SysPlan - Sistema de Planeamento de Rotas para um Robô de Transporte

Ricardo Candeias

Orientador: Prof. Nuno Leite

Relatório Final realizado no âmbito de Projeto e Seminário

Licenciatura em Engenharia de Informática e de Computadores

julho de 2021

INSTITUTO POLITÉCNICO DE LISBOA

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

SysPlan

42087 Ricardo Candeias

Orientador: Nuno Leite

Relatório Final realizado no âmbito de Projeto e Seminário

Licenciatura em Engenharia Informática e de Computadores

julho de 2021

Resumo

Com o avanço na área de IA diversos serviços que anteriormente eram apenas realizados por humanos, podem agora ser realizados por robôs. Nomeadamente serviços que envolvem robôs que realizam tarefas num circuito local, por exemplo, em edifícios. O presente projeto tem como inspiração um sistema já existente onde um robô ministra o serviço de *check-in* num aeroporto. Tem como objetivo adicional o desenvolvimento do conhecimento e das competências do autor na área de IA. Neste âmbito, pretende-se produzir três componentes de software que formam um Sistema de Planeamento de Rotas para um Robô de Transporte. Estes componentes são: Uma RN para o reconhecimento do bilhete, com fim à obtenção do terminal destino que constitui a simplificação do processo de leitura do mesmo, onde se assume um pré processamento de imagem que produz dígitos isolados e preparados para serem lidos; Um algoritmo de pesquisa para o planeamento de um caminho para este mesmo local, que simula o transporte de malas até ao terminal; Para visualização do funcionamento destes componentes, pretende-se também desenvolver um ambiente gráfico. Como resultado espera-se uma prova de conceito funcional dum sistema deste tipo.

Palavras-Chave: Algoritmo de Pesquisa, Caminho, Leitura de caracteres, Rede Neuronal.

Lista de Acrónimos

GAC	Global Assembly Cache
IA	Inteligência Artificial
MNIST	Modified National Institute of Standards and Technology database
RN	Rede Neuronal
Swipl	Swi-Prolog
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Com o crescente desenvolvimento da IA, regista-se atualmente uma grande tendência para a automação de tarefas. Tarefas que são agora realizadas por sistemas dotados de IA, em vez de serem realizadas por humanos. Existem diversas ações que máquinas executam de forma mais eficiente e consistente do que humanos. Uma das áreas que máquinas irão dominar no futuro é a do transporte de cargas e pessoas *Staff (2018)*. Mais concreto para o projeto a desenvolver, seriam então robôs de transporte de cargas ou malas, ou seja, robôs que se movimentam, por exemplo, dentro de edifícios. Pretende-se realizar um sistema de controlo para um robô deste tipo, como por exemplo um robô de transporte de malas num aeroporto. O sistema a desenvolver neste projeto, designado SysPlan, foca-se então na obtenção de um destino e no cálculo da rota para o mesmo, tendo por base conceptual um robô que efetua o *check in* num aeroporto e transporta as malas dos passageiros para o terminal correto.

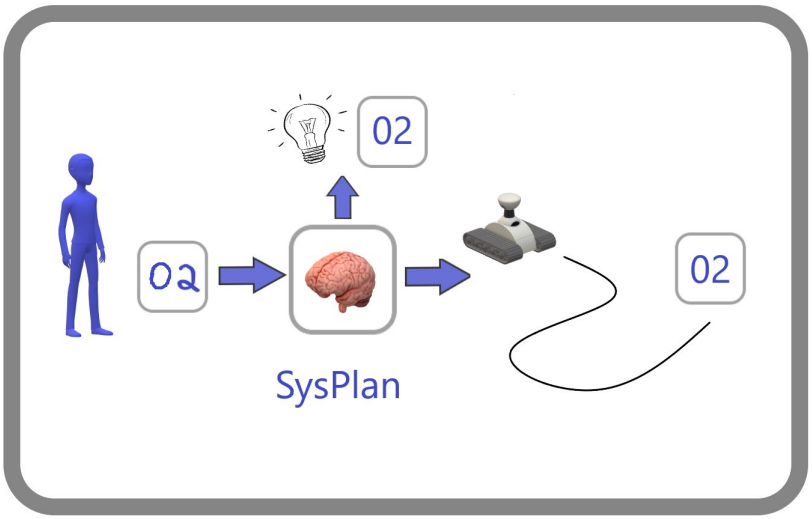

Figura 1.1: Cenário de utilização do sistema SysPlan

A figura 1.1 ilustra o cenário de utilização do sistema SysPlan, que utiliza uma rede neuronal para efetuar a leitura de um código manuscrito e calcula a rota que um robô teria de seguir para se deslocar para o destino lido no código. A pesquisa da rota mais curta

entre dois pontos é feita usando um algoritmo ótimo, ou seja, é garantido que o caminho mais curto entre dois pontos, caso exista, é encontrado pelo algoritmo.

O resto do texto está organizado em 3 capítulos. No capítulo 2, Formulação do Problema, é feito o enquadramento do projeto. Começa-se com a descrição de outros sistemas similares ao sistema proposto. De seguida, definem-se os requisitos do sistema a implementar bem como decisões tomadas na escolha de tecnologias e metodologias.

No capítulo 3, Implementação, é exibida a elaboração dos diversos componentes do projeto, tendo em conta as tecnologias e os métodos escolhidos.

Por fim, no capítulo 4 formulam-se as conclusões retiradas da elaboração do projeto e propõe-se trabalho futuro e melhorias que possam enriquecer o mesmo.

Capítulo 2

Formulação do Problema

Conteúdo

No presente capítulo, é feito o enquadramento do projeto, onde se começa com a descrição de outros sistemas similares ao sistema proposto. De seguida, definem-se os requisitos do sistema a implementar bem como decisões tomadas na escolha de tecnologias e metodologias.

2.1 Estado da Arte

Hoje em dia existem uma multitude de serviços que robôs conseguem providenciar, tais como rececionistas, robôs de limpeza, arrumadores de malas em hotéis, entre outros. Já existem empresas que comercializam robôs deste tipo *Chang (2015); Servicerobots (2021)*. Existem também serviços mais delicados como cuidadores em lares, diversos indicadores demonstram a necessidade destes robôs tornarem-se uma realidade comum, devido ao envelhecimento da população *Walker (2021)*. O presente projeto foi inspirado num outro sistema existente, um robô de transporte de malas em aeroportos, o robô, designado por '*Leo*', foi apresentado em Loughran onde desempenhou as suas tarefas durante uns dias, este tem funcionalidades de fazer *check-in*, imprimir etiquetas de malas, transportá-las, bem como a capacidade de evitar obstáculos e manobrar numa área de elevada densidade de pessoas *Loughran (2016)*.

2.2 Requisitos

O sistema desenvolvido neste projeto tem por base concetual uma simplificação do sistema que rege o robô '*Leo*'. Como grandes problemas inclui a obtenção do terminal de destino a partir de uma imagem de um bilhete e calcular a rota para o mesmo.

Um robô de transporte de malas real, como o robô 'Leo', poderá ter capacidades de reconhecimento OCR dos bilhetes dos utilizadores. No presente projeto foi feita a seguinte simplificação. O bilhete é representado por dois caracteres manuscritos, que são desenhados no ambiente gráfico desenvolvido, onde são processados segundo uma

formatação pré-definida.

Assim, o SysPlan é composto pelos seguintes componentes:

- Leitura e reconhecimento de caracteres manuscritos/bilhete indicando o destino, o terminal de *check in*;
- Sistema de planeamento de caminho até ao terminal do aeroporto;
- Interface Gráfica.

O processamento e reconhecimento dos dígitos serão efetuados através de uma RN. Foi tomada a decisão da leitura de dígitos manuscritos para aumentar a complexidade da RN e obrigar a um estudo mais aprofundado da mesma. Futuramente, pretende-se desenvolver um módulo para o processamento de imagem, para fazer o reconhecimento dos dígitos presentes num bilhete de avião real.

