

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Integração do algoritmo TEA* com o simulador FlexSim

Francisco João Neiva Araújo Rocha Damas

Mestrado em Engenharia Eletrotécnica e de Computadores

Orientador: Pedro Gomes da Costa

Co-orientador: José Lima

18 de novembro de 2022

Resumo

Um estudo da Grand View Research afirma que até 2030 o mercado de *Automated Guided Vehicles* possa alcançar um valor de 9.38 mil milhões *USD*. Este estudo vem revelar a importância que não só os AGV, bem como os algoritmos de otimização de rotas associados aos mesmos desempenham e representarão em diferentes setores a nível mundial, as suas aplicações e características funcionais, vêm acompanhar as tendências de otimização e automatização impostas pela onda da Indústria 4.0. Com esta filosofia em mente, esta dissertação tem como principal objetivo a integração do algoritmo Time-Enhanced A*, baseado no algoritmo A* com um simulador que se aproximasse das condições reais de implementação dos AGV. Para a replicação dessas condições, foi escolhido o simulador FlexSim, pelas suas características funcionais. A integração deste simulador com o algoritmo foi o objetivo principal desta dissertação.

Para que este objetivo fosse cumprido começou-se inicialmente pelo desenvolvimento das formatações de informação, troca de mensagens e outras condições necessárias para uma integração entre os dois sistemas. De seguida foi realizado o desenvolvimento do modelo tridimensional em FlexSim, onde se realizou a construção de mapa-grafo constituído com 3 AGV, submetidos a tarefas, que representassem um sistema logístico Carga/Descarga.

A construção do modelo em FlexSim permitiu realizar a comparação do modelo em que se realizou integração, com um modelo em que se utilizou o algoritmo A* presente em FlexSim, realizaram-se testes ao nível temporal, eficiência de tarefas e espaço percorrido para as mesmas. Estes testes confirmaram o pressuposto teórico, que o algoritmo TEA*, se revela uma ferramenta bastante vantajosa no controlo de sistemas multi-agv.

Abstract

A study by Grand View Research states that by 2030 the Automated Guided Vehicles market could reach a value of USD 9.38 billion. This study reveals the importance that not only AGV's, as well as the route optimization algorithms associated with them, play and will represent in different sectors worldwide, their applications and functional characteristics, come to accompany the optimization and automation trends imposed by the wave of Industry 4.0. With this philosophy in mind, this dissertation has as main objective the integration of the Time-Enhanced A* algorithm, based on the A* algorithm, with a simulator that approximates the real conditions of implementation of AGV's. The FlexSim simulator was used for this purpose, this integration between the algorithm and the simulator was the main focus of this dissertation.

In order to achieve this objective, initially the development of information formats, message exchange and other conditions necessary for an integration between the two systems began. Then the development of the three-dimensional model in FlexSim was carried out, where the construction of a graph-map was carried out with 3 AGV's, submitted to tasks, which represented a loading/unloading logistic system.

The construction of the model in FlexSim made it possible to compare the model in which the integration was carried out, with a model in which the A* algorithm present in FlexSim was used, tests were carried out at the temporal level, task efficiency and space traveled to the same. These tests confirmed the theoretical assumption that the TEA* algorithm proves to be a very advantageous tool in the control of multi-agv systems.

Agradecimentos

É neste termino do Mestrado de Engenharia Eletrotécnica e Computadores, marcado pela realização desta dissertação, que quero começar por agradecer, ao Prof. Pedro José Gomes da Costa pelo esforço, disponibilidade e compreensão, mesmo quando eu duvidada da minha capacidade de cumprir os objetivos nos prazos propostos, as palavras "tens de conseguir" nas reuniões semanais deram-me a motivação necessária. Quero agradecer também ao meu Co-Orientador Prof. José Lima pela sua disponibilidade e apreciação ao longo do processo.

Gostaria também de agradecer ao Eng. Diogo Matos, Eng. Romão Santos e ao Eng. Paulo Rebelo, pela disponibilidade e pelo apoio nas diferentes fases do projeto em que me senti mais perdido, a sua ajuda foi essencial para a realização desta dissertação.

Quero agradecer ao Miguel Teixeira, João Francisco Rocha, Eng. João Afonso e Eng. Manuel Lessa Pereira, o percurso tornou se mais fácil com a sua companhia, amizade e aconselhamento. Aos meus colegas das noites de estudo Vítor Navega, André Moreira, José Lino e João Burmester Campos, pela motivação e companheirismo ao longo de todas as noites de estudo, sem eles o percurso tinha sido bastante mais difícil e solitário.

Quero também agradecer especialmente ao meu amigo, Eng. José Araújo, pela inspiração, companheirismo e conselhos que me passou ao longo de todo o meu percurso académico.

Gostaria também de agradecer a toda a minha família, pelo apoio e motivação que deram.

Por fim quero acima de tudo agradecer às duas pessoas pessoas mais relevantes e importantes para mim, os meus pais. À minha Mãe, obrigado pelo esforço que dedicou em toda a minha educação, pelo carinho e pelo afeto. Ao meu Pai, por sempre demonstrar o seu amor nas ações mais simples na vida, obrigado por me tornar o homem que sou hoje, por me inspirar com a sua história de vida e pelos valores de trabalho e profissionalismo que me passou.

Um obrigado nunca será suficiente para conseguir agradecer, o que me deram. Tudo o que alcancei e o que sou, é deles.

Francisco Damas

*“Sometimes I’ll start a sentence, and I don’t even know where it’s going.
I just hope I find it along the way.”*

Michael Scott

Conteúdo

1	Introdução	1
1.1	Contextualização	1
1.2	Motivação	1
1.3	Objetivos	2
2	Revisão de Literatura	3
2.1	Automated Guided Vehicles	3
2.1.1	Vantagens	3
2.1.2	Categorias de AGV	4
2.1.3	Métodos de Navegação	6
2.1.4	Configuração das Rodas	8
2.2	Arquitetura de Controlo	11
2.2.1	Arquitetura Centralizada	11
2.2.2	Arquitetura Descentralizada	12
2.3	Planeamento de Trajetória	12
2.3.1	Espaço de Configuração	12
2.3.2	Métodos de Planeamento de Trajetória	12
2.3.3	Roadmap	13
2.3.4	Decomposição em Células	15
2.4	Algoritmo de Pesquisa por grafos	18
2.4.1	Algoritmos sem Informação	18
2.4.2	Algoritmos com Heurísticas	20
2.5	Software de Simulação	25
2.5.1	Comparação de simuladores	25
2.5.2	Software - FlexSim	28
2.5.3	Conclusão	31
3	Metodologia	33
3.1	Construção do Modelo Tridimensional	33
3.1.1	Ligações e Nós	34
3.2	Comunicação com Algoritmo	35
3.2.1	Comunicações Off-line	37
3.2.2	Mensagens Online	41
3.2.3	Process Flow	44
3.2.4	Conclusão	47

4	Caso de Estudo	49
4.1	Modelo TEA*	49
4.2	Análise do modelo - 1 Tarefa de <i>Load</i>	49
4.3	Análise do modelo - 2 Tarefas de <i>Load</i>	53
4.3.1	<i>Load</i> - 2 CP Diferentes	53
4.3.2	<i>Load</i> de 3 CP's	56
4.3.3	<i>Load</i> e <i>Unload</i> - Control Points	58
4.4	Construção do modelo A*	60
4.4.1	Alocação de Tarefas	61
4.5	Comparação de Resultados	68
5	Conclusões e Trabalho Futuro	69
5.1	Conclusões do Trabalho Desenvolvido	69
5.2	Trabalho Futuro	70
A		71
A.1	DataGraph	71
A.2	DataMission	72
A.3	TaskAllocationRequest	74
A.4	Task Allocation Response	76
A.5	Travel - Process Flow	77
A.6	Decide Block - Process Flow	78
A.7	Decide II	79
A.8	Last Trip - Process Flow	79
A.9	Robot Unavailable - Process Flow	80
A.10	Robot Available	80
A.11	DataMissions.JSON - teste	82
	Referências	85

Lista de Figuras

2.1	Exemplo de um Mini-AGV- KATE (desenvolvido pela Gotting) [1]	4
2.2	Exemplo de dois AGV <i>Forklift</i> com produção em série (Esquerda EK, Direita DPM)[1]	5
2.3	Exemplo de um <i>Tugger</i> construído pela DPM[1]	6
2.4	Desenho representativo do sistema de navegação com recurso a condutores elétricos[1]	7
2.5	Desenho representativo do sistema de navegação por faixas com recurso a sensores óticos (direita) ou indutivos (esquerda) [1]	7
2.6	Desenho representativo do sistema de navegação por triangulação laser [1]	8
2.7	Desenho representativo de um sistema de marcadores em matriz[1]	8
2.8	Configuração de rodas em triciclo [2]	9
2.9	Configuração de rodas em Ackerman[2]	10
2.10	Configuração de rodas diferencial [2]	10
2.11	Configuração de rodas omnidirecionais [3]	11
2.12	Diferentes classificações de arquitetura (a) Descentralizada (b) Centralizada (c) Híbrida [4]	11
2.13	Métodos de Planeamento de Trajetórias	13
2.14	Exemplo de um Visibility Graph [5]	14
2.15	Exemplo de um Diagrama Voronoi[6]	15
2.16	Exemplo de uma decomposição em polígonos convexos[7]	16
2.17	Exemplo de uma decomposição em trapézios e o grafo associado ao mesmo [5]	16
2.18	Representação gráfica da decomposição Quadtree[5]	17
2.19	Representação gráfica da decomposição em células fixas[6]	18
2.20	Representação gráfica do algoritmo de pesquisa DFS (<i>Depth First Search Algorithm</i>) - a) laranja: nós visitados b) azul: nós que faltam visitar c) cinzento: nós já visitados em que se realizou <i>backtracking</i>	19
2.21	Representação gráfica do algoritmo de pesquisa BFS - a) laranja: nós na lista a visitar b) azul: nós que estão por associar c) cinzento: nós já visitados d) vermelho: nó que está a ser analisado	20
2.22	Representação esquemática do custo das células no algoritmo A*	21
2.23	Representação esquemática das camadas temporais (esquerda) e representação do AGV na célula juntamente com as suas células vizinhas na camada temporal seguinte (direita)	22
2.24	Representação das curvas de bezier numa parte de um mapa [8]	23
2.25	Exemplo de uma linha de produção em 2D no simulador Simio	26
2.26	Exemplo de uma linha de montagem simulada em Simul8	26
2.27	Ferramenta de Process Flow do FlexSim	27
2.28	Exemplo de um loop logístico simulado em FlexSim	28
2.29	Recursos fixos do FlexSim	29

2.30	Ferramentas Task Executer	30
2.31	Ferramentas AGV	30
2.32	Ferramenta Process Flow	31
3.1	Mapa Snapshoht em RVIZ	34
3.2	Processo de construção dos <i>paths</i> e CP	34
3.3	Modelo Tridimensional construído em FlexSim	35
3.4	Diagrama de Comunicação Servidor(TEA*) - Cliente(Simulador)	36
3.5	Treenode - Label do Control Point 1	38
3.6	Treenode - Label do Path 9	39
3.7	Exemplo de Tabela com a informação das Tarefas	41
3.8	Modelo de PF de controlo do sistema multi-agv	44
3.9	Propriedades do Event - Trigger Source	45
3.10	Propriedades da Schedule Source	45
3.11	Propriedades do primeiro Assign Label	46
3.12	Propriedades do segundo Assign Label	46
3.13	Ciclo de leitura do <i>Array</i> do AGV em PF	46
3.14	Label do agv2 - Travel de CP59->CP61	47
4.1	Pedido de alocação de tarefa para o CP55	50
4.2	Resposta ao pedido de alocação CP55	50
4.3	Distância percorrida pelo AGV	50
4.4	Distância percorrida p/ segundo pelo AGV2	51
4.5	Frame de demonstração da simulação	51
4.6	Pedido de alocação de tarefa para o CP61	52
4.7	Resposta ao pedido de alocação CP61	52
4.8	Distância pelo AGV3 para a deslocação até CP61	52
4.9	Distância percorrida p/segundo pelo AGV3 para a deslocação até CP61	52
4.10	Chegada do AGV2 ao CP61	53
4.11	Pedido de alocação de tarefa para o CP61	54
4.12	Pedido de alocação de tarefa para o CP57	54
4.13	Resposta ao pedido de alocação de tarefa para o CP61	54
4.14	Pedido de alocação de tarefa para o CP57	54
4.15	Distância percorrida pelo AGV3 e AGV1	54
4.16	Distância percorrida por segundo pelo AGV3 e AGV1	55
4.17	Chegada do AGV2 ao CP55	55
4.18	Pedido de Alocação de Tarefa para a TaskID=0	56
4.19	Pedido de Alocação de Tarefa para a TaskID=1	56
4.20	Pedido de Alocação de Tarefa para a TaskID=2	56
4.21	Resposta ao Pedido de Alocação de Tarefa para a TaskID=0	56
4.22	Resposta ao Pedido de Alocação de Tarefa para a TaskID=1	56
4.23	Resposta ao Pedido de Alocação de Tarefa para a TaskID=2	56
4.24	Distância percorrida pelos 3 AGV	57
4.25	Distância percorrida por segundo pelos AGV	57
4.26	Chegada dos AGV 3 ao CP57	58
4.27	Pedido de alocação	58
4.28	Resposta ao pedido de alocação	58
4.29	Envio da mensagem available para o algoritmo	59
4.30	Resposta à mensagem available	59

4.31	Chegada do AGV2 ao CP61	59
4.32	Modelo Tridimensional com recurso ao Algoritmo A*	60
4.33	Propriedades do AGV3	61
4.34	Simulação do modelo com A* com alocação de CPs - Deadlock	62
4.35	Simulação do modelo com A* com alocação de CP61	63
4.36	Simulação do modelo com A* com alocação de CPs - Deadlock	63
4.37	Simulação do modelo com A* com alocação de CPs - Deadlock	64
4.38	Simulação do modelo com A* com alocação de CPs - Deadlock	64
4.39	Resultados da simulação do modelo com A* com alocação de CPs - Deadlock . .	65
4.40	Simulação do modelo com A* com alocação de CPs - Deadlock	66
4.41	Simulação do modelo com A* - Distância percorrida pelos AGVs	66
4.42	Simulação do modelo com A* com recurso de Sink/Source - Deadlock	67
4.43	Simulação do modelo com A* com recurso de Sink/Source - Distância percorrida pelos AGVs	67

Abreviaturas e Símbolos

AGV	<i>Automated Guided Vehicles</i>
CAGR	<i>Compound Annual Growth Rate</i>
3D	<i>Three Dimensional</i>
USD	<i>United States Dollar</i>
DV	<i>Diagrama de Voronoi</i>
DFS	<i>Depth First Search Algorithm</i>
BFS	<i>Breadth First Search Algorithm</i>
EVB	<i>Electric Vehicle Batteries</i>
GPS	<i>Global Positioning System</i>
A*	<i>A-Star</i>
CP	<i>Control Point</i>
PF	<i>Process Flow</i>
ID	<i>Identification</i>
IP	<i>Identification Protocol</i>
TEA*	<i>Time-Enhanced A-star</i>

Capítulo 1

Introdução

1.1 Contextualização

Num mundo cada vez mais automatizado, com uma indústria cada vez mais focada na otimização das diferentes fases dos seus processos, a utilização de robôs móveis que permitam transportar bens de um ponto inicial a um ponto destino tornou-se ao longo dos anos algo de enorme interesse para diferentes indústrias. O primeiro *Automated Guided Vehicle* foi implementado como um trator com reboque por Barrett-Cravens of Northbrook, Illinois no início de 1950. Desde essa altura que os AGV têm sido atualizados e hoje em dia constituem uma ferramenta de grande valor para a indústria, um estudo de mercado publicado pela Grand View Research, avaliou a evolução que a utilização dos *Automated Guided Vehicle* (AGV) a nível industrial pode ter ao longo dos próximos anos e estima-se que até 2030 a utilização de AGV possa ter uma CAGR de cerca de 10,2% [9]. Atualmente o mercado de veículos autónomos está avaliado em cerca de 4,3 mil milhões de USD. Consegue-se então perceber que os AGV são de grande valor nos sistemas logísticos de diferentes indústrias, estes robôs promovidos pelo paradigma atual da Indústria 4.0 permitem reduzir custos, minimizar os erros associados a recursos humanos e otimizar os tempos de processamento. Com estes objetivos em mente torna-se difícil abordar o tema de AGVs sem abordar a forma como as rotas dos mesmos são planeadas de forma a otimizar diferentes parâmetros, tempo, distância percorrida, segurança, entre outros. Este planeamento de caminhos é feito por algoritmos que utilizam as diferentes localizações dos robôs.

