FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Simulador de Batalhão de Limpeza

Vasco Cardoso Espinheira Rio

Programa de Mestrado em Engenharia Electrotécnica e Computadores

Orientador: Armando Jorge Sousa (Professor Doutor)

Junho de 2010

Resumo

Nesta dissertação é proposto um método para a limpeza de edifícios que pode ser classificado como dinâmico e inteligente. Espera-se ter demonstrado que a proposta é adequada para a solução do problema de limpeza de edifícios tal como o do Departamento de Engenharia Electrotécnica e de Computadores. Este método utiliza cooperação de robôs para minimizar o tempo de limpeza do edifício.

Para a limpeza de compartimentos é utilizada a Decomposição Baseada em Tabelas e posteriormente evoluida a ordem e o método de limpeza das áreas resultantes com Algoritmos Genéticos.

Na Navegação é introduzido o conceito de Navegação Local e Global, sendo a navegação Local responsável por deslocações dentro de cada compartimento e a navegação Global através de um grafo de pontos importantes, faz a navegação entre compartimentos diferentes, minimizando desta forma a complexidade da navegação resultante.

A alocação de tarefas foi feita com recurso a Leilões - Os recursos "Robôs" são leiloados pelos Compartimentos com necessidade de limpeza, obetendo-se assim cooperação e dinamismo na limpeza de edifícios.

Abstract

This dissertation proposes a method for cleaning of buildings that can be classified as dynamic and intelligent. It is expected to have demonstrated that the solution is suitable for solving the problem of cleaning in buildings such as the Department of Electrical and Computer Engineering. This method uses co-operation of robots to minimize the wast of cleanining time.

For the cleaning of compartments is used a Table Based Decomposition and subsequently evolved the order and method of cleaning the resulting areas with genetic algorithms.

In Navigation is introduced the concept of Local and Global Navigation. Local navigation is responsible for movements within each compartment and Global Navigation enables the navigation between different compartments, thereby underscoring the complexity of the resulting navigation.

Task allocation has been made resorting to the Auctions method. Resources "Robots" is auctioned by compartments in need of cleaning, thus obtain cooperation and dynamism in buildings cleaning.

Agradecimentos

Aos meus pais, ao meu irmão e à minha avó, que são as únicas pessoas no mundo com as quais uma zanga nunca dura mais de 15min. Pelo apoio que me deram, por me darem educação e amizade.

À minha namorada, aos meus amigos da faculdade, da terra, amigos de infância ou amigos mais recentes, pela fantástica vida que me têm proporcionado.

Às pessoas que sem remuneração económica desenvolveram Software que utilizei no desenvolvimento desta tese: Ubuntu, Latex, Debian, Gnome, Python, Psyco, Pygame, Dropbox, Google Code, Geany, Inkscape, Gimp, Openoffice Draw, Firefox, Tonktanks, Activestate e outros que ficaram esquecidos.

Vasco Rio

"Inventing is a combination of brains and materials.
The more brains you use, the less material you need."

Charles F. Kettering

Conteúdo

Abreviaturas e Símbolos **xvii**

1 Introdução **1**
1.1 Enquadramento e Motivação . . . 2
1.2 Objectivos . . . 2
1.3 Estrutura do Documento . . . 2

2 Revisão Bibliográfica **5**
2.1 Modelação de um Robô . . . 5
2.1.1 Robô diferencial . . . 6
2.2 Path Planning . . . 7
2.2.1 Algoritmo de *Dijkstra* . . . 7
2.2.2 A* . . . 7

3 Cooperação de Sistemas Robóticos **9**
3.1 Definição . . . 9
3.2 Arquitectura . . . 10
3.2.1 Centralização . . . 10
3.2.2 Diferenciação . . . 10
3.2.3 Comunicação . . . 10
3.2.4 Arquitecturas Relevantes . . . 11
3.3 Conflito de Recursos . . . 13
3.4 A Origem da Cooperação . . . 13
3.4.1 A Aprendizagem . . . 14
3.4.2 Distributed Artificial Inteligence . . . 15
3.5 Alocação de Tarefas . . . 15
3.5.1 Greedy . . . 15
3.5.2 Auctions . . . 16
3.6 Conclusões . . . 16

4 Area-Covering Algorithms **17**
4.1 Padrões de Varrimento . . . 17
4.2 Áreas Elementares . . . 18
4.3 Circundar Recursivamente . . . 19
4.4 Cellular Decomposition . . . 20
4.4.1 Trapezoidal Cellular Decomposition . . . 20
4.4.2 Boustrophedon Cellular Decomposition . . . 21
4.5 Genetic Algorithms . . . 22

4.6 Outras estratégias de *Area-Covering* . . . 22
4.7 Conclusão . . . 22

5 Implementação **25**
5.1 Escolha da linguagem . . . 25
5.2 Arquitectura da Cooperação . . . 26
5.3 Estrutura dos Dados . . . 27
5.4 Estrutura do Processamento . . . 28
5.5 Obter mundo . . . 30
5.5.1 Processamento do projecto CAD . . . 30
5.5.2 Criar objectos a partir da imagem . . . 31
5.5.3 Guardar Resultados . . . 32
5.6 Navegação . . . 33
5.6.1 Navegação Local - Malha . . . 34
5.6.2 Navegação Global . . . 35
5.7 Simulação da Limpeza . . . 37
5.7.1 Grelha de Sujidade . . . 38
5.7.2 Robô . . . 38
5.8 Estratégia da Limpeza . . . 41
5.8.1 Área a limpar . . . 41
5.8.2 Decomposição Celular . . . 42
5.8.3 Algoritmos Genéticos . . . 46
5.9 Alocação das Tarefas . . . 54
5.9.1 Partir um Grafo . . . 55
5.9.2 Auction . . . 55

6 Resultados **59**
6.1 Area-Covering . . . 59
6.1.1 Evolução Genética . . . 59
6.1.2 Limpeza de Um Compartimento . . . 61
6.2 Task Alocation . . . 63
6.3 Navegação . . . 65
6.4 Utilização de Robôs com Características Diferentes . . . 66
6.5 Múltiplos mapas . . . 68
6.5.1 Limpeza do Piso 1 Oeste . . . 68
6.5.2 Limpeza do Piso 0 . . . 69
6.5.3 Limpeza do Piso -1 . . . 70
6.5.4 Limpeza de Pisos . . . 71
6.6 Limpeza do Departamento de Electrotecnia . . . 71

7 Conclusões e Trabalho Futuro **75**
7.1 Conclusão . . . 75
7.2 Lições Aprendidas . . . 76
7.3 Satisfação dos Objectivos . . . 77
7.4 Trabalho Futuro . . . 77
7.4.1 Acrescentar realismo ao simulador: . . . 77
7.4.2 Acelerar a simulação . . . 78
7.4.3 Melhorias nos algoritmos genéticos . . . 78

Referências **81**

Lista de Figuras

2.1 Modelo matemático de um robô que se desloca num plano horizontal. 6
2.2 A média das velocidades de um robô diferencial consiste na sua velocidade linear. 6
2.3 Composição das velocidades de um robô diferencial. 7

3.1 Sistema robótico com uma Unidade Central de Processamento. 10
3.2 Exemplo de uma arquitectura CEBOT: *MTRAN 3* [1]. 11
3.3 Exemplo de uma arquitectura *Swarm*: *Jasmine micro-robots* [2]. 12
3.4 Exemplo de um GOFER: *Small Soccer League* [3]. 12

4.1 Exemplo de varrimento circundante. 17
4.2 Exemplo de varrimento em *S*. 18
4.3 É impossível limpar uma área não elementar com uma estratégia elementar. . . . 18
4.4 Área não elementar antes da decomposição. 20
4.5 Área Dividida pelo método da decomposição celular trapezoidal. 20
4.6 Área Dividida pelo método da decomposição celular de *Boustrophedon*. 21
4.7 Trajectória em *S* na divisão celular de *Boustrophedon*. 22

5.1 Estrutura dos Dados do *cleanSim* (Simulador desenvolvido). 27
5.2 Estrutura do Processamento do *cleanSim*. 29
5.3 Ficheiro do mapa original. 30
5.4 Imagem obtida a partir do CAD 31
5.5 Imagem obtida a partir da sombra dos compartimentos a limpar 31
5.6 Piso com todos os Objectos identificados. 32
5.7 Mapa segmentado de um compartimento 34
5.8 Navegação Local - 1ª Iteração 34
5.9 Navegação Local - 2ª Iteração 35
5.10 Trajectória de navegação da posição do Robô "R"a porta marcada a azul na direita. 35
5.11 Malha de um edifício **sem** divisão de compartimentos. 36
5.12 Malha de um edifício **com** divisão de compartimentos. 36
5.13 Exemplo de um edifício 36
5.14 Exemplo de um grafo da navegação Global do edifício de exemplo. 37
5.15 Camadas utilizadas para a simulação da sujidade no mundo. 38
5.16 Fotografia do cleanRob. 39
5.17 Representação do modelo do Robô de limpeza. 40
5.18 Representação do modelo do Robô de limpeza com uma rotação. 40
5.19 cleanAreas do Roomba 41
5.20 Área Inicial 43
5.21 Área Segmentada 43
5.22 Renumeração das células 44

5.23 Agrupar em células da tabela 44
5.24 Agregação das células. 45
5.25 Renumeração das células 45
5.26 Área Final 46
5.27 Pormenor do mapa, referente ao compartimento *r*1 49
5.28 Áreas do Exemplo 50
5.29 Estrutura do cromossoma para a limpeza do compartimento *r1* 50
5.30 Método de limpeza das áreas de *r1* 51
5.31 Caminho da área 1 para a área 7. 51
5.32 Gene não evoluido. 52
5.33 Melhor gene obtido para o compartimento *r1*. 54
5.34 Calculo das distâncias do robô amarelo a cada um dos compartimentos. 56
5.35 Calculo da soma das distâncias dos robôs verdes a cada um dos compartimentos. 56
5.36 Tempo de limpeza de cada compartimento. 56
5.37 Valor licitado por cada um dos compartimentos no robô Amarelo 57

6.1 Evolução do tempo de limpeza face à evolução genética. 60
6.2 Pormenor da parte da imagem que deu origem ao compartimento 6 do piso 1. . . 61
6.3 Caminho Empírico. 61
6.4 Caminho Circundante 62
6.5 Caminho Genético. 62
6.6 Desempenho comparativo dos diferentes métodos de alocação de tarefas. 64
6.7 Navegação entre dois compartimentos. 65
6.8 Navegação Óptima entre dois compartimentos. 66
6.10 Comparativo entre a malha de limpeza do *cleanRob* (lado esquerdo) e do *Roomba* (lado direito). 67
6.9 Robô com modelo do *Roomba* limpa o piso 0 do DEEC 67
6.11 Piso 1 Oeste da Departamento de Electrotecnia 68
6.12 Piso 0 da Departamento de Electrotecnia 69
6.13 Piso -1 da Departamento de Electrotecnia 70
6.14 Desempenho comparativo dos diferentes métodos de alocação de tarefas. 72

Lista de Tabelas

5.1 Connectividade-8 32
5.2 Combinações de métodos de limpeza de cada área 47

6.1 Evolução Genética no piso 0 do Departamento de Electrotecnia da FEUP 60

Abreviaturas e Símbolos

CAD	*Computer-aided design*
CBR	*Case Based Reasoning*
CCPP	*Complete Coverage Path Planning*
CR	Circundar Recursivamente
CTPS	*Central Task Planning System*
DEEC	Departamento de Engenharia Electrotécnica e de Computadores
FEUP	Faculdade de Engenharia da Universidade do Porto
TBD	*Table Based Decomposition*

Capítulo 1

Introdução

O tema desta dissertação está inserido no desenvolvimento de sistemas robóticos cooperativos para a limpeza de edifícios. A limpeza de edifícios automatizada é um tema que, apesar de ter já mais de 20 anos, continua a ser de dificuldade acrescida dado a sua complexidade.

Nesta dissertação procura-se encontrar uma estratégia para a limpeza de edifícios com recurso à cooperação de robôs. Esta limpeza está enquadrada na Faculdade de Engenharia da Universidade do Porto (FEUP) e o edifício de Estudo é o Departamento de Engenharia Eletrotécnica e de Computadores (DEEC). Será dada ainda importância ao dinamismo da solução desenvolvida, pelo facto de suportar robôs e edifícios com características diferenciadas.

A dissertação centra-se em três grandes temas: O primeiro são os *Area-covering Algorithms* os quais são algoritmos que procuram encontrar uma trajectória que passe em todos os pontos de uma área. O segundo grande tema é o *Path-Planning* o qual consiste na navegação dentro de um edifício. O terceiro é a alocação de tarefas a robôs, no qual cada robô é responsabilizado pela limpeza de um segmento.

Esta dissertação enquadra-se ainda no seguimento do projecto desenvolvido pelo Professor Doutor Armando Jorge Sousa - o *cleanRob* [4] [5] [6] [7] [8] [9] [10]. O *cleanRob* é um robô de limpeza desenvolvido na FEUP.

O DEEC (edifício de estudo) é um edifício com construção industrial: corredores largos e amplos e a ausência móveis ou cadeiras nos espaços comuns. Por esse motivo, as tarefas de limpeza neste edifício são em grande parte a limpeza do chão; ideal para um sistema de limpeza de robótico.

Nesta dissertação será procurada uma arquitectura para a limpeza deste tipo de edifícios que inclua a cooperação e a navegação dentro do edifício. Será também desenvolvido um simulador que permita testar diferentes estratégias e que a partir destas possibilite obter medidas de desempenho.

1.1 Enquadramento e Motivação

Os Robôs de Limpeza serão certamente, num futuro próximo, dos sistema autónomos mais presente no nosso dia a dia [11]. Em 1991 *Hitachi* e *Sanyo* apresentaram os primeiros protótipos, que mostravam que a tecnologia já existia. Na verdade, uma década passou até que se começasse a ver os primeiros robôs comerciais. Apesar das altas expectativas em robôs de limpeza para utilização caseira, os primeiros protótipos foram principalmente utilizados para fins lúdicos/académicos [11]. Um dos motivos que levam os robôs de limpeza a não serem ainda vistos como produto para o mercado grossista é a enorme complexidade que é limpar: Desde coisas simples como encravar num tapete, a situações complexas como arrastar cadeiras ou distinguir um telemóvel de uma lata. As pessoas que procuram um robô de limpeza para as suas casas, procuram um que resolva todos estes problemas e tal ainda não existe. Assim, aquelas que realmente compraram um para este fim, acabaram por ficar desiludidas com o mesmo e a publicitá-lo de forma negativa [12]. Já na limpeza de áreas comerciais e edifícios industriais a realidade não é a mesma uma vez que em grandes áreas livres os robôs de limpeza apresentam bons resultados.

1.2 Objectivos

Os objectivos desta dissertação focam-se em encontrar uma solução para a limpeza de edifícios com uma equipa de robôs. A solução terá que incluir um simulador, de forma a testar resultados e a ser possível testar diferentes estratégias. O simulador desenvolvido deverá prezar pelo dinamismo e a estruturação, de modo a que seja um projecto com continuidade.

Assim, o projecto deverá cumprir os objectivos:

- Desenvolver um simulador para uma equipa de robôs de limpeza.
- O simulador deve ser dinâmico e facilmente reconfigurado.
- Deverão ser ponderadas estratégias de cooperação diferenciadas.
- O simulador deve permitir acelerar o tempo e retirar resultados da limpeza.
- Deve ser dada a resposta à questão: *Qual é o número de cleanRobs necessário para limpar o Departamento de Electrotecnia durante uma noite?*

1.3 Estrutura do Documento

O Documento desenvolvido é composto por 7 capítulos que pretendem relatar e estruturar o trabalho desenvolvido. No capítulo 1 é dada uma introdução ao problema a desenvolver, bem como são descritos os objectivos do projecto. No capítulo 2 é mostrado o estado da arte de temas que apesar de não serem essência do projecto, foram igualmente utilizados. Os tópicos principais da cooperação robótica, desde de arquitecturas à inteligibilidade de robôs são dados a conhecer no capítulo 3. As técnicas actualmente utilizadas para *Area-covering* são apresentadas no capítulo 4.

O capítulo 5 contém os pontos essenciais do Software desenvolvido, incluindo alguns algoritmos inventados. Os resultados dos testes ao *software* desenvolvido são apresentados no capítulo 6. No capítulo 7 é dada uma conclusão a todo o trabalho, referindo-se os pontos a reter do sistema desenvolvido.

Capítulo 2

Revisão Bibliográfica

Neste capítulo são sumariados tópicos relevantes utilizados na dissertação. O capítulo começa com a modelação de um robô diferencial, partindo para o Planeamento de trajectórias. Com a modelação de um robô diferencial pretende-se mostrar o paralelismo existente entre o mundo simulado (velocidades em v e ω) e o mundo real (velocidades nas rodas V_1 e V_2). No planeamento de trajectórias são apresentadas as bases do algoritmo de *Dijkstra* e do *A**.

2.1 Modelação de um Robô

Segundo [13], o modelo matemático para modelar um robô que se desloca num plano horizontal contem três graus de liberdade:

$$(x, y, teta) \tag{2.1}$$

onde teta é o ângulo que o robô faz com o sistema de coordenadas x, y.

Este modelo matemático, parte da simplificação de que o robô responde instantaneamente a um controlo gerado pelo controlador, não sendo considerados os efeitos dinâmicos resultados do movimento do sistema robótico [14]. As velocidades em cada um dos graus de liberdade são dadas por:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \omega \\ v \end{bmatrix} \tag{2.2}$$

Para instantes próximos, a posição em função da anterior pode ser aproximada por:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x_0 + v dt \cos(\theta_0) \\ y_0 + v dt \sin(\theta_0) \\ \theta_0 + \omega dt \end{bmatrix} \tag{2.3}$$

onde v e ω são, respectivamente, as velocidades linear e angular.

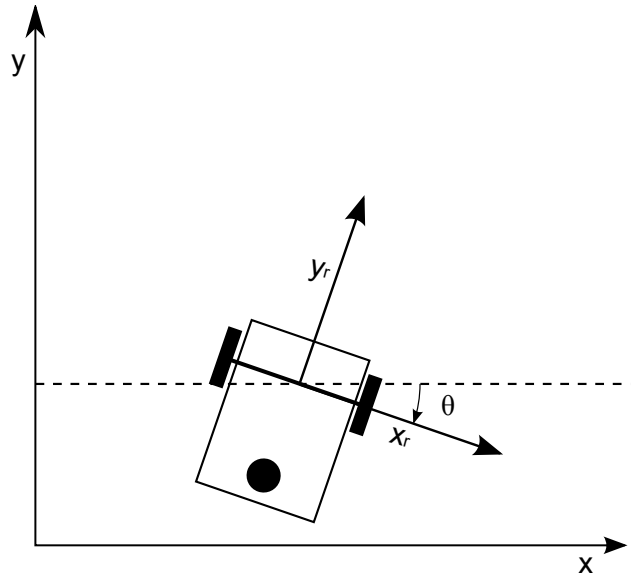


Figura 2.1: Modelo matemático de um robô que se desloca num plano horizontal.

2.1.1 Robô diferencial

O robô diferencial 2.2 é constituído por duas rodas motrizes (V_1 e V_2) e uma roda livre.

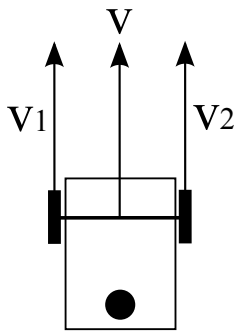


Figura 2.2: A média das velocidades de um robô diferencial consiste na sua velocidade linear.

Como se pode ver na figura 2.3, as rodas motrizes são controladas em conjunto, de forma a controlar v e ω.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{V_2+V_1}{2} \\ \frac{V_2-V_1}{d} \end{bmatrix} \tag{2.4}$$

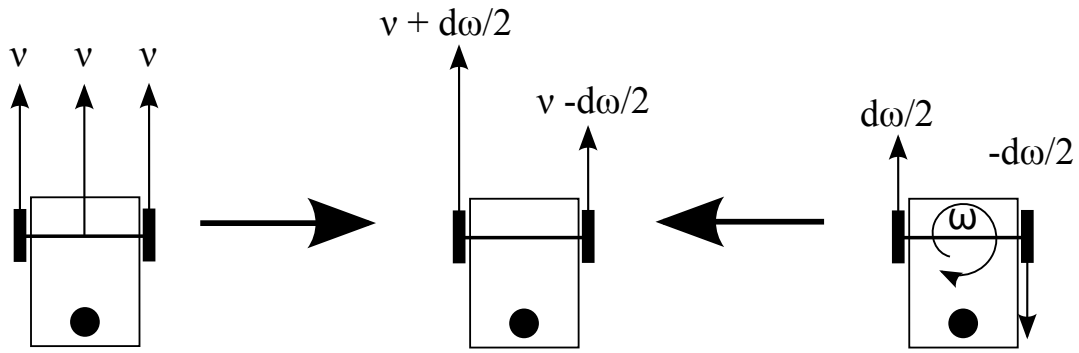


Figura 2.3: Composição das velocidades de um robô diferencial.

2.2 Path Planning

O planeamento de trajectórias é a técnica utilizada para a partir de dois pontos - "partida" e "chegada" - Encontrar o melhor caminho que os une. Nas secções seguintes são apresentadas técnicas para encontrar tais caminhos.

2.2.1 Algoritmo de *Dijkstra*

O algoritmo de Dijkstra foi desenvolvido por *Edsger Dijkstra* em 1959 para a resolução do problema de *Shortest Path* (Caminhos Mínimos). O algoritmo de *Dijstra* tem por base um grafo no qual os nós são posições e os segmentos entre nós são os caminhos. Neste grafo, o peso dos caminhos pode ser dado pela distância, pelo tempo, pelos limites de velocidade, entre outros, dependendo da aplicação.

O Algoritmo de *Dijkstra* é um procedimento iterativo que partindo de um nó inicial do grafo, cresce em todos os sentidos, guardando o caminho para cada ponto bem como o peso do caminho total até cada um destes. Após inicialização, o mesmo procedimento é chamado no ponto com menor peso. Se um ponto for encontrado várias vezes, só a primeira é considerada (a de menor peso). O algoritmo termina quando é encontrado o ponto do destino. [15]

2.2.2 A*

O *A** tem por base o Algoritmo de *Dijkstra*, mas necessita da noção do espaço global ou seja, da noção da posição de cada ponto no mundo. A partir das posições consegue calcular a distância de cada ponto descoberto ao destino pretendido e usa-la para escolher que ponto explorar [16].

Capítulo 3

Cooperação de Sistemas Robóticos

Este capitulo procura instruir o leitor sobre os princípios da Cooperação em Sistemas Robóticos. Para tal, e à semelhança de [17], este está organizado por tópicos orientadores. Partindo das Arquitecturas até Múltiplo planeamento de Trajectórias, passando pela Resolução de Conflitos, a Origem da Cooperação e os Métodos da Aprendizagem.

Existem duas razões chaves que levaram ao desenvolvimento de sistemas robóticos móveis: A primeira é a redução da necessidade da presença humana em locais perigosos tais como: limpeza de resíduos tóxicos, desactivação de resíduos nucleares, exploração planetária, missões de procura e salvamento ou segurança; A segunda chave é a redução das tarefas repetitivas feitas pelo homem: Vigilância, limpeza de pavimentos, corte de relva.