O sistema a desenvolver tem os seguintes requisitos funcionais:

2.2.1 Requisitos obrigatórios funcionais

- desenvolver uma RN de raiz, com a função de ler caracteres manuscritos, programada em *C#*;
- implementar um algoritmo ótimo em *Prolog* para o cálculo da rota a tomar;
- integração *C#* e *Prolog* para a comunicação entre os componentes;
- interface gráfica para interação e demonstração do sistema também em *C#*.

2.2.2 Requisitos obrigatórios não funcionais

- ajustamento da posição do dígito na imagem por centro de massa, de imagens de 20 por 20 píxeis, para imagens de 28 por 28 píxeis.
- permitir a interação com a rota ao colocar um obstáculo na mesma.

2.2.3 Requisitos Opcionais

- acrescentar módulo com reconhecimento de imagem para a leitura de uma fotografia do bilhete com o terminal *check-in*;
- desenhar uma RN para reconhecer caracteres para além de dígitos.

2.3 Representação do mapa

Nesta fase determinou-se também que cada nó está disposto numa grelha uniforme, podendo esta ser vista como um mapa de duas dimensões. Com isto tem-se um mapa *2D* como o ilustrado pela figura 2.1.

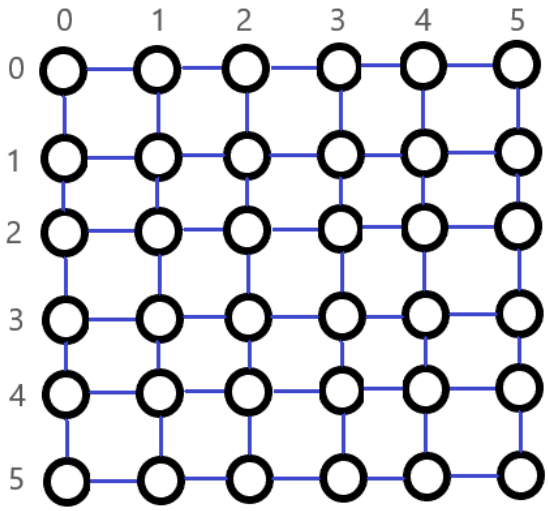

Figura 2.1: Esquema do mapa de pesquisa.

2.4 Cálculo do caminho mais curto

Escolheu-se a linguagem *Prolog* por esta constituir um sistema de dedução lógica o que facilita a programação, pois apenas se tem de escrever os factos e as regras do algoritmo, sendo a parte de inferência realizada pelo sistema.

Os três principais algoritmos considerados para realizar o cálculo da rota, foram o *Dijkstra*, o *Hill Climbing* e o *A**.

O algoritmo HC já foi estudado e utilizado anteriormente na cadeira de Lógica de Computação. Para implementar o HC é preciso representar dois aspetos: o ponto onde estamos atualmente, designado de *solução*, e o conjunto de vizinhos desta *solução*. O HC é um algoritmo iterativo que obtém os vizinhos da *solução* corrente e evolui para o melhor vizinho (o que tem melhor custo), passando este a ser próxima *solução* corrente. Apesar da sua simplicidade, este algoritmo não é ótimo dado que pode alcançar um ponto com um custo x e existirem outros pontos com um melhor custo no espaço de pesquisa que o algoritmo não conseguiu alcançar. Dado que este algoritmo não é ótimo e por já ter sido estudado, não foi considerado no projeto. Contudo, considera-se a sua implementação em trabalhos futuros.

Comparando os algoritmos restantes *Dijkstra* e *A** sabe-se que ambos têm uma fila de prioridade para o próximo vértice, que tem em conta a distância ao próximo nó, sendo que *A** tem a informação adicional de uma heurística calculada, o que lhe permite encontrar o caminho ótimo mais rapidamente que o algoritmo de *Dijkstra*. Esta heurística calculada tem de ser admissível, ou seja, tem de ser melhor ou igual ao custo ótimo. Tal heurística nunca seria pior que o melhor caminho possível.

A complexidade temporal de *Dijkstra* é $O(V + E * log_2(V))$ onde 'E' é o total de arestas dado por $maxAdjacentEdgesOfAVertex * totalVertices >= totalEdges(E)$ e 'V' o número de vértices.

A complexidade temporal de *A** é $O(b^d)$ onde 'b' é o fator de ramificação de um nó, que num contexto de um mapa de duas dimensões seria 4, e 'd' seria a profundidade de pesquisa, que depende da distância a que se encontra a solução. No fim escolheu-se o algoritmo de *A** pela rapidez em boas condições.

2.5 Arquitetura da solução

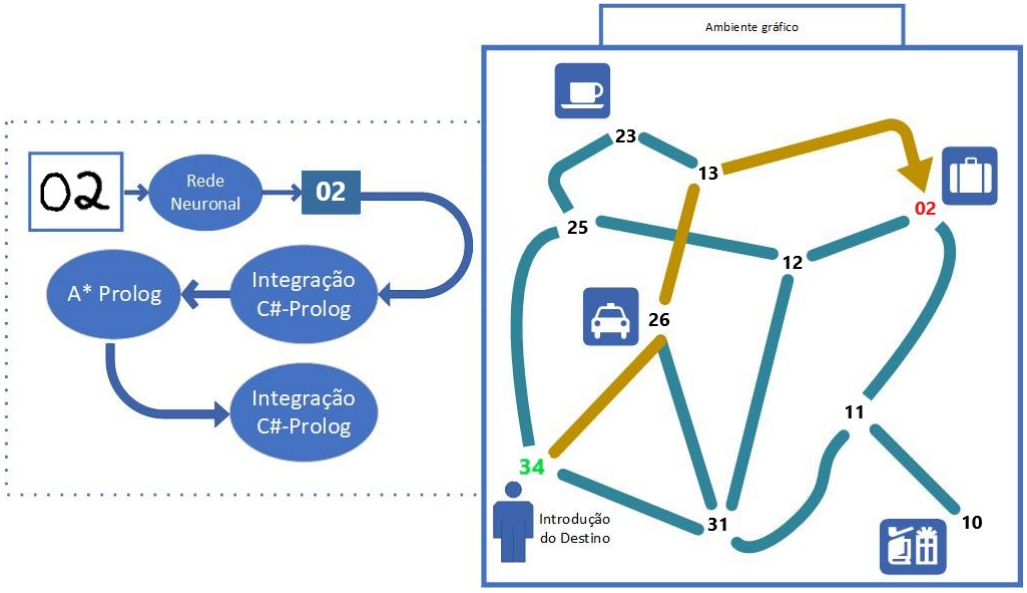

Figura 2.2: Esquema do funcionamento do sistema.

Como ilustrado na figura 2.2 o sistema consiste em 3 módulos principais: uma RN; um algoritmo de pesquisa; um ambiente gráfico. O módulo da integração nasce da necessidade de comunicação entre o *Prolog* e o C#. A RN realiza um pré processamento de imagem. Este processamento é ilustrado na figura 2.3.

No que respeita ao módulo correspondente ao cálculo da rota, decidiu-se que este seria desenvolvido em *Prolog* devido a esta linguagem ter sido aprendida durante o curso e se pretender expandir a experiência de uso da mesma. Escolheu-se então o algoritmo A* para efetuar a pesquisa devido a este ser ótimo mediante a escolha de uma heurística adequada, o que é compreensível de determinar neste contexto pois pode ser simplesmente a distância em linha reta de um nó qualquer para o nó destino. O descrito pode-se encontrar demonstrado na figura 2.4.