1.2 Motivação

Para a implementação de um conjunto de AGV e o algoritmo que está associado ao seu planeamento de rotas, tornou-se aqui como fator motivador a criação de um ambiente simulado que permiti-se avaliar o comportamento dos AGV.

1.3 Objetivos

Esta dissertação tem como objetivo principal a integração do algoritmo Time-Enhanced A* num ambiente de simulação em FlexSim, reforçando a validação do algoritmo num ambiente próximo da realidade e a comparação do Time-Enhanced A* com um algoritmo A*, presente como ferramenta no *software* de simulação.

Capítulo 2

Revisão de Literatura

O capítulo de revisão de literatura tem como objetivo ser um sumário compreensivo de alguns dos estudos desenvolvidas ou soluções presentes no mercado, para as diferentes áreas que o projeto de dissertação possa tocar. Inicialmente são analisados os AGV em diferentes frentes, as vantagens dos mesmos quando comparados com soluções mais tradicionais da indústria, métodos de navegação e por fim as diferentes categorias de AGV. Ao nível do controlo dos AGV's analisa se também os tipos de arquiteturas de controlo que existem.

Nos capítulos seguintes, são analisados alguns estudos e métodos nas áreas mais referentes ao planeamento de trajetórias, bem como algoritmos de pesquisa. Por fim são analisadas algumas das soluções existentes do mercado ao nível de *softwares* de simulação, e o posicionamento comparativo do FlexSim, com esses outros *softwares*.

2.1 Automated Guided Vehicles

Nos dias de hoje os robôs móveis industriais, AGV, têm-se tornado uma ferramenta de grande valor na logística interna de grandes indústrias. Desde o seu primeiro desenvolvimento em 1953 no EUA, implementado por Barrett-Cravens of Northbrook, Illinois (atualmente Savant Automation Inc., Michigan), que possuía apenas sensores mais rudimentares de choque e atuadores de emergência, os AGV têm vindo a acompanhar a evolução tecnológica [1], sendo hoje possível ter grandes frotas de AGV a funcionar em paralelo, com algoritmos e tecnologias bastante diferentes umas das outras. Atualmente as principais áreas de aplicação como referido em [1], focam-se principalmente na logística interna da indústria, focando-se na organização, controlo e otimização de fluxos internos dos bens materiais. Os AGV utilizados, são maioritariamente elétricos utilizando como fonte de energia baterias, que variam consoante a aplicação dos mesmos [1].

2.1.1 Vantagens

A utilização de AGV na indústria confere na sua utilização um grande número de vantagens comparativamente a métodos que utilizem recursos humanos para transporte. Na literatura [1] são referidas algumas das vantagens de utilização dos AGV entre elas:

1. melhor organização da informação de fluxo interno de materiais que conduz a um aumento da produtividade, favorecido pela transparência no processo logístico;
2. as fases de transporte são calculadas e planeadas de forma bastante precisa;
3. Maior disponibilidade e fiabilidade (consoante o número de robôs disponíveis)
4. menor uso de recursos humanos no transporte;
5. minimização de erro humano, como danificação de material;

2.1.2 Categorias de AGV

Os AGV podem ser divididos em diferentes categorias consoante diferentes cargas que os mesmos transportem e as características de construção que os mesmos possuem, na tabela 3.6 da literatura [1] são apresentados os diferentes tipos de AGV. De entre as diferentes categorias dos AGV enumerados, destacam se os Mini-AGV, "Forklift" AGV e *Tugger* AGV, estes três AGV distinguem se essencialmente pela carga que transportam.

Mini-AGV Os Mini-AGV são usualmente utilizados em indústrias que recorrem a grandes frotas de AGV inteligentes e com bastante flexibilidade, capazes de efetuar transporte de cargas mais leves, comparativamente com outros AGV, de forma rápida. Na figura 2.1 está representado um exemplo de um Mini-AGV.



Figura 2.1: Exemplo de um Mini-AGV- KATE (desenvolvido pela Gotting) [1]

Forklift AGV Os Forklift AGV são semelhantes ao formato convencional de uma empilhadora, sendo que possuem a característica de não ser necessário um recurso humano para a sua utilização. A utilização desta categoria de AGV na logística, esta essencialmente centrada no transporte de cargas em paletes compatíveis com empilhadoras e a sua aplicação varia entre o transporte simples entre dois pontos ou o transporte de cargas mais complexas com diferentes pontos destino. Dentro desta categoria de AGV pode ainda ser dividida em duas categorias, AGV de série que utilizam empilhadoras existentes no mercado e automatizam as mesmas consoante as necessidades do cliente ou AGV especialmente desenhados, a escolha entre estas duas categorias varia consoante as exigências das necessidades de aplicação em que os mesmos são inseridos, salientado que os AGV de série apresentam custos mais baixos, maior fiabilidade e uma maior disponibilidade de peças de substituição. Na figura 2.2 estão representados dois *Forklift* AGV com produção em Série.



Figura 2.2: Exemplo de dois AGV *Forklift* com produção em série (Esquerda EK, Direita DPM)[1]

Tugger AGV Os *Tugger*, caracterizam se por serem AGV de transporte de múltiplos reboques. Dentro desta categoria de AGV os mesmos poderiam ser divididos eles também em série ou especialmente desenhados, todavia este tipo de AGV é menos comum no mercado do que as empilhadoras sendo que não é feita essa divisão categórica. Usualmente utilizados na indústria automóvel este tipo de robô, caracteriza se por ser capaz de transportar um ou mais reboques, ao longo de uma rota pré-determinada, com pontos de descolamento dessas cargas ao longo da sua rota, uma das limitações deste tipo de AGV é o seu baixo raio de viragem. Na figura 2.3 está representado um exemplo de um *Tugger*, acoplado com 3 reboques ao longo de uma rota pré determinada.



Figura 2.3: Exemplo de um *Tugger* construído pela DPM[1]

2.1.3 Métodos de Navegação

Os métodos de navegação de AGV podem ser divididos em dois tipos, fixos ou dinâmicos. A implementação de cada um dos tipos de navegação vai depender de diversos fatores, como os orçamentos que se têm para o projeto, flexibilidade que o mesmo necessita e o tipo de AGV que se pretende implementar.

2.1.3.1 Trajetórias Fixas

Nas trajetórias fixas existem dois métodos [10] que podem ser utilizados para navegar os AGV:

Sistema Filoguiado: Num sistema de navegação filoguiado [10], são colocadas condutores elétricos, embutidos ou sobre o chão, que vão definir as trajetórias possíveis dos robôs. Estes condutores elétricos criam um campo elétrico que é detetado pelo robô, este método de navegação apesar da sua simplicidade apresenta uma baixa flexibilidade no caso de se desejar alterar as trajetórias que os robôs podem tomar, deste modo este tipo de navegação não será favorável em aplicações que necessitem de alterar o layout de uma forma frequente. Na figura 2.4 está representada a forma como é realizada a navegação por sistema filoguiado.

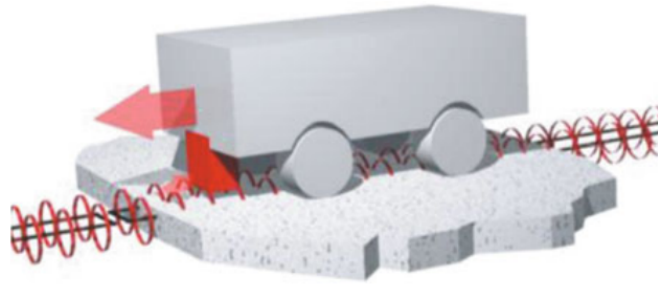


Figura 2.4: Desenho representativo do sistema de navegação com recurso a condutores elétricos[1]

Sistema de Faixas: Um sistema de faixas, pode ser alcançado de duas formas, ou pela colocação de fitas magnéticas, ou pela colocação de linhas pintadas, dependendo do tipo de sensor, indutivo ou ótico, respetivamente. O AGV vai seguir as trajetórias definidas por essas faixas. Comparativamente ao Sistema Filoguiado, o sistema de faixas apresenta uma maior flexibilidade, na medida em que a alteração das trajetórias, tem um custo menor. Por outro lado dada a menor resistência mecânica deste tipo de método, as faixas estão mais propensas a serem desgastadas o que pode dar origem a falha de identificação das mesmas por parte do robô. Na Figura 2.5 estão representados os dois métodos de navegação por sistema de faixas, ótico e magnético.

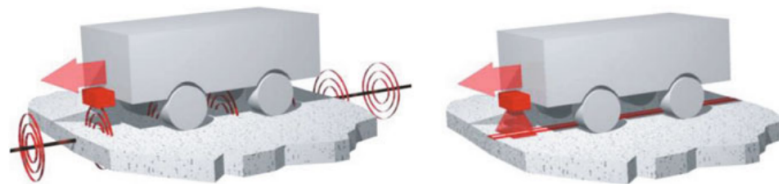


Figura 2.5: Desenho representativo do sistema de navegação por faixas com recurso a sensores óticos (direita) ou indutivos (esquerda) [1]

2.1.3.2 Trajetórias Dinâmicas

Um sistema de navegação por trajetórias dinâmicas, pela sua natureza de funcionamento apresenta uma maior flexibilidade quando comparado com trajetórias fixas. As trajetórias dinâmicas podem ser alcançadas por Triangulação Laser, Sistema de Marcadores ou GPS.

Triangulação Laser: A triangulação laser como representado em 2.6 é caracterizada como sendo a solução com maior flexibilidade porém é também a mais cara. Neste método [10][1], os refletores são colocados a uma altura superior do nível dos operadores, o laser montado no AGV, realiza um scan de forma a medir a posição de no mínimo 3 refletores fixos e compara com os valores inseridos na configuração, deste modo é alcançada a posição e orientação atual do robô ao longo da rota. A área é mapeada e guardada na memória do AGV [10].

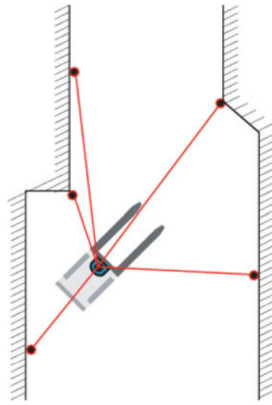


Figura 2.6: Desenho representativo do sistema de navegação por triangulação laser [1]

Sistema de Marcadores: Neste tipo de navegação dinâmica são colocados vários marcadores magnéticos no chão, este posicionamento pode variar entre, uma matriz (representado na figura 2.7), que oferece uma maior flexibilidade de trajetórias, ou em série, este posicionamento vai depender do objetivo e trajetória pretendida para o AGV. A localização destes marcadores magnéticos é guardada previamente numa fase de configuração garantindo que quando o robô passar por estes marcadores, fique a ser conhecida a sua posição. Este tipo de navegação, esta sujeita a erros que levam a que o AGV não encontre o próximo marcador para onde se deve dirigir pelo que é usualmente associado um giroscópio ao AGV de forma a analisar mais detalhadamente as variações de direção do AGV minimizando assim desvios de trajetória por parte do mesmo [10].

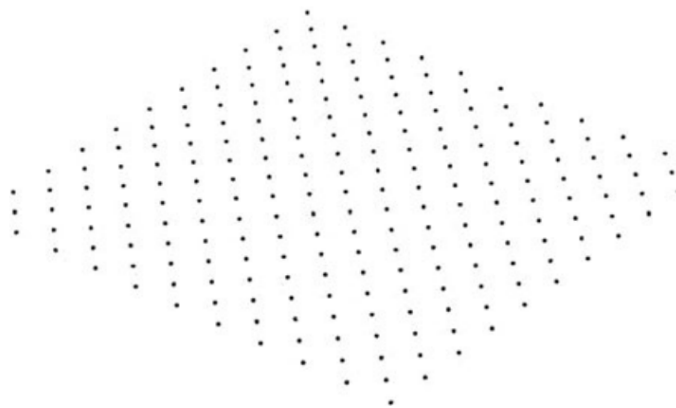


Figura 2.7: Desenho representativo de um sistema de marcadores em matriz[1]

2.1.4 Configuração das Rodas

A configuração das rodas nos AGV, tem um impacto muito grande no comportamento de um sistema de transporte autónomo. A configuração diz respeito ao número de rodas, o seu posicio-

namento bem com o tipo de atuação das mesmas [1], esta configuração será tão melhor, quanto melhor for a eficiência dos movimentos do robô. Esta capacidade de o robô realizar alterações de direção num menor espaço de tempo e numa área inferior. Dentro das diferentes configurações de rodas para os AGV destacam-se , a diferencial, triciclo, Ackerman e omnidirecional, que serão abordadas de seguida. A escolha do tipo de AGV para um determinado sistema, dependerá da área disponível para o AGV se poder movimentar bem como o orçamento disponível para o AGV, esta escolha tem também influência ao nível do algoritmo, já que o tempo de alteração de direção varia para diferentes configurações, pelo que exigirá uma adaptação do algoritmo.

Triciclo Este tipo de veículo caracteriza-se por possuir uma roda frontal de tração e direcional e duas rodas no eixo traseiro, esta disposição garante a este tipo de veículo a capacidade de rodar sobre si próprio, todavia apresenta duas limitações, a perda de tração na mudança de direções, como referido em [2] e o facto de a movimentação preferencial ser para a frente [1]. Na figura 2.8 está representada a típica configuração de um AGV em triciclo.