A natureza de muitos destes desafios leva a que os sistemas robóticos sejam completamente autónomos a atingir os objectivos. Uma solução para este tipo de problema passa por criar um robô que consiga cumprir completamente a tarefa especificada. Porém, existem vários factores que levam a que esta nem sempre seja possível haver um só robô: Requisitos temporais apertados, complexidade das tarefas a realizar, dimensões dos robôs. Assim a solução adoptada para o problema passa por criar um sistema cooperativo robótico com características heterogéneas e homogéneas, que colaborem para atingir um objectivo. [18]

3.1 Definição

Dada uma determinada tarefa especificada pelo *designer*, um sistema multi-robô demonstra comportamentos cooperativos se, devido a algum mecanismo próprio, houver aumento na utilidade total do sistema [17].

À semelhança de [17], nesta dissertação não é feita distinção entre cooperação e colaboração.

3.2 Arquitectura

Na definição de um sistema cooperativo robótico, o primeiro passo é definir a sua arquitectura.

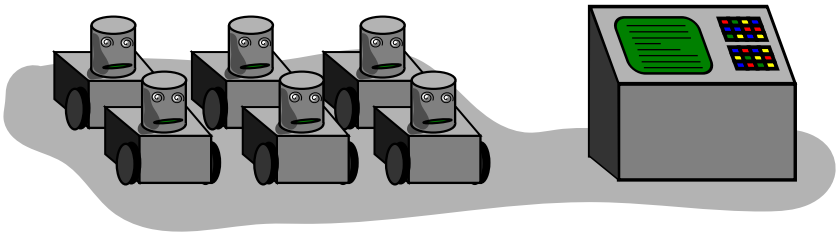


Figura 3.1: Sistema robótico com uma Unidade Central de Processamento.

A arquitectura de um sistema de computação é definida como a parte de um sistema que se mantém inalterada, excepto na intervenção de um agente externo. [17]

3.2.1 Centralização

Pode-se separar o mundo das arquitecturas de sistemas robóticos em dois grandes grupos:

Centralizado: Apenas um elemento de controlo.

Descentralizado: Sistemas com vários elementos que controlam. Podem ainda ser divididas em hierárquicos e distribuídos.

As arquitecturas Centralizadas tem a grande vantagem de serem mais simples de implementar, mas exigem uma constante comunicação entre todos os elementos envolvidos. As descentralizadas tem a vantagem de serem mais resistentes a falhas e maior exploração do paralelismo.

3.2.2 Diferenciação

Num sistema robótico, muitas vezes há a necessidade de interagir com robôs iguais (homogéneos) ou com características diferentes (heterogéneos) por forma a atingir o objectivo esperado. Esta diferenciação tem que ser bem ponderada no planeamento e escalonamento das acções dos robôs.

3.2.3 Comunicação

A comunicação é a base da interacção num sistema robótico, e como tal determina as capacidades globais do mesmo. *Uni Cao* em [17] propõe a separação dos tipos de comunicação em três grupos de interacção:

Comunicação por Ambiente

Este tipo de comunicação ocorre quando o ambiente em si é o meio de comunicação. Ou seja não há necessidade de definir a comunicação entre os intervenientes, pois esta está implícita. Numa situação prática, é a partilha de memória. Este modo de comunicação é o mais simples e mais poderoso, embora por vezes seja impraticável.

Comunicação por Percepção

Na interacção por sentidos, à semelhança da por Ambiente, não há a necessidade de uma comunicação explícita. Um robô por exemplo olha para um segundo e apercebe-se que o segundo está atarefado.

Comunicação Explícita

Uma comunicação Explícita é uma comunicação na qual há uma troca de informação entre intervenientes e ambos se apercebem disso. Uma comunicação explícita pode ser ainda direccionada ou por *broadcast*.

3.2.4 Arquitecturas Relevantes

Existem várias arquitecturas de sistemas robóticos cooperativos, sendo que grande parte delas são baseadas em modelos existentes na Natureza. Em seguida são apresentadas algumas destas arquitecturas.

CEBOT

Figura 3.2: Exemplo de uma arquitectura CEBOT: *MTRAN 3* [1].

CEllular roBOTics System é uma arquitectura descentralizada hierárquica inspirada pela organização das células. Nesta arquitectura, cada robô é uma célula autónoma que pode ser acoplada fisicamente a outras células. A partir desta reconfiguração o CEBOT espera adaptar-se às necessidades.

Na hierarquia CEBOT, existem células mestre que coordenam sub-tarefas e que comunicam com outras células mestre [17] [1].

Na figura 3.2 é representado uma arquitectura CEBOT, neste caso *MTRAN 3*. Os *MTRAN 3* conseguem-se conjugar num robô de maior tamanho e desenvolver tarefas como dançar ou serpentear.

Swarm

Figura 3.3: Exemplo de uma arquitectura *Swarm*: *Jasmine micro-robots* [2].

O *Swarm* (Enxame) é um sistema distribuído com um grande número de robôs. Numa arquitectura *Swarm*, os robôs não demonstram inteligência separadamente, mas sim quando cooperam. Normalmente as arquitecturas *Swarm* utilizam robôs homogéneos [17] [2].

Na figura 3.3 está representado um sistema de robôs *Swarm*, os *Jasmine micro-robots*. Estes robôs têm a forma de um cubo com lado em torno dos 2cm. Na figura alguns deles deslocam-se para a borda onde recarregam as suas baterias.

GOFER

Figura 3.4: Exemplo de um GOFER: *Small Soccer League* [3].

A Arquitectura GOFER é constituída por uma equipa de robôs e um CTPS (*Central Task Planning System*). O CTPS tem uma visão global do mundo (Interacção via Ambiente), das tarefas

e das disponibilidades. Inicialmente o CTPS recebe um objectivo e parte-o em tarefas. Depois utiliza um algoritmo de alocação de tarefas para determinar o papel de cada robô. Tendo um robô uma tarefa alocada, utiliza técnicas da Inteligência Artificial para escolher a melhor forma de a cumprir [19].

Na figura 3.4 é mostrada *Small Soccer League*. Nesta liga, um computador central processa a posição da bola e de todos os jogadores. Depois informa cada um dos robôs com a táctica a utilizar.

ALLIENCE

ALLIANCE é uma arquitectura completamente distribuída que utiliza selecção de acções adaptativa para atingir controlo cooperativo em missões do robô envolvendo baixo acoplamento e independência de tarefas. Os robôs desta arquitectura possuem uma variedade de funções de alto nível, que podem utilizar durante a missão, e devem seleccionar a acção certa baseados nos requisitos da missão, nas actividades dos outros robôs, nas condições do ambiente e nos seus próprios estados internos [18].

3.3 Conflito de Recursos

Na secção anterior 3.2 foram apresentadas várias arquitecturas para a cooperação entre robôs. Um tema de grande importância na implementação dessas arquitecturas é como se distribuem os recursos entre os diferentes robôs, quando os requisitam, quando os devem abandonar.

Quando um recurso indivisível é requisitado por múltiplos entidades, nasce um conflito de recursos. No caso dos robôs, acontece conflito de recursos quando por exemplo há necessidade de partilhar espaço, manipular objectos ou de comunicar.

Para resolver este tipo de problemas, utilizam-se técnicas já utilizadas anteriormente em Software (Sistemas Operativos) na eliminação de *Deadlock*: *Avoidance*, *Detection* e *Recovering*.

3.4 A Origem da Cooperação

A Origem da cooperação é um tema de enorme importância na Robótica Cooperativa: *A cooperação deve estar implícita no design ou deverá surgir com a aprendizagem do robô?* Na maioria das soluções utilizadas hoje, a cooperação é implícita no design. Alguns autores [17] [20] acreditam que a cooperação deve surgir pela necessidade dos Robôs cumprirem os seus objectivos.

No mundo não robótico, a verdade é que ambas se verificam. Os seres humanos são seres programados, mas com capacidades adaptativas. A cooperação é implícita mas a forma como é aplicada depende das vivências dos espécimes.

3.4.1 A Aprendizagem

Encontrar os parâmetros ideais para o controlo dos robôs é muitas vezes impossível dada a complexidade de alguns problemas. Para se encontrar bons parâmetros, garantindo a robustez da solução, é altamente desejável que estes parâmetros sejam obtidos pela experiência do robô de forma a optimizarem a sua performance e a adaptarem-se ao ambiente envolvente.

Para se encontrar parâmetros de forma inteligente, existem várias técnicas com princípios bastante diferentes.

3.4.1.1 Case Based Reasoning

O método *Case Based Reasoning* (Sistema Baseado em Casos) é uma forma de resolver problemas (casos) com base em problemas anteriormente encontrados. O CBR é um método baseado na experiência muito utilizado em diagnóstico médico e mecânico. Dado se tratar de um método utilizado por máquinas e por humanos, é particularmente interessante em situações em que existe a interacção entre ambos [21].

O procedimento do CBR é composto por 4 passos:

Recolha Dado um problema, é necessário recolher da memória todos os problemas relevantes na resolução deste.

Reutilização Adapta uma solução anterior ao problema actual.

Revisão Testa a solução adaptada.

Retenção Retém o resultado da solução adaptada para ser utilizado posteriormente.

Em [22] é apresentada uma optimização de um plano de inspecção em robôs, com a utilização de CBR.

3.4.1.2 Genetic Algorithms

Algoritmos Genéticos é um procedimento exploratório baseado na Teoria da Evolução de *Darwin*, que procura encontrar soluções próximas da óptima em problemas complexos e em tempos reduzidos. Para isso, ele mantém um conjunto de amostras (chamadas de indivíduos) e força-os a evoluir no sentido da solução aceitável [23] [24].

O procedimento dos algoritmos genéticos é composto por:

Avaliação da população Classifica-se o quão bem um individuo resolve um problema.

Reprodução É gerado um número de cópias iguais a cada individuo (com o mesmo genoma), directamente proporcionais à sua avaliação.

Recombinação São combinados, de forma aleatória, os genes dos indivíduos reproduzidos.

Mutação Para expandir o campo de pesquisa são introduzidos bits aleatórios em alguns dos novos indivíduos.

O procedimento anterior corre indeterminadamente ou até uma condição definida pelo designer.

Este algoritmo é utilizado por exemplo em jogos de computador, desenvolvendo em relativamente pouco tempo estratégias que o computador utiliza contra humanos. É também utilizado na robótica, em algoritmos de cobertura de superfícies como será visto na secção 4.5 [25].

3.4.1.3 Neural Network

Tal como os Algoritmos genéticos, As redes neuronais são uma abordagem biológica da Inteligência Artificial. Ao contrário dos algoritmos genéticos que assentam na evolução do genoma, as redes neuronais têm por base o funcionamento do cérebro: Os neurónios e as respectivas ligações.

As redes neuronais são constituídas por um conjunto de neurónios dispostos em rede, podendo o tamanho desta ser dinâmico ou estático. A aprendizagem da rede é dada pela variação do valor dos neurónios e das ligações entre eles. Uma rede neuronal pode ter várias entradas e saídas. O treino é dado por uma medida do resultado obtidos das acções anteriores [26].

3.4.2 Distributed Artificial Inteligence

Sistemas com *Inteligência Artificial Distribuida* (DAI) podem ser definidos como grupos de agentes que actuam em conjunto para resolver um problema. Na DAI os agentes desenvolvem comportamentos sociais, comportamentos que beneficiam o grupo [27].

3.5 Alocação de Tarefas

Um dos métodos utilizados na robótica para distribuir problemas complexos, é a alocação de tarefas a cada robô. Dois dos métodos utilizados na alocação de tarefas são o *Greedy* (Guloso) e os *Auctions* (Leilões).

O *Greedy* é um método que apenas olha para o presente, é um método que atribui ao robô a tarefa mais próxima. Solução que pode não ser compatível com um bom resultado global. Nas *Auctions*, cada tarefa licita um valor num robô, valor este que representa a necessidade de resolução da tarefa, proximidade ao robô e outros valores que necessitem de ser modelados.

3.5.1 Greedy

No Greedy existem tarefas a resolver e um robô disponível. Ao robô é atribuída a tarefa mais próxima dele [28].

3.5.2 Auctions

Uma das dificuldades nos sistemas multi-robô é muitas vezes decidir qual o robô que deve desenvolver que tarefa. Conforme referido em [29], o *Auction* (Leilão) é um protocolo distribuído de negociação que aloca recursos a tarefas. Este protocolo é estruturado em 4 pontos:

Anúncio de nova tarefa Surge uma tarefa para desenvolver e é iniciado um leilão para atribuir um Agente que resolva a tarefa.

Avaliação Todos os Agentes avaliam a tarefa, tentando da melhor forma caracterizar o seu desempenho a resolve-la.

Submissão de ofertas Cada Agente faz uma oferta no leilão proporcional ao seu desempenho.

Fechar o leilão É terminado o Leilão. O Agente que licitou mais fica responsável pela tarefa.

3.6 Conclusões

A cooperação robótica é um tema de grande actividade e talvez um dos principais temas neste início do século XXI. Neste capítulo mostrou-se arquitecturas de sistemas robóticos, a origem da cooperação e a alocação de tarefas. É notório que a abordagem dada normalmente a estes sistemas é muitas vezes inspirada na vida, inspirada em técnicas que seres vivos utilizam com sucesso.

Neste capítulo também se pode destacar que dependendo da abordagem existem soluções para a cooperação robótica muito diferentes. Compare-se por exemplo a abordagem do CEBOT com a SWARM. Apesar de ambos serem compostos por robôs com poucos centímetros, os CEBOT comportam-se como células, que se unem para constituir um robô maior; os SWARM comportam-se como formigas no qual todos trabalham com vista a obter o bem da colmeia, mas sem constituírem um ser maior.

Capítulo 4

Area-Covering Algorithms

Area-Covering é um tipo de *Complete Coverage Path Planning* (CCPP). Robôs que seguem um algoritmo de *Area-Covering* são orientados a passar por todos os pontos de um plano. Este tipo de algoritmo é particularmente interessante em cortadores de Relva, situações de busca e salvamento, pintura, procura de minas terrestres ou limpeza [30].

Na limpeza de um compartimento normalmente não é evidente qual a forma mais rápida de o limpar, qual a forma na qual se percorre uma menor distância ou de que forma se pode evitar múltiplas passagens pelo mesmo ponto. Assim é muitas vezes complexo encontrar um caminho óptimo para cobertura total da superfície. Neste capítulo são apresentadas técnicas para a cobertura de superfícies que procuram de alguma forma minimizar as perdas de tempo e energia.

4.1 Padrões de Varrimento

Defina-se um padrão de varrimento como uma técnica CCPP que é composta por uma malha e uma método para adaptar a malha a uma superfície, tornando-a numa trajectória. Neste documento fala-se de dois padrões de varrimento: os varrimentos circundantes e os varrimentos em *S*.

Varrimento Circundante

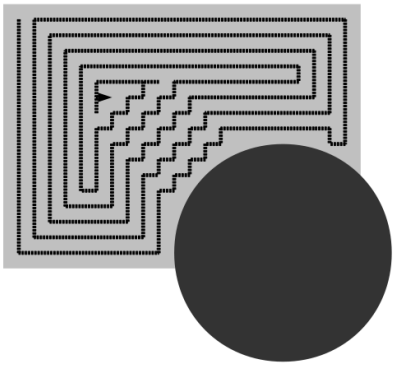


Figura 4.1: Exemplo de varrimento circundante.

No varrimento circundante o robô percorre uma superfície a varrer sempre encostado à margem exterior que ainda não foi varrida. Na figura 4.1 pode-se ver um exemplo deste tipo de varrimento.

Varrimento em *S*

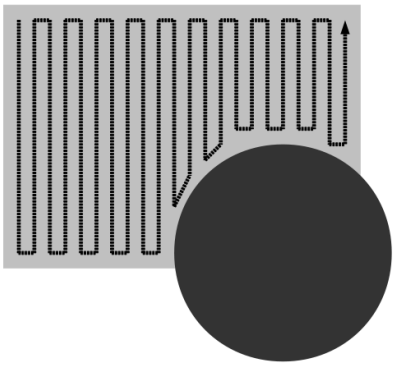


Figura 4.2: Exemplo de varrimento em *S*.

No varrimento em *S* o robô encaixa uma malha de linhas paralelas na área a cobrir. Após isso, une as linhas paralelas de forma a obter uma trajectória única. Na figura 4.2 pode-se ver um exemplo deste tipo de varrimento.

4.2 Áreas Elementares

Defina-se uma área elementar como uma área que pode ser facilmente coberta por um padrão de varrimento. Ou seja, uma área é elementar se existir um caminho composto por uma, e uma só, trajectória em elipse ou em *S* que cobre completamente a área . É possível demonstrar que todas as áreas convexas são Elementares.

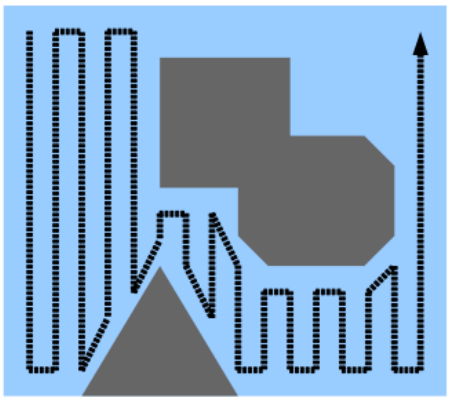


Figura 4.3: É impossível limpar uma área não elementar com uma estratégia elementar.

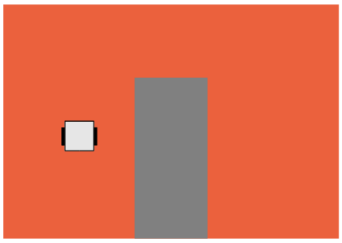
Em 4.3 pode-se ver um exemplo de uma área não elementar. Como se verifica, a trajectória em S não consegue cobrir completamente a área.

4.3 Circundar Recursivamente

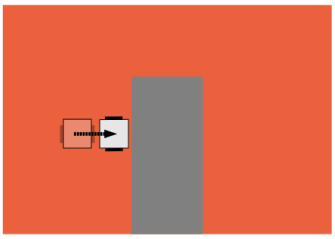
Circundar Recursivamente (CR) é um método dinâmico e recursivo mas que não contém qualquer Inteligência Artificial. O algoritmo CR é utilizado muitas vezes por operadores humanos de máquinas de corte de relva. Apesar da sua simplicidade, o algoritmo mostra-se muito versátil e com bons resultados (ver algoritmo 1).

Algorithm 1 Circundar Recursivamente

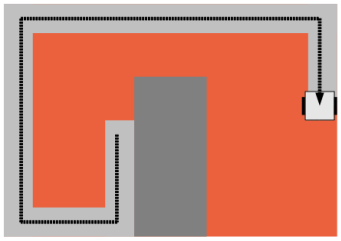
1: $a =$ área a limpar.



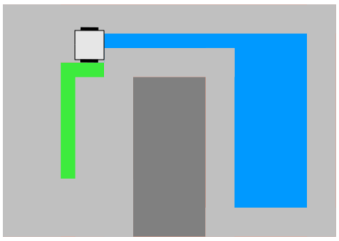
2: Atingir o ponto da borda exterior da área mais próximo do robô.



3: Apartir desse ponto, limpar a circundando-a até que esta termine ou seja partida em várias.



4: **if** a se partir em várias áreas $(a_0, a_1, ..., a_n)$ **then**
5: Chamar recursivamente o Algoritmo para cada uma das novas áreas.



6: **else**
7: Terminar
8: **end if**

4.4 Cellular Decomposition

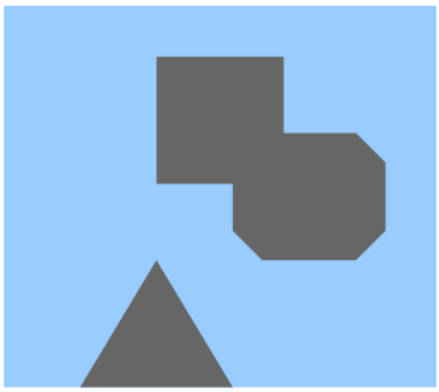


Figura 4.4: Área não elementar antes da decomposição.

No dia a dia, não são raras as vezes em que é necessário utilizar-se um algoritmo de cobertura de superfícies: quer seja em limpeza ou no corte de relva. O que o operador procura normalmente é encaixar um método de limpeza de Áreas Elementares (ver cap. 4.1) na superfície. Como a área muitas vezes não é elementar, o operador procura dividi-la em áreas elementares e depois utiliza a limpeza elementar. A esta divisão chama-se *Cellular Decomposition* [31].

4.4.1 Trapezoidal Cellular Decomposition

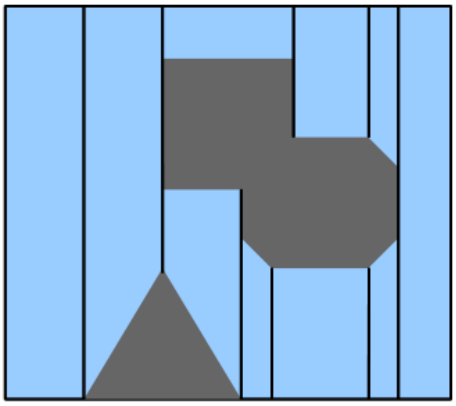


Figura 4.5: Área Dividida pelo método da decomposição celular trapezoidal.

Trapezoidal Cellular Decomposition (Decomposição em Células Trapezoidais) é uma técnica para obter Áreas elementares [31], que garante que todos as áreas resultantes são trapezoidais e como tal são convexas e Elementares (ver alg. 2).

Algorithm 2 Trapezoidal Cellular Decomposition

1: $a =$ área a limpar (ver fig. 4.4).
2: Marca-se um sentido.
3: Perpendicular ao sentido marcado, traça-se todos os segmentos de recta que iniciam num vértice dos obstáculos a terminar numa zona não marcada para limpeza (ver fig. 4.5).

A Decomposição Trapezoidal é um método simples e funcional. Porém dificuldades surgem normalmente quando o mapa não é geométrico ou quando estão presentes bordas curvas. Nesses casos a solução é aproximar aproximar os obstáculos por polinómios e só depois utilizar a decomposição.

4.4.2 Boustrophedon Cellular Decomposition

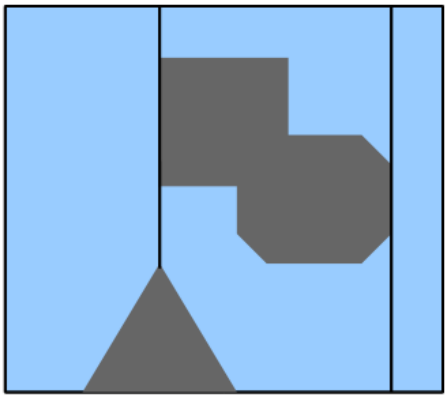


Figura 4.6: Área Dividida pelo método da decomposição celular de *Boustrophedon*.

Boustrophedon Cellular Decomposition (Divisão Celular de *Boustrophedon*) é também uma técnica de decomposição celular. A estratégia de *Boustrophedon* é tentar partir a área em poucas células. Quanto menos forem as células, maiores serão e consequentemente melhor será o rendimento da limpeza elementar [31]. O algoritmo para obter a divisão de *Boustrophedon* é descrito em 3.

Algorithm 3 Boustrophedon Cellular Decomposition

1: $a =$ área a limpar (ver fig. 4.4).
2: Marca-se um sentido.
3: Perpendicular ao sentido marcado, traça-se todos os segmentos de recta que passam tangente a um vértice dos obstáculos, sem atravessar o obstáculo (ver fig. 4.4.2).

O resultado da Decomposição Celular de *Boustrophedon* não são áreas puramente elementares (ver fig 4.4.2). As áreas resultantes podem sempre ser cobertas por uma trajectória em S paralela à divisão ou por uma trajectória Circundante Recursiva. Na figura 4.7 é mostrado um exemplo de uma trajectória obtida a partir deste tipo de divisão.