Em relação à interface gráfica pretende-se que esta seja desenvolvida em C# com uma representação do mapa, obstáculos que possam existir e locais para desenho de dígitos de *input* das coordenadas. Tendo em conta a diferença entre C# e *Prolog* é necessário

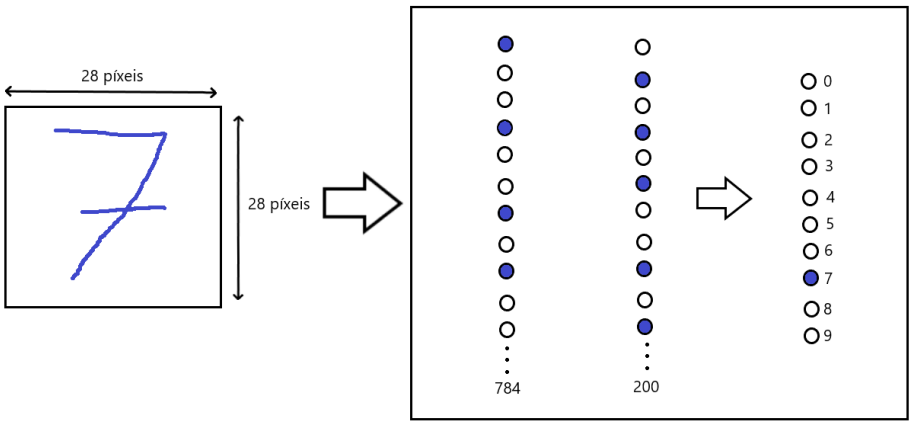

Figura 2.3: Esquema do funcionamento da Rede Neuronal, com a simulação das diversas ativações neuronais da mesma.

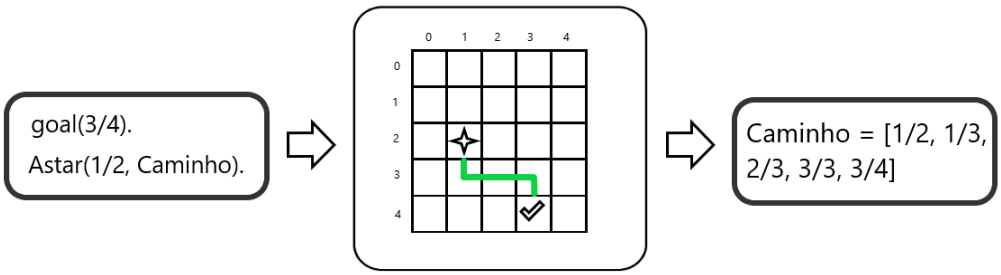

Figura 2.4: Esquema do funcionamento do algoritmo *A**.

uma ponte de comunicação entre as duas linguagens. Para o efeito é utilizada a biblioteca (Swi-prolog (2015)) providenciada pelo Swipl e mantida por *Uwe Lesta*. Esta biblioteca fornece uma interface em C# para comunicar com o Swipl, construída sobre a interface em C com o Swipl. Esta lida com conversão de tipos automática de e para *Prolog*, mapeamento de exceções e a realização de consultas ao *Prolog*. Há um *call-back* de *Prolog* para C#. Este último módulo de integração é salientado na figura 2.5.

Figura 2.5: Esquema do funcionamento da comunicação entre o *Prolog* e o *C#*

Capítulo 3

Implementação

Conteúdo

O presente capítulo descreve a implementação do SysPlan, designadamente os componentes de software que o compõem e os recursos usados, tendo em conta as tecnologias e metodologias escolhidas, descritas no capítulo 2.

3.1 Ferramentas usadas

Relativamente a ferramentas, foram usados o Visual Studio para o desenvolvimento da rede neuronal e da interface gráfica, sendo esta última sido desenvolvida com WPF. É usado também o *swi-prolog* para o desenvolvimento do algoritmo *A**.

3.2 *Data Set*

A escolha do *data set* para treino e teste da rede neuronal foi feita com base na reputação e disponibilidade, tendo sido escolhido o *data set* denominado MNIST disponível online *LeCun (2021)*. Este *data set* é a versão ajustada pelo Professor Yann LeCun do Instituto de ciências matemáticas da universidade de Nova York, onde é feito um reprocessamento das imagens para as centrar pelo centro de massa numa janela maior. O MNIST é usado geralmente como base de treino e teste para avaliar novas novas técnicas de aprendizagem e reconhecimento de padrões. Para o manuseamento do *data set* seguiu-se o exemplo de leitura das imagens do MNIST presente num artigo online *McCaffrey (2015)*. No MNIST as imagens estão formatadas com um tamanho de 28x28 píxeis e guardadas num formato próprio que consiste num ficheiro binário para guardar matrizes, onde se encontram a lista das imagens, e um segundo ficheiro com o mesmo número de entradas mas com a informação das etiquetas (*labels*), ou seja, a que dígito corresponde a imagem do ficheiro das imagens.

3.3 Rede Neuronal

Para a implementação da rede neuronal foram estudados livros e material de referência sobre o tema, tendo sido escolhidos dois livros como base da implementação *Kim (2017); Rashid (2016)*.

A base conceptual de uma RN é o neurónio. Cada entrada tem associada um valor denominado por *peso*, que é simplesmente um fator multiplicativo que vai definir a importância dessa entrada para o resultado que o neurónio irá produzir.

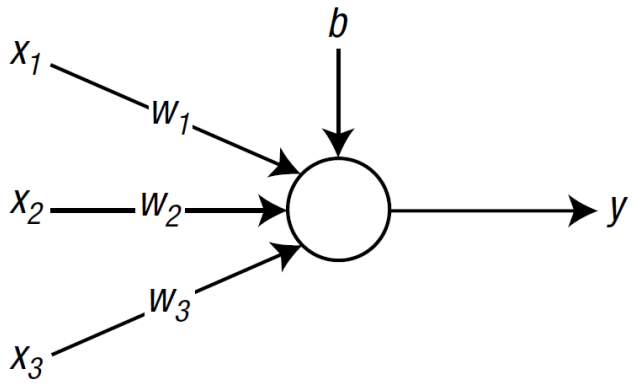

Figura 3.1: Um neurónio com três entradas.

A figura 3.1 ilustra um neurónio com três entradas e uma entrada adicional (b) designada por *bias*. A combinação das entradas do neurónio é dada pela seguinte equação (3.1):

$$v = (w_1 * x_1) + (w_2 * x_2) + (w_3 * x_3) + b \tag{3.1}$$

Com o intuito de facilitar os cálculos e a leitura das expressões são usadas matrizes. Assim, a equação anterior toma a forma da equação (3.2):

$$v = wx + b \tag{3.2}$$

onde *w* é uma matriz *(1,3)* com os pesos do neurónio e *x* uma matriz *(3,1)* com as entradas

do mesmo. Para realmente obter o resultado do neurónio, que consiste na ativação ou não do mesmo, passa-se o valor *v* por uma função de ativação φ, como demonstrado pela equação (3.3).

$$y = \varphi(v) \tag{3.3}$$

Existem diversas funções de ativação, mas aquela que é usada é das mais utilizadas para o efeito e corresponde à função *sigmoid* com a famosa curvatura em forma de '*S*'. Um exemplo da função *sigmoid* é a função logística (*logistic function*), que se ilustrada na figura 3.2.

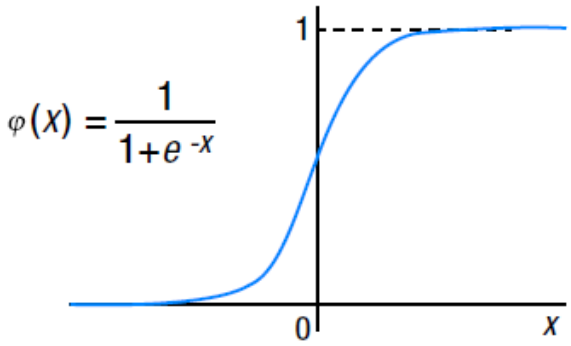

Figura 3.2: A função *sigmoid* definida, função logística.

A RN desenvolvida é designada por *shallow Multi-layer Neural Network*, e está representada na figura 3.3, o termo *shallow* é usado para distinguir entre RN multi-camada contendo apenas uma camada escondida e RN multi-camada contendo mais do que uma camada escondida.