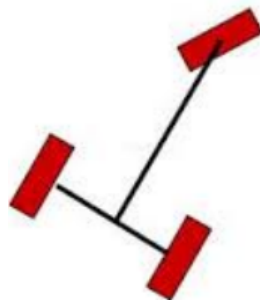


Figura 2.8: Configuração de rodas em triciclo [2]

Ackerman A configuração Ackerman ou quadriciclo, caracteriza-se por uma configuração típica de veículos de quatro rodas com dois eixos frontais e dois traseiros, podendo o tipo de tração variar entre estes dois. A distância entre estes dois eixos vai definir o ângulo de viragem do AGV, este ângulo de viragem quando comparado com os outros tipos de configurações, apresenta-se como uma grande limitação [2], já que vai exigir uma maior área para efetuar essas mudanças direcionais. Na utilização deste tipo de AGV exige uma maior atenção à definição dos trajetos a ser efetuados pelo mesmos, garantido que os caminhos que os mesmo pode efetuar não ultrapassem o seu raio de curvatura máximo. Na figura 2.9 está representada a típica configuração de rodas em Ackerman.

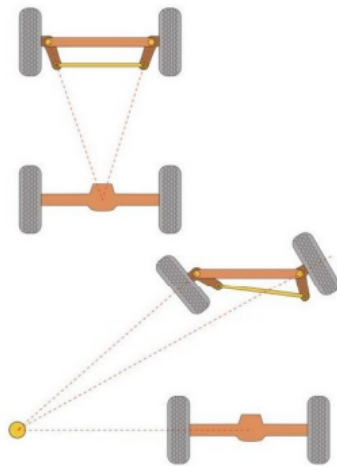


Figura 2.9: Configuração de rodas em Ackerman[2]

Diferencial Este tipo de configuração é um tipo de configuração bastante utilizada na indústria, quer pelo seu baixo raio de curvatura, quer pela facilidade de programação. A configuração diferencial caracteriza-se por ser bastante semelhante ao triciclo, sendo que não possui a roda frontal, possui apenas duas rodas de tração, que conseguem controlar a direção do robô pela diferença de velocidades entre as rodas. Na figura 2.10 estão representados os diferentes movimentos numa configuração diferencial de um AGV.

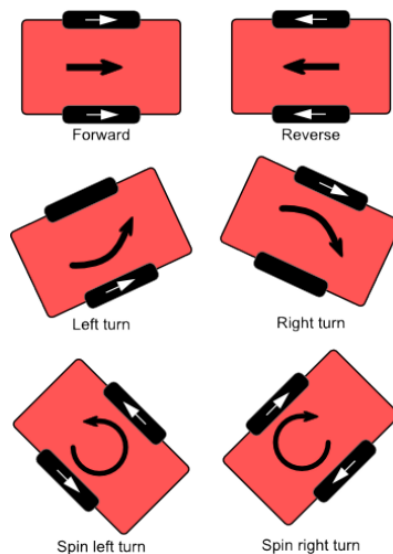


Figura 2.10: Configuração de rodas diferencial [2]

Omnidirecional Uma configuração omnidirecional pode possuir ou três rodas afastadas em ângulos de cerca de 120 graus ou uma configuração de 4 rodas que usualmente afastadas por 90

graus entre as rodas.

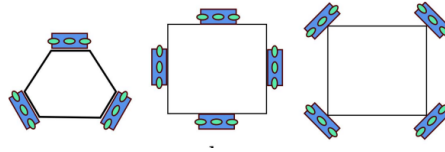


Figura 2.11: Configuração de rodas omnidirecionais [3]

2.2 Arquitetura de Controlo

A arquitetura de controlo de um sistema pertence a uma componente do sistema que não sofre alterações a menos que um agente externo atue sobre o mesmo. No sistema de multi AGV a arquitetura de controlo representa um componente muito importante no mesmo, que vai definir não só as limitações do sistema como também as capacidades do mesmo. Esta arquitetura de controlo pode ser dividida em dois segmentos, arquitetura centralizada e arquitetura descentralizada, tal como descrito em [4]. Nas secções seguintes as arquiteturas são descritas de forma mais detalhada bem como as consequências que as mesmas têm na independência dos AGV para decidirem as suas rotas.

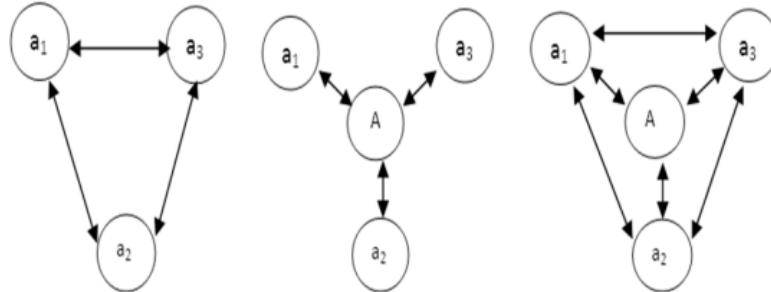


Figura 2.12: Diferentes classificações de arquitetura (a) Descentralizada (b) Centralizada (c) Híbrida [4]

2.2.1 Arquitetura Centralizada

Numa arquitetura de multi robôs centralizada o controlo das rotas é realizado por um agente central, sendo que os robôs não comunicam entre si. Nesta abordagem são combinados os planos individuais dos diferentes robôs, de maneira a que se encontre uma rota otimizada e pré determinada que respeite os objetivos individuais de cada robô, tal como descrito em [11], neste método verifica-se que em ambientes mais congestionados, em que existe um maior número de robôs móveis, o sistema tem uma taxa de sucesso inferior causada pela grande carga computacional a que o sistema está a operar, que consequentemente leva a tempos de processamento mais elevados. Na

abordagem de arquitetura centralizada tomada em [12] foi feita uma coordenação de diagramas de forma a reduzir os tempos os tempos de computação. Por outro lado em [13] foi realizado um método dinâmico de rotas baseado em janelas temporais, de forma a assinalar a sobreposição de rotas quando elas existem, para prevenir não só situações de colisão como também *deadlocks*.

2.2.2 Arquitetura Descentralizada

Numa arquitetura descentralizada o método caracteriza-se não só por uma distribuição da computação como também um alto nível de autonomia dos robôs móveis. Este tipo de distribuição onde os robôs planeiam as suas rotas e decisões, garante a este tipo de arquitetura uma elevada robustez e facilidade de implementação de novos robôs no sistema. Por outro lado, o facto de os robôs possuírem esta independência entre eles e operando no mesmo ambiente, leva a que seja exigida especial atenção no desenvolvimento de algoritmos de forma a que sejam minimizadas situações de colisão e de *deadlock* [14]. Uma arquitetura descentralizada pode ser dividida em dois tipos de métodos, [11], acoplada e desacoplada. Numa abordagem acoplada os robôs móveis planeiam as suas trajetórias de forma simultânea, isto garante que quando uma solução existe ela é encontrada (método completo).

2.3 Planeamento de Trajetória

A escolha do método de planeamento de trajetórias, caminhos ou movimentos, utilizado tem uma influência muito grande no comportamento do sistema.

O planeamento de caminho está associado a uma representação matemática, ao percurso que o robô deve percorrer até ao seu destino sem colidir com obstáculos, por outro lado o planeamento de trajetória descreve esse caminho em função do tempo. Por fim no planeamento dos movimentos, as limitações dinâmicas do robô móvel já são tidas em consideração.

2.3.1 Espaço de Configuração

De forma a que seja possível planear o caminho dos robôs móveis é preciso conhecer o ambiente físico em que os mesmos estão inseridos e mapear esse ambiente. O espaço total em que os robôs estão inseridos representa-se por $C_{\text{espaço}}$, inseridos neste espaço está o C_{free} , que representa a área em que os robôs se podem mover e onde os caminhos poderão ser definidas por outro lado a área ocupada por obstáculos é definida por C_{obstacle} e pode ser calculada de diferentes métodos tal como referido em [6].

2.3.2 Métodos de Planeamento de Trajetória

Na escolha de um método existem alguns parâmetros que se deve ter em consideração [6], tais como o tipo de variável que se pretende otimizar ou a complexidade computacional do método

que se pretende implementar, já que uma elevada complexidade pode levar a tempos de execução elevados ou a uma insuficiência de memória.

Para um método ser completo tem de encontrar uma solução quando de facto ela existe, ou não a encontre e de facto não exista, é completo em resolução se existir uma solução para determinada discretização do ambiente ou probabilisticamente completo se a probabilidade de encontrar uma solução converge para um à medida que o valor temporal cresce. [6]

Um método pode ser também dividido em *online* ou *offline*. Caso a solução seja encontrada previamente, ou seja antes de o robô iniciar o seu trajeto, o método diz se *offline*. Por outro lado se o robô for encontrar a solução enquanto se encontra em processamento o método diz se, *online*. Dado que o objetivo desta dissertação é a implementação do algoritmo TEA*, os métodos de planeamento de caminhos analisados serão apenas os que resultam em grafos no qual podem ser aplicados algoritmos de pesquisa que conseguem obter os caminhos com o menor custo total possível de modo a obter rotas otimizadas, nestes métodos realça-se o Roadmap e a Decomposição em células que foram analisados de forma mais detalhada em 2.3.3 e 2.3.4, respetivamente. Na figura 2.13 encontram se representado alguns dos métodos de planeamento de trajetórias.

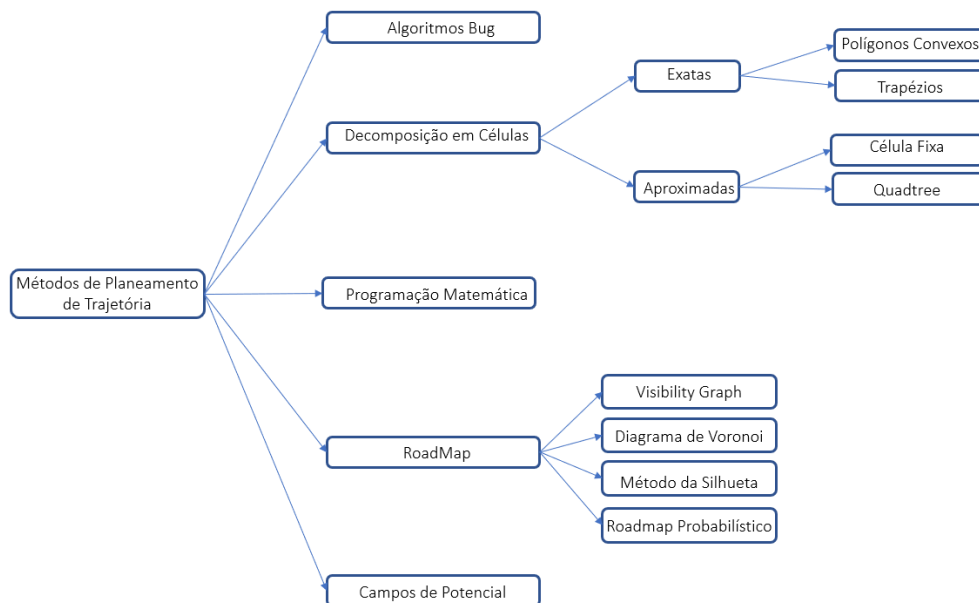


Figura 2.13: Métodos de Planeamento de Trajetórias

2.3.3 Roadmap

O método de Roadmap, caracteriza-se pela modelação do C_{free} em nós e ligações esta modelação topográfica leva a que seja possível sintetizar o espaço em grafos de forma a facilitar a implementação de algoritmos de pesquisa [6]. A construção do roadmap pode ser feita com recurso a diferentes técnicas, que serão analisadas nas secções seguintes 2.3.3.1 e 2.3.3.2.

2.3.3.1 Visibility Graph

Este tipo de gráfico caracteriza-se por uma representação bidimensional do espaço C_{free} , onde os nós representam não só os pontos de partida e destino como também os vértices dos obstáculos entre esses dois pontos. Para existir um caminho entre dois nós é necessário que não exista interseção com nenhum obstáculo. O facto de os nós corresponderem a pontos muito próximos dos obstáculos, pode criar situações de colisão dos robôs [6].

Na figura 2.14 está representado um visibility graph num espaço bidimensional, os polígonos a preto representam os obstáculos, as linhas a tracejado e de espessura maior representam o caminho mais curto, passando pelos vértices dos obstáculos, entre os pontos q_{init} e q_{goal} , de acordo com a métrica euclidiana [5].

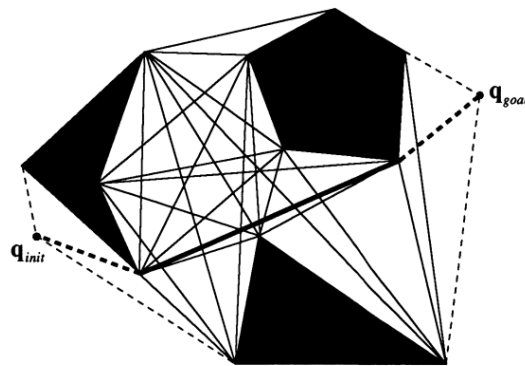


Figura 2.14: Exemplo de um Visibility Graph [5]

2.3.3.2 Diagrama de Voronoi

Ao contrário do visibility graph analisado anteriormente um diagrama de Voronoi caracteriza-se por encontrar um caminho do ponto inicial ao ponto destino mantendo uma distância equidistante entre os obstáculos, este conjunto de pontos médios dita a rota que o robô deve adotar até ao destino [5]. O facto de este tipo de roadmap manter sempre uma determinada distância entre os obstáculos faz com que este método seja bastante preventivo nas colisões do robô com os obstáculos, ainda que a rota não seja a mínima. Na figura 2.15 está representado um exemplo de um Diagrama de Voronoi, o ponto inicial liga-se ao diagrama e a partir daí encontra o o melhor caminho até ao ponto destino.

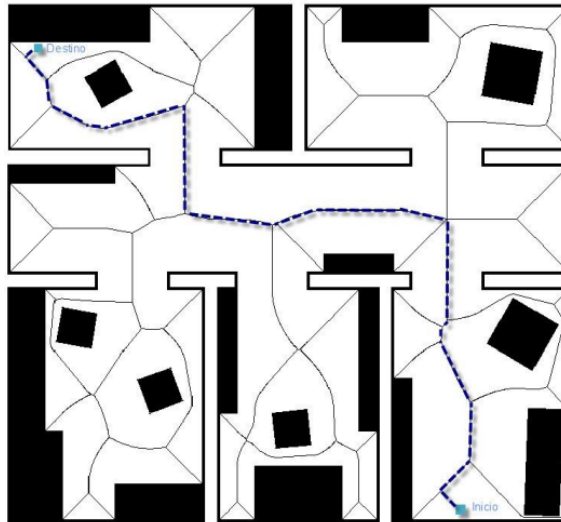


Figura 2.15: Exemplo de um Diagrama Voronoi[6]

2.3.4 Decomposição em Células

O método de decomposição de células é um dos métodos mais estudados, esta abordagem baseia-se na decomposição bidimensional do $C_{\text{espaço}}$, avaliando se cada uma das células está livre ou ocupada, bem como as ligações entre elas. Esta decomposição pode ser representada por um grafo que facilitara a utilização de algoritmos de pesquisa de grafos. A decomposição do $C_{\text{espaço}}$ pode ser realizada de duas formas, decomposição em células aproximadas 2.3.4.2 ou decomposição em células exatas 2.3.4.1.