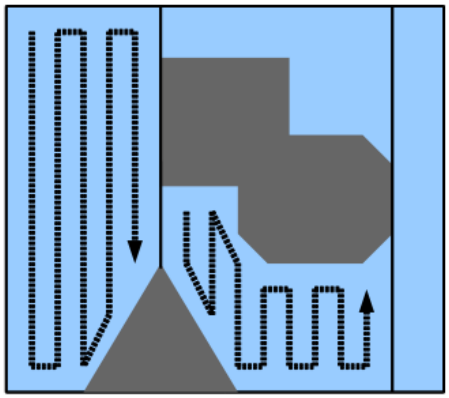


Figura 4.7: Trajectória em *S* na divisão celular de *Boustrophedon*.

4.5 Genetic Algorithms

No artigo [25] *Ann Nicholson Paulo A. Jimenez*, *Bijan Shirinzadeh* e *Gursel Alici* defendem um método que mistura algoritmos genéticos, áreas elementares, padrões de varrimento e decomposição celular. O método consiste partir uma área complexa em áreas elementares (utilizando decomposição celular). De seguida utilizar algoritmos genéticos para anexar um padrão de varrimento a cada área elementar, bem como encontrar a melhor ordem para fazer o varrimento. No algoritmo 4 é explicado detalhadamente a aplicação de *Genetic Algoritms* em *Area-Covering*.

4.6 Outras estratégias de *Area-Covering*

Em [32] e [33] *Simon Yang't* sugere a utilização de redes neuronais com desvio de obstáculos imprevistos. Neste, a dinâmica de cada neuronio é caracterizada por uma *shunting equation* que partiu do modelo biológico de uma membrana. Os neurónios são distribuídos pela superfície, sendo as suas ligações apenas as locais e consequentemente a complexidade depende linearmente do número de neurónios. Em [34] *Michel Taix* propõe uma estratégia para *Area-Covering* em máquinas agrícola. Nesta estratégia é entrada em consideração com a inclinação do terreno, evitando subir e descer muitas vezes o terreno. No artigo [35] o auto propõe a utilização de *Fuzzy* para resolver o problema de *Area-Covering*.

Em [36] os autores propõem utilizar a decomposição de *Boustrophedon* e posteriormente atribuir a cada uma das áreas, um dos robôs - individualmente. Desta forma obtém-se cooperação na limpeza pela distribuição de tarefas.

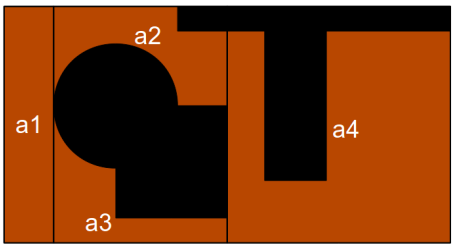
4.7 Conclusão

Neste capítulo foram apresentados técnicas utilizadas para conseguir a cobertura de superfícies. *Area-Covering* é um tipo de planeamento de trajectórias no qual o caminho óptimo não é

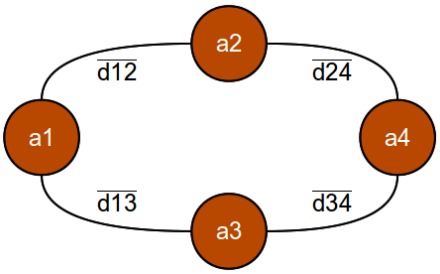
Algorithm 4 Aplicação de Algoritmos Genéticos em Area-Covering

1: $a =$ área a limpar.

2: Utilizar uma decomposição celular (neste caso *Boustrophedon*) e obter $A = [a_0, a_1, ..., a_n]$.



3: Criar um grafo com as áreas de A em que todas as áreas que fazem fronteira são ligadas com peso = distância entre o centro destas.

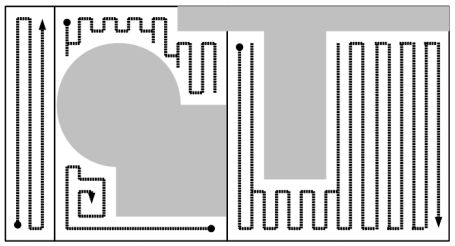


4: O Genoma do Algoritmo Genético contem um método para o varrimento da área completa. Começando pela ordem das Áreas Elementares a Limpar, seguido pelo tipo de Varrimento Padrão a aplicar em cada área.

ordem				Padrões de Varrimento			
a1	a2	a3	a4	a1	a2	a3	a4
1º	*2º*	*4º*	*3º*	*S vertical ↑*	*S vertical ↑*	*Circundar ←*	*S vertical ↓*

O valor dado a cada execução do algoritmo genético é dado pelo inverso do tempo de limpeza total.

5: Correr o Algoritmo Genético até haver pouca variação nas pontuações dos melhores espécimes de gerações consecutivas.



evidente e por isso mesmo, as técnicas utilizadas procuram obter uma aproximação desse caminho óptimo, com utilização de técnicas locais ou aproximações a partir de múltiplas tentativas. Este tema tem interesse em aplicações práticas como limpeza de superfícies, corte de relva, na resolução do problema do caixeiro viajante - no qual é preciso igualmente passar por todos os pontos.

Capítulo 5

Implementação

Neste capítulo são apresentados pormenores da implementação utilizada, bem como os algoritmos desenvolvidos. De forma resumida, é apresentada a estrutura do processamento e do código desenvolvido. Seguidamente é apresentado o método utilizado para a partir de um projecto em CAD de um piso, obter o mapa no simulador. Nesse mapa é apresentado método de navegação e a simulação da sujidade/limpeza do mundo. Ainda é apresentada a estratégia de limpeza de um compartimento, desde a segmentação de áreas até a utilização dos algoritmos genéticos na limpeza do mesmo. O capítulo termina com o método de atribuição de tarefas de limpeza a cada robô.

5.1 Escolha da linguagem

A linguagem escolhida como base para este projecto foi o *Python*. A escolha da linguagem para o desenvolvimento de um projecto é um ponto com importância: A velocidade de desenvolvimento e a de execução estão normalmente em lados opostos da balança.

No projecto desenvolvido, apesar do processamento ser pesado (principalmente devido à grande utilização do algoritmo de *Dijkstra* e algoritmos genéticos), o tempo de execução não se revelou um factor primário. Isto devido a dois factores: Primeiro, *path planning* ocorre poucas vezes por minuto; Segundo, a evolução do algoritmos genéticos corre à parte da execução da limpeza, ou seja, não corre em tempo real.

Por outro lado, o tempo de desenvolvimento neste projecto é essencial. A tese desenvolvida é grande e com muitos algoritmos complexos de implementar, gastando tempo para correcção de bugs que não pode ser desprezado [37].

Como já foi referido acima, a linguagem escolhida foi o *Python*. Entre os factores que motivaram a escolha do *Python* está:

- Como se trata de uma linguagem interpretada é possível executar código a partir da consola em tempo real, o que acelera o desenvolvimento e o *debug*.

- Corre em qualquer máquina com Windows, Linux/Unix, OS/2, Mac, Amiga, AIX, AROS, BeOs, iPod, MorphOS, OS/2, OS/390, z/OS, Palm OS, PlayStation, PSP, RISC OS, Series 60, Solaris, VMS, Pocket PC, entre outros.
- É *Open-Source*.
- Suporta a biblioteca *PyGame* que facilita a criação de ambientes de jogo.
- O *Python* é uma linguagem de muito alto nível: extremamente simples de aprender e de rápido desenvolvimento.
- Existe imensa documentação disponível *online* (www.python.org/doc/, www.pygame.org/docs/).
- Existe muito código *Open-Source* o qual é possível e legal consultar. Em sites como www.google.com/codesearch ou www.pygame.org é possível pesquisar código e consultar programas e jogos *Open-Source*.

De forma a garantir que o tempo de execução não se torna um problema, é possível ainda utilizar duas tecnologias:

- Utilizar a biblioteca *Psyco* que permite pré-compilar todas as função que ocupem tempos não desprezáveis. Com em média reduções de tempos na ordem dos 20% [37].
- Utilizar código em *c++*. Como *Python* é desenvolvido por base em c++, todas os objectos base do *Python* existem em *c++*. Assim é simples desenvolver bibliotecas em *c++* e utiliza-las em *Python*, aliando-se as vantagens de ambas linguagens.

5.2 Arquitectura da Cooperação

A arquitectura escolhida para a Cooperação entre Robôs foi GOFER. Conforme foi referido em 3.2.4 a arquitectura GOFER é centralizada num CTPS (*Central Task Planning System*). Todos os robôs comunicam tudo ao CTPS, este como tem conhecimento global do mundo a cada momento, toma as opções que acha mais correctas - Decisões estas que consistem em ordens para cada robô.

Na arquitectura utilizada, o CTPS comunica a cada robô que tarefa ele deve realizar e como o fazer. Por outro lado, o robô comunica ao CTPS todas as avarias/anomalias, estado das tarefas e estado do robô. Dados estes utilizados pelo CTPS para tomar as novas decisões.

Decisões como desvio de obstáculos, robôs e pessoas são tomadas ao nível do robô, devendo ser comunicadas ao CTPS para que este consiga encontrar padrões nas anomalias no mundo e posteriormente utiliza-las; Por exemplo muita gente na zona da máquina de café durante a hora seguinte ao almoço obrigam a que o robô se desvie, mas também permite que o CTPS decida que deve limpar aquela área em horas que não as seguintes ao almoço.

As vantagens da tecnologia utilizada são principalmente:

- Registo de ocorrências.

- Facilidade em alterar o número dos robôs.
- Consulta do estado do mundo por parte de um robô.
- Afastar o processamento do planeamento dos robôs, permitindo que estes sejam mais simples; Centralizar o processamento.
- O facto de não haver comunicação directa entre robôs, liberta a rede de tráfego excessivo. Cada robô só é informado com dados que lhe são relevantes.
- Permite criar facilmente um paralelismo entre a simulação e uma situação prática - Na situação prática o CTPS, em simulação a memória partilhada e um alocador de tarefas.

5.3 Estrutura dos Dados

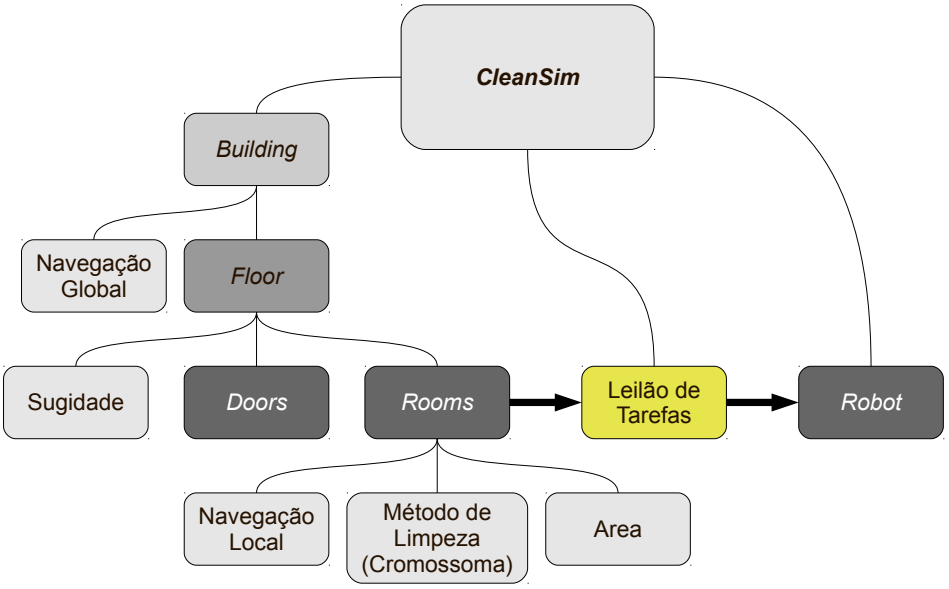


Figura 5.1: Estrutura dos Dados do *cleanSim* (Simulador desenvolvido).

Com vista à possível extensão do código e à adaptação a um sistema de robôs do simulador desenvolvido, a estrutura do mesmo foi cautelosamente planeada. Na figura 5.1 é possível observar dois grupos principais: O *Robot* e o *Building* (Edifício). O ***Robot*** é um modelo de um robô diferencial normal, com a capacidade de receber ordens simples (como "vai para um ponto"ou "segue estes pontos"), com um recipiente de bateria e um recipiente de lixo.

Já o ***Building*** é constituído por um grafo de *Doors* (portas) e *Rooms* (compartimentos), utilizado na Navegação Global. Este grafo permite calcular o caminho mínimo entre dois dos seus elementos. Para isso foi utilizado *Dijkstra* (como se verá mais à frente). Para além da Navegação Global, o *Building* contem ainda vários *Floors* (pisos). Assim é possível posteriormente adicionar

funções ao programa que necessitem da utilização de elevadores ou escadas. É possível também ter vários *Buildings* sendo cada um deles, por exemplo, um Departamento; e cada um destes com vários pisos.

Cada um dos ***Floor*** referidos acima é ainda subdividido em *Doors* , *Rooms* e uma camada de sujidade - *Dirt*. O *Floor* permite a partir de uma imagem retirada de um projecto (e posteriormente tratada) obter todas as portas e compartimentos, bem como as suas posições. O mesmo com a ***Sujidade***, que partindo de uma *layer* separada, obtém em função do tempo a sujidade em cada ponto do compartimento.

Cada ***Room*** é constituído por uma área a limpar e um Cromossoma que indica a melhor forma de o fazer. Sempre que é lançado um Algoritmo Genético sobre o método de limpeza do compartimento, o Cromossoma anteriormente calculado é reutilizado.

Por fim, no **Leilão de Tarefas** cada *Room* com necessidade de limpeza, licita nos *Robot* disponíveis. O que licita com valor mais alto, ganha o leilão e é limpo pelo *Robot* licitado. O valor licitado por cada *Room* é função da sujidade distância aos *Robot* e da sua urgência de limpeza.

Com vista a facilitar a implementação e a interacção entre os diferentes módulos, foram criadas algumas regras básicas a seguir durante o desenvolvimento de todo o código. Por exemplo, $room.pos$, $robot.pos$, $door.pos$ são as posições de cada um dos blocos, sendo esta dada por uma classe chamada ***Position***. A classe *Position*, para além de (x, y, θ) contém ainda alguns métodos essenciais: medir a distância entre duas posições, medir ângulos entre duas posições, somar ou subtrair posições, somar um versor a uma posição, entre outros. Um exemplo mais avançado será por exemplo $room.area.getPositions()$ no qual se obtém todos as posições de um compartimento (separadas por uma distância predefinida) e se pode passar directamente ao robô.

A estrutura base, permite distribuir problemas maiores em subproblemas simples e de fácil resolução. Assim, imagine-se que tínhamos o objectivo de limpar o compartimento *r23*. Primeiro atribuía-se um robô ao compartimento por *Auctions* (Leilões) dada a disponibilidade e a proximidade dos robôs. Depois, utilizava-se o a Navegação Global para encontrar os nós entre o compartimento em que o robô se encontra e o compartimento *r23* - [*room 13*, *door 5*, *door 7*, *door 3*, *room 23*]. Para cada par destes nós, utilizava-se novamente o algoritmo de *Dijkstra* para se encontrar o caminho mais próximo, através de um compartimento - [d5.pos, Position(5,32;7,49), ..., Position(3,12;2,92), d7.pos]. Uma vez atingido o *r23*, utiliza-se o genoma do melhor método de limpeza encontrado até o momento para gerar o caminho que lhe permite a cobertura de toda a área.

5.4 Estrutura do Processamento

A Estrutura do Processamento do *cleanSim* é fulcral para a compreensão do simulador. Esta foi estruturada em 3 Grupos:

- Simulador do Mundo
- Actualiza a Imagem

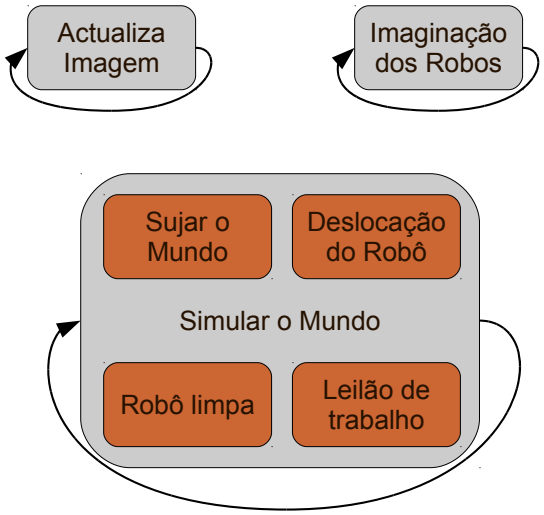


Figura 5.2: Estrutura do Processamento do *cleanSim*.

- Imaginação do Robô

O **Simulador do Mundo** é responsável por movimentar o robô, lançar os *triggers* das mudanças de ordens/tarefas, simulação do tempo, limpar o chão sob o robô e sujar o chão com o decorrer do tempo.

O bloco **Actualiza a Imagem** é o procedimento que agarra todos os *Sprites* e os pinta no ecrã. Este procedimento tem que estar livre do Simulador do Mundo, quando é necessário acelerar o Simulador sem que para isso se tenha que pintar mais imagens no ecrã - evitando assim *Frame Lock*.

Um dos motivos pelo qual o cérebro humano é tão grande (proporcionalmente ao tamanho do seu corpo) é a capacidade de simulação [38]. O ser humano antes de executar qualquer acção não instintiva simula-a; Por exemplo, se se considerar a hipótese de conduzir de olhos fechados, depressa pensamos nas possíveis consequências disso. O mesmo acontece com tirar um gelado a uma criança ou uma cadeira a uma velhinha, não precisamos de viver estes momentos para conseguirmos utilizar os seus resultados nas nossas decisões. A essa capacidade de simular chamemos-lhe de **Imaginação**. Com a **Imaginação do Robô**, é criado um robô que não existe no mundo real (ou neste caso, não existe no Simulador do Mundo) e que de um modo acelerado, consegue testar as várias hipóteses que o robô poderia executar. Desta forma, para o robô limpar um compartimento não precisa de experimentar os caminhos e tentar encontrar um melhor, o robô faz-lo como o homem o faria - Imagina-se a fazê-lo. De forma concreta, é criado um robô simplificado que consegue executar as tarefas de forma mais rápida e sem alterar realmente o mundo de simulação.

5.5 Obter mundo

Neste projecto, foi tido em conta que o mundo é estático, que não existem objectos que se moveram, que não existem pessoas e que o projecto do edifício está perfeito. Numa situação real, tal não acontece.

Numa situação real, é necessário utilizar os *inputs* dos robôs para alterar o mapa de uma forma dinâmica. Se uma secretária foi mudada de lugar permanentemente, o sistema tem que se adaptar e recalcular os caminhos para limpar aquela zona. Tal forma de recalcular os caminhos é sugerido no último capítulo na secção 7.4.

A obtenção do mapa utilizada foi elaborada em 3 passos essenciais e não é actualizada com o decorrer do programa. Os 3 passos foram:

- Edição manual do projecto CAD - Eliminando e acrescentando informações necessárias.
- Criar objectos a partir da imagem - Processamento da imagem obtida do projecto CAD.
- Guardar Resultados para evitar perdas de tempo em reprocessamento.

5.5.1 Processamento do projecto CAD

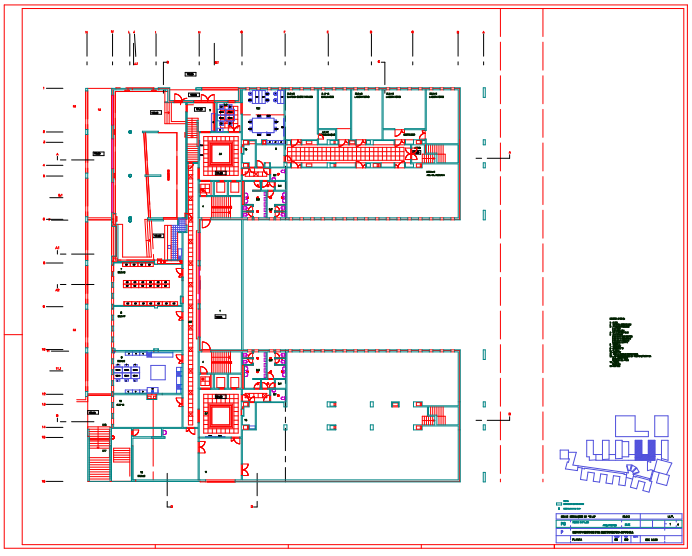


Figura 5.3: Ficheiro do mapa original.

Um projecto CAD de um edifício está repleto de *layers* com informações que não são importantes ao simulador - sejam elas declives nos tectos, pormenores de caixas eléctricas ou armários embebidos (ver figura 5.3). Um primeiro processamento do projecto CAD é remover todas estas *layers*. Posteriormente, deve-se converter o desenho vectorial para um *bitmap* e mudar as cores de forma a que: Todas as portas sejam azuis, todas as áreas possivelmente livres estejam a branco e

todo o resto esteja de cor diferente (mapa resultante está representado na figura 5.4). Estas cores são utilizadas pelo simulador para interpretar o mapa.

Figura 5.4: Imagem obtida a partir do CAD

Paralelamente a esta imagem é criada uma nova, na qual a *layer* de cor *alpha* (camada responsável pelas transparências em imagens) tem em cada pixel um valor que é directamente proporcional à quantidade de pó que aparece no ponto em função do tempo. Nesta camada é possível distinguir zonas pela sua sujidade.

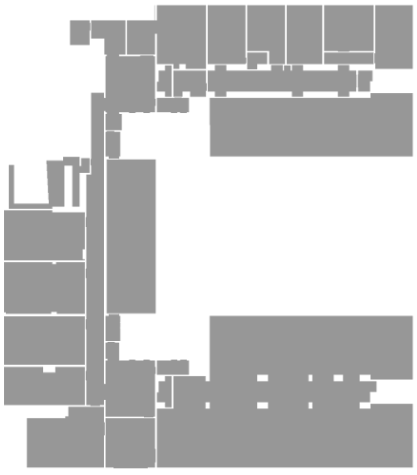


Figura 5.5: Imagem obtida a partir da sombra dos compartimentos a limpar

Quando se geram as imagens finais é necessário apontar a relação pixel/metro utilizada na importação da imagem.

5.5.2 Criar objectos a partir da imagem

Para obter os Objectos presentes em cada mapa - *Rooms*, *Doors* e *Paths* é preciso fazer varrimentos à imagem. Inicialmente são efectuados varrimentos à imagem à procura de *Rooms* e

Doors.

Para se encontrar os pontos pertencentes a cada objecto, foi utilizada uma técnica de etiquetagem com conectividade-8 (ver tabela 5.1). É efectuado um varrimento da imagem em que cada ponto (X) pertencente aos objectos é comparado com os pontos A,B,C e D (relativamente ao ponto a comparar). No caso de algum destes pontos estar etiquetado, a etiqueta é copiada para o ponto X; No caso de vários pontos estarem etiquetados, as duas etiquetas são marcadas como sinónimas; No caso de nenhum ponto estar etiquetado, é lançada uma nova etiqueta para X. Por fim, todas as etiquetas marcadas como sinónimas são substituídas por uma eleita como mestra [39].

B	C	D
A	X	

Tabela 5.1: Connectividade-8

O resultado deste algoritmo são vários grupos de pontos, marcados como *Doors* ou *Rooms*. É necessário ainda calcular informações como a posição destes objectos ou mesmo a malha de limpeza.