Como regra de aprendizagem da RN usou-se a *Delta Rule*. O algoritmo de aprendizagem, denominado de algoritmo de *back propagation*, consiste nos seguintes passos. Durante o treino da rede, os valores colocados nas entradas são propagados para os neurónios das camadas seguintes, até se produzir o resultado na camada de saída. Depois a resposta obtida é usada para calcular o erro comparando com a resposta esperada. Este

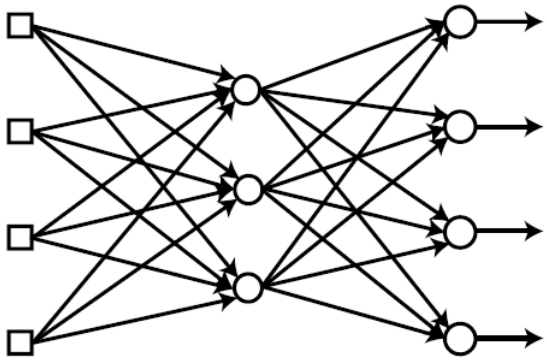

Figura 3.3: RN Multi-camada (*shallow*).

erro é propagado de trás para a frente usando a regra de aprendizagem.

A figura 3.4 demonstra a forma de obtenção do erro na última camada, a camada de *output*.

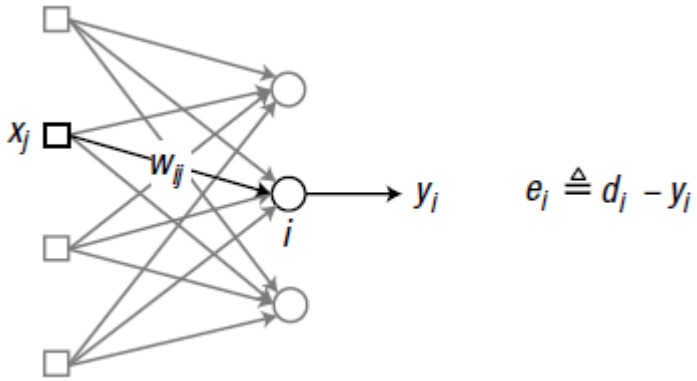

Figura 3.4: Obtenção do erro onde d_i é o resultado esperado e y_i é o resultado obtido.

A *Delta Rule*, aplicada ao exemplo da figura 3.4, pode ser resumida da seguinte maneira:

"Se um nó (neurónio) de *input* contribui para o erro do nó de *output*, o peso entre os dois nós é ajustado em proporção do valor de *input* x_j, e o erro de *output*, e_i."

Com isto em mente as expressões que especificam a *Delta Rule* são as seguintes:

$$w_{ij} = w_{ij} + \Delta w_{ij} \tag{3.4}$$

$$\Delta w_{ij} = \alpha \delta_i x_j \tag{3.5}$$

$$\delta_i = \varphi'(v_i) e_i \tag{3.6}$$

A equação (3.4) especifica o ajuste dos pesos com o termo Δw_{ij}. A equação (3.5) explica como se obtém o valor de ajuste final, multiplicando o ritmo de aprendizagem α com o valor da variação obtido do erro δ e os valores de input do neurónio *j*, x_j. Por fim a equação (3.6) demonstra como se calcula o valor de variação a partir do erro, obtendo o resultado da passagem da saída do neurónio v_i pela função derivada da função de ativação $\varphi(v_i)$, multiplicado depois pelo valor do erro e_i. A derivada da função logística pode ser expressa por $\varphi'(v) = \varphi(v)(1 - \varphi(v))$ podendo assim a função de ajuste dos pesos ser descrita como na seguinte equação (3.7):

$$w_{ij} = w_{ij} + \alpha \varphi(v_i)(1 - \varphi(v_i)) e_i x_j \tag{3.7}$$

A variável α corresponde ao *learning rate*, ou seja, o ritmo de aprendizagem. Este fator multiplicativo vai determinar a dimensão do acerto efetuado à RN por cada passo no ciclo de treino. Um valor grande de α tornaria muito difícil o processo de treino atingir uma taxa de acerto próxima do valor ideal. Por outro lado um valor muito baixo tornaria o processo de treino demasiado lento. Estas expressões funcionam bem para a camada de saída pois o erro é dado por $e_i = d_i - y_i$. Contudo para as camadas escondidas este erro tem de ser calculado de outra forma, usando, neste caso, o algoritmo de *back propagation*. As figuras 3.5 e 3.6 exemplificam a aplicação do algoritmo de *back propagation* onde o delta é calculado da forma explicada anteriormente.

Como se tem o *delta* de cada nó de *output*, aplica-se o mesmo processamento para a próxima camada à esquerda.

Resolvendo o problema de como calcular o erro das camadas escondidas, usando o algoritmo de *back propagation*, o erro do nó é definido como a soma dos *deltas* multiplicados pelos pesos da camada imediatamente à direita. Este processo pode ser expresso

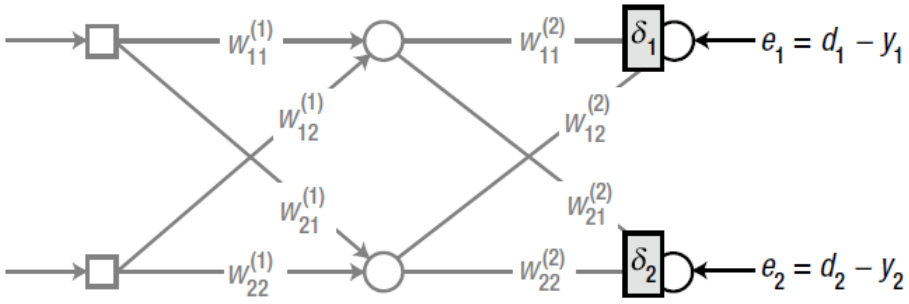

Figura 3.5: Treinar a RN usando o algoritmo de *back propagation*.

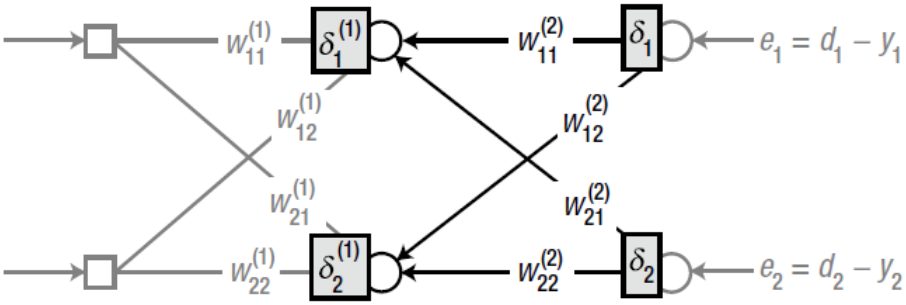

Figura 3.6: Prosseguir para a esquerda, para os nós escondidos, e calcular o delta.

pelo conjunto de equações (3.8), (3.9), (3.10) e (3.11):

$$e_1^{(1)} = w_{11}^{(2)}\delta_1 + w_{21}^{(2)}\delta_2 \tag{3.8}$$

$$\delta_1^{(1)} = \varphi'(v_1^{(1)})e_1^{(1)} \tag{3.9}$$

$$e_2^{(1)} = w_{12}^{(2)}\delta_1 + w_{22}^{(2)}\delta_2 \tag{3.10}$$

$$\delta_2^{(1)} = \varphi'(v_2^{(1)})e_2^{(1)} \tag{3.11}$$

onde v_1^1 e v_2^1 são as somas das multiplicações dos sinais de entrada com os respetivos pesos, no sentido da esquerda para a direita (*foward*). Assim a única diferença entre o algoritmo de correção dos pesos entre a camada de *output* e a as camadas subsequentes é o cálculo do erro.

Este processo na forma matricial corresponde à equação (3.12):

$$\begin{bmatrix} e_1^1 \\ e_2^1 \end{bmatrix} = W_2^T \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \tag{3.12}$$

Para os cálculos matriciais em C# foi usada a biblioteca '*MathNet.Numerics.4.15.0*' que permite efetuar todos os cálculos até aqui demonstrados.

