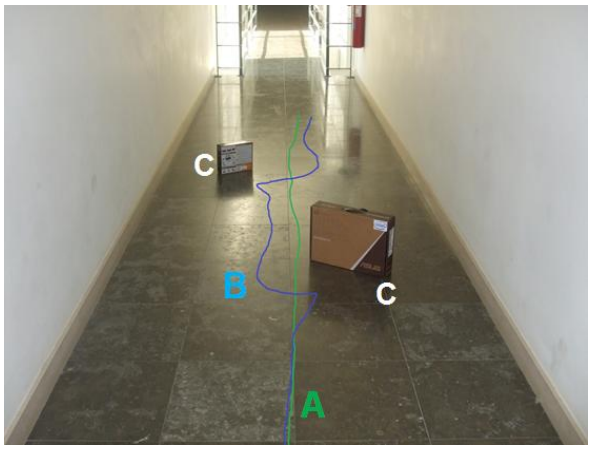


Figure 11- Example of obstacle avoidance (the lines were drawn using a pen attached to the rear of the robot, so they actually mark the motion of its rear, not its centre of mass). A) Path followed without obstacles. B) Path followed with obstacles. C) Obstacles.

Figure 11 shows an example of a straight path previously taught and later followed with two obstacles introduced in that path. After avoiding collision with the first obstacle, the robot resumes memory-based navigation maintaining its original heading, performing lateral drift correction for a while. It then detects and avoids the second obstacle and later resumes memory-based navigation maintaining its original heading. Note that in the figure, because the lines were drawn using a pen attached to the rear of the robot, when the robot turns to the left it draws an arc of a line to the right.

If the motions performed to circumvent the obstacle were not logged, or the stack not emptied, the robot would still most of the times correctly return to its original heading and path. But the use of the stack greatly improves the autonomous navigation performance, by making it possible to work with shorter safety distances and reducing the amount of time necessary for the robot to get back to the correct path.

B. Obstacles in curves

The stack method described in the previous subsection works correctly if the obstacle is not detected during a turn. If the obstacle interferes when the robot is adjusting its heading, then the stack method is not guaranteed to work, for it does not guarantee that the robot takes the correct heading after the obstacle is circumvented. Thus if obstacles are detected in turns (points where there are heading changes), the stack method is not used – the robot just circumvents the obstacle and gets back to vision-based navigation as soon as possible. In turns, the probability of confusion of images is not very high, even if the images are captured at different distances, the camera angle often has a more important impact than the distance. Figure 12 shows an example where the robot avoided an obstacle placed in a turn. The wall corner at the left is also detected as an obstacle, thus there is actually a second heading adjustment.

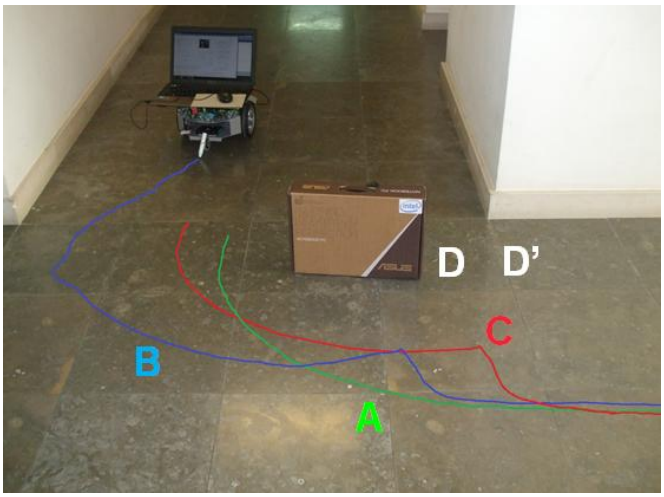


Figure 12 - Example of obstacle avoidance in a curve. A) Taught path. B) Path followed by the robot avoiding the obstacle D and the left wall corner. C) Path followed by the robot when the obstacle was moved to D'.

VII. CONCLUSIONS

A method of navigating a robot, using visual information stored into a SDM, has been proposed. The SDM is implemented in parallel in a GPU for better performance. The images, along with motion information, are stored into the SDM after a learning stage and used for navigation during the autonomous run. A sliding window is used to segment the search space and improves reliability.

An obstacle-avoidance algorithm was also proposed. The use of a stack to store the robot motions when circumventing obstacles showed good performance in straight paths. However, this method is not appropriate to avoid obstacles located in turns. In this case, the stack is not used, and experimental evidence shows that, after circumventing the obstacle, the robot adjusts its heading faster using memory-based navigation for localization and the drift-correction algorithm for heading adjustment.

Experimental results show excellent performance of the system. The robot is able to perform autonomously tasks such as surveillance, guiding people inside a building or carrying objects to previously taught places.

Future work will include development of higher level modules to implement advanced surveillance and cicerone behaviors. In the patrol mode the robot must detect changes in the environment which should trigger an alert, such as intruder detection. In the cicerone mode, the robot must be able to interact with people.

REFERENCES

[1] P. Kanerva, (1998). *Sparse Distributed Memory*. Bradford Books. ISBN 978-0262514699.

[2] Mateus Mendes, A. Paulo Coimbra, Manuel M. Crisóstomo, "Robot Navigation based on view sequences stored in a Sparse Distributed Memory," in *Robotica*, vol. 30, n.º4, Cambridge University Press, UK, July 2011.

[3] Rajesh P. N. Rao and Olac Fuentes, "Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory", *Machine Learning*, 31(1-3), pp.87-113, April 1998.

[4] Yoshio Matsumoto, Katsuhiro Sakai, Masayuki Inaba and Hirochika Inoue, "View-based approach to robot navigation", in *Proceedings of 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), 2000.

[5] Mateus Mendes, Manuel M. Crisóstomo, A. Paulo Coimbra, "Assessing a Sparse Distributed Memory Using Different Encoding Methods," in *Proc. of the 2009 World Congress on Engineering*, London, UK, July 2009.

[6] Bohdana Ratitch and Doina Precup, "Sparse distributed memories for on-line value-based reinforcement learning", In *ECML*, 2004.

[7] J. Snaider, S. Franklin, S. Strain and E. O. George, "Integer sparse distributed memory: Analysis and results", *Neural Networks, 46*, pp.144-153, 2013.

[8] André Rodrigues, André Brandão, Mateus Mendes, A. Paulo Coimbra, Fernando Barros and Manuel Crisóstomo, "Parallel Implementation of a SDM for Vision-based Robot Navigation," *13th Spanish-Portuguese Conference on Electrical Engineering* (13CHLIE), Valencia, Spain, July 2013.