O grupo de pontos obtido acima é ainda utilizado o para encontrar todos os objectos que estão encostados (*Doors* que dão acesso a *Rooms*). Nestes objectos é criada uma ligação, que é utilizada para criar os *Paths* - Grafo de ligações e distâncias entre portas e compartimentos.

Na figura 5.6 está representado um piso com todas as portas e compartimentos encontrados. As portas que não ligam vários compartimentos são automaticamente excluídas.

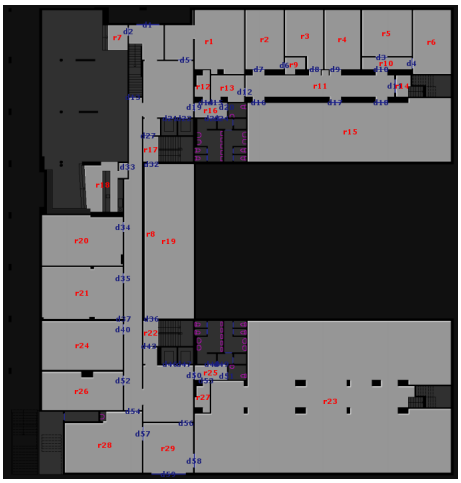


Figura 5.6: Piso com todos os Objectos identificados.

5.5.3 Guardar Resultados

Como o processamento utilizado nas funções anteriores é vasto, e como o tempo de abrir o programa é muito importante para quem o desenvolve, houve a necessidade de acelerar este

processo. Para isso foram criados *backups* em disco dos resultados obtidos. Assim, sempre que o programa liga, antes de gerar os objectos, procura por eles no disco.

Para tais backups foi utilizada a biblioteca *cPickle* disponível originalmente com a distribuição original do *Python*. Para acelerar ainda este processo foi utilizada a biblioteca *Psyco* que, por pré-compilar funções mais demoradas, conseguiu retirar aproximadamente 80% do tempo original.

Os backups realizados ficam guardados numa pasta com o nome da imagem original.

5.6 Navegação

Imagine-se que um viajante que procura deslocar-se ao Porto para provar o melhor Vinho do Porto, de uma marca conceituada. O viajante com certeza não procurará o caminho até a cadeira onde se sentará a provar um delicioso Vinho; ele procura o caminho para Portugal, procura o caminho para o Porto, procura o caminho para as Caves e, por fim, procura a cadeira onde se sentará. Esta simples e funcional estruturação, permite ao viajante resolver um complexo problema de deslocação na área da superfície Terrestre num tempo reduzido. A abordagem à navegação utilizada foi um *Bottom Up* da estratégia de navegação utilizada por nós no nosso dia a dia.

Conforme já foi referido em 5.3, por forma a simplificar todo o desenvolvimento, o programa foi extremamente bem estruturado, não sendo a Navegação uma excepção. A navegação é separada em 2 grandes camadas: **Local** e **Global**. A navegação Local é a forma como (sem abandonar um compartimento) se vai de um ponto a outro do compartimento. A navegação Global é como se vai de um compartimento até outro, ou até uma porta. Desta forma, o que inicialmente poderia ser visto como um problema de uma complexidade computacional grande, acaba por se tornar bastante simples e de rápida resolução, mesmo para uma linguagem interpretada como o *Python*.

Antes da explicação de cada um dos métodos de navegação, é importante deixar claro qual a necessidade de uma navegação leve e rápida. Se um robô precisa de ir de *A* para *B*, demora 2 min a fazê-lo e 0.1 segundos a decidir como o faz, o tempo de decisão não é problema. As facilidades que trabalhar num ambiente simulado acrescentam, são de alguma forma compensadas aqui. Num ambiente simulado como este, é necessário muitas vezes efectuar medidas de desempenho. Por exemplo, se na aplicação de algoritmos genéticos for criada uma população de 1000 robôs em 50 gerações que necessitam de uma medida de desempenho, só a deslocação para o ponto inicial (digamos "ir de *A* para *B*") ocuparia 1h23. É bom ter em conta que ir para a posição inicial é apenas um dos muitos caminhos que é necessário percorrer para avaliar o desempenho de um robô a limpar um compartimento. Se considerarmos os vários compartimentos envolvidos, depressa chegaria a um tempo que ultrapassaria a data de entrega da Dissertação. É necessário portanto ter em conta que os possíveis cortes de processamento efectuados não se trataram de um clichê mas sim de um bem essencial e necessário para o desenvolvimento deste projecto.

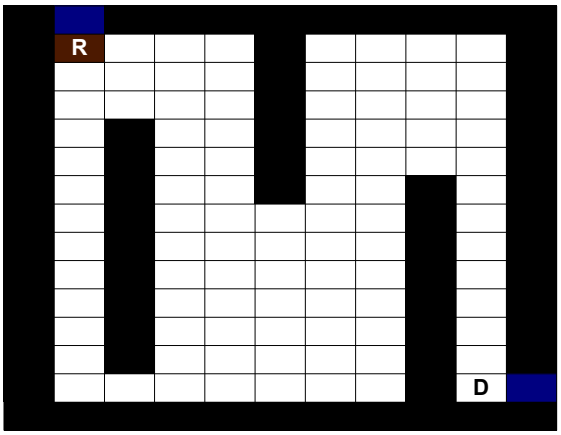


Figura 5.7: Mapa segmentado de um compartimento

5.6.1 Navegação Local - Malha

Conforme referido acima, a navegação local não permite atravessar portas, e como tal, não permite abandonar um *Room*. A abordagem para a resolução do problema da Navegação Local foi criar um grafo com todos os pontos atingíveis no *Room* e liga-los com a passagem entre eles. Posteriormente, aplicando o Algoritmo de *Dijkstra* (ver secção 2.2.1), encontra-se o caminho de menor peso que une dois pontos - O de origem e o de destino.

Na figura 5.7 pode ser visto um exemplo de um *Room* já segmentado. Neste compartimento é possível destingir-se as células: a preto - as paredes; a branco - os pontos atingíveis; a azul - as portas; com um "R"está marcado o Robô; com um "D"está marcado o destino.

3	2	3
2	R	2
3	2	3

Figura 5.8: Navegação Local - 1ª Iteração

Conforme explicado na secção 2.2.1, o algoritmo de *Dijkstra* inicia-se no ponto de origem e propaga-se iterativamente até atingir o ponto de destino ou esgotar as hipóteses. Um robô que percorre a diagonal de um quadrado de lado 2 percorre exactamente $2\sqrt{2}$. Porém, na implementação do algoritmo foi feita uma aproximação, considerando-se que o robô para percorrer a diagonal tem um custo de 3 e para percorrer a ortogonal de 2 (ver figura 5.8). Na figura 5.9 pode-se observar a aplicação da segunda iteração deste método. Importante lembrar que em cada posição fica guardado o caminho para a atingir.

A resolução do exemplo proposto na figura 5.7 pode ser vista na figura 5.10. Nesta figura a salmão está marcado o caminho escolhido pelo algoritmo. A cinzento estão marcados pontos

6	5	4	5	6
5	3	2	3	5
4	2	R	2	4
5	3	2	3	5
6	5	4	5	6

Figura 5.9: Navegação Local - 2ª Iteração

de outros caminhos igualmente óptimos. Como a nossa intuição mostra, a escolha do caminho realmente é óptima, considerando as limitações causadas pela aplicação da malha inicial.

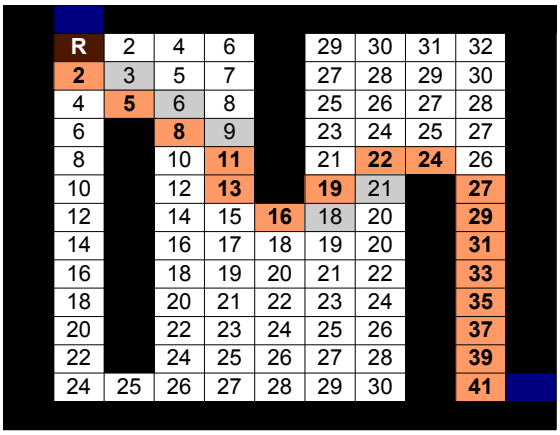


Figura 5.10: Trajectória de navegação da posição do Robô "R"a porta marcada a azul na direita.

Na aplicação deste método considerou-se a distância entre pontos de $0,4m$, o que num compartimento de $25m * 16m$ (o maior compartimento utilizado - Os Armazéns do Departamento de Electrotecnia da FEUP) dá um total de $63 * 40 = 2520$ pontos - O que é relativamente rápido a calcular. Se em vez de se utilizar este algoritmo ao nível do *Room* se utilizasse directamente num piso do Departamento ($40m * 48m$) o resultado rapidamente atingiria os 10 mil pontos; Se em vez disso considerássemos o maior edifício da FEUP (o Bloco B), teríamos em torno dos 20 mil pontos só no piso inferior.

Num compartimento de medidas mais usuais ($6m * 6m$) obtém-se uma malha com 225 pontos, o que comparando com os possíveis 10 mil pontos que se teria se o compartimento não fosse segmentado é um resultado bom.

5.6.2 Navegação Global

No final da secção da Navegação Local foram mostradas as vantagens de partir o problema do planeamento de trajectórias em vários elementos. No entanto, ficou por explicar como é feita a navegação entre compartimentos - dita Navegação Global. Nesta secção, será explicada tal navegação.

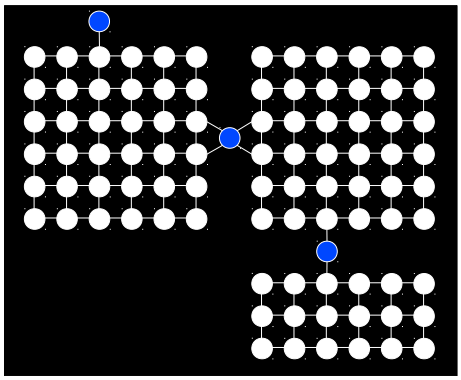


Figura 5.11: Malha de um edifício **sem** divisão de compartimentos.

A Navegação Global, é sem dúvida a chave pela qual foi possível uma navegação tão leve. Se se criar uma malha semelhante à da Navegação Local (secção 5.6.1) mas para todo um edifício, é notório que as portas separam grandes aglomerados de pontos (ver figura 5.11).

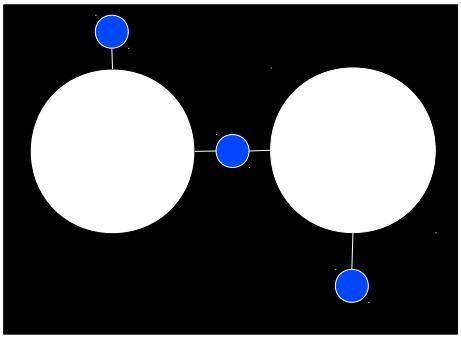


Figura 5.12: Malha de um edifício **com** divisão de compartimentos.

A abordagem da Navegação Global é agrupar o conjunto de pontos de um compartimento num só ponto (com a posição do centro de massa do compartimento) e adicionar os pontos resultantes, bem como os pontos das portas num grafo (ver figura 5.12). Neste grafo, o peso dos caminhos é dado pela distância entre pontos. Esta redução de complexidade consegue que o número de nós do grafo da navegação global seja apenas de $n_{\text{portas}} + n_{\text{compartimentos}}$.

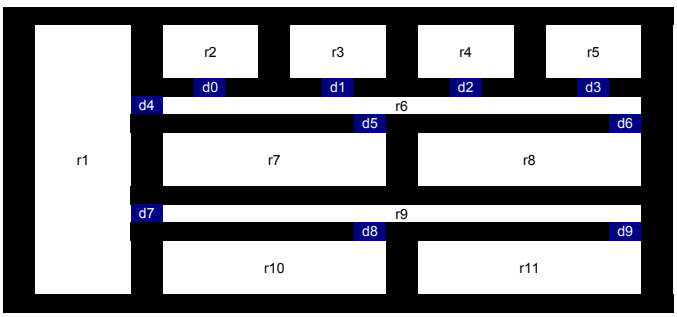


Figura 5.13: Exemplo de um edifício

Na figura 5.13 pode-se ver um exemplo de um edifício. A aplicação inicial do método de Navegação global neste edifício, consiste em encontrar o centro de cada compartimento e com uso das distâncias entre os pontos, gerar um grafo (ver figura 5.14). A partir do grafo gerado, utiliza-se o Algoritmo de *Dijkstra* (ver 2.2.1) para se encontrar o caminho mais curto entre dois pontos.

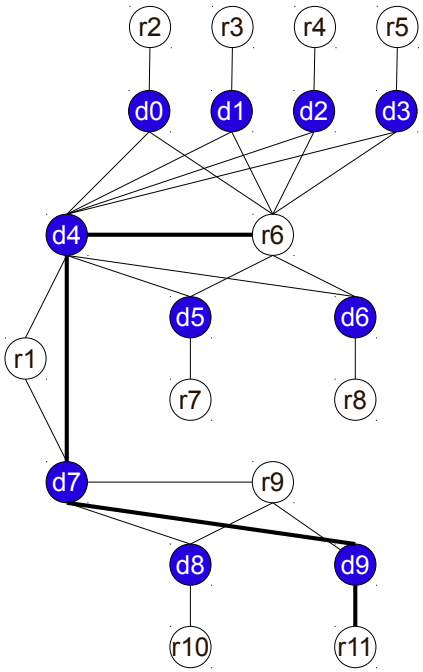


Figura 5.14: Exemplo de um grafo da navegação Global do edifício de exemplo.

No grafo da figura 5.14 está destacado o caminho entre $r6$ e $r11$. A presença dos compartimentos no grafo é principalmente para pontos de início ou término; os pontos intermédios são em norma portas.

5.7 Simulação da Limpeza

Para simular a sujidade de um piso, foram criadas duas imagens de sujidade sobrepostas ao mapa do piso (ver figura 5.15). A primeira, que corresponde à **Variação da Sujidade**, é uma imagem que em função do valor de cada pixel diz o quanto se suja o ponto correspondente (em função do tempo) - Sendo portanto cada ponto a derivada da sujidade em função do tempo do ponto de sujidade correspondente.

$$P_{x,y} = \frac{\mathrm{k} * \mathrm{dS_{x,y}}}{\mathrm{d}t} \tag{5.1}$$

A segunda imagem corresponde à **Sujidade** de cada pixel no instante actual.

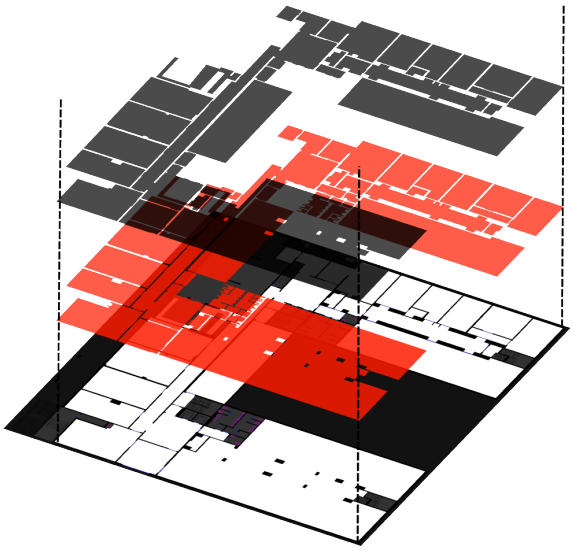


Figura 5.15: Camadas utilizadas para a simulação da sujidade no mundo.

5.7.1 Grelha de Sujidade

Como Grelha de sujidade foi considerada uma imagem na qual cada pixel corresponde a um bloco de $10cm * 10cm$. Esta dimensão da grelha é um compromisso entre a rapidez da simulação e a resolução da simulação obtida. Como a superfície de limpeza dos robôs mais pequenos anda em torno dos $30cm$ este resultado consegue ao mesmo tempo uma boa resolução e rapidez na simulação.

Considerou-se ainda que o edifício fica com a sujidade da imagem da **Variação da Sujidade** ao fim de 3 dias sem limpeza. Como o processo de actualizar completamente a sujidade da imagem é demoroso para ser chamado muito regularmente, optou-se por considerar que só é notória a diferença na sujidade de um piso de 8 e 8 horas de simulação - O que é um valor razoável.

$$k = \frac{8horas}{3dias * 24horas} \tag{5.2}$$

Assim sendo, sempre que se actualiza a sujidade (de 8 em 8 horas), a nova sujidade é dada por:

$$S_{x,y} := S_{x,y} + k * P_{x,y} \tag{5.3}$$

5.7.2 Robô

Com vista a mostrar o dinamismo do simulador, foram criados dois robôs distintos: *Roomba* e o *CleanRob*. Para simplificar a implementação a diferentes robôs foi criada uma classe chamada *CleanArea*. Cada robô tem uma ou mais *cleanAreas*. Esta área não é mais que um conjunto de pontos flutuantes, que quando chamados com uma posição central, efectuam uma limpeza numa área. Por outras palavras, é como um carimbo de limpeza que pode se marcado em qualquer

posição. Um robô de limpeza de ruas com 2 escovas e dois aspiradores, pode ser facilmente modelizado por 4 *cleanAreas*.

Na secção anterior (secção 5.7.1) foram apresentadas estratégias de limpeza de compartimentos, na qual foi apresentada a camada que representa a sujidade no instante actual $S_{x,y}$. Como esta imagem tem valor apenas na *layer Alpha* (camada de visibilidade) de uma imagem e tem apenas 16 bits, $S_{x,y} \in [0,255]$: Sendo que 0 corresponde a um ponto sem qualquer sujidade e 255 corresponde ao máximo da sujidade. Considerando a frequência das acções sobre o robô de 4Hz, uma área é limpa a 100% se numa exposição de $0.25s$ $(1/(4Hz))$ a área ficar completamente limpa.

5.7.2.1 cleanRob

O *cleanRob* é um robô de limpeza desenvolvido na FEUP (ver figura 5.16) [4] [5] [6] [7] [8] [9] [10]. Este robô é constituido por dois blocos distintos: O bloco traseiro, responsável pela locomoção diferencial. Este bloco contem toda a inteligência do robô. O bloco frontal (a Amarelo) é responsável pela limpeza do chão. A parte de limpeza é composta por um aspirador e uma escova ao canto.

Figura 5.16: Fotografia do cleanRob.

O *cleanRob* é constituído por duas *cleanAreas* (ver figura 5.17). A principal *cleanArea* está disposta no centro do bloco de limpeza e é responsável pela maioria da limpeza. A segunda *cleanArea* está disposta no canto e serve para limpar as zonas não acessíveis pela *cleanArea* principal, como zonas encostadas à parede.

Na figura 5.17 pode-se ainda ver as distâncias do centro das *cleanAreas* ao centro do eixo de rotação do robô.

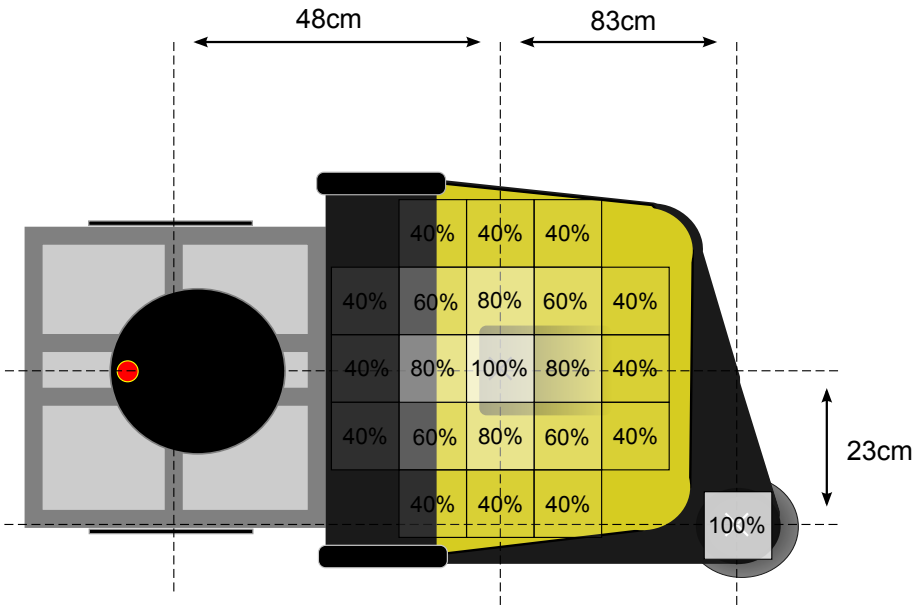


Figura 5.17: Representação do modelo do Robô de limpeza.

Como se pode verficiar na figura 5.18, o facto de rodar o robô, não roda a *cleanArea*. Garantindo assim a integração da zona de limpeza com o *bitmap* que representa a sujidade em cada compartimento.

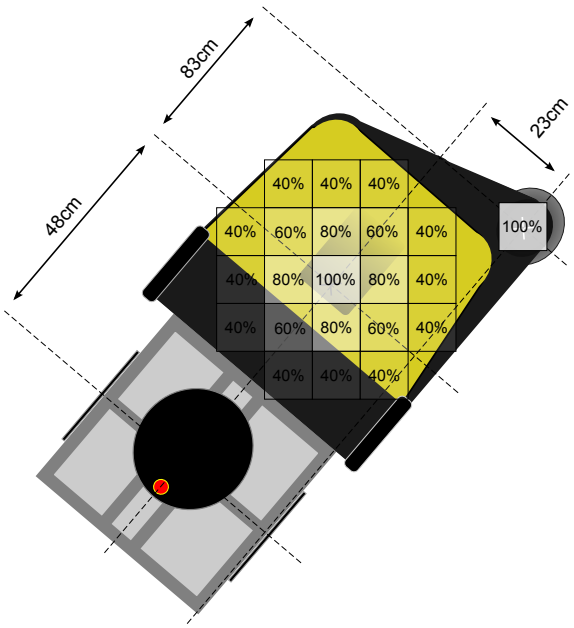


Figura 5.18: Representação do modelo do Robô de limpeza com uma rotação.

Assim, a partir da posição central do robô, da sua orientação e das distâncias relativas do centro das *cleanAreas* ao robô, é possível lançar a limpeza nos pixeis da grelha de sujidade.

5.7.2.2 Roomba

O *Roomba* é um robô de limpeza comercial (33 cm diâmetro), utilizado para a limpeza de casas e escritórios planos e preferencialmente, sem tapetes ou irregularidades. Este robô é composto por dois mecanismos de limpeza: Uma escova do lado direito e um aspirador central. O modelo da superfície que este robô limpa é composto por quatro áreas sendo a primeira a responsável pela escova e as restante responsáveis pela esquerda, centro e direita do aspirador.

Como equação de limpeza em cada ponto, considerou-se 100% no ponto central do aspirador, e 60% nos restantes pontos.

Figura 5.19: cleanAreas do Roomba

5.8 Estratégia da Limpeza

Tal como apresentado no capítulo 4 a limpeza de superfícies é um problema de *Area Covering*.

Conforme referido em [12], um dos problemas dos actuais robôs de limpeza é estes muitas vezes praticarem trajectórias imprevisíveis a visão do ser humano; A previsibilidade das actuais estratégias é baixa e isso dificulta a interacção com as pessoas. Por outro lado, a falta de multi-funcionalidade dos actuais robôs fazem com que esta interacção seja muitas vezes obrigatória. Com um seccionamento da área de limpeza em áreas rectangulares, para além de obter áreas elementares (ver secção 4.2) obtém-se também áreas bem limitadas nas quais pessoas conseguem prever a deslocação do robô.