3.3.1 Configuração e treino da Rede Neuronal

Como já mencionado anteriormente a RN tem uma camada de *input*, uma escondida e uma de *output*. A de *input* tem $28 * 28 = 784$ neurónios que correspondem ao número de píxeis de uma imagem que seria processada pela rede. A de *output* tem *10* neurónios de dimensão, que corresponde ao número de *outputs* possíveis. O número de neurónios da camada escondida é o mais crítico, pois é aquele que vai afetar mais o desempenho e principalmente a velocidade de treino da RN. Foi escolhido *200* como número de neurónios desta camada com base no estudo feito no livro *Rashid (2016)*, que determina este valor como o suficiente para atingir um nível de desempenho próximo do melhor possível para a rede. Um gráfico com a influência do número de neurónios da camada escondida no desempenho da rede pode ser observado na figura 3.7.

A escolha do *learning rate* foi ponderada tendo em conta dois fatores, como a quantidade de ciclos de treino e o desempenho obtido. Um ciclo de treino da RN completo denomina-se de *epoch*. Ao pensar no *learning rate* tem de se ter em conta o número de *epochs*, pois estes dois valores estão interligados. O gráfico com a relação entre o rácio de aprendizagem e o desempenho encontra-se na figura 3.8. Tendo em conta o mesmo poderia-se escolher um valor entre 0.1 e 0.2. A escolha no entanto vai ser influenciada pelo número de *epochs*.

Em Rashid (2016), o autor levou a cabo um estudo (figura 3.9) para analisar a influência do *learning rate*. Na figura 3.9 pode observar-seque o número de *epochs* seria

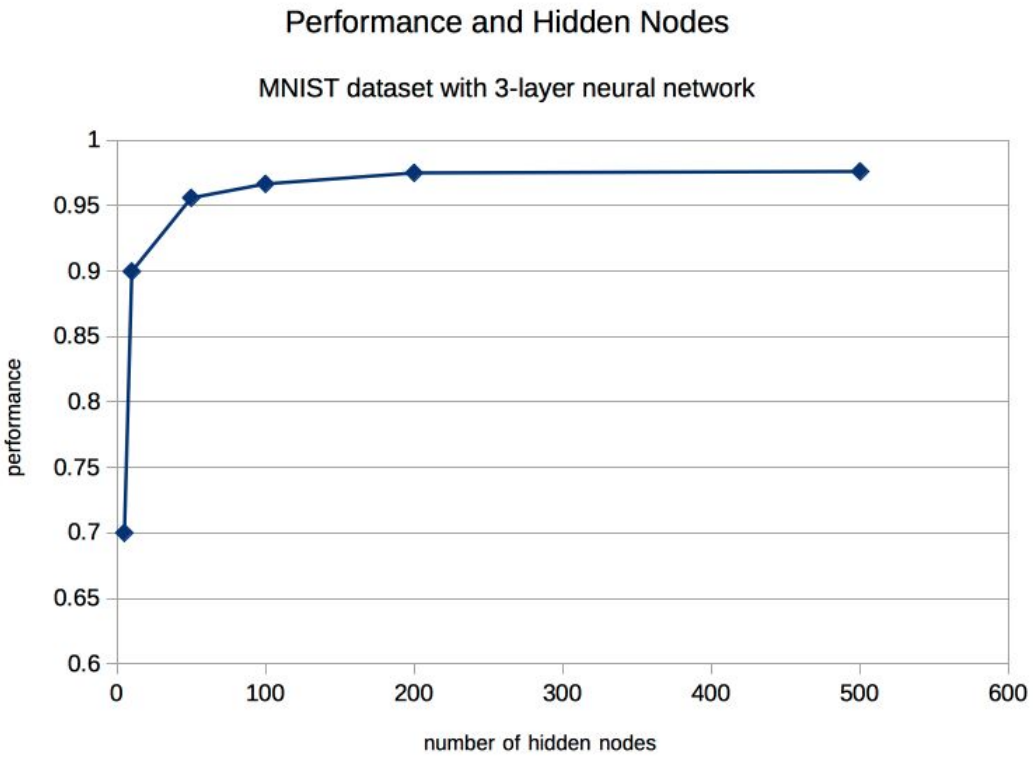

Figura 3.7: Desempenho por número de neurónios da camada escondida.

aproximadamente 5 para um rácio de aprendizagem de 0.1, por forma a obter o melhor desempenho possível.

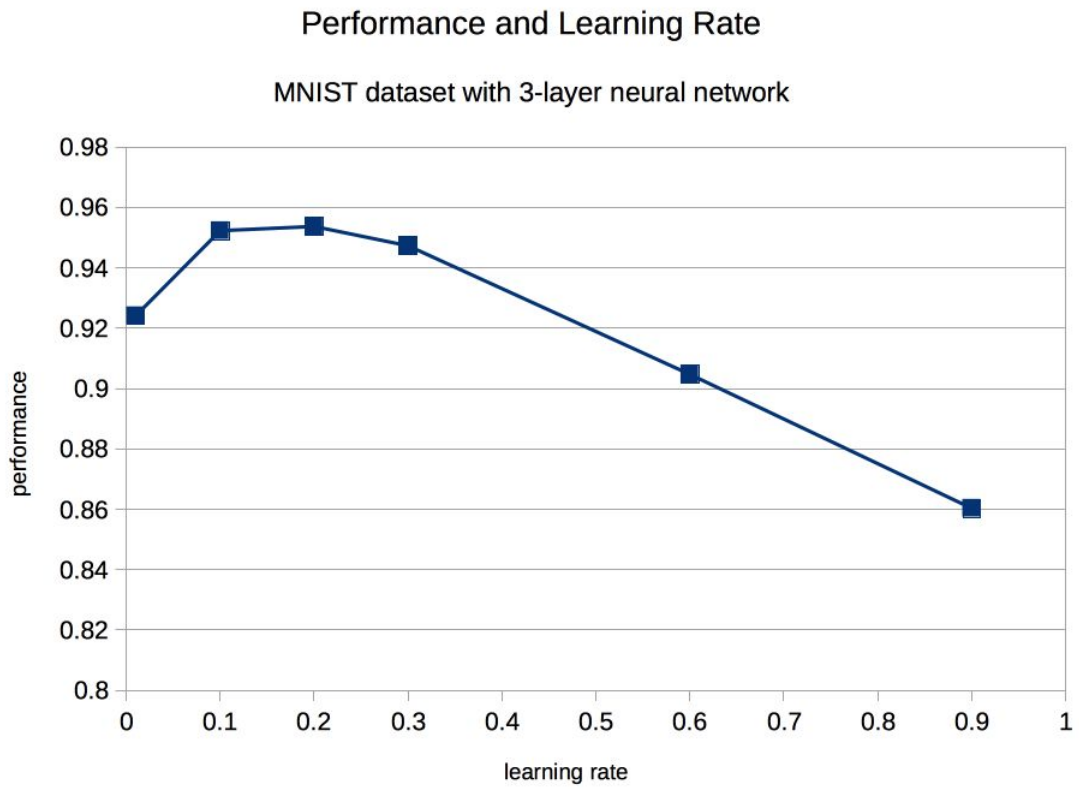

Figura 3.8: Desempenho em relação ao *learning rate*.

3.4 Algoritmo *A**

A implementação do algoritmo *A** foi baseada na implementação descrita no livro *Bratko (2012)* no capítulo 12. Este algoritmo é também denominado de '*best first*'. O nome do algoritmo advém da forma como o próximo nó a expandir é escolhido, é sempre escolhido o nó com o menor valor da função objetivo ou função de custo. A função objetivo é a soma de duas funções: uma função g que calcula o custo do caminho realizado até ao momento e uma segunda função, h, designada por função heurística, que representa o custo do caminho que ainda falta percorrer. O esquema lógico do cálculo do custo é ilustrado na figura 3.10. O código *Prolog* do algoritmo A* encontra-se no apêndice A.