2.3.4.1 Decomposição em células exatas

Neste tipo de método, a decomposição do espaço em células é feita através de uma representação geométrica simples que separa o C_{free} do C_{obstacle} , que reflete o espaço físico que está a ser decomposto [5]. Esta decomposição em células exatas pode ser feita de duas formas ou por uma representação geométrica em polígonos convexos ou por uma representação geométrica em trapézios.

Decomposição em polígonos convexos No seguimento da secção anterior 2.3.4.1, nesta metodologia as células são representadas por polígonos convexos compostos por vértices como representado em 2.16 que representam os limites de $C_{\text{espaço}}$ bem como os vértices dos obstáculos, os pontos médios destes segmentos de reta, ditam o trajeto pelo qual o robô deve passar [6].

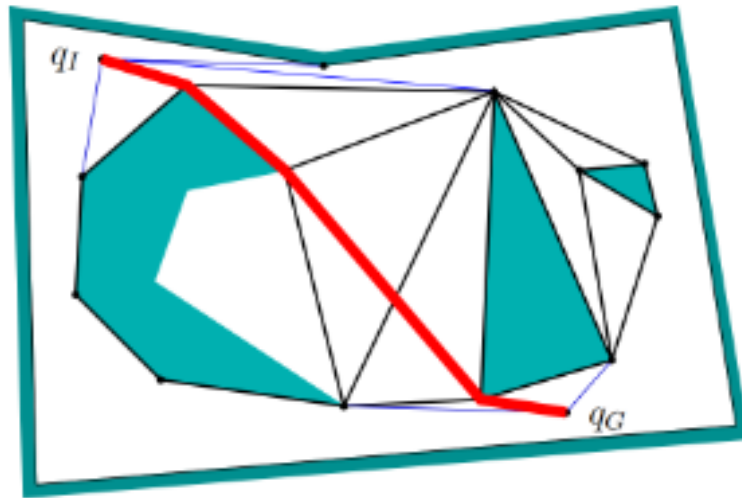


Figura 2.16: Exemplo de uma decomposição em polígonos convexos[7]

Decomposição em trapézios Na decomposição em trapézios são traçadas linhas verticais nos vértices dos $C_{obstacle}$, estas linhas vão formar células com a forma de trapézios, depois de o grafo de conectividade estar construído o trajeto é configurado através dos pontos médios das várias células adjacentes, até que seja atingido o ponto destino. [5] [6]

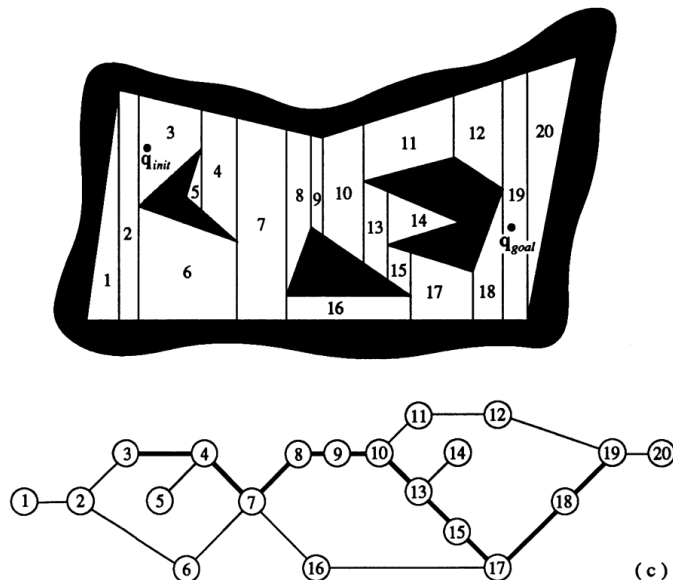


Figura 2.17: Exemplo de uma decomposição em trapézios e o grafo associado ao mesmo [5]

2.3.4.2 Decomposição em células aproximadas

Numa decomposição em células aproximadas ao contrário do que acontecia com a decomposição exata, neste método as células retangulares, são divididas em livres, ocupadas ou semi-ocupadas e têm um tamanho pré determinado, esta estratégia confere uma representação do espaço físico menos precisa, que varia consoante o tamanho definido das células. Quanto menor o tamanho das células, maior será o numero de células livres, que levará não só, a uma maior precisão na tradução do espaço físico, como também a uma maior hipótese de ser encontrada uma solução. A decomposição em células pode ser feita de algumas formas, dada a natureza da dissertação focou-se apenas nos métodos de Quadtree e Células Fixas.

Quadtree: A metodologia Quadtree é caracterizada por uma divisão recursiva [15] das células que não estão livres, essas células dividem-se em 4 células com o mesmo tamanho, este processo é feito iterativamente até que seja atingido o tamanho mínimo das células retangulares, anteriormente determinado. Comparativamente ao método de Células Fixas, o facto de existir um menor número de células facilita o processamento computacional do algoritmo. Na figura 2.18 está representada uma representação gráfica da decomposição Quadtree.

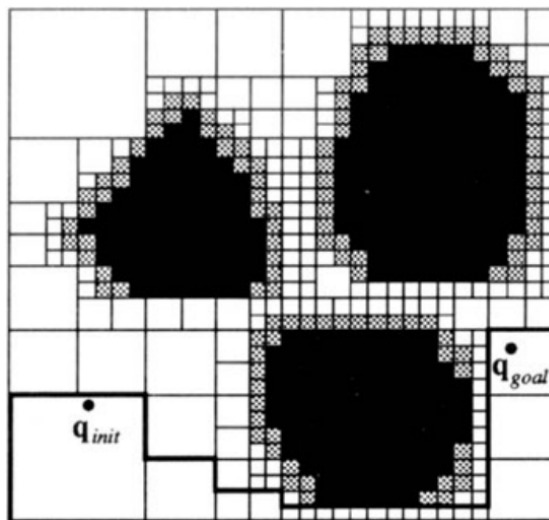


Figura 2.18: Representação gráfica da decomposição Quadtree[5]

Células Fixas: No método de células fixas o $C_{\text{Espaço}}$ é dividido em células com um tamanho pré-determinado, ao contrário do que acontece com o método de Quadtree, as células retangulares possuem todas o mesmo tamanho, todavia o facto de este método decompor o espaço num maior número de células leva a um tempo de computação mais elevado comparativamente ao método de Quadtree. Na figura 2.19 está representado um método de células fixas.

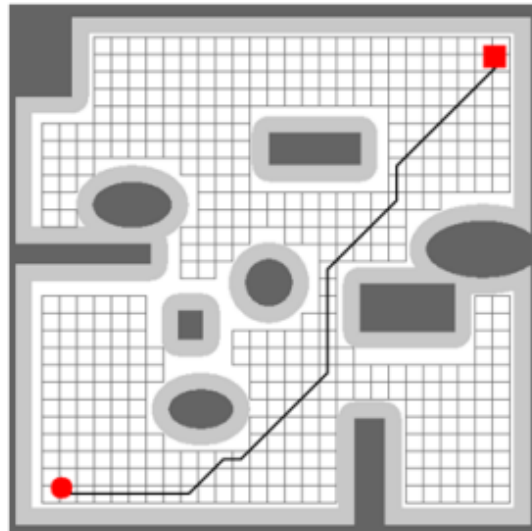


Figura 2.19: Representação gráfica da decomposição em células fixas[6]

2.4 Algoritmo de Pesquisa por grafos

Nas abordagens analisadas anteriormente em 2.3.4, analisou-se a decomposição do $C_{\text{espaço}}$ em grafos. Os algoritmos de pesquisa por grafos vão agora determinar o comportamento e eficiência do sistema, depois de estarem determinados os pontos iniciais e finais os algoritmos de pesquisa ditam qual o melhor caminho do ponto inicial ao ponto final, a escolha do tipo de algoritmo utilizado determina a eficiência dos caminhos dos robôs. Os algoritmos de pesquisa por grafos podem ser divididos em dois tipos, algoritmos sem informação 2.4.1 e algoritmos com heurística 2.4.2 que se distinguem pela facto de conseguirem atingir soluções com com menor uso de espaço e tempo, através da análise de custo entre nós do grafo.[16]

2.4.1 Algoritmos sem Informação

Os algoritmos de pesquisa sem informação caracterizam-se pela forma que é realizada a pesquisa nos grafos. Neste tipo de algoritmos, a única informação que possuem são os nós, inicial e final e com base nisso efetuam uma busca nos grafos. A pesquisa pode ser realizada com recurso a diferentes algoritmos [6], todavia foram apenas abordados os algoritmos de pesquisa por profundidade e por largura, que se distinguem pela forma como é realizada a pesquisa nos grafos e que são analisados em 2.4.1.1 e 2.4.1.2

2.4.1.1 Profundidade - DFS

O algoritmo de pesquisa por profundidade, é um dos principais algoritmos de pesquisa, usado para explorar nós e ligações de grafos. A complexidade de tempo para este algoritmo é de $O(N^\circ \text{ de vértices} + N^\circ \text{ de ligações})$. A pesquisa é realizada começando pelo nó da raiz e visitando todos nós de um ramo o mais fundo quanto possível, até encontrar um nó sem ligações, ou um nó já visitado, nesse caso, o algoritmo retrocede até a um nó com ramos por explorar e repete o processo até ser encontrado o nó destino. Na figura 2.20 esta representada esquematicamente a forma como a pesquisa por profundidade é realizada.

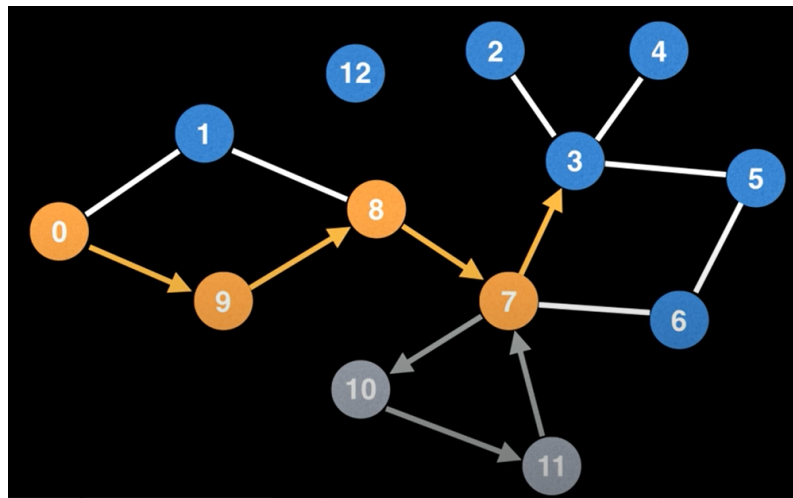


Figura 2.20: Representação gráfica do algoritmo de pesquisa DFS (*Depth First Search Algorithm*) - a) laranja: nós visitados b) azul: nós que faltam visitar c) cinzento: nós já visitados em que se realizou *backtracking*

2.4.1.2 Largura - BFS

O algoritmo de pesquisa por largura, tal como o algoritmo de profundidade analisado na secção anterior, o BFS (*Breadth First Search Algorithm*) é um dos principais algoritmos de pesquisa utilizado também para explorar os nós e ligações de um grafo, quando apenas possui conhecimento do nó inicial e do nó final que se pretende alcançar. Este método diferencia-se do DFS, na forma como a pesquisa do grafo é realizada, tem como objetivo principal de encontrar o caminho mais curto em grafos sem custo associado às ligações. O método de pesquisa começa ele também na raiz do grafo (nó inicial), todavia a pesquisa é realizada visitando todos os nós conectados a esse nó, este processo é realizado de forma iterativa até que seja encontrado o nó final que representa o destino que se pretende alcançar ou o grafo ser todo percorrido. A figura 2.21 representa de forma teórica a forma como o processo é realizado, a partir do nó inicial, são guardados numa lista os nós associados ao mesmo, que representam os nós que faltam visitar, este processo repete-se até encontrar o nó destino ou serem visitados todos os nós do grafo.

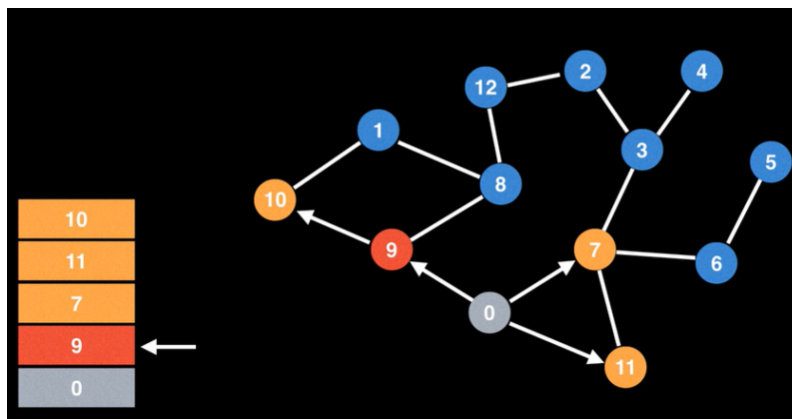


Figura 2.21: Representação gráfica do algoritmo de pesquisa BFS - a) laranja: nós na lista a visitar b) azul: nós que estão por associar c) cinzento: nós já visitados d) vermelho: nó que está a ser analisado

2.4.2 Algoritmos com Heurísticas

Os algoritmos com heurística, são bastante utilizados devido à sua capacidade de reduzir o tempo de execução, diferenciam-se de algoritmos sem informação na medida em que existe um custo associado entre os nós, este custo representa usualmente o tempo que demora a passar de um nó para o outro, todavia visto que a velocidade dos robôs será toda igual este custo pode ser associado à distância física entre as células, representadas pelos nós do gráfico. Dentro dos algoritmos com heurísticas realçam-se o algoritmo Dijkstra, Greedy, A* e a derivada deste algoritmo que terá maior relevo na dissertação Time Enhanced A*.

2.4.2.1 Algoritmo Dijkstra's

Neste método introduzido inicialmente por Dijkstra em 1959, são avaliados os nós mais próximos do nó inicial, ou seja é explorado em cada vizinhança do nó o que tem o custo mais baixo, este processo é feito de forma iterativa, para um número finito de nós, até se encontrar uma rota para o nó final. Este algoritmo apenas encontra soluções ótimas para casos em que o custo de todas as ligações é igual[17].