5.8.1 Área a limpar

Como área a limpar entenda-se, um conjunto de pontos pertencentes a um compartimento que se o robô atingir todos os pontos enquanto limpa, o compartimento é considerado limpo. Tendo

em conta a dimensão da *cleanArea* do *cleanRob* (ver secção 5.7.2.1), foi considerada que esta área é composta por uma malha de pontos que distam 40 cm entre si.

5.8.2 Decomposição Celular

Conforme referido anteriormente na secção 4.4 a decomposição celular é um conjunto de técnicas que permitem a partir de uma área complexa obter um conjunto de Áreas Elementares. Áreas estas que são facilmente limpas por um Padrão de varrimento.

Para efectuar a decomposição celular, utilizou-se um método que foi denominado *Table Based Decomposition* (Decomposição Baseada em Tabelas). Este tipo de decomposição tem algumas vantagens comparativamente às restantes:

- Todas as áreas resultantes são rectangulares - Aumentando assim a previsibilidade dos robôs que as limpam.

- Por reescrever as áreas em tabelas de menor dimensão, facilita e acelera o processamento das mesmas.

O algoritmo que implementa a *Table Based Decomposition* é apresentado em no Algoritmo 5.

Algorithm 5 Table Based Decomposition

1: Encontrar linhas/colunas (l/c) de transição entre parede e passagem.
2: Utilizar as linhas encontradas para partir a área. Ficando a área inicial partida em $n_l + 1 * n_c + 1$ áreas. Agregar as áreas resultantes, passando cada uma a ser representada por um pixel apelidado de célula. As células são numeradas.
3: Por ordem crescente do número de célula, tentar agrupar em todos os sentidos cada célula até que não seja possível agrupar mais nenhuma.
4: Fazer a operação inversa à do ponto 2, obtendo de novo as áreas de cada célula, cada ponto com a numeração da célula correspondente.

Para melhor se perceber a *Table Based Decomposition*, de seguida mostra-se um exemplo da utilização do mesmo. Entenda-se que o objectivo do mesmo é a partir de uma área, decompor-la num reduzido número de subáreas rectangulares.

Exemplo da utilização de *Table Based Decomposition*: Conforme descrito no algoritmo 5, a TBD (*Table Based Decomposition*) inicia-se com a entrada de uma área em formato de tabela. Cada ponto corresponde por exemplo a uma área de 40cm*40cm (ver figura 5.20).

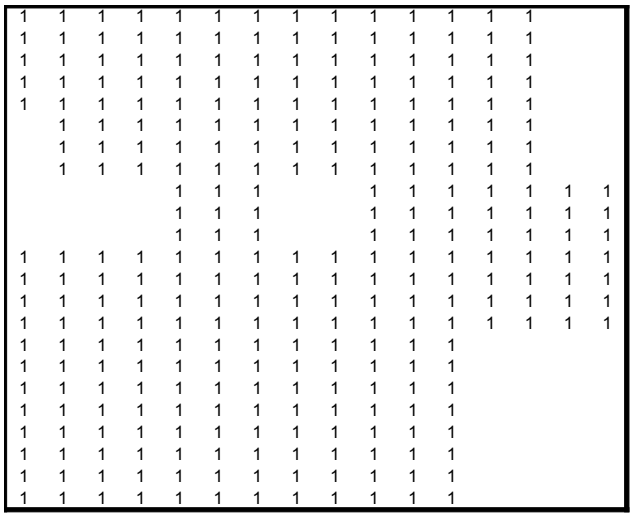


Figura 5.20: Área Inicial

De seguida, a área é segmentada (ver figura 5.21).

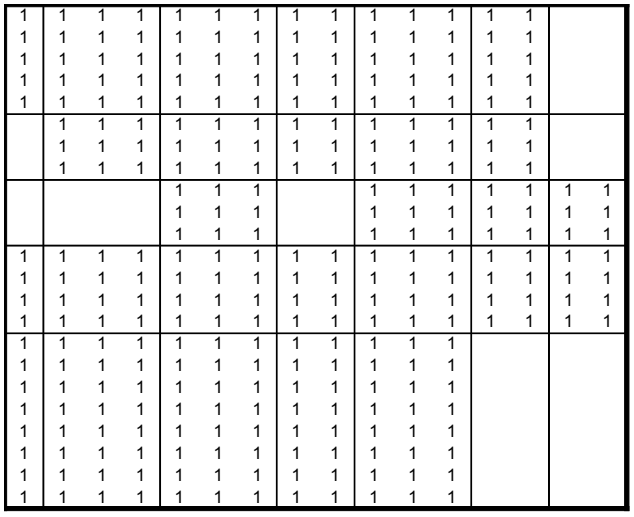


Figura 5.21: Área Segmentada

O terceiro passo é a uma numeração das células (ver figura 5.22).

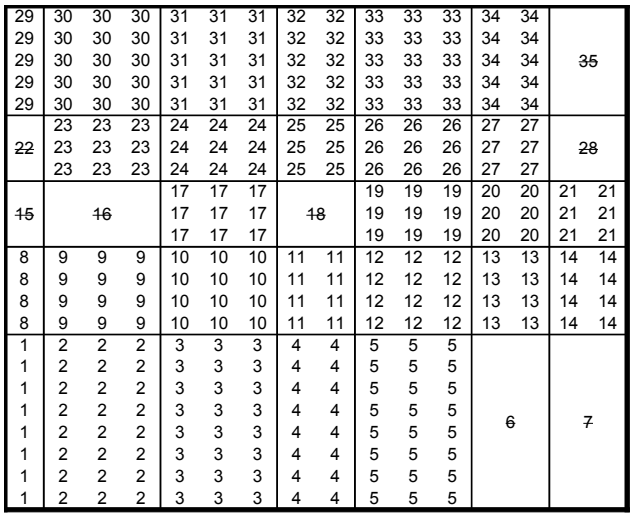


Figura 5.22: Renumeração das células

O quarto passo é o passo que dá o nome ao método - agrupar em tabelas. A vantagem deste passo é poder-se esquecer a quantidade de pixeis que cada área segmentada tem, tornando todas as áreas em simples células (ver figura 5.23).

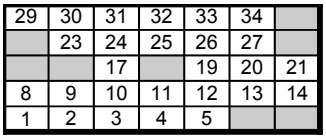


Figura 5.23: Agrupar em células da tabela

Na figura 5.24 é mostrado o processo de agregação de áreas. Começando dos números mais baixos para os mais altos, agregando para a direita e depois para cima.

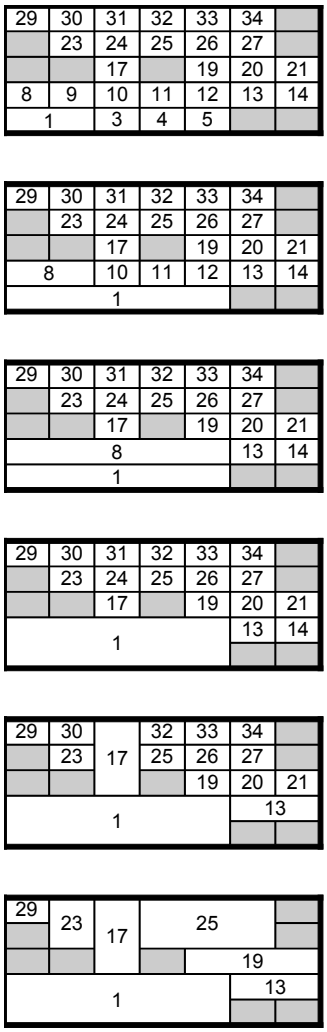


Figura 5.24: Agregação das células.

Após todas as células estarem agrupadas ao máximo, é efectuada uma renumeração das células, eliminando os números excluídos (ver 5.25).

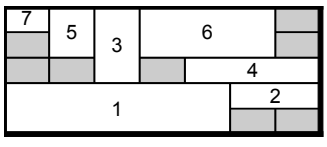


Figura 5.25: Renumeração das células

Por fim, as células são desagrupadas, obtendo então a área inicial segmentada por TBD (ver figura 5.26).

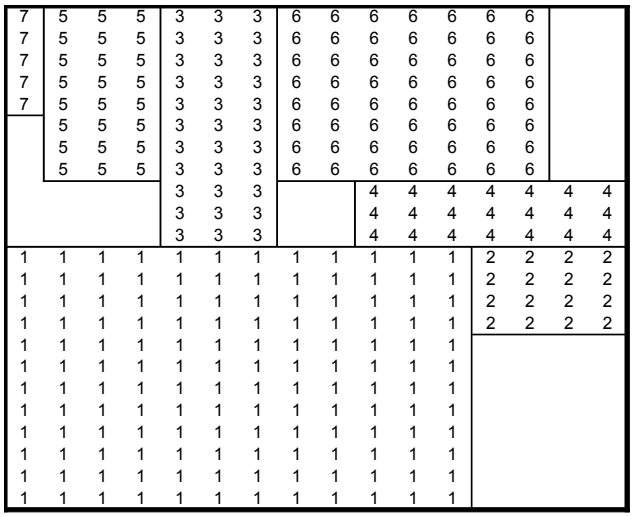


Figura 5.26: Área Final

5.8.3 Algoritmos Genéticos

Uma vez obtidas todas as áreas a limpar, resta encontrar a melhor ordem pela qual se limpam as áreas e o melhor caminho para limpar cada uma delas. Para nos localizarmos na necessidade computacional envolvida é conveniente calcular o número total de combinações possíveis.

Assim, num compartimento com n áreas temos:

- Para a ordem das áreas de limpeza um total de combinações dado por:

$$C_{ordem} = n! \tag{5.4}$$

- Para o percurso, o método pelo qual se limpa cada uma das áreas, o total de combinações é dado por:

$$C_{percurso} = (2*2*2*2)^n \tag{5.5}$$

Cada área pode ser limpa de 16 formas diferentes: O total de combinações pode ser obtido pela equação:

$$C = C_{ordem} * C_{percurso} = n! * (2*2*2*2)^n \tag{5.6}$$

		gene[0]	gene[1]	gene[2]	gene[3]
0000	0	circular	anti clockwise	top	right
0001	1	circular	anti clockwise	top	left
0010	2	circular	anti clockwise	bot	right
0011	3	circular	anti clockwise	bot	left
0100	4	circular	clockwise	top	right
0101	5	circular	clockwise	top	left
0110	6	circular	clockwise	bot	right
0111	7	circular	clockwise	bot	left
1000	8	S	y Axis	top	right
1001	9	S	y Axis	top	left
1010	10	S	y Axis	bot	right
1011	11	S	y Axis	bot	left
1100	12	S	x Axis	top	right
1101	13	S	x Axis	top	left
1110	14	S	x Axis	bot	right
1111	15	S	x Axis	bot	left

Tabela 5.2: Combinações de métodos de limpeza de cada área

Num compartimento com 22 áreas distintas (o mais complexo dentro do edifício considerado) temos $22! * (2 * 2 * 2 * 2)^{22}$ genes diferentes. Isto corresponde a um total de 10^{47} caminhos possíveis. Se considerarmos que cada simulação demora 1 segundo e que todos os casos têm que ser simulados - com o objectivo de encontrar o melhor cromossoma, ainda assim temos 10^{40} anos de simulação. Na tentativa de solucionar este problema e conforme referido na secção 4.5, neste documento é utilizada a convergência genética.

Os algoritmos genéticos permitem a partir de uma amostra de população fazê-la convergir no sentido de minimizar uma função de custo. Assim, com um número que não é de forma alguma semelhante às 10^{47} simulações, consegue-se uma estratégia de limpeza boa. Apesar de muitas vezes não ser efectivamente a melhor solução, da maioria das vezes aproxima-se.

5.8.3.1 Estrutura Genética

A base do algoritmo genético é estruturada em 4 camadas: genes, cromossoma, individuo e população.

Genes Os genes são a base dos algoritmos genéticos. Cada gene representa uma variável que quando modificada (mutação) pode alterar o desempenho do cromossoma. Cada gene é independente dos restantes, podendo portanto sofrer uma mutação individualmente.

Cromossoma O cromossoma é um conjunto de genes ordenados. Um cromossoma tem que permitir ser quebrado, de forma a que a combinação de dois cromossomas pais, originem um cromossoma filho.

Individuo Cada individuo contem um cromossoma. Contem também o reflexo da utilização do cromossoma no mundo, tendo portanto a medida do sucesso do seu cromossoma.

População Uma população, é um conjunto de indivíduos que coexistem no mesmo instante de tempo. Os indivíduos de uma população podem cruzar os seus cromossomas.

Na implementação dos algoritmos genéticos utilizada, talvez fosse de esperar que cada robô seria um Indivíduo. No entanto, cada Compartimento é um indivíduo; Desta forma, apesar da solução ter que ser procurada para cada compartimento, fica mais adaptada ao mesmo. Para cada compartimento é decidido uma forma de o limpar. Quando um robô se desloca para um compartimento, pergunta ao CTPS qual a melhor forma de o limpar - ao que o CTPS responde com o cromossoma do compartimento.

Conforme referido acima, cada cromossoma é constituído por genes. Este cromossoma é constituído por n conjuntos de 4 genes (*cleanMethod*) dados pela tabela 5.2 e um gene que indica a ordem pela qual se limpam as áreas (*sortGene*). O *cleanMethod* indica o método como se limpa cada uma das áreas que constituem o compartimento.

CleanMethod Método de limpeza de uma área. É composto por 4 genes booleanos nos quais uma mutação corresponde a uma inversão do valor.

SortGene É um gene com a ordem pela qual se limpam as áreas de um compartimento. Contem um *array* com os números inteiros no intervalo $[0, n-1]$ (n é o número de áreas que constituem o compartimento). Existem duas formas de mutar este Gene: sortear a ordem de um grupo de números do array e trocar dois elementos do array.

Cada mutação num tipo de gene tem uma probabilidade de ocorrência associada.

5.8.3.2 Evolução Genética

A Evolução Genética consiste em 4 pontos:

- **Crossover:** Cruzamento dos cromossomas de indivíduos da população anterior - resultando num individuo com um cromossoma semelhante ao dos pais. Este cruzamento pode ser feito entre quaisquer pais mas mantendo a regra: Um pai com sucesso tem uma probabilidade maior de passar o seu cromossoma. A esta concorrência entre pais chama-se **tournment** (Torneio).

- **Mutation:** Consiste em modificar aleatoriamente genes de um indivíduo novo. Após ser criado um indivíduo, os genes não mais podem ser modificados.

- **Evaluation:** É verificar o quão bem sucedido é um Indivíduo. Neste caso é verificado quanto tempo um indivíduo demora a limpar o compartimento em questão. O resultado é tão melhor quanto menor for o tempo.

- **Goal**: Os objectivos que levam ao fim do algoritmo são: ser atingido o tempo pretendido por algum dos indivíduos ou ter passado gerações suficiente.

A comparação entre o sucesso dos Indivíduos nos algoritmos genéticos e os seres vivos são de alguma forma interessantes. Um ser vivo como o ser humano, é sem dúvida um ser bastante adaptado e com um bom sucesso relativo no mundo. No entanto, será que algum homem tem **o melhor** cromossoma? Com certeza não o tem. Para referência, existem $10^{1800000000}$ combinações possíveis no cromossoma humano, imagine-se que se pretendia testar o sucesso de todas as combinações possíveis; Neste caso cada avaliação do sucesso de um indivíduo demoraria algo como 50 anos. Mesmo com uma população de 6 mil milhões de indivíduos, desde o início do universo ainda não houve tal tempo. No caso da evolução genética em seres vivos, tal como na evolução genética aplicada neste projecto, não há o tempo suficiente para correr todas as hipóteses e o melhor resultado não é óbvio.

A *evaluation* (avaliação) de um indivíduo foi feita com recurso a **Robôs Imaginados**. Este tipo de robôs são semelhantes aos robôs normais, no entanto têm simplificações como a não interferência real com o mundo. Desta forma, quando o código for adaptado a robôs reais, em vez de ser necessário um Indivíduo efectivamente limpar um compartimento para obter uma avaliação, pode simplesmente imaginar-se um robô a limpa-la. Este processamento separado é também referido na secção 5.4.

5.8.3.3 Exemplo da Utilização dos Algoritmos Genéticos

Neste exemplo pretende-se demonstrar o modelo genético, bem como aplica-lo num compartimento. Neste caso optou-se pelo compartimento *r1* (ver figura 5.27).

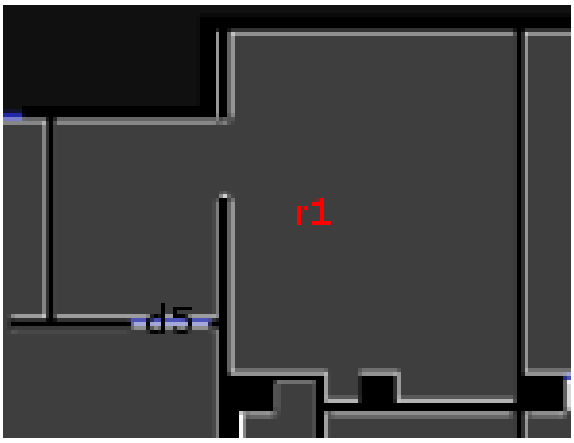


Figura 5.27: Pormenor do mapa, referente ao compartimento $r1$

O compartimento deve estar já decomposto em pontos e segmentado (a explicação da decomposição reside na secção 5.8.2). O resultado desta segmentação pode ser visto na figura 5.28.

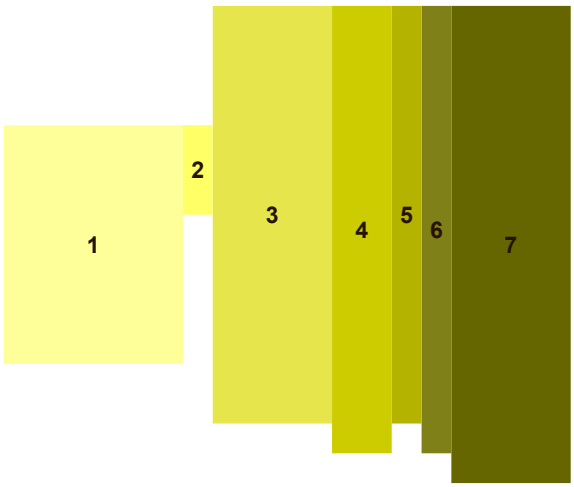


Figura 5.28: Áreas do Exemplo

Para representar o modo de limpar o *r1* é necessário criar uma carcaça para o cromossoma. Na figura 5.29 é representada a estrutura do cromossoma utilizado. Como se pode ver, é composto por $1+7*4$ (29) genes e contem $7!*(2*2*2*2)^7$ (1352914698240) combinações possíveis.

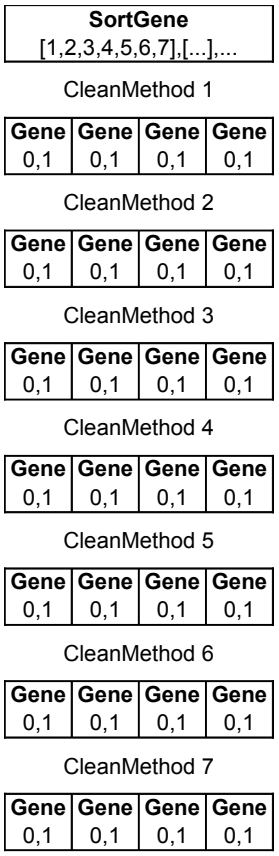


Figura 5.29: Estrutura do cromossoma para a limpeza do compartimento *r1*

Na figura 5.30 é representado o exemplo de um método de limpeza para cada uma das áreas de *r1*. Os genes correspondentes à área 1 tomaram os valores `1000`, ou seja: [Trajectória em S, translação no sentido do eixo dos *yy*, começou no cimo, começou encostado à direita] (consultar a tabela 5.2).

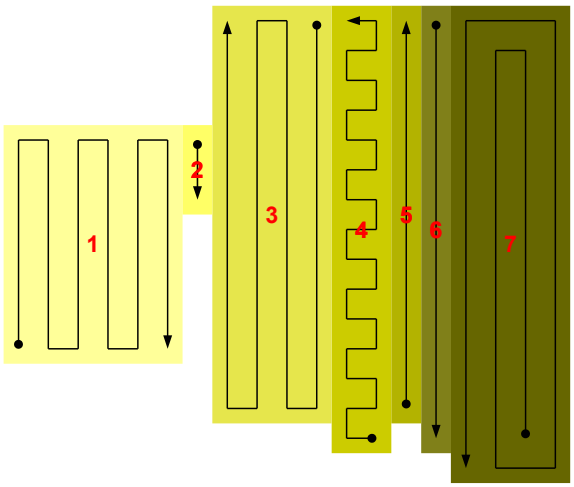


Figura 5.30: Método de limpeza das áreas de *r1*

Entre cada uma das áreas representadas é ainda necessário utilizar um *Path-Planning*, que neste caso nunca abandona a Navegação Local (ver secção 5.6.1).

Para limpar as áreas com os métodos indicados na figura 5.30 é necessário calcular um caminho que parte da posição final de método de limpeza de cada área para a posição inicial do método de limpeza da área seguinte. O exemplo de um caminho desses pode ser visto em 5.31 (neste caso da área 1 para a área 7).

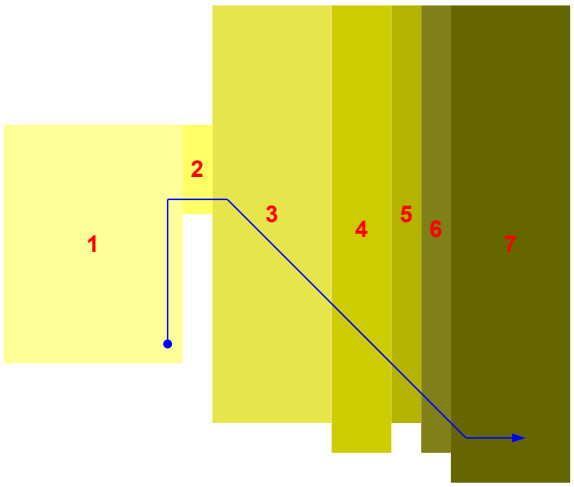


Figura 5.31: Caminho da área 1 para a área 7.

O exemplo do resultado da aplicação de um cromossoma, pode ser visto em 5.32. Neste caso o cromossoma é representado por:

`[1,6,4,5,3,7,2], 1011 1111 1000 1101 1010 0100 1111`

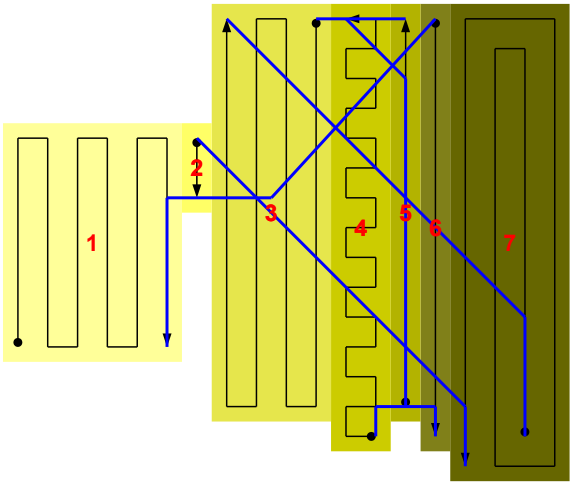


Figura 5.32: Gene não evoluido.

Como se pode verificar, este gene não consegue de todo um bom resultado. Os factor que mais contribuem para isso são: por um lado o facto da área 4 ser limpa com um *S* no eixo dos *xx* causa muitas inversões no sentido da marcha do robô; Por outro lado a ordem pela qual as áreas são limpas faz com que o robô passeie de de um lado para o outro.