A heurística utilizada para a resolução de um problema como o aqui abordado, a pesquisa do caminho mais curto até ao destino, coincide com a distância euclidiana do nó

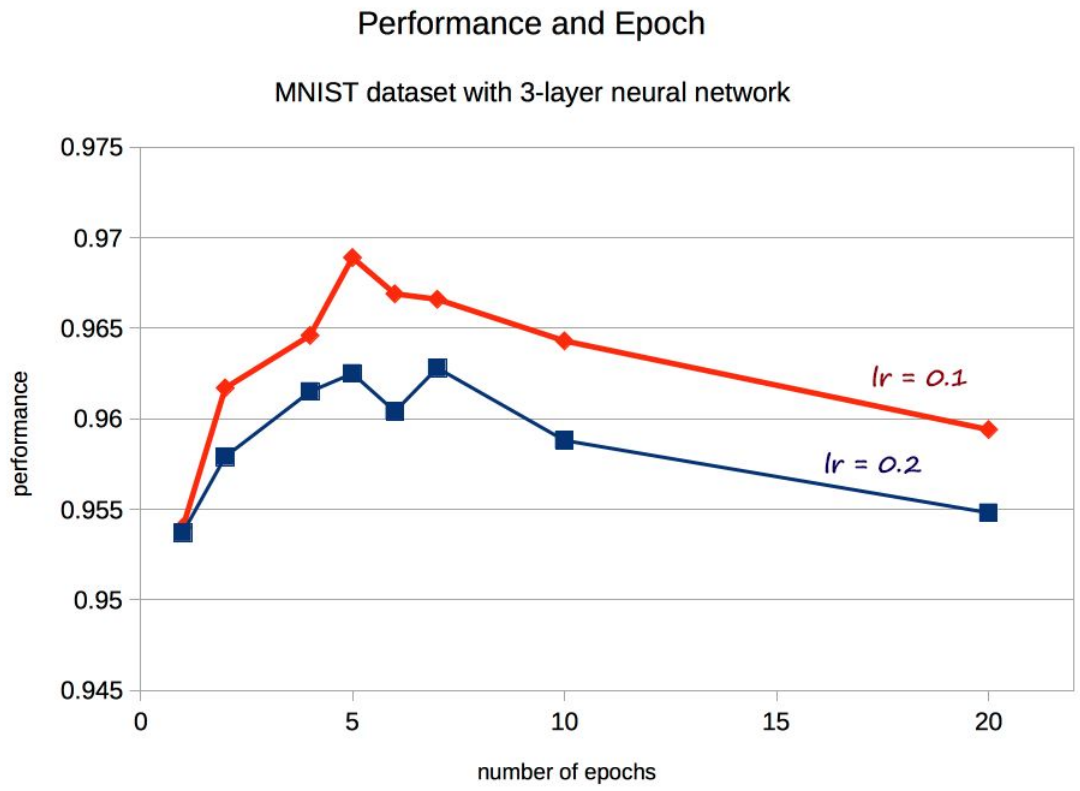

Figura 3.9: Desempenho em relação ao número de *epochs* por valor de *learning rate*(*lr*).

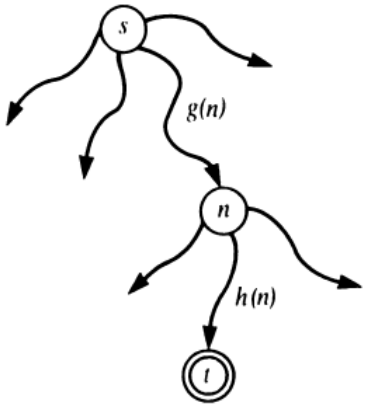

Figura 3.10: Construção da função de custo $f(n)$ que mede o custo do caminho ótimo de *s* a *t* via n: $f(n) = g(n) + h(n)$.

a considerar até ao destino.

3.5 *Prolog*

Para a utilização do algoritmo de pesquisa *A** é necessário determinar o predicado *s(A, B, D)* onde *B* é o nó sucessor de *A* e *D* a distância entre eles, e também o predicado *h(A, X)* onde para qualquer nó *A*, *X* é o valor da estimação heurística do mesmo. Para a definição dos predicados *s/3* e *h/2* é necessário conceptualizar o contexto de utilização do algoritmo de pesquisa. Decidiu-se que cada nó seria um par de coordenadas num mapa de duas dimensões. Os nós sucessores válidos são nós adjacentes do corrente que não sejam paredes ou obstáculos. Para o predicado *h/2* calcula-se a distância em linha reta ao nó objetivo através do teorema de Pitágoras. O código produzido encontra-se na figura 3.11. A distância entre nós sucessores é igual a 1 unidade.

3.6 Integração *Prolog* C#

De modo a que haja comunicação entre *Prolog* e C# é utilizada a biblioteca Swi-prolog (2015). A utilização desta biblioteca é mediada pela classe *PrologComm* desenvolvida com esse propósito. O código desta classe encontra-se no apêndice C.

3.7 Interface gráfica

Para a construção da interface gráfica é utilizado o *framework* de interface de utilizador, WPF. WPF trabalha sobre XAML que é uma linguagem declarativa baseada em XML desenvolvida pela Microsoft. Com recurso a esta esta *framework*, construiu-se uma janela para albergar as funcionalidades do projeto, o que inclui duas janelas de desenho para a obtenção dos dígitos representativos das coordenadas do mapa, bem como uma representação do mapa *2D*. Na figura 3.12 encontra-se uma fase de utilização que coincide com a colocação do robô na sua posição inicial, após desenhados os dígitos correspondentes às coordenadas desejadas.

```
:- dynamic(obstacle/1).
:- dynamic(goal/1).

walls([0/0,0/1,2/0,2/1,1/3,3/3,5/5,8/8,6/9]).

xMaxCoord(9).
yMaxCoord(9).

adjacent(X1/Y1, X2/Y1) :-
    xMaxCoord(Xmax),
    ((X2 is X1 + 1,
    X1 < Xmax)
    ;
    (X2 is X1 - 1,
    X1 > 0)).

adjacent(X1/Y1, X1/Y2) :-
    yMaxCoord(Ymax),
    ((Y2 is Y1 + 1,
    Y1 < Ymax)
    ;
    (Y2 is Y1 - 1,
    Y1 > 0)).

s(A, B, 1) :- adjacent(A, B),
    walls(Set),
    \+ member(B, Set),
    \+ obstacle(B).

h(X1/Y1, R) :-
    goal(X2/Y2),
    R is sqrt((X2 - X1)**2 + (Y2 - Y1)**2).

:- ensure_loaded(['Astar.pl']).
```

Figura 3.11: Código Prolog produzido para utilizar *A** num mapa *2D*.

Para calcular a rota para o nó destino é necessário desenhar novamente um par de coordenadas e pedir para que o sistema represente o caminho calculado na janela. Esta fase de utilização está representada na figura 3.13.

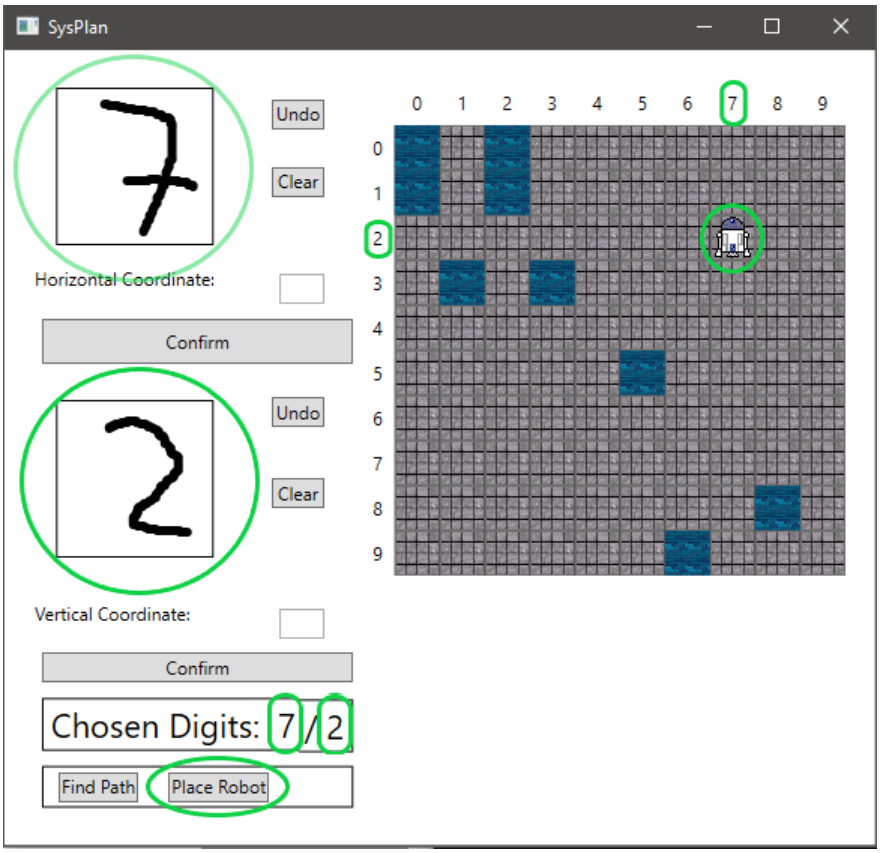

Figura 3.12: Interface Gráfica: Colocação do robô na sua posição inicial.