2.4.2.2 Algoritmo Greedy

O algoritmo Greedy usa metodologia de pesquisa local em cada nó, onde procura o nó seguinte com o menor custo possível para o nó final, ótimo local. O método é feito de forma iterativa, para um número finito de ligações, até ser encontrado o nó destino. [17].

2.4.2.3 Algoritmo A*

O algoritmo de procura A* é uma das melhores e mais utilizadas técnicas no planeamento de trajetórias e na pesquisa de grafos. Considerando um grafo com um nó inicial, nó final destino e nó

n que representa o nó que está a ser visitado, e tendo como objetivo, para o caso desta dissertação chegar ao nó final no menor tempo possível, este processo pode ser alcançado através do algoritmo A* [17]. Nesta metodologia, é visitado e atribuído a cada um dos nós um determinado custo, este custo representado por $F(n)$ representa a soma de valores de $G(n)$ e $H(n)$. $G(n)$ representa o custo do caminho do nó inicial até ao nó n que está a ser visitado, por outro lado $H(n)$ representa a heurística que dita o custo do nó n visitado, até ao nó final que se pretende alcançar.

$$F(n) = G(n) + H(n) \quad (2.1)$$

- Se não estiverem presentes em nenhuma das listas, são inseridos numa lista, O-list
- Se já estiverem presentes na O-list (células já visitadas) e o seu valor de custo $F(n)$ for inferior ao nó anterior, o valor de F é alterado para esse novo valor.
- Se por outro lado já estiver presente na C-list e o seu custo de de F for inferior, é inserido na O-list.

Quando a célula que contém as coordenadas pretendidas é alcançada, significa que foi atingido o caminho ótimo do ponto inicial ao ponto final e sabendo que cada uma das células contém informação relativamente à célula pai que lhe deu origem, o caminho ótimo é descoberto através de uma pesquisa inversa.

Na figura 2.22, está representado um exemplo da forma como o algoritmo opera de forma a encontrar o caminho com o menor custo, representado a cor azul do ponto A ao ponto B, o valor central nas células representa o custo $F(n)$, o valor superior esquerdo representa o custo $G(n)$ e por fim o valor superior direito representa o custo $H(n)$. As células a vermelho dizem respeito aos nós da C-list, e as células a verde representam os nós da O-list.

		B	58 10 68	48 20 68	38 30 68	34 40 74
58 24 82						24 44 68
54 28 82	44 24 68	34 20 54	24 24 48	14 28 42	10 38 48	14 48 62
58 38 96	40 34 74	30 30 60	20 34 54	10 38 48	A	10 52 62

Figura 2.22: Representação esquemática do custo das células no algoritmo A*

2.4.2.4 Algoritmo Time Enhanced A* (TEA*)

Num sistema de coordenação de multi-AGV, é necessário realizar uma coordenação temporal dos AGV de modo a que sejam planeadas as trajetórias dos vários AGV ao longo das diferentes camadas temporais, que variam de $k=[0;TMax]$ ($TMax$ representa a última camada temporal), como referido em 2.23. Esta coordenação temporal garante que se evitem colisões entre robôs antes de os mesmos começarem os seus trajetos. O algoritmo Time Enhanced A* [18], é um algoritmo de planeamento de caminhos recalculados de forma contínua, sendo assim um método *online*, aplicável a cenários industriais em que existe mais do que um robô móvel, multi-AGV. Durante a pesquisa de trajeto para um AGV, é utilizada a mesma abordagem do algoritmo A* onde se avalia o custo dos vértices vizinhos, sendo que neste algoritmo as vértices vizinhos e a posição atual do AGV pertencem à camada temporal seguinte $K+1$, como representado na figura 2.23, o facto de se incluir a posição atual do AGV na camada temporal seguinte, permite que o AGV mantenha a sua posição para diferentes camadas temporais, caso não existam vértices livres.

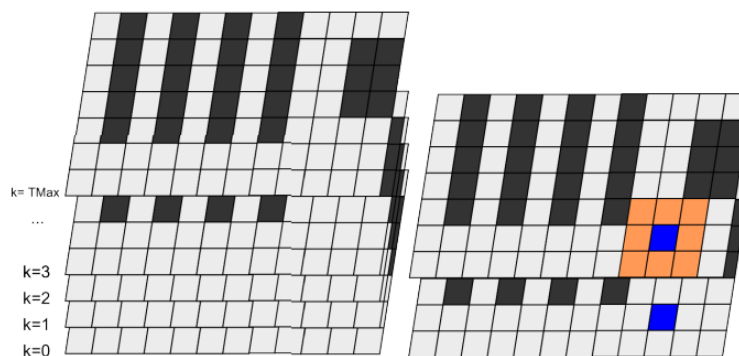


Figura 2.23: Representação esquemática das camadas temporais (esquerda) e representação do AGV na célula juntamente com as suas células vizinhas na camada temporal seguinte (direita)

[18]

O número de camadas temporais será tanto maior quanto maior forem as iterações necessárias para que os robôs cheguem ao seu destino, pelo que quanto maior for o ambiente em que os mesmos estão inseridos, maior será o número de camadas temporais K . Cada um dos AGV tem atribuída uma determinada prioridade, esta prioridade definirá a hierarquia de planeamento do caminho de cada um dos AGV, cada um desses pontos dos caminhos bem como a posição que os mesmos se encontram são identificados como obstáculos, para os robôs na posição hierárquica inferior. De forma a evitar situações de *deadlock*, as células referentes às posições dos AGV são identificadas como obstáculos apenas nas camadas temporais $K=0$ e $K=1$.

Metodologia Como referido anteriormente o algoritmo TEA*, utiliza a mesma abordagem do algoritmo A* para atribuição de custos dos diferentes vértices. De forma semelhante ao algoritmo

A*, o primeiro termo $\alpha g(n,k)$, corresponde à distância entre o vértice atual e inicial na camada temporal K 2.3, por outro lado $\beta h(n,k)$ corresponde ao valor da heurística que determina a distância até ao vértice final 2.4 . Os coeficientes α e β correspondem a diferentes pesos atribuídos a cada um dos termos.

$$F(n,k) = \alpha g(n,k) + \beta h(n,k); k \in [0, TMax]; n \in [0, N_{maxvertices}] \quad (2.2)$$

$$g(n,k) = dis(n, n_0, k) + dis(n, n+1, k) \quad (2.3)$$

$$h(n,k) = dis(n, n_f, k) \quad (2.4)$$

A partir da literatura [8], para a aplicação do algoritmo TEA*, é antes necessário modelizar a disposição do chão de fábrica, numa série de vértices e arestas correspondentes a um grafo, que representem os diferentes caminhos em que os AGV se podem movimentar. As diferentes arestas, são obtidas sob a forma de uma curva cúbica de Bezier [19], tal como representado na figura 2.24.

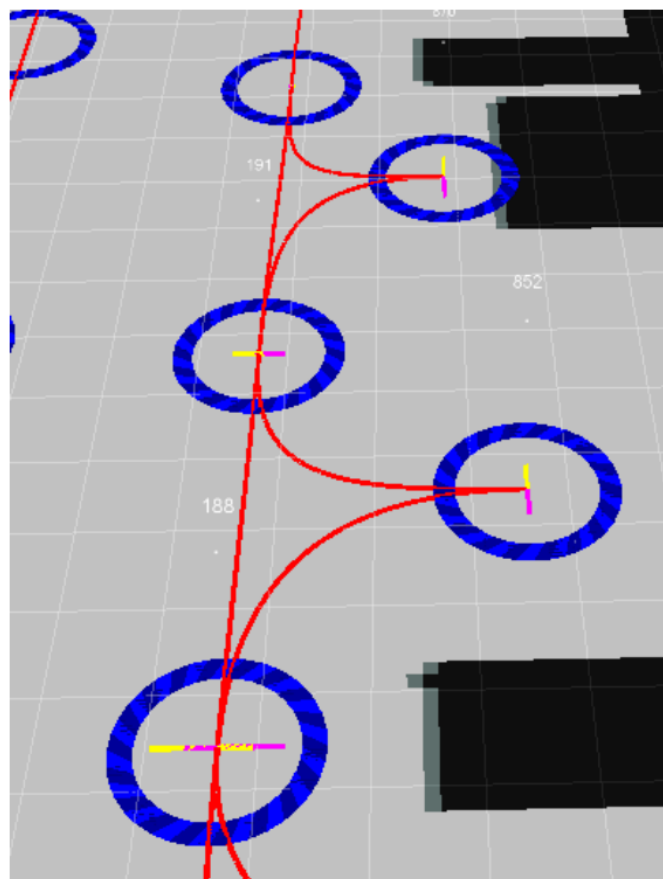


Figura 2.24: Representação das curvas de bezier numa parte de um mapa [8]

Este método distingue-se, pelo facto de ser um algoritmo aplicado de forma *online*, sempre que as posições dos vários robôs são atualizadas, as suas rotas são também atualizadas, esta característica garante a este método uma elevada segurança no controlo das diferentes rotas. Relativamente à complexidade computacional deste algoritmo e às consequências que o mesmo pode ter no comportamento do sistema, pelos resultados obtidos na literatura [18] consegue-se perceber que o algoritmo TEA* é um algoritmo com um comportamento bastante versátil às mudanças do ambiente e que fornece contribuições significativas na coordenação de multi-AGV.

2.5 Software de Simulação

2.5.1 Comparação de simuladores

A simulação de um sistema cresce com a necessidade de prever e estudar as consequências de possíveis implementações no sistema, sem comprometer o sistema real, isto leva a que os *softwares* de simulação sejam uma ferramenta cada vez mais útil em ambiente industrial. Nesta seção são mencionados alguns dos *softwares* de simulação disponíveis, bem como uma enumeração das principais características de cada um, justificando assim a escolha de simulador a ser implementada. Os *softwares* de simulação podem ser divididos em três categorias: Simulação de Eventos Discretos, Simulação de Eventos Contínuos, ou Simulação baseada em agentes [20].

Na simulação de eventos contínuos é normalmente analisada a resposta de um sistema ao longo da duração da simulação, é usualmente aplicada a processos de fluxo contínuo onde se pretende analisar elementos que possuem relações de não linearidade entre eles [20].

Simulação baseada em agentes, é caracteriza-se pela realização de uma simulação de agentes autônomos e as interações entre os mesmos de maneira a que seja obtida uma melhor compreensão do sistema e do seu comportamento [20], são maioritariamente aplicados em áreas de ciências sociais.

Simulação de eventos discretos, são uma escolha de simulação usualmente aplicada a nível industrial, podem ser definidos como a modelação de operações que ocorrem em determinados instantes no tempo e as alterações de estado inerentes ao mesmo, dada a natureza da dissertação, as ferramentas de *software* analisadas, são simuladores de eventos discretos.

Visual Components O simulador Visual Components, foi inicialmente fundado em 1999, com a principal função de facilitar o design de produção e tecnologias de simulação, a diferentes indústrias. Hoje em dia o *software* Visual Componentes, permite o desenvolvimento e criação de layouts de produção tridimensionais, programados em linguagem Python ou Dot.NET. Este *software* esta dividido em 3 tipos de produtos, Essencial, Professional e Premium, que variam entre si essencialmente pelas ferramentas que possuem.

Simio: O Simio é um simulador de modelação dinâmico, desenvolvido em 2007 utilizado para a simulação de de modelos dinâmicos, sendo a criação de objetos é totalmente gráfica sem recurso a nenhuma linguagem de programação. A simulação pode ser realizada através de animação 3D ou 2D, que simula o comportamento do sistema ao longo do tempo. O simulador permite ainda alternar entre um modelo interativo onde se pode visualizar o comportamento de sistemas, e um modelo experimental que permite realizar alterações de prioridade de forma a avaliar o comportamento do mesmo a essas alterações. O simulador permite replicar o modelo real, mantendo as relações espaciais dos objetos de forma bastante precisa, esta funcionalidade é alcançada dada a biblioteca pertencer ao Google Warehouse. Em termos de aplicabilidade deste *software*, o mesmo pode ser aplicado a diferentes tipos de sistema, manufaturação, logística, cuidados de saúde, entre

outros. Uma das desvantagens que o Simio apresenta, na atratividade para os utilizadores comparativamente com outros, é o facto de não ser possível a importação de objetos *Computer Aided Design*.

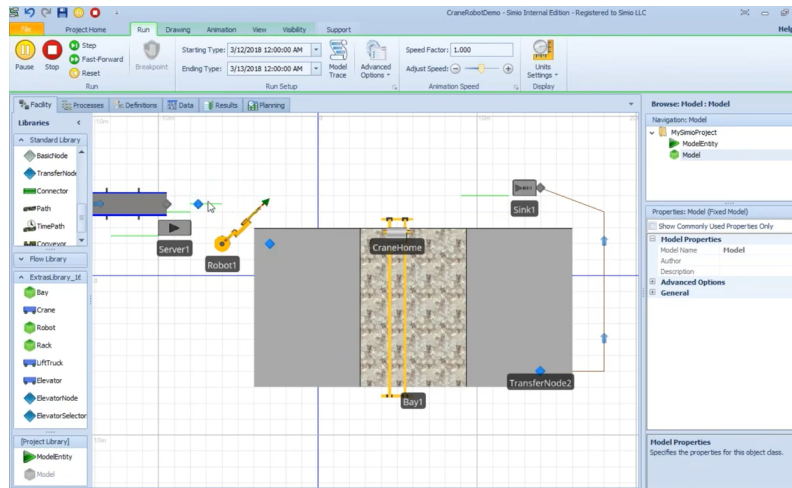


Figura 2.25: Exemplo de uma linha de produção em 2D no simulador Simio

Simul8: O Simul8 é um simulador desenvolvido em 2012, que permite a modelação de projetos através do uso de uma interface de *drag and place* com recurso a blocos de construção. A simulação em Simul8, é realizada através de uma linguagem à base de Visual Basic e permite a partilha e análise dos resultados com outras ferramentas como Excel [21]. O Simul8 apresenta duas grandes desvantagens, o facto de o ambiente gráfico deste *software* permitir apenas uma visualização em 2D e da análise realizada ao *software*, não foi encontrada uma ferramenta que permitisse a utilização de robôs móveis, o que exclui este *software* para o estudo da dissertação.