Como optimizar este resultado? Correr o algoritmo genético. Temos portanto um indivíduo anterior que, apesar de ser obviamente mau, é o melhor até o momento. Como cada indivíduo tem que ter uma pontuação associada é necessário simular o cromossoma do individuo anterior. Neste caso, a pontuação obtida foi de 787 segundos.

Criar população: Numa iteração inicial, é necessário criar uma população aleatória. Neste caso, utiliza-se uma população de 3 elementos. Numa situação real utiliza-se uma população de mais de 100 elementos.

`[1, 6, 4, 5, 3, 7, 2], 10111111100111011111001101111` score = 787s
`[7, 6, 4, 5, 1, 3, 2], 10010011111111100000110001000` score = ?
`[5, 1, 7, 3, 4, 6, 2], 11111001110101100110011101100` score = ?

Avaliar População: Para avaliar a população é necessário simular cada indivíduo e apontar o tempo decorrido.

Geração 1:

1º `[7, 6, 4, 5, 1, 3, 2], 10010011111111100000110001000` score = 771s
2º `[1, 6, 4, 5, 3, 7, 2], 10111111100111011111001101111` score = 787s
3º `[5, 1, 7, 3, 4, 6, 2], 11111001110101100110011101100` score = 827s

Na primeira população do algoritmo genético, não foi possível remover grande tempo à solução inicial.

Crossover e Mutação Para gerar uma nova geração, é necessário utilizar os métodos de *crossover* (cruzamento de indivíduos) e de mutação. Para cada indivíduo existem 3 hipóteses de

crossover: Cruzar informação genética com o próprio indivíduo (não cruzar), cruzar com outro indivíduo ou cruzar com um indivíduo gerado aleatoriamente. No cruzamento, cada gene tem uma probabilidade de 50% de copiar cada um dos pais. Neste caso vamos fazer o cruzamento dos indivíduos do 1º e do 2º lugares:

```
pai.......[7, 6, 4, 5, 1, 3, 2], 10010011111111100000110001000
mae.......[1, 6, 4, 5, 3, 7, 2], 10111111100111011111001101111
MASC.M....[........pai........], pppmpppmmpmmmppmmmpmpppppmppp
filho0....[7, 6, 4, 5, 1, 3, 2], 10010011100111011011110001000
```

Seguidamente é necessário mutar genes aleatórios do filho:

```
filho0....[7, 6, 4, 5, 1, 3, 2], 10010011100111011011110001000
MASC.C....[...S.....S.........], ..M.......MM.............M..
filho.....[7, 5, 4, 6, 1, 3, 2], 10110011101011011011110001100
```

Com "M"estão marcados os genes que simplesmente invertem o seu valor na mutação. Com "S"estão marcados dois elementos do *sortGene* que foram trocados. Todos estas posições foram seleccionadas aleatoriamente. Da mesma forma é criado ainda um outro filho com pais aleatórios que é adicionado à população. Em conjunto é também adicionado o melhor gene da população anterior - sem qualquer mutação ou cruzamento.

Procedimento cíclico Após avaliados os novos filhos, a nova população fica dada por:

Geração 2:

```
[1, 6, 5, 3, 2, 4, 7], 10010011100111011011110001000 score = 675s
[7, 6, 4, 5, 1, 3, 2], 10010011111111100000110001000 score = 771s
[7, 5, 4, 6, 1, 3, 2], 10110011101011011011110001100 score = 772s
```

O procedimento anterior é repetido até atingir as gerações pretendidas. Neste caso a 3ª geração é dada por:

Geração 3:

```
[3, 6, 5, 1, 7, 4, 2], 10010011100111011011110001000 score = 622s
[1, 6, 5, 3, 2, 4, 7], 10010011100111011011110001000 score = 675s
[6, 4, 2, 3, 7, 1, 5], 10010011101111100000110001000 score = 708s
```

Em apenas 3 gerações com populações de 3 indivíduos foi possível reduzir o tempo de 826s para 621s. No entanto o total de 7 $(3+2+2)$ indivíduos é muito pouco e com uma população reduzida é fácil o resultado não convergir directamente. É sempre melhor manter diversidade genética durante muitas gerações. Com populações reduzidas é normal que todos os indivíduos se tornem semelhantes e que não beneficiem de diversidade genética.

Após correr a optimização genética em 30 gerações com uma população de 100 indivíduos foi obtido:

```
[1, 2, 3, 4, 5, 6, 7], 00110011100110011001101101 01 score = 463s
```

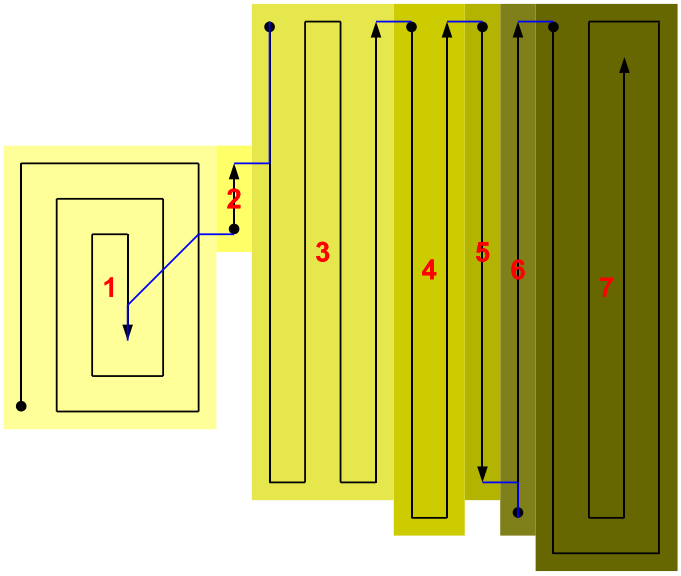


Figura 5.33: Melhor gene obtido para o compartimento *r1*.

Como se pode ver pela figura 5.33, o resultado apesar de provavelmente não ser o óptimo, tem muito poucas sobreposições de caminhos e relativamente poucas curvas.

5.8.3.4 Optimização da Convergência:

A maior dificuldade da implementação genética é fazer o *sortGene* convergir para o resultado óptimo. Na verdade, uma simples alteração no *sortGene*, tem implicações drásticas no resultado de muitos dos restantes Genes.

Para acelerar a convergência do algoritmo genético, optou-se por procurar uma solução inicial boa para o *sortGene* e depois diminuir a probabilidade das mutações do mesmo. Para se encontrar esta solução inicial utilizou-se uma solução *greedy* (ver secção 3.5.1): Seguidamente à segmentação das áreas, escolhe-se uma área inicial (próxima de uma porta); Dessa 1ª área, procura-se a área mais próxima, marcando-a como utilizada; Este procedimento é repetido até esgotar todas as áreas. A ordem de áreas gerada é o valor inicial do *sortGene*.

5.9 Alocação das Tarefas

Tendo em conta o referido nas secções anteriores deste mesmo capítulo, nesta altura os robôs são já capazes de navegar entre compartimentos e limpar cada compartimento com uma boa estratégia. Falta portanto a cooperação. Com base na arquitectura utilizada, a cooperação estrutura-se em duas camadas: *Colision avoidance* (evitar colisões) na qual os robôs trabalham em conjunto de forma a não colidirem e distribuição de necessidades de limpeza pelos robôs. A primeira o robô resolverá localmente, não havendo planeamento para isso. A distribuição de tarefas será feita pelo CTPS com vista a responder às necessidades de limpeza.

Para a alocação de tarefas foram utilizados dois métodos. Um primeiro mais complexo e estático **Partir um Grafo** e um segundo com mais simples e dinâmico **Leilões**.

5.9.1 Partir um Grafo

Este método consiste em a partir do grafo de Navegação Global passar-lhe um conjunto de portas que permitam partir o grafo em *n* grafos. Cada um dos sub-grafos gerados é atribuído a um robô, que se dirige para ele para o limpar. Dada a partição do grafo ter que ser passada por uma pessoa, este método pode ser utilizado para fazer limpezas periódicas de um piso, preferencialmente limpezas completas ou seja, limpezas nas quais todo o piso necessita de ser limpo.

Dado este método ser estático e necessitar de uma segmentação feita por uma pessoa, decidiu-se procurar um outro método.

5.9.2 Auction

A atribuição de tarefas baseada em Leilões (*Auctions*) consiste em leiloar um recurso, atribuindo-o a uma necessidade (ver secção 3.5.2). Neste caso os recursos são os robôs, as necessidades são os compartimentos. A necessidade de limpeza de um compartimento nasce da simulação do compartimento ou da introdução directa por um utilizador - "O compartimento xpto precisa de ser limpo". Quando um compartimento precisa de ser limpo requisita a sua limpeza ao CTPS. O CTPS leiloa cada robô disponível aos compartimentos que necessitam de limpeza. O compartimento "que der mais" fica com um robô atribuído que o limpa de seguida.

A equação que decide o valor licitado por cada compartimento é função de 3 valores:

- d_R - Distância do robô licitado ao compartimento.
- d_{Rs} - Distância dos restantes robôs ao compartimento.
- t_{clean} - Tempo que o compartimento demora a ser totalmente limpo.

$$Auction_{Room} \equiv -d_R + \frac{d_{Rs}}{n_{robots} - 1} + t_{clean}/k \tag{5.7}$$

Assim esta equação contempla: Por um lado procura o robô mais próximo; Por outro lado, se tiver muitos robôs disponíveis, não licita muito alto; A terceira parcela é para o caso de uma limpeza global, na qual é dada alguma prioridade aos compartimentos maiores, facilitando assim a que a tarefa seja terminada com brevidade.

O k na equação 5.7 serve como termo de relação entre a distância e o tempo, permitindo assim valorizar mais ou menos o tempo *versus* as distâncias a percorrer.

Exemplo da Utilização de Auctions: Na figura 5.34 pode-se ver um piso com alguns compartimentos. Neste exemplo, é mostrada a forma como é calculada a licitação de cada compartimento com necessidade de limpeza (nesta caso todos os compartimentos) para obter o recurso "Robô

Amarelo". A verde estão representados os restantes robôs. Na figuras seguintes é dado especial destaque ao *Room x*, compartimento no qual são demonstrados os valores calculados.

Ainda na figura 5.34 é representada a distância do "Amarelo" a cada compartimento. Esta distância é medida com auxílio do Grafo de Navegação Global (ver secção 5.6.2).

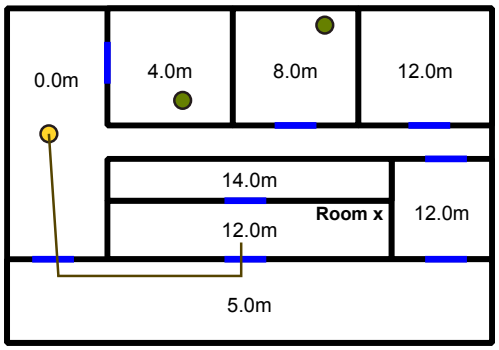


Figura 5.34: Calculo das distâncias do robô amarelo a cada um dos compartimentos.

Na figura 5.35 é representada a soma das distâncias dos "Verdes" a cada compartimento. A verde está ainda traçado os caminhos dos robôs ao *Room x*.

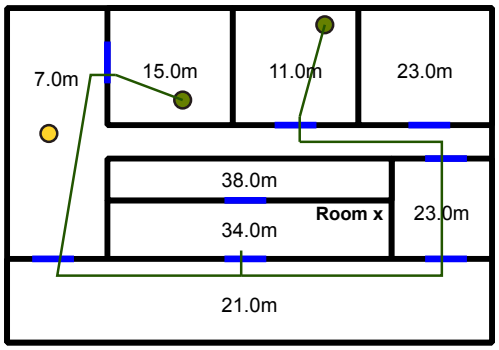


Figura 5.35: Calculo da soma das distâncias dos robôs verdes a cada um dos compartimentos.

Na figura 5.36 é representado o tempo de limpeza de cada compartimento. Este tempo é dado pelo *score* do Genoma de cada compartimento, sendo dependente do caminho que foi decidido utilizar na limpeza do compartimento (ver secção 5.8.3).

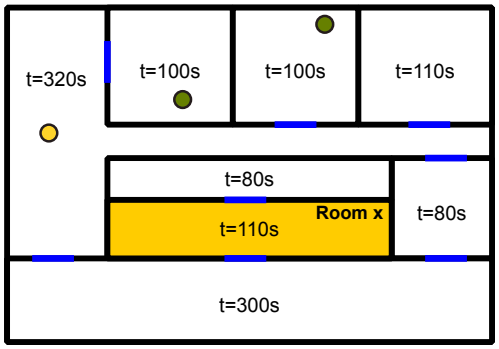


Figura 5.36: Tempo de limpeza de cada compartimento.

Por fim, na figura 5.37 é mostrado o valor licitado ao "Robô Amarelo" por cada compartimento. Neste exemplo foi utilizado $k = 30$.

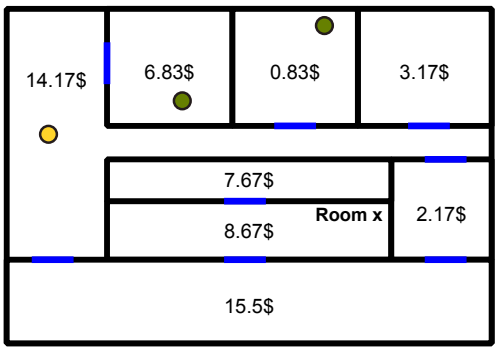


Figura 5.37: Valor licitado por cada um dos compartimentos no robô Amarelo

Pontos a reter do exemplo anterior:

1. O compartimento de baixo ganhou a licitação porque está próximo do robô amarelo, está longe dos restantes robôs e tem um tempo de limpeza elevado.

2. O robô procura limpar os compartimentos que estão longe dos outros robôs. Assim quando os outros robôs forem leiloados, vão ter compartimentos próximos para limpar.

3. O compartimento onde está o "Amarelo" não ganhou. Isto deve-se ao compartimento estar perto de todos os robôs, todos os robôs o podem limpar com pequenas deslocações.

Capítulo 6

Resultados

Neste capítulo são descritos resultados que procuram demonstrar o funcionamento do simulador desenvolvido. As demonstrações dos resultados começam pela Cobertura de Superfícies com Convergência da Evolução Genética e são seguidos por um comparativo entre o caminho obtido pelo algoritmo genético e outros caminhos para a limpeza de compartimentos. Nos resultados relativos à Navegação é comparado o caminho obtido pela navegação implementada com o caminho óptimo. De seguida é demonstrado que o simulador consegue utilizar robôs com dinâmicas e áreas de limpeza diferentes. É também demonstrado que o simulador consegue operar com pisos diferentes e foi de igual forma representado o seu desempenho em tais pisos.

No fim deste capítulo é apresentado o desempenho de um sistema de *cleanRob* naes limpeza do Departamento de Engenharia Electrotécnica e de Computadores.

6.1 Area-Covering

6.1.1 Evolução Genética

A evolução genética é um método que permite, a partir de algumas iterações, encontrar uma solução boa para um problema. No entanto, nem sempre existe a noção de solução boa; Neste caso não existe. Assim foi necessário estimar quando é que o resultado tinha deixado de evoluir e já se podia considerar uma solução boa.

A estratégia utilizada para estimar uma boa solução foi iterar a evolução genética com populações de 20 elementos até a variação se tornar pouco significativa. Partindo de uma população inicial foram feitas 4^n evoluções e a partir do gráfico dos segundos de limpeza removidos por geração, foi estimado o tempo que se removeria na iteração seguinte.

Na tabela 6.1 pode-se verificar que a evolução genética tende a estabilizar. Nas primeiras gerações encontra-se variações de 27s por geração, já nas últimas esse valor desce para 0.018s. Prevendo o mesmo tipo de comportamento nas gerações seguintes, é espectável que nas próximas 1024 gerações se obtenha uma variação total em torno de 0,47s. Comparativamente, nas primeiras

Gerações	1	4	16	64	256	1024
Indivíduos	20	80	320	1280	5120	20480
$\sum$ Tempo (s)	10294	10213	10103	9978	9874	9860
$\frac{\Delta \text{Tempo}}{^{n}\text{Gerações}}$ (s)	-	27,167	9,167	2,609	0,542	0,018

Tabela 6.1: Evolução Genética no piso 0 do Departamento de Electrotecnia da FEUP

1024 gerações obteve-se uma redução de tempo de 434s. Assim, duplicando o número de testes apenas se deve obter uma melhoria nos genomas não superior a 0,11%. Dito isto, 1024 evoluções genéticas neste piso são suficientes, uma vez que os ganhos obtidos por aumentar este valor são reduzidos face ao tempo de simulação.

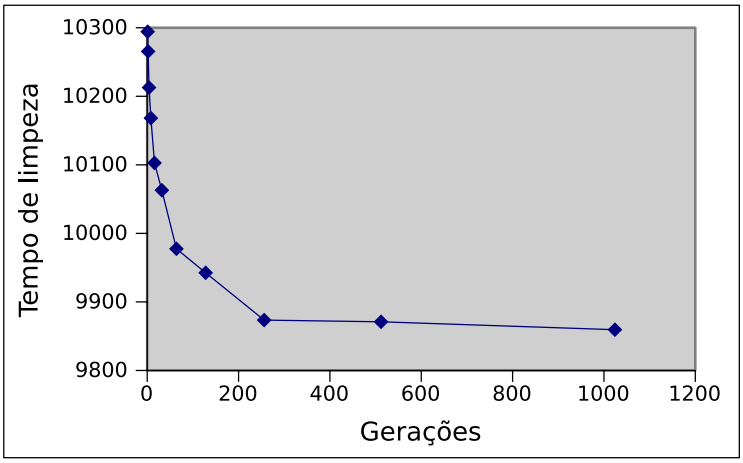


Figura 6.1: Evolução do tempo de limpeza face à evolução genética.

Na figura 6.1 está representado o gráfico que relaciona a evolução do tempo de limpeza com o número de gerações. Como se pode ver, a variação tende a estabilizar entre os 9800 e os 9900 segundos.

Com os resultados anteriores não só se demonstra que é fácil encontrar o número de iterações para terminar o treino genético, como também se conclui que a aplicação dos algoritmos genéticos converge para um resultado. Na secção seguinte avalia-se o resultado para o qual o algoritmo convergiu.

A título de curiosidade, o simulador levou aproximadamente 21 horas para obter as 1024 simulações. Cada uma destas simulações implica testar 20 robôs a limpar os 47 compartimentos do piso, o que significa que em média o simulador levou 79ms a testar uma estratégia de limpeza num compartimento. Pode-se ainda concluir que, na evolução genética, o simulador esteve acelerado a aproximadamente 8000x. Para desenvolver esta simulação foi utilizado um computador modesto, equipado com um *intel T2330* (1,6GHz) e 2GB de RAM.

A soma das combinações possíveis dos genomas dos compartimentos totalizam $2,08 * 10^{14}$. Desse total apenas foram testados $9,83 * 10^{5}$ combinações (sem contar com possíveis repetições).

Assim conclui-se ainda que o número de testes realizados foi muito mais baixo que o número de combinações possíveis.

6.1.2 Limpeza de Um Compartimento

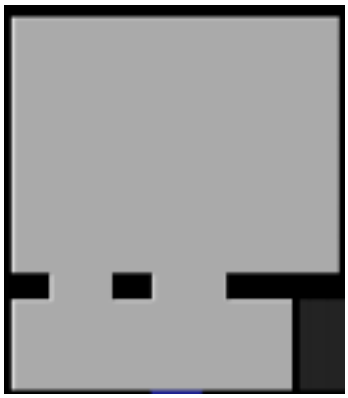


Figura 6.2: Pormenor da parte da imagem que deu origem ao compartimento 6 do piso 1.

Conforme foi referido no Capítulo anterior (cap 5) para a limpeza de cada compartimento foram utilizados algoritmos genéticos. Nesta secção, será comparado o resultado do algoritmo utilizado, com o método circundante (ver secção 4.3) e com um resultado encontrado empiricamente. Estes resultados terão base no compartimento r6 do piso 1 (ver figura 6.2), o qual foi escolhido devido à sua complexidade.

Caminho Empírico: Inicialmente foi criado um caminho que se considera bom para a limpeza do compartimento - **Caminho empírico**. Para se criar esse caminho teve-se em conta os seguintes factores:

- Evitar mudar de direcção.
- Evitar várias passagens no mesmo ponto.

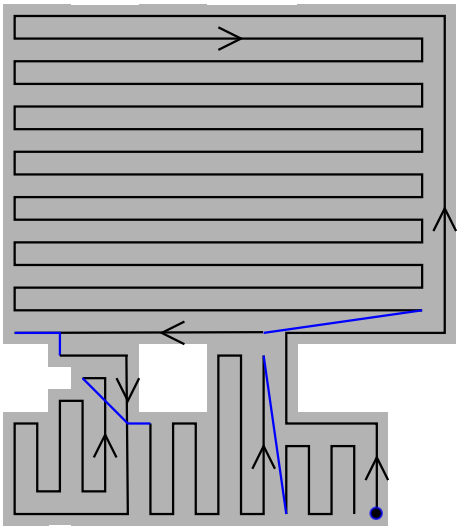


Figura 6.3: Caminho Empírico.

Na figura 6.3 está representado o Caminho Empírico escolhido. É possível verificar que este caminho tem poucas sobreposições e não muitas mudanças de sentido.

Caminho Circundante: Com o método descrito na secção 4.3 foi criado o **Caminho Circundante**. Este caminho é gerado sem necessidade de inteligência artificial ou interacção humana. Como se pode ver pela figura 6.4, o resultado não tem muitas sobreposições tendo no entanto muitas mudanças de sentido.

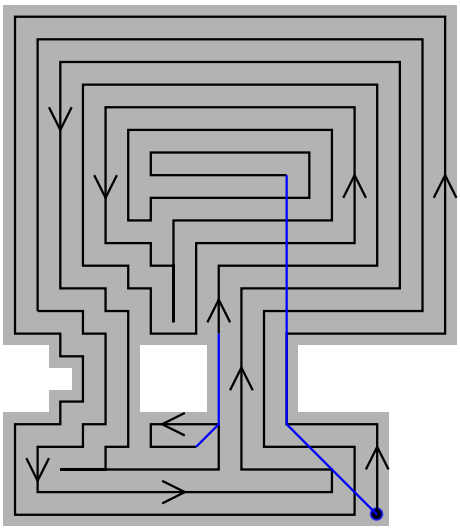


Figura 6.4: Caminho Circundante

Caminho Genético: Foi ainda criado criado um método de limpeza do compartimento utilizando os algoritmos genéticos - **Caminho Genético**. Este caminho foi uma selecção de 5000 indivíduos (100 gerações de 50 indivíduos).

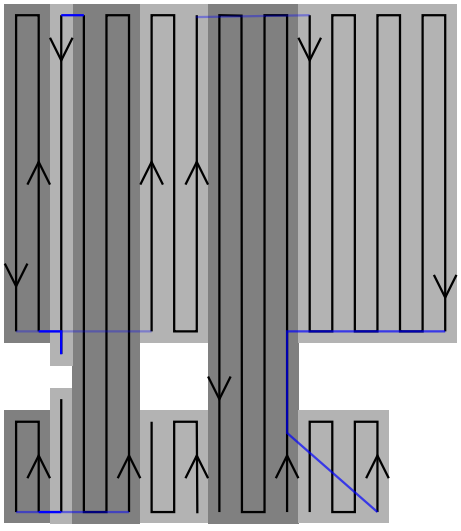


Figura 6.5: Caminho Genético.