É possível também colocar um obstáculo no caminho do robô para forçá-lo a adaptar a rota. Para o efeito todos os quadrados que representam um espaço válido de deslocamento podem ser interagidos de modo a colocar um obstáculo. Tal comportamento é representado na figura 3.14.

Quantos mais obstáculos se encontrarem nos caminhos de menor custo existentes, mais lento será o algoritmo por este testar todos eles antes de encontrar um caminho desimpedido. Deste modo o pior cenário possível é o destino estar bloqueado na sua vizinhança deixando uma multitude de caminhos possíveis mas bloqueados. Este processamento num mapa 10 por 10 demora cerca de 5 segundos, o que é bastante relevante numa aplicação com interface de utilizador.

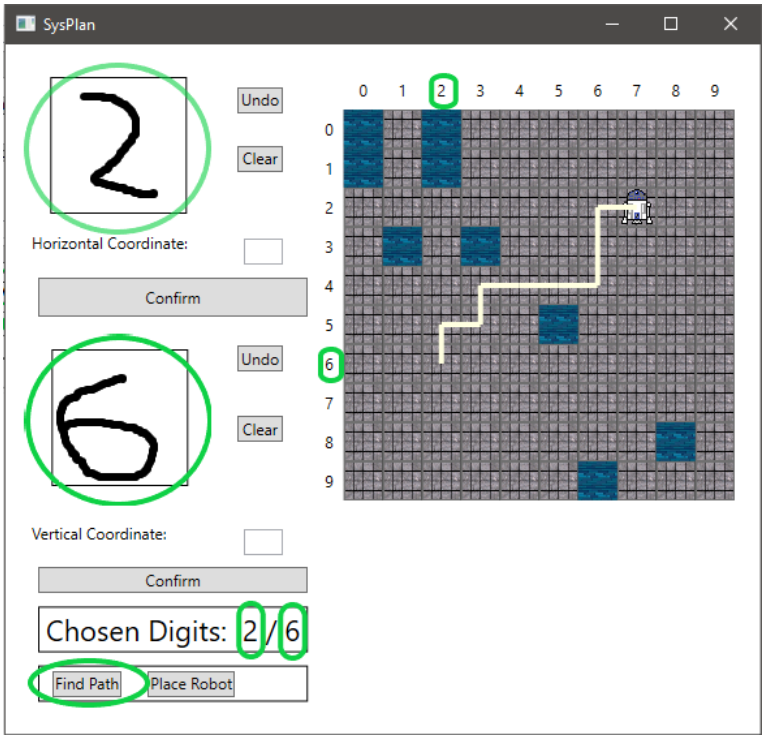

Figura 3.13: Interface Gráfica: Cálculo do caminho para o destino.

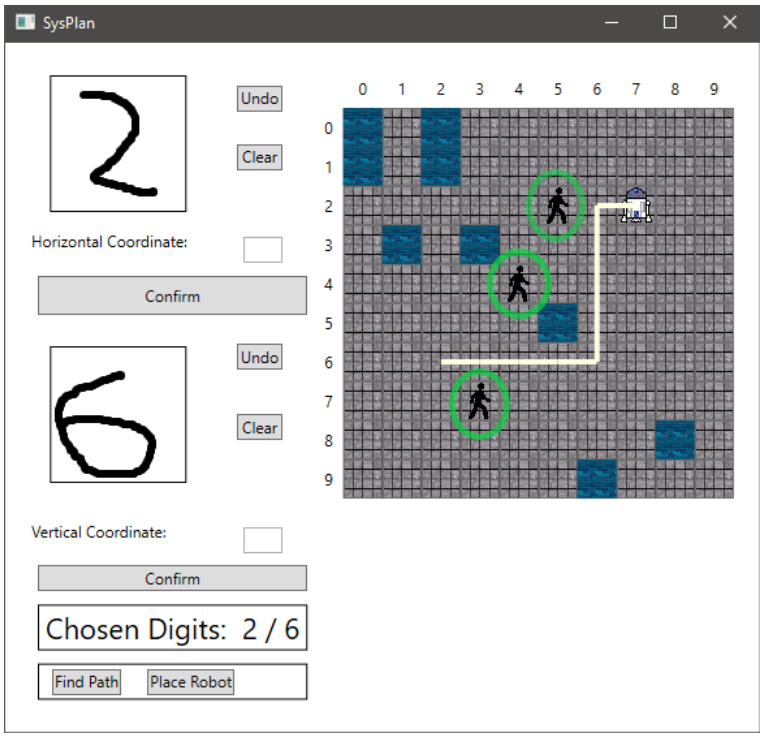

Figura 3.14: Interface Gráfica: Adaptação da rota devido ao surgimento de um obstáculo na rota calculada.

Capítulo 4

Conclusões e Trabalho Futuro

Conteúdo

Neste capítulo enumeram-se algumas conclusões do trabalho efetuado e aspetos de desenvolvimento futuro.

Durante o desenvolvimento do projeto, houve uma fase em que se teve em conta o valor de *bias* na RN. Das experiências realizadas com a inclusão do valor de *bias*, o desempenho baixou em cerca de 0,5%. Assim, a rede final não inclui o valor de *bias*.

Ao calcular o caminho, usando $A*$, para um destino impossível de alcançar bloqueado perto do mesmo, verifica-se que o programa demora cerca de 5 segundos a processar todos os caminhos possíveis antes de determinar como falha a obtenção do caminho. Apenas neste caso é que o algoritmo demora um tempo considerável.

4.1 Trabalho Futuro

Para trabalho futuro tem-se os objetivos opcionais mencionados anteriormente no relatório. Pretende-se melhorar a RN para ler caracteres para além de dígitos, e acrescentar um componente de software para o reconhecimento de imagens, de modo a permitir ler o código do destino a partir de uma fotografia.

Atualmente os nós que representam locais válidos de deslocamento são dados em grelha, limitados pelas dimensões da mesma. No futuro tornar o processo de obtenção destes nós num processo dinâmico, variando conforme as posições determinadas como válidas, por exemplo pela câmara de um robô e um módulo de reconhecimento de imagem, seria interessante do ponto de vista de uma aplicação real do sistemas.