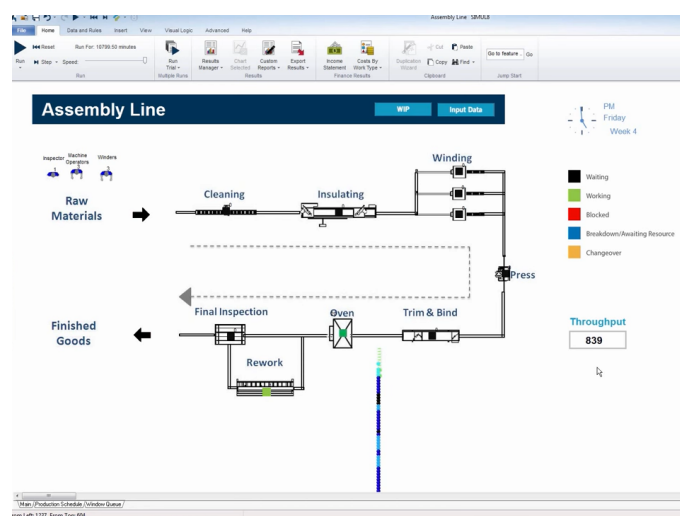


Figura 2.26: Exemplo de uma linha de montagem simulada em Simul8

FlexSim: O *software* FlexSim, foi inicialmente lançado em 2003, e ao longo dos anos tem vindo a ser um *software* com bastante presença em diferentes setores como a fabricação, assistência médica, armazenamento, entre outros. O *software* permite modelar um sistema real em 3D e possibilita a importação de objetos personalizados e tridimensionais em CAD, que se aproximem do modelo real. O *software* permite ao utilizador executar simulações usando o modelo base para testar diferentes cenários "e se", permitindo assim uma análise do mesmo a diferentes *inputs*. O *software* permite ainda a simulação de um sistema Multi-AGV que permite configurar e modelar sistemas complexos que utilizem AGV [22].

O modelo do projeto é realizado com base em objetos tridimensionais presentes no ambiente gráfico, sendo que as suas interações são realizadas através da implementação do modelo com base em diagramas de blocos, através da ferramenta de programação por fluxograma representado na figura 2.27. Por fim, o FlexSim tem ainda a possibilidade de programação em C++, que concede uma maior liberdade na definição de parâmetros que o utilizador pretende implementar.

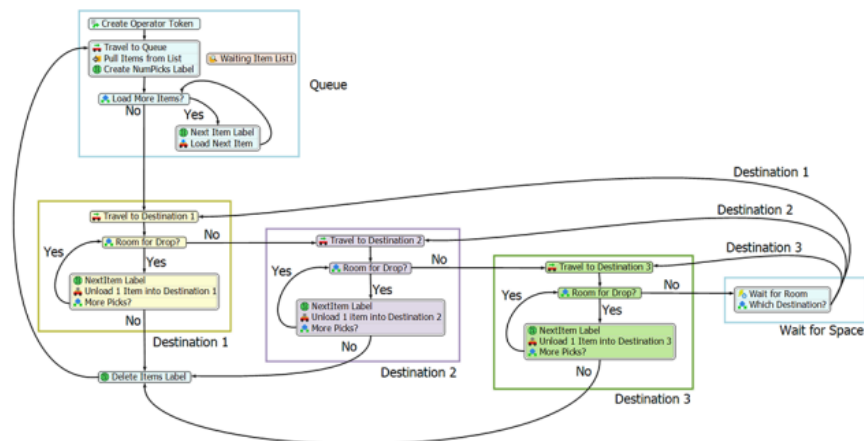


Figura 2.27: Ferramenta de Process Flow do FlexSim

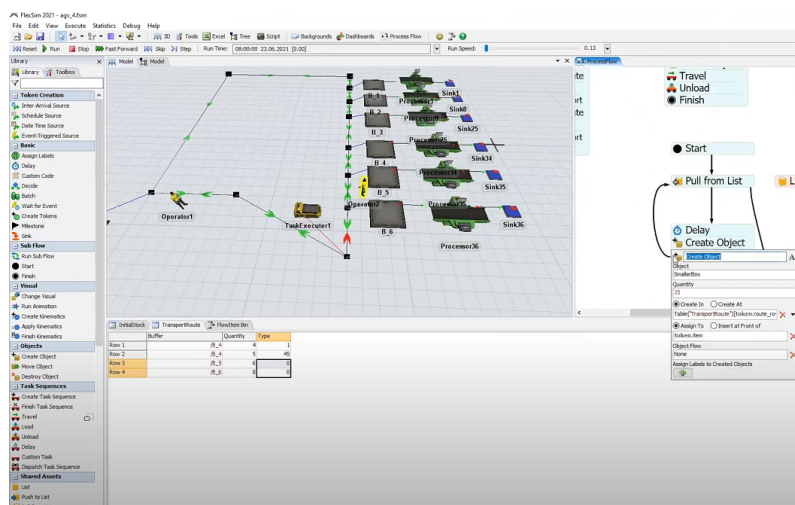


Figura 2.28: Exemplo de um loop logístico simulado em FlexSim

2.5.2 Software - FlexSim

Dos *softwares* analisados anteriormente consegue-se comprovar que o FlexSim é de facto o *software* mais indicado para a implementação do algoritmo TEA*. O facto de o FlexSim possibilitar a definição de parâmetros de implementação, ser um dos *softwares* com maior presença no mundo industrial, de permitir a utilização e planeamento de trajetórias de robôs móveis, a utilização de um ambiente gráfico 3D e por fim o facto de o INESC-TEC possuir uma licença flutuante que permite o acesso a todas as ferramentas, faz deste *software* o mais indicado para a utilização do estudo da dissertação. Nesta secção é analisada de forma mais detalhada algumas das ferramentas, importantes ao desenvolvimento da dissertação, presentes neste *software*.

2.5.2.1 Modelação

A modelação em FlexSim pode ser realizada, ou com recurso a uma modelação através do mapa modelo que permite replicar num ambiente 3D, as condições físicas do ambiente real que se pretende simular, ou no caso de se pretender uma simulação de um modelo mais teórico, usar de forma singular a ferramenta de Process Flow, em modelos mais complexos estas duas ferramentas são usadas de forma complementar.

2.5.2.2 Model

Como já referido anteriormente o Model do FlexSim permite a construção de modelos de processos tridimensionais. Esta ferramenta do simulador, é constituída por uma biblioteca, composta por componentes reutilizáveis e pré-configurados, representativos de vários componentes existentes em indústria. Na biblioteca estão presentes diferentes tipos de objetos [23], todavia dada a

natureza da dissertação, destacam-se os Fixed Resources, Task Executors e a ferramenta AGV necessária para a construção dos caminhos dos AGV.

- Flow Items: Flow items [23], não estão incluídos na biblioteca como um objeto a ser utilizado de forma singular todavia estão presentes no modelo e representam os objetos que se movem entre estações, este tipo de objetos varia consoante o modelo que se pretende simular ainda assim são geralmente representativos de bens de produção ou logística que necessitam de ser transportados ou clientes, no caso de ser um modelo representativo de serviço ao cliente.
- Fixed Resources: Representam recursos que se mantêm estáticos no modelo tridimensional [23] e que interagem com o fluxo de objetos do modelo. Dentro deste tipo de recursos destacam-se, as Sources, que criam os flow items, Queue que armazenam os flow items até que os mesmos possam ser transportados para outro objeto, os Processors que representam algum tipo de processamento nos flow items e por fim a Sink, que retira os flow items do modelo de simulação.

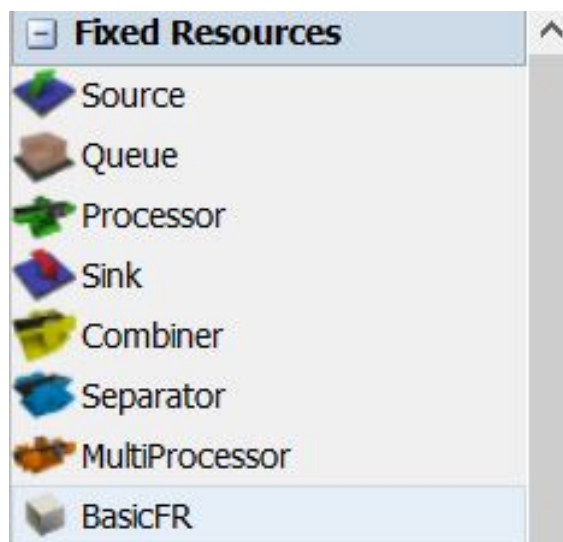


Figura 2.29: Recursos fixos do FlexSim

- Task Executors: Este tipo de objetos representa a classe de nível superior para vários objetos na biblioteca e são utilizados para desempenharem funções de operação, transporte, entre outros. Este objetos têm a capacidade de se movimentar, carregar e descarregar flow items e executar muitas outras tarefas dentro do modelo de simulação. Entre os objetos existentes, destacam se os AGV que são os objetos utilizados para a representação da integração do algoritmo no simulador. [23].

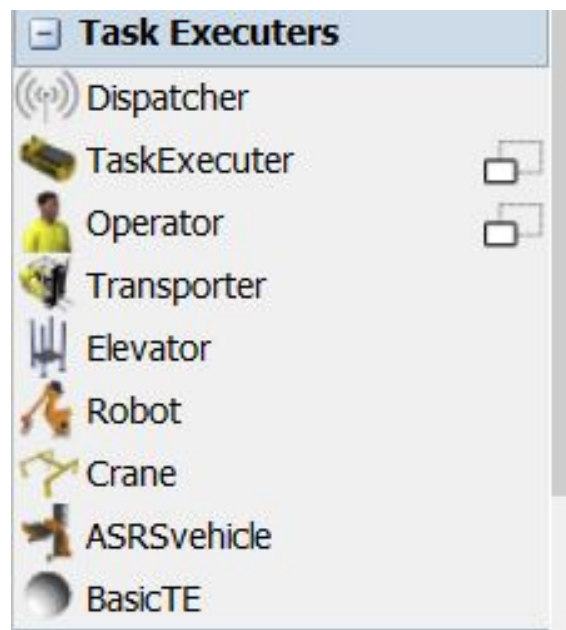


Figura 2.30: Ferramentas Task Executer

- AGV: Na biblioteca está também presente a ferramenta AGV, constituída por elementos, que formam os caminhos que os robôs se podem movimentar. Os caminhos, representados com a ferramenta Paths e os Control Points, utilizados como pontos de referência para a passagem dos robôs. Nos modelos construídos na dissertação os Paths e Control Points representam as ligações e os nós do mapa grafo, respetivamente.

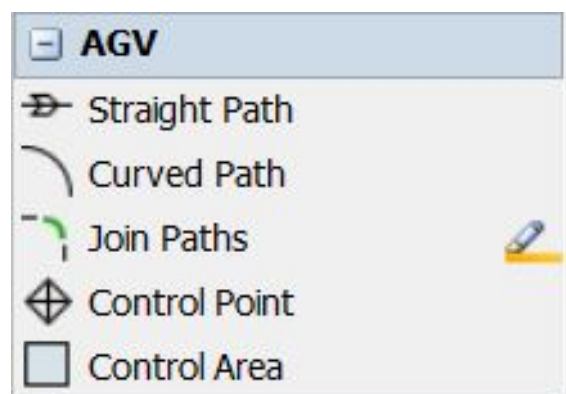


Figura 2.31: Ferramentas AGV

2.5.2.3 Process Flow

A ferramenta de Process Flow, apresenta-se como uma ferramenta não só mais abstrata, como também pelo facto de existir uma maior facilidade de criação de modelos com lógicas personalizadas, quando comparada com as ferramentas de modelação em três dimensões. A modelação

lógica de um sistema, com uma boa precisão, é uma tarefa muitas vezes exigente do ponto de vista de implementação, que usualmente requer bastantes linhas de código, deste modo a ferramenta Process Flow apresenta-se como uma alternativa bastante vantajosa. A programação é realizada com recurso a fluxogramas, através da utilização de blocos pré-construídos, a lógica do processo rege-se pelo fluxo do sistema [24]. De forma semelhante à modelação tridimensional, o process flow possui uma biblioteca com diferentes blocos, pré-construídos, cada um dos tipos de blocos possui diferentes tipos de personalização [25]. A *trigger* de cada bloco do Process Flow, são os "token" presentes, nesta ferramenta, que se inicializam na source e que percorrem os blocos, consoante a modelação implementada no diagrama de fluxo.

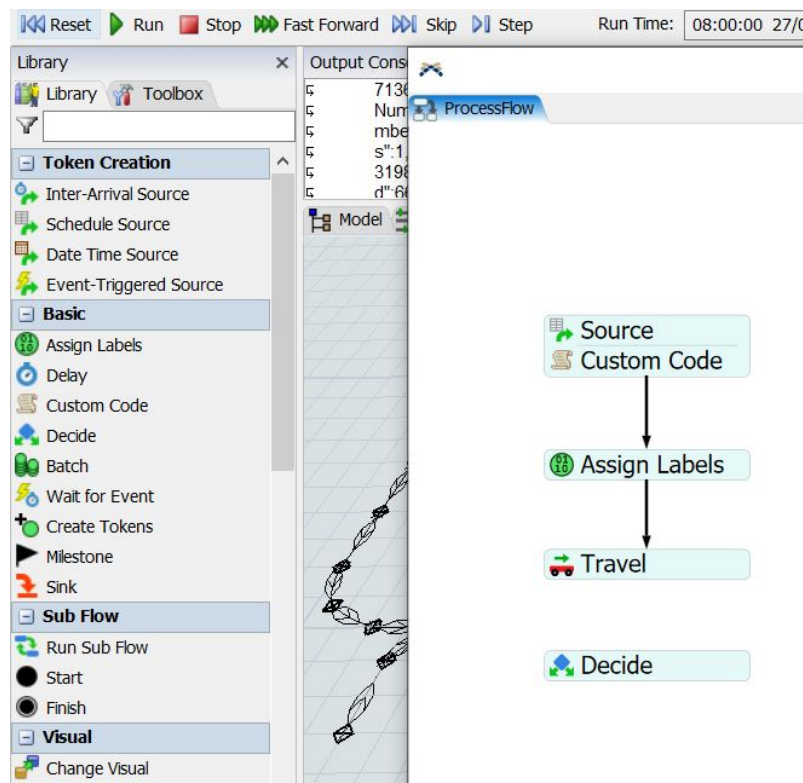


Figura 2.32: Ferramenta Process Flow

2.5.3 Conclusão

Neste capítulo foram mencionadas, algumas das soluções já existentes relativamente a tipos de AGV, algoritmos de otimização de rotas, decomposição de mapas e alguns dos simuladores existentes no mercado. Os conceitos teóricos anteriormente mencionados são importantes para a compreensão dos diferentes pontos e conceitos científicos que o projeto de dissertação toque.

Capítulo 3

Metodologia

Este capítulo tem como principal objetivo descrever o trabalho desenvolvido para a implementação do algoritmo no simulador FlexSim. O desenvolvimento do modelo exigia uma escolha quantitativa de AGV, deste modo dada a natureza desafiante do projeto para o tempo fornecido para a realização o mesmo, bem como a troca de informação exigida entre os dois sistemas e o consequente nível de computação requerido do *hardware*, para correr a simulação, foi construído um modelo de simulação com 3 AGV. Este capítulo divide-se em diferentes secções começando com uma descrição da plataforma de simulação e o trabalho desenvolvido na mesma, mapa de simulação, AGV e toda a configuração no simulador. De seguida é descrita a forma como as trocas de mensagens entre o simulador e o algoritmo são realizadas e por fim é descrito o PF que foi construído, este PF pode ser visto como um "Organizador de Tarefas", dentro do FlexSim, é responsável pela organização temporal da troca informações com o algoritmo e por ordenar o movimento dos AGVs.