Na figura 6.5 está representado o Caminho Genético vencedor de uma selecção de 5000. Como se pode ver, o caminho é realmente bom. Não tem muitas sobreposições e tem poucas inversões de sentido.

A comparação dos dois resultados pode ser feita de 3 formas: Pelo comprimento percorrido, pelo ângulo girado e pelo tempo de limpeza total.

Comprimento O comprimento percorrido foi de $170,0$m para o empírico (1^o), 174.4m para o circundante (3^o) e 171.6m para o genético. Este resultado significa que o caminho empírico foi o que repetiu menos vezes pontos.

Ângulo O rotação efectuada foi de 6300° para o empírico (2^o), 7650° para o circundante (3^o) e de 5850° para o genético (1^o). Assim, na rotação, o algoritmo genético possui o melhor resultado. Por outro lado, o circundante teve um resultado muito mau neste ponto.

Tempo O tempo é a avaliação que engloba o resultado das anteriores. Neste ponto, o resultado foi de 780s para o empírico (2^o), 863s para o circundante (3^o) e de 761s para o genético (1^o).

Em suma, o melhor resultado foi o do Caminho Genético com 761s, seguido pelo Caminho Empírico com 780s ($+2,5\%$) e por fim o Caminho Circundante com 863s ($+13.4\%$).

Talvez ao contrário do que se estaria à espera, apesar das limitações impostas pela segmentação das áreas ao caminho genético, este conseguiu ser melhor que um dimensionado por uma pessoa. Não que uma pessoa não conseguisse encontrar um caminho melhor que o representado, mas este resultado mostra que o caminho genético é um bom caminho, próximo do caminho óptimo.

Pode-se ainda ver neste exemplo que na maioria dos casos os caminhos escolhidos para limpeza de uma área são trajectórias em "S"e não circulares. As trajectórias circulares aparecem quando é mais conveniente terminar a limpeza da área no meio do compartimento do que numa extremidade. Isto é particularmente frequente em cantos de salas, situações em que é necessário voltar atrás após a limpeza. No piso 0, verificou-se que em 35% das áreas foi utilizada a limpeza circular.

6.2 Task Alocation

Neste trabalho foram utilizados *Auctions* (Leilões) na atribuição das tarefas. Neste caso os compartimentos licitam num robô disponível; Ao vencedor do leilão, é-lhe atribuído o robô. Os *Auctions* implementados foram uma expansão dos algoritmos *greedy*, acrescentando outros factores. Os três factores utilizados são: Proximidade do robô ao compartimento, a distância dos outros robôs ao compartimento e o tempo de limpeza do compartimento. Nesta secção procura-se demonstrar a vantagem da utilização destes três factores em simultâneo.

Na figura 6.6 é representado o somatório dos tempos não produtivos (utilizado em deslocações/esperas) na limpeza de um piso. O tempo de limpeza é sempre 9857s, o tempo de não produtivo é sempre superior a 2000s e sobe com o aumento do número de robôs.

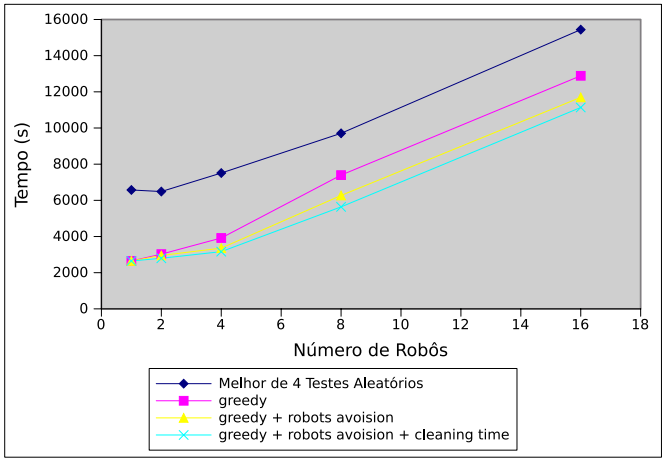


Figura 6.6: Desempenho comparativo dos diferentes métodos de alocação de tarefas.

Porque é que o aumento do número de robôs induz o aumento do tempo não produtivo? Por dois motivos:

1. Como os robôs partem todos do mesmo sítio, têm que se deslocar mais até começarem a limpar - Com um número de compartimentos igual ao número de robôs, um robô pode fazer uma deslocação de 500s para efectuar uma limpeza de 100s.

2. Esperar que os últimos robôs terminem a limpeza - Se o último compartimento demora 1000s a ser limpo e se enquanto um robô limpa esse compartimento 5 robôs estão à espera (terminaram o trabalho), isso corresponde a um desperdício de 5*1000s.

Greedy No gráfico pode-se ver que com a utilização de um algoritmo *Greedy* obtém-se um bom resultado. Comparativamente à melhor de 4 soluções aleatórias teve uma melhoria média de 40% e máxima de 60%. O *Greedy* é também extremamente simples de implementar e tem um custo baixo em processamento.

Robot Avoision O método *Robot Avoision* tenta distribuir os robôs por todas as zonas com necessidade de limpeza. Este método tem melhores resultados comparativamente ao greedy quando o número de robôs aumenta. As melhorias face ao *Greedy* puro são em média de 8% e no máximo de 15%. Este método é pesado porque necessita calculo de distâncias de todos os robôs para todos os compartimentos, utilizando os métodos de *path-planning* para 50 compartimentos por 16 robôs, o desempenho do simulador cai drasticamente. A solução para este problema de desempenho passa por criar uma *lookup table* com a distância entre todas as posições onde o robô pode estar (aproximação feita ao metro) e a posição da porta mais próxima de cada compartimento.

Cleaning Time Este método procura fazer com que os compartimentos maiores sejam limpos antes, evitando assim o 2º problema de produtividade enumerado acima. Este método consiste em utilizar directamente o valor de limpeza de cada compartimento (anteriormente calculado) na

licitação. Com a utilização deste valor obteve-se, face ao *Greedy* com *Robot Avoision*, um aumento de desempenho de médio de 5% e máximo de 10%. De forma semelhante ao *Robot Avoision*, este método também tem maior impacto quando aumenta o número de robôs.

Os *Auctions* provaram ser um excelente método para a alocação de tarefas, não só pela sua simplicidade, como pelos seus resultados. Têm ainda a vantagem, de ser simples acrescentar parâmetros, sendo uma atribuição de tarefas muito modificável. Se se quisesse contemplar também o tempo de bateria de cada robô, ou o recipiente do lixo, os *Auctions* estavam por natureza preparados. O maior problema dos *Auctions* é a calibração dos pesos que nem sempre é intuitiva ou simples. "Quanto mais vale limpar um compartimento próximo, face a afastar de outros robôs?". A resposta não é fácil ou directa. Uma solução para isto poderia ser correr um novo algoritmo genético no qual o valor destes parâmetros constituía o Genoma. Na solução desenvolvida, os parâmetros foram ajustados por tentativa-erro.

6.3 Navegação

Conforme referido em 5.6 o *Path Planning* foi estruturado em Navegação Local e Navegação Global e tem base numa Grelha de Navegação. Nesta secção é comparado o desempenho da navegação desenvolvida com a navegação óptima - a que minimiza a distância entre dois pontos.

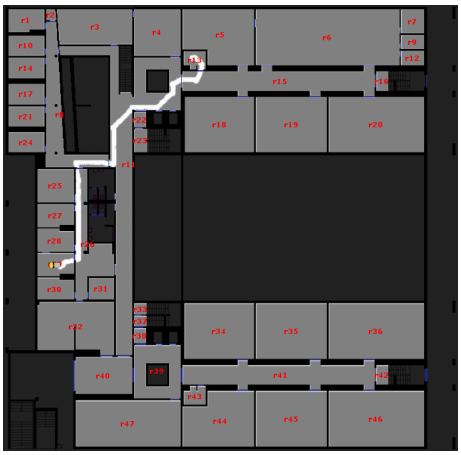


Figura 6.7: Navegação entre dois compartimentos.

Na figura 6.7 é representado o caminho que o robô escolheu para cumprir a navegação do compartimento *r13* para o compartimento *r29*. Como se pode verificar, o que limita a navegação é a malha de navegação, na qual o robô apenas se desloca com ângulos múltiplos de 45°. Dentro dessa limitação, o caminho escolhido parece ser realmente o caminho que minimiza a distância entre dois pontos.

Na figura 6.8 é representado o caminho óptimo para os mesmos compartimentos. A diferença entre as duas é o facto de esta não cumprir a malha de navegação.

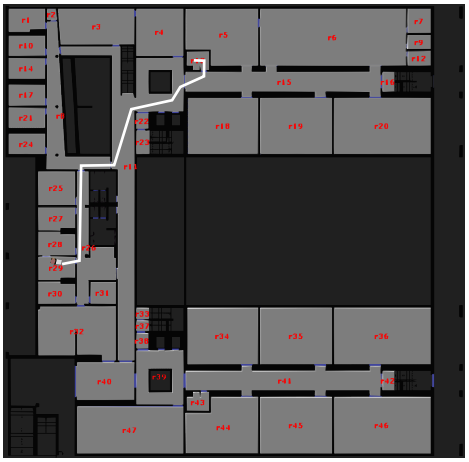


Figura 6.8: Navegação Óptima entre dois compartimentos.

Na solução implementada, a distância percorrida são $32,2$m; O caminho óptimo é de $30,7$m - o que corresponde a uma perda de 5% de tempo. Esta perda deve-se ao facto da navegação local ser constituída por uma malha com uma granularidade muito grande. A solução para esta perda passaria por aumentar a resolução da malha, aumentando também a necessidade de processamento, mas fazendo no limite o caminho tender para o óptimo. Este aumento de processamento podia ser crítico em simulação, tornando-a mais lenta, aumentando assim a demora de testes de desempenho (ver secção 5.6). No entanto, na aplicação prática, como o planeamento de trajectórias ocorre poucas vezes por minuto, o aumento desta grelha beneficiaria o resultado final.

A estruturação do problema de navegação em Local e Global permitiu distribuir um problema com complexidade conjunta grande em dois problemas simples. Esta estrutura permite também facilmente contemplar os elevadores do edifício (através de ligação ao nível da navegação global), permitindo que este método seja aplicado a edifícios mais complexos. Este método permite ainda utilizar métodos diferentes de navegação em cada um dos compartimentos: Por exemplo utilizar-se ruas com sentidos em corredores ou $A*$ em algum compartimento que o justificasse. Desta forma, a navegação deixa de ser igual para o universo e passa a adaptar-se às necessidades locais/globais. Este método permite ainda bloquear um corredor quando este está a ser limpo, deslocando os robôs para um corredor paralelo.

Em suma, a distribuição da navegação em Local e Global foi um dos trunfos deste projecto, que lhe permitiu ter um bom desempenho e ao mesmo tempo permite que este seja expandido, contemplando novos factores.

6.4 Utilização de Robôs com Características Diferentes

Com vista a demonstrar a capacidade do simulador de utilizar robôs com características diferentes, neste exemplo é mostrado o desempenho de um robô com as características do *Roomba* a limpar um piso do DEEC.

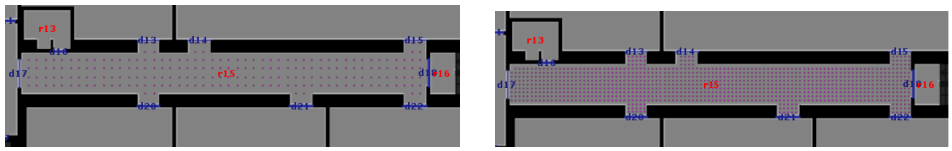


Figura 6.10: Comparativo entre a malha de limpeza do *cleanRob* (lado esquerdo) e do *Roomba* (lado direito).

Pelo facto de a área de limpeza do *Roomba* ser inferior à do *cleanRob*, a grelha de limpeza terá consequentemente de ser menor (ver secção 5.8.1). Assim para a distância entre pontos da grelha de limpeza utilizou-se 20cm. Por outras palavras, se o *Roomba* passar em duas trajectórias paralelas rectilíneas que distam 20cm, toda a área entre essas duas linhas é considerada limpa.

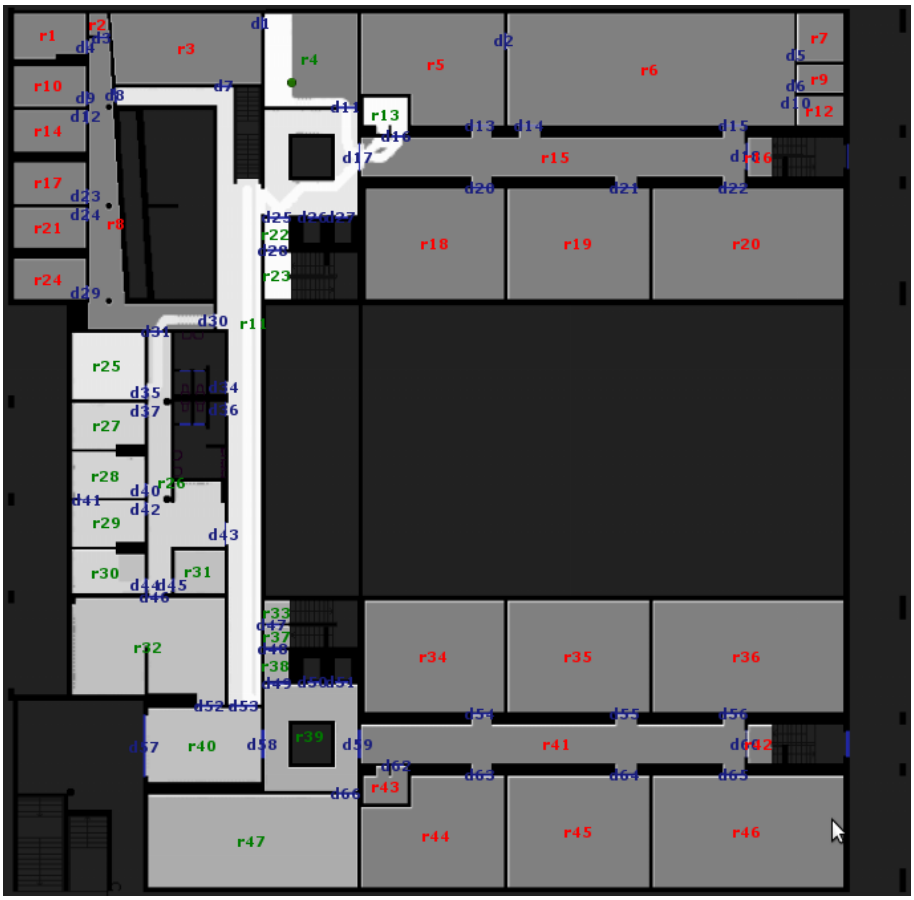


Figura 6.9: Robô com modelo do *Roomba* limpa o piso 0 do DEEC

O facto da grelha de limpeza do *Roomba* ter uma distância entre pontos inferior, faz com que haja mais pontos e com que o robô tenha que percorrer uma distância superior para cobrir todos os pontos. Isto, aliado a uma velocidade inferior, implica que o tempo de limpeza do piso seja superior. Na figura 6.10 é representada a comparação entre os pontos da malha de limpeza do *cleanRob* com os do *Roomba*.

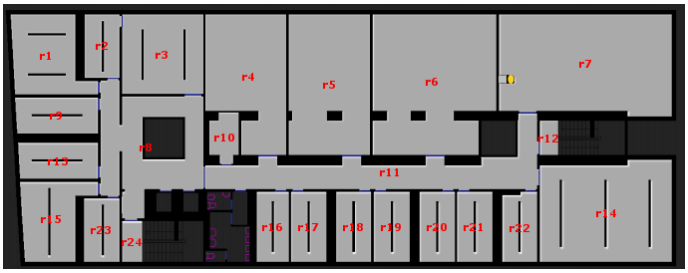
Na figura 6.9 é representado o *Roomba* a limpar. Com o *Roomba*, o tempo de limpeza do piso foi de 34831s. Como era de esperar este tempo é superior ao de limpeza com um *cleanRob* (12900s).

6.5 Múltiplos mapas

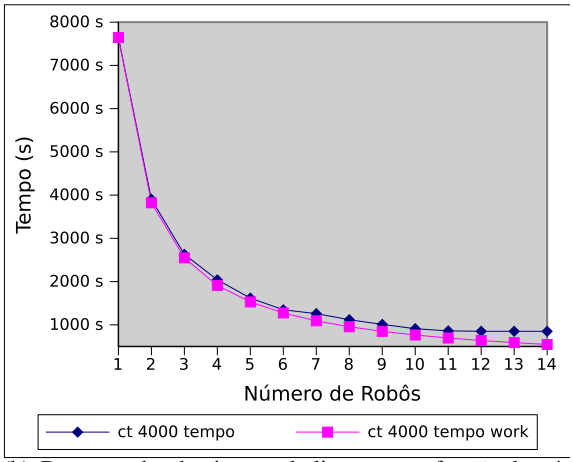
Uma das vantagens do dinamismo do código desenvolvido é permitir passar ao programa um mapa e um factor de escala sendo que este encarrega-se de criar tudo o que depende desse mapa: compartimentos, malhas de navegação global e locais, segmentação das áreas e genomas.

Para demonstrar esse dinamismo, foram feitos testes de desempenho com n robôs em 3 pisos do DEEC.

6.5.1 Limpeza do Piso 1 Oeste



(a) Mapa do piso.



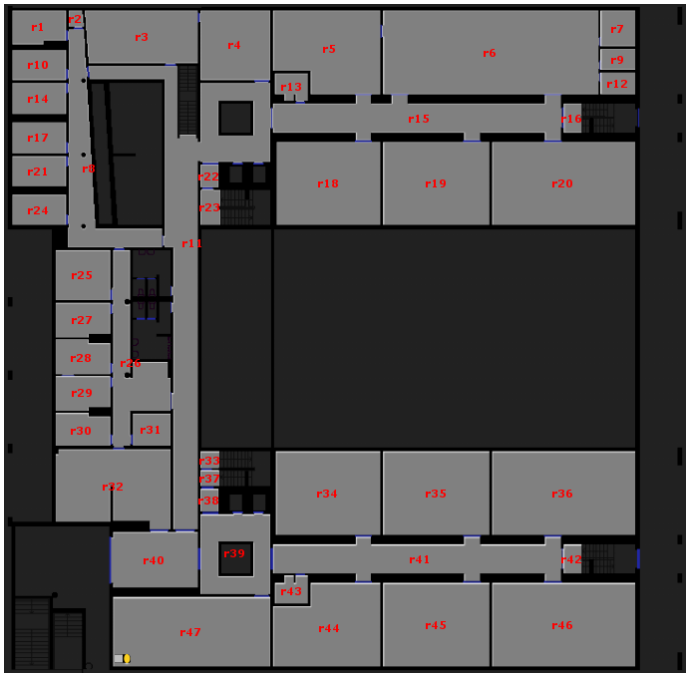
(b) Desempenho do sistema de limpeza em função do número de robôs.

Figura 6.11: Piso 1 Oeste da Departamento de Electrotecnia

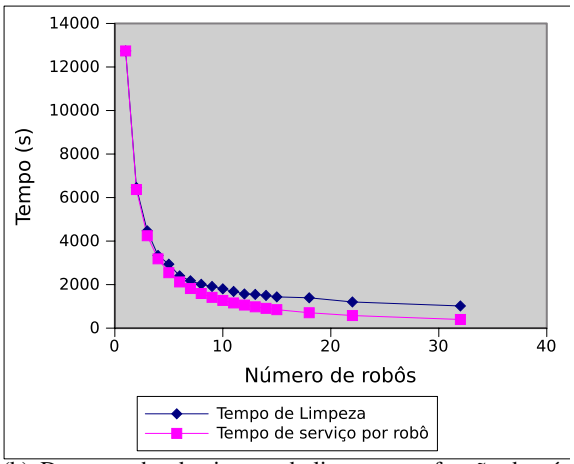
No ponto de vista da simulação da limpeza, o piso 1 Oeste é provavelmente o mais simples dos pisos; Tem poucos compartimentos de tamanhos reduzidos. Na figura 6.11(a) pode-ser ver o mapa do piso. É de destacar que são encontrados convenientemente todos os compartimentos e é marcado a vermelho (no seu centro geométrico) o nome atribuído a cada.

No gráfico da figura 6.11(b) podem-se destacar duas curvas. A rosa está marcado o tempo esperado na limpeza com n robôs (dado pelo tempo de limpeza com um robô a dividir por n). A azul está o tempo efectivamente despendido. Conforme seria de esperar, quando se aumenta o número de robôs, o tempo de limpeza total diminui. Por outro lado, ao aumentar o número de robôs, a produtividade de cada robô diminui; Ou seja, a percentagem do tempo realmente passado a limpar diminui.

6.5.2 Limpeza do Piso 0



(a) Mapa do piso.



(b) Desempenho do sistema de limpeza em função do número de robôs.

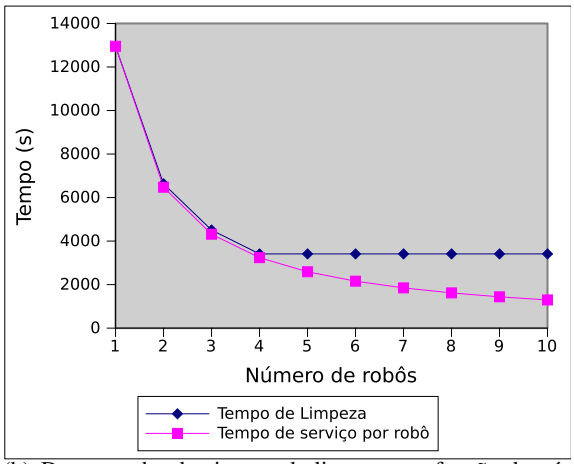
Figura 6.12: Piso 0 da Departamento de Electrotecnia

O piso 0 tem uma complexidade acrescida comparativamente ao piso 1, uma vez que enquadra os Lados Este e Oeste do edifício. Conforme se pode ver na 6.12(a) este piso tem sensivelmente o dobro dos compartimentos e com tamanhos mais variáveis.

Os resultados são semelhantes aos do piso 1: Com o aumento dos robôs o tempo de limpeza diminui, mas também a produtividade. Como se pode constatar, apesar da complexidade acrescida deste piso, o software não teve dificuldade em utilizá-lo, mantendo a qualidade dos resultados.

6.5.3 Limpeza do Piso -1

(a) Mapa do piso.



(b) Desempenho do sistema de limpeza em função do número de robôs.

Figura 6.13: Piso -1 da Departamento de Electrotecnia

A figura 6.13 mostra o mapa do Piso -1. A característica mais marcante deste piso é o facto do compartimento *r23* ser de grande dimensão (aproximadamente um campo de basquetebol). O tempo de limpeza deste compartimento são 3358s, o que vai limitar a atribuição de tarefas. Por outras palavras, 10 robôs são mais rápidos a limpar o resto do piso que um robô a limpar este compartimento, estando assim o tempo total de limpeza limitado pelo tempo de limpeza do compartimento *r23*. Uma solução para esta dificuldade é apresentada no capítulo 7.4.

6.5.4 Limpeza de Pisos

Como se pode verificar, pelo facto de ser construído um simulador dinâmico, foi possível, e em tempo reduzido, apresentar resultados com vários pisos. Desta forma, garante-se que o simulador testa pisos muito diferenciados sem que para isso seja necessário descrever o piso extensivamente. Outra vantagem deste método é permitir uma adaptação mais rápida do sistema desenvolvido a espaços diferenciados.