Apêndice A

Algoritmo *A**

```
bestfirst(Start, Solution) :-
        expand([], l(Start, 0/0), 9999, _, yes, Solution).

expand(P, l(N, _),_, _, yes, [N|P]) :- goal(N).

expand(P, l(N, F/G), Bound, Tree1, Solved, Sol) :-
        F =< Bound,
        (
                bagof( M/C, ( s(N, M, C), \+ member(M, P)), Succ),
                !, % Node N has successors
                succlist( G, Succ, Ts), % Make subtrees Ts
                bestf( Ts, F1), % f-value of best successor
                expand(P, t(N, F1/G, Ts), Bound, Tree1, Solved, Sol)
                ;
                Solved = never % N has no successors - dead end
        ).
```

```
expand( P, t(N, F/G, [T | Ts]), Bound, Tree1, Solved, Sol) :-
        F =< Bound,
        bestf(Ts, BF), Bound1 = min( Bound, BF),
        expand([N | P], T, Bound1, T1, Solved1, Sol),
        continue(P, t(N, F/G, [T1 | Ts]), Bound, Tree1, Solved1,
           ↪ Solved, Sol).

expand( _, t(_, _, []), _, _, never, _) :- !.

expand(_, Tree, Bound, Tree, no, _) :-
        f(Tree, F), F > Bound.

continue(_, _, _, _, yes, yes, _).

continue(P, t(N, _/G, [T1|Ts]), Bound, Tree1, no, Solved, Solution)
   ↪ :-
        insert(T1, Ts, NTs),
        bestf(NTs, F1),
        expand(P, t(N, F1/G, NTs), Bound,Tree1,Solved,Solution).

continue( P, t(N, _/G, [_ | Ts]), Bound, Tree1, never, Solved, Sol)
   ↪ :-
        bestf( Ts, F1),
        expand( P, t(N, F1/G, Ts), Bound, Tree1, Solved, Sol).

succlist(_, [], []).
```

```
succlist(G0, [N/C | NCs], Ts) :-
        G is G0 + C, %calculo do custo incluindo o successor
        h(N, H), %heuristica para o successor
        F is G + H,
        succlist(G0, NCs, Ts1),
        insert( l(N, F/G), Ts1, Ts).

insert(T, Ts, [T | Ts]) :-
        f(T, F),
        bestf(Ts, F1),
        F =< F1, !.

insert(T, [T1 | Ts], [T1 | Ts1]) :-
        insert(T, Ts, Ts1).

f(l(_, F/_), F). %F-value of leaf

f(t(_, F/_, _), F). %f-value of tree

bestf([T | _], F) :- %best value of list of trees
        f(T, F). %garanteed by insert

bestf([], 9999). %no tree bad F value
```

Apêndice B

Guia de instalação

Repositório: '$https://github.com/RAACandeias/PS\_2021\_RicardoCandeias\_G34\_42087$'

Na diretoria do executável (SysPlan.exe) encontra-se uma biblioteca (SwiPlCs.dll), que tem de ser referenciada no GAC. Como tal para a utilização do projeto é necessário:

- Instalar o SWI-Prolog versão 6.6.6.
- Correr o *Developer Command prompt for Visual Studio* como administrador.
- Mudar a diretoria da consola para a diretoria da biblioteca SwiPlCs.dll.
- Utilizar a ferramenta *gacutil.exe* correndo um comando semelhante $\rightarrow$ "$C:/ProgramFiles(x86)/MicrosoftSDKs/Windows/v10.0A/bin/NETFX4.8Tools/gacutil" -i\ SwiPLCs.dll$

A biblioteca mencionada não é atualizada à bastante tempo, como tal, a última versão de Swipl para a qual a mesma foi atualizada foi a v6.6.6, e é por isso que se necessita da mesma.

Apêndice C

Classe PrologComm

```
public static class PrologComm
{
        public class Coords
        {
                public int x;
                public int y;
                public Coords(int x, int y)
                {
                        this.x = x;
                        this.y = y;
                }
                public override string ToString()
                {
                        return "(" + x + "," + y + ")";
                }
                public bool Equals(Coords c)
                {
                        return c.x == x && c.y == y;
```

```
        }
    }

    private static Coords[] StringToCoords(string s)
    {
        string newS = s.Trim('[', ']');
        string[] s2 = newS.Split(',');
        Coords[] c = new Coords[s2.Length];
        for (int i = 0; i < s2.Length; i++)
        {
            string[] coords = s2[i].Split('/');
            c[i] = new Coords(Int32.Parse(coords[0]), Int32
                ↪ .Parse(coords[1]));
        }
        return c;
    }

    public static void Begin()
    {
        if (!PlEngine.IsInitialized)
        {
            string[] param = { "-q" };
            PlEngine.Initialize(param);
            _ = PlQuery.PlCall("ensure_loaded('./Prolog/
                ↪ Program.pl'])");
        }
    }
```

```
public static void End()
{
        if (PlEngine.IsInitialized)
        {
                PlEngine.PlCleanup();
        }
}
public static int[] GetXYmax()
{
        if (PlEngine.IsInitialized)
        {
                int[] ret = new int[2];
                using (PlQuery q1 = new PlQuery("xMaxCoord(S)")
                    ↪ )
                {
                        PlQueryVariables v = q1.
                            ↪ SolutionVariables.First();
                        string s = v["S"].ToString();
                        ret[0] = int.Parse(s);
                }
                using (PlQuery q2 = new PlQuery("yMaxCoord(S)")
                    ↪ )
                {
                        PlQueryVariables v = q2.
                            ↪ SolutionVariables.First();
                        string s = v["S"].ToString();
```

```
                        ret[1] = int.Parse(s);
                }
                return ret;
        }
        return null;
}
public static Coords[] GetWallInfo()
{
        if (PlEngine.IsInitialized)
        {
                using (PlQuery q = new PlQuery("walls(S)"))
                {
                        PlQueryVariables v = q.SolutionVariables
                            ↪ .First();
                        string s = v["S"].ToString();
                        return StringToCoords(s);
                }
        }
        return null;
}
public static void SetGoal(Coords goal)
{
        if (PlEngine.IsInitialized)
        {
                _ = PlQuery.PlCall("retract(goal(_))");
                _ = PlQuery.PlCall($"asserta(goal({goal.x}/{
                    ↪ goal.y}))");
```

```
        }
    }
    public static void AddObstacle(Coords obstacle)
    {
        if (PlEngine.IsInitialized)
        {
            _ = PlQuery.PlCall("asserta(obstacle(" +
                ↪ obstacle.x + "/" + obstacle.y + "))");
        }
    }
    public static void RemoveObstacle(Coords obstacle)
    {
        if (PlEngine.IsInitialized)
        {
            _ = PlQuery.PlCall("retract(obstacle(" +
                ↪ obstacle.x + "/" + obstacle.y + "))");
        }
    }
    public static Coords[] GetPath(Coords from)
    {
        if (PlEngine.IsInitialized)
        {
            using (PlQuery q = new PlQuery($"bestfirst({
                ↪ from.x}/{from.y}, S)"))
            {
                try
                {
```

```
                                PlQueryVariables v = q.
                                    ↪ SolutionVariables.First();
                                string s = v["S"].ToString();
                                return StringToCoords(s);
                            }
                            catch(InvalidOperationException _)
                            {
                                return null;
                            }
                        }
                    }
                    return null;
                }
            }
```

Referências

Bratko, I. (2012). *Prolog programming for artificial intelligence by Ivan Bratko*. (4th ed.).

Chang, A. (2015). NY midtown robots allow for conversation-free hotel service. https://www.cnbc.com/2015/05/13/ny-midtown-robots-allow-for-conversation-free-hotel-service.html, last accessed on 14/06/21.

Kim, P. (2017). *MATLAB Deep Learning*.

LeCun, Y. (2021). The mnist data base. http://yann.lecun.com/exdb/mnist/, last accessed on 14/06/21.

Loughran, J. (2016). Luggage robot leo autonomously checks in airport baggage. https://eandt.theiet.org/content/articles/2016/06/luggage-robot-leo-autonomously-checks-in-airport-baggage/, last accessed on 14/06/21.

McCaffrey, J. (2015). Test run: Working with the mnist image recognition data set. https://docs.microsoft.com/en-us/archive/msdn-magazine/2014/june/test-run-working-with-the-mnist-image-recognition-data-set, last accessed on 14/06/21.

Rashid, T. (2016). *Make your own Neural Network*.

Servicerobots (2021). Service robots. https://www.servicerobots.com/, last accessed on 14/06/21.

Staff, R. (2018). How automated transportation will change our lives. https://www.roboticsbusinessreview.com/supply-chain/how-automated-transportation-will-change-our-lives/, last accessed on 14/06/21.

Swi-prolog (2015). Swiplcs.dll. https://www.swi-prolog.org/contrib/CSharp.html, last accessed on 30/07/21.

Walker, J. (2021). Does our future depend on elder care robots? https://waypointrobotics.com/blog/elder-care-robots/, last accessed on 14/06/21.