3.1 Construção do Modelo Tridimensional

Para a integração e aplicação do algoritmo TEA* no simulador é inicialmente necessária a construção de uma mapa tridimensional representativo de um sistema multi-agv. Deste modo, inicialmente foi construído um mapa em FlexSim a partir da planta de chão do laboratório do INESC-TEC.

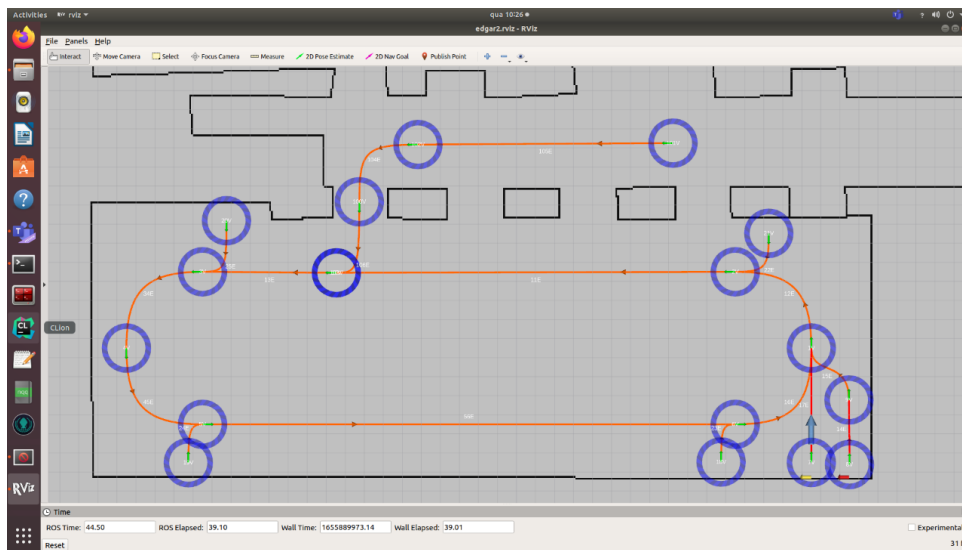


Figura 3.1: Mapa Snapshot em RVIZ

3.1.1 Ligações e Nós

Para a construção do mapa modelo, é inicialmente importante definir os caminhos nos quais os AGVs se podem movimentar. Deste modo foram definidos os nós e ligações do mapa com as ferramentas de *paths* e CP do FlexSim. Na construção dos *paths* e CP do mapa tridimensional do chão de produção tentou se respeitar as medidas reais, com uma escala de 1:2, ou seja um comprimento de 2 metros no simulador, representa um comprimento de 1 metro no sistema real.



Figura 3.2: Processo de construção dos *paths* e CP

O processo de decomposição em nós e caminhos no mapa passou por inicialmente se colocar o mapa obtido em RVIZ, no chão do 3D model do FlexSim, após os nós definidos no mapa serem posicionados com os CP respetivos, os restantes CP são colocados ao longo do caminho definido, sendo que o distanciamento entre os CP deve ser igual, ou seja os *paths* devem possuir todos o mesmo comprimento. Após o modelo tridimensional, dos caminhos possíveis dos AGVs, ser definido foram colocados 3 AGVs, que correspondentes a 3 CP iniciais distintos.

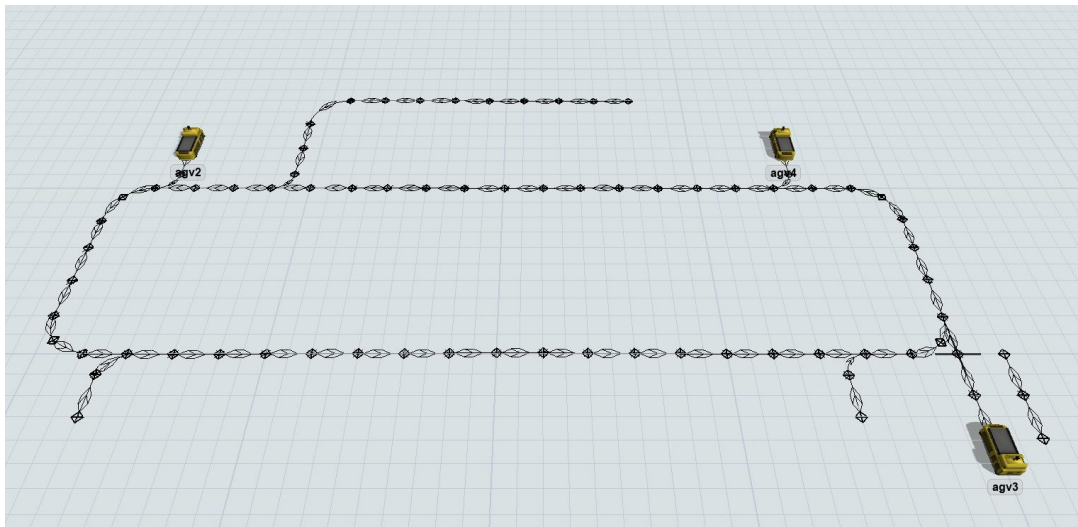


Figura 3.3: Modelo Tridimensional construído em FlexSim

3.2 Comunicação com Algoritmo

O sistema Simulador-Algoritmo, utiliza um arquitetura centralizada, , sendo que o algoritmo TEA*, é o agente central de decisão, que procura encontrar um rota otimizada de forma *online*, que vá ao encontro dos objetivos individuais de cada um dos AGVs.

Inicialmente o servidor começa por aceder e compilar de forma Off-line os ficheiros JSON de "Data Graph" e "Data Mission", esta fase garante ao algoritmo um conhecimento não só do mapa ("Data Graph"), bem como da posição dos diferentes AGVs e as diferentes tarefas que necessitam de ser realizadas ("DataMissions"). A comunicação com o agente central (algoritmo), é realizada por *sockets* com um modelo cliente (simulador) - servidor(algoritmo). A comunicação é iniciada pelo lado do servidor, que ficará à escuta, na *socket* 8888 no IP local-host (127.0.0.1 IPV4). Após o cliente (simulador) se conectar com o servidor por essa *socket*, e ser posteriormente aceite pelo servidor, pode-se dar início à troca de mensagens entre o algoritmo e simulador.

A troca de mensagens *online* é inicializada pelo Cliente (Simulador) que envia a primeira mensagem, nesta mensagem é realizado um pedido de trajetória para uma determinado ID de uma tarefa, referida em "DataMission". O algoritmo responde a esta mensagem com o "TaskAllocationResponse", que contém os ID dos CP que o AGV deve percorrer até ao CP final da tarefa. Após o AGV, chegar a este Control Point, o mesmo está pronto para realizar um carregamento, é nesta fase que é enviada para o algoritmo a mensagem "RobotUnavailable", que sinaliza ao algoritmo a indisponibilidade do AGV. Assim que a o carregamento é realizado, é enviado para o algoritmo a mensagem de "RobotAvailable", que informa o algoritmo que o robô terminou a tarefa de carregamento, e está pronto para prosseguir o transporte. O algoritmo responde ao "RobotAvailable" com a mensagem "TaskAllocationResponse", que retorna a rota de CP's, até à Rest Station de descarga, após está descarga estar concluída, é enviada novamente a mensagem de disponibilidade do AGV,

específico. Assinala-se que esta troca de mensagens entre o algoritmo e simulador é realizada não só de forma paralela para os diferentes AGVs, de forma a que exista uma coordenação multi-AGV, como também de forma cíclica, até que a conexão seja fechada ou que as tarefas sejam todas completadas e não seja enviada para o algoritmo nenhuma "TaskAllocationRequest".

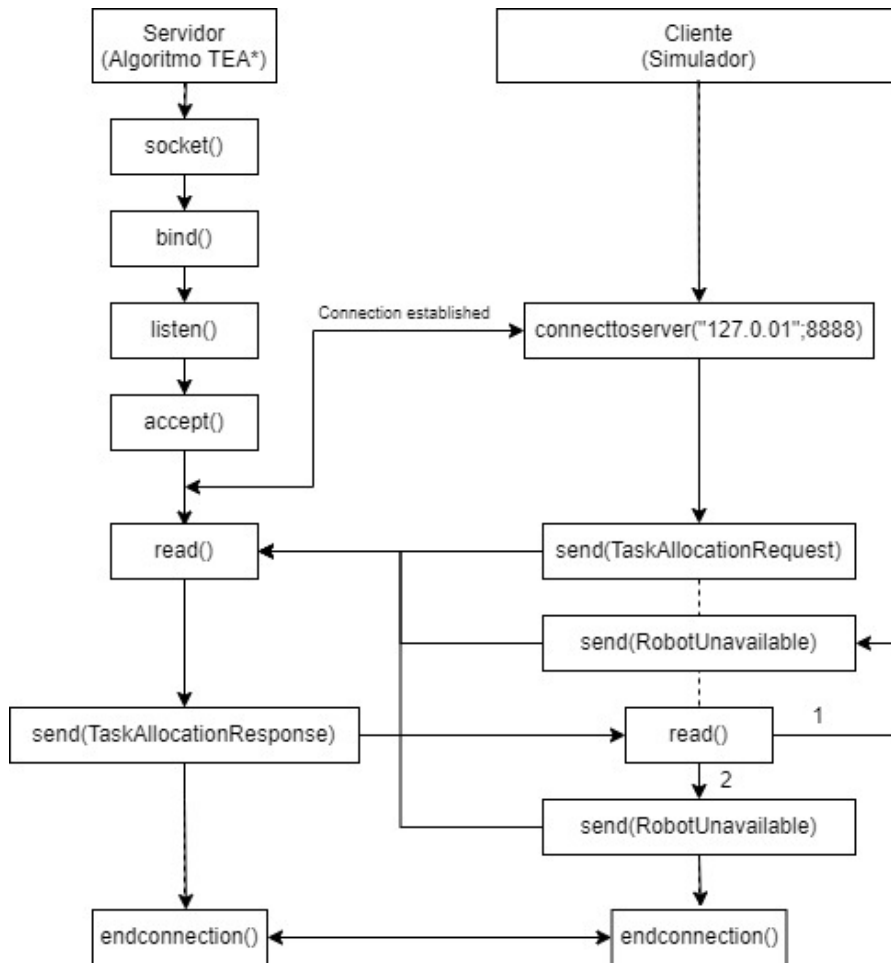


Figura 3.4: Diagrama de Comunicação Servidor(TEA*) - Cliente(Simulador)

3.2.1 Comunicações Off-line

Para um correto planeamento das trajetórias dos AGV, o algoritmo necessita de ter conhecimento de diferentes informações, não só do mapa-grafo do modelo tridimensional, como de diferentes informações relativas aos AGV e os seus posicionamentos no mapa, antes de os mesmos sequer iniciarem o seu trajeto. A obtenção desta informação e a sua posterior formatação em formato JSON, foi um processo que exigiu bastante trabalho, dadas as limitações de acesso presentes em FlexSim na obtenção de informação do mapa, estas limitações são explicadas ao longo desta secção. Neste subcapítulo são explicados, os processos necessários para a obtenção e formatação da informação necessária às mensagens de "DataGraph" e "Data Missions".

3.2.1.1 Informação do Mapa-Grafo

A mensagem "DataGraph", fornece ao algoritmo uma informação de todos os nós e ligações, presentes no mapa tridimensional por onde os AGV se poderam movimentar. A informação dos diferentes nós e ligações do mapa, como o seu posicionamento referencial, o numero de ligações em cada Control Point, comprimento dos "paths" e a sua bidirecionalidade, são algumas das informações essenciais ao algoritmo para o correto funcionamento do mesmo. O envio desta informação deve respeitar uma estrutura em formato JSON à qual o algoritmo está preparado para ler.

Nome: Datagraph

Descrição: Informação do mapa

Sentido da comunicação: Simulador->Algoritmo;

Trigger: Mensagem Enviada *Offline*

Estrutura:

- "MessageID"(string): "DataGraph";
- "Map"(JSON Object)
 - "Nodes"(JSON Array)
 - * "Id" (número): Node ID;
 - * "x" (número): XX cordenada da posição do nó;
 - * "y" (número): YY cordenada da posição do nó;
 - * "Number_of_Links" (número): Número de ligações conetadas ao nó;
 - "Links"(JSON Array)
 - * "Id" (Número): ID da ligação;
 - * "Node_1_Id" (Número): ID do primeiro nó da ligação;
 - * "Node_2_Id" (Número): ID do segundo nó da ligação;
 - * "Distance" (Número): Comprimento da ligação;
 - * "Bidirectional" (Número): 0 se for unidirecional, 1 se for bidirecional;

Nós: Para a organização e obtenção das informações dos CP/Nós, foram realizadas diferentes tarefas. Inicialmente, foi atribuída nas Labels de cada CP, um determinado número inteiro que definisse uma ID a cada um, isto permitiu que cada um destes CP tivesse uma identidade única e constante durante a sua vida útil. De forma semelhante, visto que o FlexSim não possui nenhuma informação, relativamente ao número de ligações conectadas a cada um dos CP, após diferentes tentativas a única solução encontrada, foi contabilizar o número de *links* de cada um dos CP e adicionar esse valor inteiro, às *labels* dos CP.