O defeito deste método é que os mapas muitas vezes não correspondem à realidade. Na altura da construção ou até por obras posteriores, muitas vezes são criadas alterações pequenas ou grandes ao mapa. Por este motivo, numa implementação real deste sistema é necessário ter este factor em conta, tendo os robôs que estar capacitados de métodos que lhes permitam actualizar o mapa. Tal método é sugerido secção 7.4.

6.6 Limpeza do Departamento de Electrotecnia

Uma das questões colocadas no início do projecto foi: *Qual é o número de cleanRobs necessário para limpar o Departamento de Electrotecnia durante uma noite?*

Em posse do simulador é possível encontrar uma estimativa deste número.

Para desenvolver a solução para o problema é necessário entrar com algumas considerações:

- A noite tem 6 horas [1h, 7h].
- O DEEC Ala *i* é constituído por duas torres de 5 pisos, sendo as duas torres unidas nos pisos -1 e 0.
- Os pisos 1,2 e 3 sãos constituídos cada um por dois blocos semelhantes ao *Piso 1 Oeste*. Estes dois blocos não têm acesso directo.
- De acordo com os itens anteriores, o edifício é composto por 8 blocos (1,1,2,2,2 nos pisos -1,0,1,2,3 respectivamente).
- Foi estimado que um robô demora 3 minutos a mudar de bloco.
- O tempo de limpeza de cada bloco varia com o número de robôs do modo referido na secção 6.5.

Para resolver o problema acima foram considerados 3 modos:

Limpeza Sequencial *n* robôs limpam ao mesmo tempo um bloco de cada vez. Os blocos são limpos sequencialmente.

Limpeza Paralela São atribuídos robôs a cada bloco. Todos os pisos são limpos ao mesmo tempo.

Limpeza Mista A cada grupo de robôs é atribuído um grupo de blocos. Neste caso colocou-se o mesmo número de robôs em cada piso (Os robôs dos pisos 1,2 e 3 limpam 2 blocos).

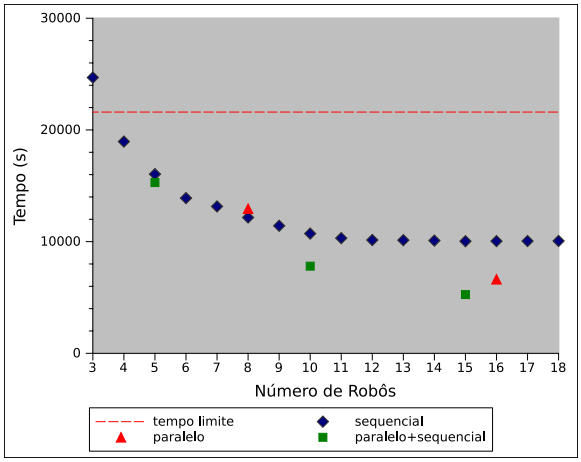


Figura 6.14: Desempenho comparativo dos diferentes métodos de alocação de tarefas.

Na figura 6.14 está representado o desempenho de *n* robôs a limpar o edifício. Neste gráfico está marcado a tracejado o tempo a partir do qual o resultado é aceitável (inferior às 6 horas nocturnas).

Como se pode constatar pelo gráfico, a limpeza sequencial é a mais versátil pois permite adicionar ou remover um robô sem prejudicar drasticamente o desempenho global. Já na limpeza paralela, adicionar um robô muitas vezes não conduz sequer a um melhor resultado, isto porque é acelerada apenas a limpeza de um dos pisos. A Limpeza Mista é uma solução adaptada ao problema e que, por isso, consegue tempos mais reduzidos.

Resultados importantes:

- Com 4 ou mais robôs é possível limpar o edifício nas 6 horas
- Com limpeza sequencial, 4 robôs limpam o edifício com tolerância de 12% do tempo e produtividade de 75% (tempo utilizado para limpar).
- Com limpeza sequencial, 5 robôs limpam o edifício com tolerância de 26% e produtividade de 71%.
- Com limpeza mista, 5 robôs limpam o edifício com tolerância de 29% e produtividade de 75%.

- Com limpeza paralela, são necessários 8 robôs para limpar o edifício, limpando com tolerância de 40% e produtividade de 55%.

Tentando manter uma tolerância de 20% existem 3 soluções possíveis, cada uma com as suas vantagens:

5 robôs com limpeza sequencial: Este é certamente a aposta mais fiável uma vez que, se um robô avariar, é possível limpar o edifício com 4 robôs mantendo ainda uma tolerância de 12%. Outra vantagem é a facilidade em acrescentar um robô. A desvantagem deste método é a utilização dos elevadores por parte dos robôs. Ao todo implica no mínimo $5 * 8$ viagens de elevador. Este tipo de ocupação em principio não será drástica durante a noite, uma vez que o número de pessoas a essa hora na faculdade não é grande.

5 robôs com limpeza mista : A limpeza mista é o método que mantém produtividades mais altas ou seja, é o tipo de limpeza que mais inibe os desperdícios de tempo por parte dos robôs. Este método diminui a utilização dos elevadores para apenas 3 vezes por limpeza do edifício.

8 robôs com limpeza paralela : A limpeza paralela tem a vantagem de não ser preciso preparar os elevadores para receberem robôs e vice versa uma vez que deixa de haver essa interacção. No fundo é como se houvesse vários CTPS, um por bloco. A grande desvantagem deste método é a necessidade de um elevado número de robôs, o que no caso de robôs com um preço elevado, este pode depressa ficar desapropriado.

A meu ver a melhor solução é a dos 5 robôs com **Limpeza Sequencial**. Em horas nocturnas a utilização de elevadores não é critica, e tratando-se de um sistema que facilmente permite, caso se justifique, acrescentar mais robôs e diminuir o tempo de limpeza. Por outro lado, no caso de um robô avariar durante a noite, o sistema pode responder e distribuir as tarefas pelos outros robôs, tendo de manhã o resultado que se teria com os 5 robôs. Este é um método que utiliza a cooperação entre robôs.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusão

Esta dissertação aborda não só a cooperação robótica como o desenvolvimento de um Software de elevada complexidade e com múltiplas funcionalidades. Nos capítulos anteriores foram discutidas estratégias de cooperação, planeamento de trajectórias, algoritmos de cobertura de superfícies, segmentação de áreas, algoritmos de segmentação em visão, modelação da realidade e algoritmos genéticos. Um dos factores centrais deste projecto de um Simulador, foi a sua aplicabilidade.

Um dos factores essenciais no desenvolvimento de um Software é o seu projecto. Pode muitas vezes não ser visível num documento deste tipo, o caminho que o projecto levou deste o seu planeamento até a sua implementação. Neste caso, parte das grandes chaves do sucesso do desenvolvimento estiveram na fase inicial.

Em primeiro lugar, a escolha da ferramenta de programação. Pelo facto do *Python* ser interpretado, é uma linguagem pesada computacionalmente. No entanto, pelo mesmo facto e por permitir coexistir com outras linguagens, torna-se uma excelente linguagem para prototipagem - Em *Python* é rápido testar múltiplos algoritmos e, caso se justifique, escrever alguns em linguagens com melhores desempenhos.

Em segundo lugar, antes de ser iniciada a programação foi desenhada grande parte da estrutura do código com classes e funções vazias. Desta forma, garantiu-se a coesão do código, sistematizando o desenvolvimento das funções. Esta estruturação conduziu também a que o projecto fosse desenvolvido com calma e com um bom planeamento e organização do tempo.

Esta arquitectação e estruturação acentuada permitiu uma construção sólida do software, fazendo com que fosse relativamente fácil garantir dinamismo no simulador, permitindo que aceitasse múltiplos mapas e gravasse resultados tanto dos compartimentos como dos genomas. O planeamento inicial permitiu também que o simulador estivesse preparado para funcionar com múltiplos pisos e elevadores, sendo necessário apenas preencher código em locais específicos.

Um outro factor chave no bom desempenho do projecto foi a escolha dos Algoritmos Genéticos na limpeza dos compartimentos. Apesar de algum receio inicial que a complexidade dos Genomas envolvidos tenha levantado, os Algoritmos Genéticos mostraram-se eficazes e sólidos. Com apenas um bilionésimo dos testes das soluções possíveis, apresentaram resultados muito bons, até mesmo considerando uma possível implementação comercial. Ainda nos algoritmos genéticos, nesta dissertação foi utilizada uma técnica para obter o primeiro Genoma de cada compartimento. Esta técnica mostrou acelerar em muito a estabilização na obtenção do resultado final.

A estruturação do problema de navegação em Local e Global permitiu distribuir um problema com complexidade conjunta grande em dois problemas simples. Esta estrutura permite também facilmente contemplar os elevadores do edifício (através de ligação ao nível da navegação global), permitindo que este método seja aplicado a edifícios mais complexos. Este método permite ainda utilizar métodos diferentes de navegação em cada um dos compartimentos: Por exemplo, utilizar-se ruas com sentidos em corredores ou $A*$ em algum compartimento que o justificasse. Desta forma, a navegação deixa de ser igual para o universo e passa a adaptar-se às necessidades locais/globais. Este método permite ainda bloquear um corredor quando este está a ser limpo, deslocando os robôs para um corredor paralelo. Em suma, a distribuição da navegação em Local e Global foi um dos trunfos deste projecto, que lhe permitiu ter um bom desempenho e ao mesmo tempo permite que este seja expandido, contemplando novos factores.

Os *Auctions* provaram ser um excelente método para a alocação de tarefas, não só pela sua simplicidade, como pelos seus resultados. Têm ainda a vantagem, de ser simples acrescentar parâmetros, sendo uma atribuição de tarefas muito modificável. Se se quisesse contemplar também o tempo de bateria de cada robô, ou a percentagem do recipiente do lixo ocupada, os *Auctions* estavam por natureza preparados.

De forma resumida, neste projecto foram obtidos resultados bons e inovadores. Foram introduzidas as técnicas de navegação Global e Navegação Local. Foi ainda encontrado um método para obter uma primeira iteração do Algoritmo Genético de organização de limpeza de áreas que provou ser muito eficaz. A utilização dos Leilões para a atribuição de tarefas foi sem dúvida um ponto de sucesso nesta dissertação, permitindo que o sistema se adaptasse a quaisquer necessidades de limpeza, sem que esta tenha que ser atribuida por um operador.

7.2 Lições Aprendidas

A primeira das lições aprendidas com esta dissertação é a separação da navegação em Global e Local. Pelo facto de a navegação ser estruturada em camadas permite por um lado acelerar o Path-Planning, por outro dar ordens de alto nível como bloquear estradas ou corredores.

Nos algoritmos genéticos foi tomada a segunda lição. Os algoritmos genéticos por vezes demoram a tender para um bom resultado. Utilizando técnicas para obter uma primeira iteração consegue-se diminuir o tempo de tender para um bom resultado.

A terceira lição foi que na Engenharia as soluções estão à nossa volta, apenas temos que as saber procurar. A maioria das soluções implementadas foi baseada em estratégias já utilizadas noutros contextos (como no caso da divisão da localização na secção 5.6). Na natureza estão as respostas simples para os problemas complexos.

7.3 Satisfação dos Objectivos

Todos os Objectivos referidos em 1.2 foram concluídos com sucesso. Foi desenvolvido um sistema de controlo para equipas de robôs que visa a limpeza de um edifício e também um simulador para o mesmo sistema. Graças ao dinamismo presente no simulador, foi possível testar o mesmo com diferentes mapas e com diferentes estratégias de limpeza. Foram ponderadas várias arquitecturas de cooperação. Com vista a manter uma velocidade de simulação rápida foram desenvolvidas estratégias como a navegação estruturada e os robôs imaginados, tendo sido conseguido na evolução genética acelerações na ordem das 8000x. Foi ainda simulada a limpeza no DEEC concluindo que são necessários 4 robôs, no mínimo, para conseguir a limpeza durante uma noite.

7.4 Trabalho Futuro

Chegado o fim do projecto, há ainda alguns pontos que podiam ser melhorados. Esse pontos assentam principalmente em melhorar a simulação, tanto por acelerá-la como por introduzir-lhe mais realismo, e melhorias nos algoritmos genéticos. Apesar de no início da tese ter sido feito um esforço no planeamento do projecto de forma a escolher as melhores soluções, há sempre tópicos nos quais só se vê a melhor solução após tentar a errada. Nas seguintes secções são sugeridas alternativas a quem procurar desenvolver um simulador/estratégias para robôs de limpeza. Por outro lado, são também sugeridas alterações que permitiriam dar mais realismo à simulação.

7.4.1 Acrescentar realismo ao simulador:

Introdução de Factores Aleatórios Um simulador não permite testar a resposta a situações reais sem a introdução de factores aleatórios. Sejam eles pessoas, portas bloqueadas, mesas mexidas de lugar. Um dos pontos que permitiria acrescentar realismo ao simulador seria introduzir tais factores aleatórios.

Interacção entre Robôs: Os Robôs não se atravessam. Para aumentar o realismo da simulação seria preciso considerar este factor, não só localmente - robôs desviarem-se; como globalmente - o planeamento das trajectórias deveria evitar cruzamento de robôs em portas apertadas. As trajectórias dos robôs também devia sempre que possível evitar caminhos nos quais robôs estivessem a limpar.

Acrescentar Elevadores: Acrescentando elevadores seria possível testar a limpeza de todo o edifício. O código existente está já preparado para receber múltiplos pisos.

Acrescentar um Mapa Real e um Simulado: Com estes dois mapas, seria possível utilizar o robô para reidentificar paredes, tirar conclusões sobre zonas habitualmente ocupadas - seja com máquinas ou armários.

7.4.2 Acelerar a simulação

O *bottleneck* do desempenho da simulação é o *Path-Planning*. Isto deve-se à enorme quantidade de caminhos calculados na atribuição de tarefas dos *Auctions*. Para acelerar o *Path-Planning* exitem duas técnicas principais.

Substituir *Dijkstra* por *A:** *A** é mais rápido e tem por base *Dijkstra*. Substituir *Dijkstra* por *A** não será muito demorado e os ganhos na execução podem ser drásticos.

Fazer uma *lookup table* medir distâncias entre pontos: Considerando uma precisão ao metro e que a distância era guardada num byte, mesmo assim a tabela para o *edifício i* ocuparia em torno dos 130MB. Esta solução diminui a utilização do processamento mas aumenta os gastos de memória. Para um edifício muito maior que o *edifício i* o melhor seria guardar apenas os caminhos todos que partissem dos pontos principais (como compartimentos ou portas) para todos os outros pontos. Assim acelerava-se o processo sem grandes alocações de memória.

Escrever os módulos de *path-plannig* em c++: Conforme referido na secção 5.1, uma das vantagens do python é permitir uma simples integração de módulos escritos em c++. Como c++ permite código com os melhores desempenhos, acredito que os módulos de *path-plannign* deviam ser implementados em c++.

7.4.3 Melhorias nos algoritmos genéticos

Os algoritmos genético implementados têm algumas limitações. Nesta secção são levantadas as limitações e sugeridas melhorias.

Todas as melhorias sugeridas acima para o *Path-Planning* conduzem também a melhorias no desempenho da Evolução Genética.

Considerar porta de Início e porta de Fim: A implementação dos Algoritmos Genéticos desenvolvida não têm noção do ponto de partida, nem do ponto de término da limpeza. Em alguns compartimentos, esta limitação implica que quando o robô chega ao compartimento para limpar, se tenha que deslocar para a zona marcada como ponto de início de limpeza e quando termina, tem que se deslocar para a porta de saída. Uma solução para isso seria treinar, para cada compartimento, vários algoritmos genéticos que contemplavam cada combinação de duas portas como

entrada e saída. Assim era possível que quando fosse dada a ordem de limpeza fosse escolhido um caminho que começasse o mais próximo da porta de entrada e que terminasse igualmente próximo da porta de saída.

***Hill Climbing* na Evolução Genética:** Uma técnica utilizada para optimizar a o resultado do Algoritmo Genético e garantir que o melhor indivíduo se encontrar num máximo local do genoma é o *Hill Climbing*. Com este método, garante-se que uma variação única num dos Genes do Cromossoma do Indivíduo não conduz a uma melhoria no resultado de simulação.

Utilização de *Lookup Tables* na simulação de Limpeza de Áreas: Na implementação feita no projecto, o robô percorre cada uma das áreas do compartimento sempre que pretende avaliar o genoma. Na verdade o tempo de limpeza de uma das áreas com uma estratégia não varia. Assim, associado a uma área poderia estar um tempo, reduzindo da evolução genética uma boa parte do tempo de processamento.

Partir compartimentos em sub-compartimentos: Foi visível no exemplo de limpeza do Piso -1 (ver exemplo na secção 6.5.3) que a partir dos 4 robôs, o aumento do número dos robôs não diminuía o tempo de limpeza. Isto deve-se ao facto de um dos compartimentos ter uma dimensão muito grande e limitar o tempo total de limpeza. Um dos métodos de que os algoritmos genéticos poderiam beneficiar seria partir compartimentos grandes em sub-compartimentos. Esse sub compartimentos poderiam ser um conjunto de sub-áreas do compartimento grande. Desta forma permitia-se a distribuição de tarefas dentro de um compartimento grande.

Referências

[1] Mark Yim, Kimon Roufas, David Duff, Ying Zhang, Craig Eldershaw, Sam Homans. Modular Reconfigurable Robots in Space Applications. *Palo Alto Research Center (PARC)*.

[2] Chris Rouff Michael G. Hinchey, Roy Sterritt. Swarms and Swarm Inteligence. 2007.

[3] http://maquinaespeculativa.blogspot.com/2009/06/robocup-as-diferentes-ligas-da.html. Small Robot League. 2007.

[4] Armando Jorge Sousa. Cleanrob - robot autónomo de limpeza (feup deec) — cleanrob - robot de limpeza, feup, 2008. [Online; accessed 23-June-2010].

[5] Fernando Pedro Vieira Freitas Pinto e Gustavo Gama da Rocha Pimentel. Cleaning Robot - Robot Autónomo de Limpeza. *PSTFC da LEEC*, 2007.

[6] Daniel Tiago de Jesus Coutinho. Melhoramento da interface local de um Robô de Limpeza através da utilização do comando da consola Wii. *Dissertação MIEEC FEUP*, 2008.

[7] João Manuel Ferreira Martins. Melhoramento do Desempenho do Robot de Serviço de Limpeza - Comparação de desempenho de Filtros de Kalman e de Filtros de Partículas para Auto-Localização. *Dissertação MIEEC FEUP*, 2008.

[8] Gustavo Gama da Rocha Pimentel. Aplicação do Filtro de Partículas como sistema de fusão de informação. *Dissertação MIEEC FEUP*, 2007.

[9] Fernando Pedro Vieira Freitas Pinto. Aplicação do Filtro de Kalman na Auto-Localização de um Sistema Robótico Autónomo. *Dissertação MIEEC FEUP*, 2007.

[10] Paulo Costa Armando Sousa, Pedro Moreira. Multi hypotheses Navigation for Indoor Cleaning Robots. *iRobot 2008*, 2008.

[11] Erwin Prassler, Arno Ritter, Christoph Schaeffer, Paolo Fiorini. A Short History of Cleaning Robots. *Kluwer Academic Publishers. Manufactured in The Netherlands.*, 2000.

[12] Hyunjin Kim, Hyunjeong Lee, Stanley Chung, Changsu Kim. User-Centered Approach to Path Planning of Cleaning Robots: Analyzing User's Cleaning Behavior. *Samsung advanced institute of technology*, 2007.

[13] Roland Siegwart e Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Book, 2004.

[14] Frederico Carvalho Vieira. *Controle Dinâmico de Robôs Móveis com Acionamento Diferencial*. Tese de doutoramento, Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, 2005.

[15] Moshe Sniedovich. Dijkstra's algorithm revisited: the dynamic programming connexion. 2006.

[16] Sven Koenig, Maxim Likhachev, Yaxin Liu, e David Furcy. Incremental heuristic search in ai. *AI Mag.*, 25(2):99–112, 2004.

[17] Y. UNY CAO. Cooperative Mobile Robotics: Antecedents and Directions. Relatório té, Computer Science Department, University of California, Los Angeles.

[18] Lynne E. Parker. *Heterogeneous Multi-Robot Cooperation*. Tese de doutoramento, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, February 1994.

[19] Philippe Caloud, Wonyun Choi, Jean-Claude Latombe, Claude Le Pape and Mark Yim. Indoor Automation with many Mobile Robots. *IEEE International Workshop on Intelligent Robots and Systems, IROS*, 1990.

[20] David MarFarland. Towards Robot Cooperation. Relatório té, Department of Zoology and Balliol College.

[21] David B. Leake. CBR in Context: The Present and Future. *Case-Based Reasoning: Experiences, Lessons, and Future Directions, AAAI Press/MIT Press*, 1996.

[22] M. HORVA TH J. VA NCZA e Z. STANKO CZI. Robotic inspection plan optimization by case-based reasoning. *Journal of Intelligent Manufacturing*, 2004.

[23] James F. Frenzel. Genetic Algorithms - A new breed of optimization. *IEEE Potentials*, 1993.

[24] Mitchell e Melanie. *An Introduction to Genetic Algorithms,* . MIT Press, Cambridge, 1996.

[25] Ann Nicholson Paulo A. Jimenez, Bijan Shirinzadeh e Gursel Alici. Optimal Area Covering using Genetic Algorithms. *IEEE*, 2007.

[26] Pina Martins. Apontamentos de Sistemas Baseados em Inteligência Computacional - Redes Neuronais e Neuro-Difusas. 2007.

[27] Nikos Drakos. *Computer Based Learning Unit*. Tese de doutoramento, University of Leeds, 1993.

[28] Stuart J. Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[29] Brian P. Gerkey e Maja J. Mataric. Sold!: Auction Methods for Multirobot Coordination. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 18*, 2002.

[30] Simon X. Yang Chaomin Luo e Xiaobu Yuan. Real-time Area-Covering Operation with Obstacle Avoidance for Cleaning Robots. *Inteligent Robots and Systems*, 2002.

[31] Howie Choset. Coverage of Known Spaces: The Boustrophedon Cellular Decomposition. Relatório té, Department of Mechanical Engineering, Carnegie Mellon University, 2000.

[32] Simon X. Yang't, Chaomin Luo and Q.-H. Max Meng. Real-Time Area Covering Operation with Obstacle Avoidance for Cleaning Robots. *Advanced Robotics a n d Intelligent Systems (ARIS) Lab*, 2002.

[33] Simon X. Yang't, Chaomin Luo and Q.-H. Max Meng. Area-Covering Operation of a Cleaning Robot in a Dynamic Environment with Unforeseen Obstacles. *Advanced Robotics a n d Intelligent Systems (ARIS) Lab*, 2003.

[34] Michel Taix , Philippe Soueres , Helene Frayssinet, and Lionel Cordesses. Path Planning for Complete Coverage with Agricultural Machines. *Springer-Verlag Berlin Heidelberg 2006*, 2006.

[35] Y. Fu and S.Y.T. Lang. Fuzzy logic based mobile robot area filling with vision system for indoor environments. *IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation*, 1999.

[36] DeWitt Latimer IV, Siddhartha Srinivasa, Vincent Lee-Shue, Samuel Sonne, Howie Choset, Aaron Hurst. Towards Sensor Based Coverage with Robot Teams. *International Conference on Robotics and Automation*, 2002.

[37] Francisco Santos, Nuno Cerqueira, Ricardo Barreira, Rui Martins. The State of Python. 2006.

[38] Dan Gilbert. TED Conference. Dan Gilbert asks, Why are we happy?, 2004.

[39] Armando Jorge Sousa, Catarina Santiago, Luís Paulo Reis, M. L. Estriga. Automatic Detection and Tracking of Handball Players. *II ECCOMAS Thematic Conference on Computational Vision and Medical Image Processing*, 2009.