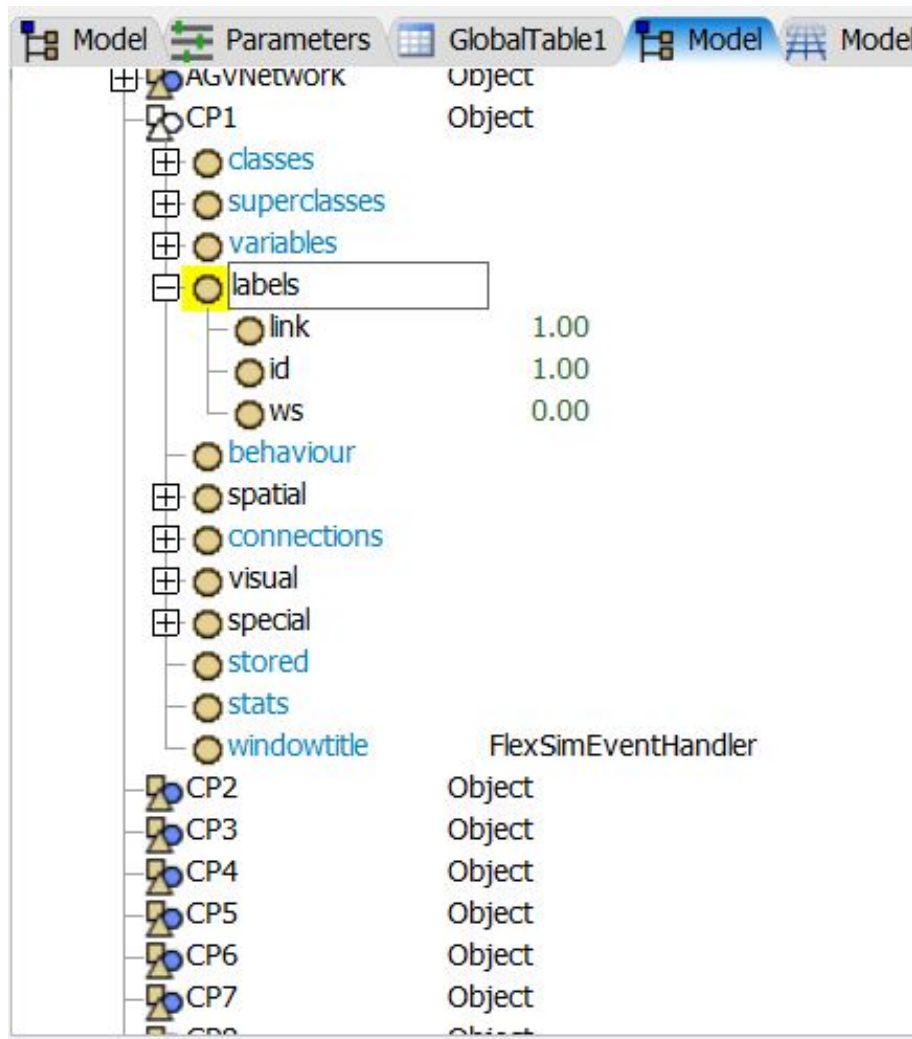


Figura 3.5: Treenode - Label do Control Point 1

Após esta definição de informação estar completa, no *script* responsável por formatar a informação "DataGraph", realizou-se um ciclo, que percorre a *treenode* do modelo FlexSim, selecionando apenas os objetos que possuíssem a string "CP", para cada um desses objetos é guardada num *array* associativo, `map[]`, as informações, relativamente ao seu ID, posição referencial (*X* e *Y*) e o número de ligações para cada um.

Ligações: Para a organização e obtenção das informações dos diferentes *paths*/ligações, de forma semelhante à dos CP começou por se atribuir um valor inteiro a cada um dos *paths*, que correspondê se ao seu ID.

Visto que o FlexSim, não possuía na sua *treenode* dos objetos, informação relativa ao valor do CP inicial (*node_i*) e final (*node_f*) de cada um dos *paths*/ligação, foi necessária inserir cada um desses valores em todos os *paths*, de forma a conseguir saber qual o ponto inicial e final de cada uma das ligações, esta atribuição de valor nas *labels* está representada na figura.

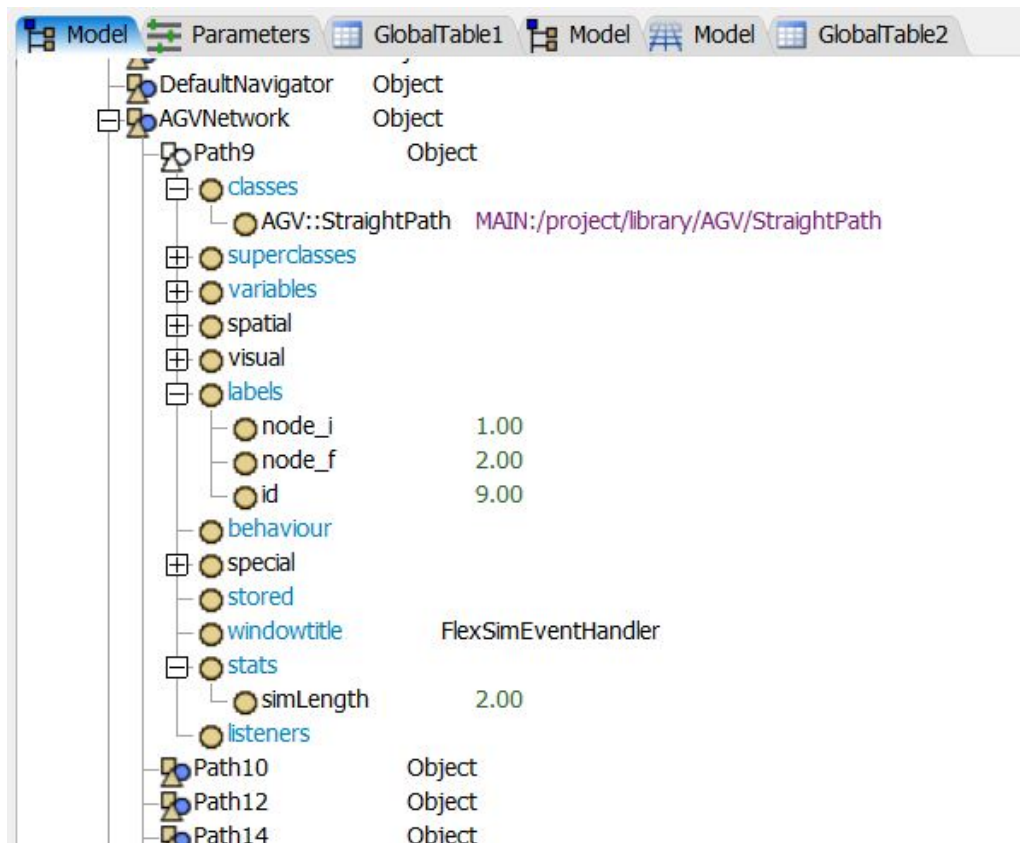


Figura 3.6: Treenode - Label do Path 9

Visto que se consegue obter a informação relativa ao comprimento e bidirecionalidade de cada um dos *paths* na *treenode* do FlexSim, de forma semelhante aos nós foi criado um ciclo *for()* que percorresse os sub-nós da *treenode* do objeto *AGVNetwork*, os diferentes valores de ID, *Node_1_Id*, *Node_2_Id*, Comprimento e Bidirecionalidade, são guardados num array associativo *map2[]*, que é depois "empurrado" para um novo array, *secArray*.

3.2.1.2 Data Mission

Nesta secção é analisada a metodologia utilizada para a organização das informações necessárias à mensagem "DataMission". De forma semelhante à informação do mapa-grafo, também a

informação presente na "DataMissions", exigiu uma inserção de informação que não se conseguiria obter pela treenode do FlexSim. Tal como a metodologia utilizada na formatação da mensagem "DataGraph", também esta mensagem respeita uma estrutura, que o algoritmo está preparado para ler.

Estrutura:

- "MessageID"(string):"DataMissions";
- "Mission"(JSON Object)
 - "Robot"(JSON Array)
 - * "Robot_Id" (número): ID do robô;
 - * "x" (número): Coordenada XX da posição inicial do robô;
 - * "y" (número): Coordenada YY da posição inicial do robô;
 - * "Direção" (número): Orientação inicial do robô;
 - "Rest Stations"(JSON Array)
 - * "node_id" (Número): ID do nó;
 - * "isactive" (Número): 0 se for um ponto de carregamento ou 1 se for um ponto de carga/descarga;
 - * "X" (Número): Coordenada XX do nó;
 - * "Y" (Número): Coordenada YY do nó;
 - "Tasks"(JSON Array)
 - * "Task_id" (Número): ID da tarefa;
 - * "priority" (Número): Prioridade da tarefa;
 - * "Nodes" (Array Numérico): Sequência de nós associados às tarefas de cada AGV;

3.2.1.3 Robô

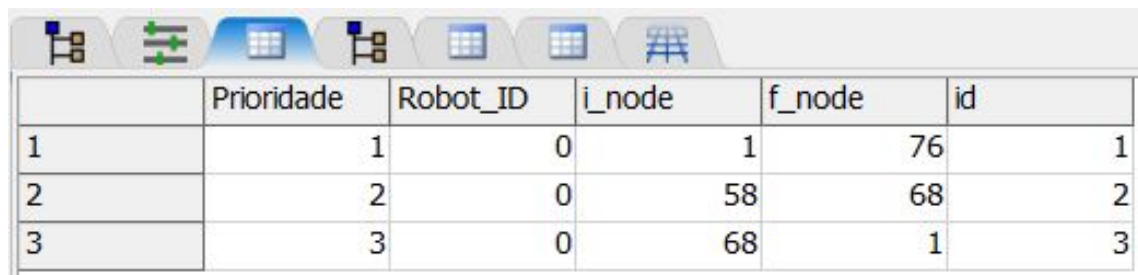
Relativamente à informação necessária dos Robôs/AGV, a obtenção das diferentes revelou-se bastante acessível, o seu posicionamento referencial (X e Y) bem como direção é informação que se obteve acedendo aos atributos de cada um dos AGV, bastando apenas ser atribuído um número inteiro nas *labels* de cada um dos AGV, que definisse um ID para os mesmos.

3.2.1.4 Rest Stations

De forma semelhante à obtenção de informação dos Nós/CP na mensagem "DataGraph", para a obtenção das informações das estações de repouso, onde os AGV poderam parar (Load/Unload), foi apenas necessário definir uma *label* para os CP que fossem representativos de estações de repouso.

3.2.1.5 Tasks

As tarefas que devem ser realizadas pelos robôs, foram inseridas numa Tabela Global no FlexSim, para cada uma destas tarefas é definida uma prioridade, ID e os CP que os AGV se devem deslocar para realizar as tarefas Load e Unload sequencialmente. Assim que o script da criação de mensagem de DataMission é inicializado, realiza-se uma leitura das diferentes colunas desta tabela, consoante a informação, que se pretende formatar. Na figura 3.7, está representado um exemplo de uma tabela de tarefas.



	Prioridade	Robot_ID	i_node	f_node	id
1	1	0	1	76	1
2	2	0	58	68	2
3	3	0	68	1	3

Figura 3.7: Exemplo de Tabela com a informação das Tarefas

3.2.2 Mensagens Online

De forma a que o algoritmo TEA* seja funcional e aplicado corretamente a um modelo multi-agv em FlexSim é essencial uma troca precisa e eficiente, entre o sistema servidor-cliente de forma *online*. Variações de posicionamento, pedidos de alocação de tarefas e informação relativamente à disponibilidade dos robôs, são mensagens que necessitam ser enviadas do modelo de FlexSim para o algoritmo, de forma a que o mesmo tenha um conhecimento alinhado com o *status-quo* do sistema, para que deste modo possa responder ao simulador as atuações a serem realizadas no modelo, para diferentes camadas temporais.

Dada esta informação, tornou-se necessário formatar 4 tipos de mensagem, o pedido de alocação de Tarefa ("TaskAllocationRequest"), a resposta de alocação de Tarefa ("TaskAllocationResponse"), a mensagem que informa que o robô que efetuou a deslocação está indisponível ("RobotUnavailable") e por fim a mensagem que informa ao algoritmo que o robô está disponível ("RobotAvailable"). Nas secções seguintes é realizada primeiramente uma análise da forma em que se obteve e que se formatou os dados relativos ao pedido de alocação de tarefas e de seguida é realizada a metodologia utilizada para a interpretação da resposta do algoritmo a esse pedido de alocação de tarefas.

3.2.2.1 Pedido de Alocação de Tarefa

Nesta primeira mensagem *online*, é transmitido ao algoritmo as diferentes tarefas associando a cada uma um determinado ID e uma prioridade. Além das tarefas a realizar é enviado também ao algoritmo uma informação do posicionamento dos AGVs. De forma semelhante à mensagen

de *Offline* "DataMission", o pedido de alocação de tarefa deve respeitar uma estrutura de formato JSON semelhante, à qual o algoritmo está pronto para receber.

Estrutura:

- "MessageId"(string) : "TaskAllocationRequest";
- "TaskID"(number) : ID da task a ser executada;
- "TaskType"(string) : "Entrada"/"Saída";
- "RobotID"(number): ID do AGV a realizar uma tarefa caso seja um pedido de carregamento de baterias ou estacionamento;
 - "RobotLocations"(JSON Array)
 - * "Robot_id" (número): ID do robô;
 - * "X" (número): Coordenada XX da posição inicial do robô;
 - * "Y" (número): Coordenada YY da posição inicial do robô;
 - * "Direção" (número): Orientação inicial do robô;
 - * "PrevNodeId"(número): ID do nó anterior do Robô;
 - * "IsAtNode"(número): 1 se estiver entre dois CP;
 - * "LinkId" (número): Localização do Path em que o AGV se encontra (Definido na Label do AGV);

3.2.2.2 Resposta de Alocação de Tarefa

A resposta de alocação de tarefa como referido anteriormente diz respeito à única mensagem enviada do algoritmo para o FlexSim e uma das principais já que uma das informações que envia é um grupo de IDs de CP (definido em "RobotRoutes") que no seu conjunto definem uma rota para diferentes em diferentes camadas temporais que um determinado AGV (definido no RobotID) deve adotar. Este array, é alocado à label do AGV.

O tipo de estrutura que o pedido de resposta de alocação de tarefa, é a seguinte:

Estrutura:

- "Messageid"(string) : "TaskAllocationResponse";
- "RobotID"(number) : ID do AGV deve realizar a tarefa (valores nulos informam o FlexSim que os AGVs se encontram todos ocupados)
- "RobotRoutes"(Array Numérico) : Este array define os IDs dos CP, pelas quais os AGVs devem seguir. Valores iguais e consecutivos de ID de CP significam uma manutenção da posição entre as duas camadas temporais;

3.2.2.3 AGV Disponível

Esta mensagem diz respeito à disponibilidade dos AGVs e é enviada sempre que um AGV termina uma tarefa, quer seja o carregamento ou descarregamento de uma peça. Nesta mensagem é enviado o ID do AGV que terminou essa determinada tarefa, bem como qual o tipo de tarefa que terminou ("LoadFinished"/"UnloadFinished"). A mensagem obedece a seguinte estrutura:

Estrutura:

- "MessageId"(string) : "RobotAvailable";
- "RobotID"(number) : ID do AGV disponível;
- "RobotLocations"(JSON Array)
 - "Robot_ID" (número) : ID do robô;
 - "X" (número) : Coordenada XX da posição inicial do robô;
 - "Y" (número) : Coordenada YY da posição inicial do robô;
 - "Direction" (número) : Orientação inicial do robô;
 - "PrevNodeId"(número) : ID do nó anterior do Robô;
 - "IsAtNode"(número) : ID do nó que o Robô se encontra;
 - "Linkid" (número) : Localização do caminho no FlexSim do AGV;

Ao contrário da "TaskAllocationRequest" que apenas necessitava de ler a *label* inicial do AGV onde tinha sido introduzida o Path em que o mesmo se encontra no início do ciclo, esta mensagem exige uma atualização deste valor de Path, de modo a que o mesmo represente a posição do AGV no momento em que terminou a tarefa.

3.2.2.4 AGV Indisponível

Esta mensagem tem como objetivo informar o algoritmo da indisponibilidade dos AGVs, e é enviada sempre que um determinado AGV chegou a uma Rest Station e está pronto para começar a realizar uma tarefa de carregamento ou de descarregamento ("LoadStarted" ou "UnloadStarted"). A mensagem obedece à seguinte estrutura:

Estrutura:

- "MessageId" (string): "RobotAvailable";
- "RobotID" (number): ID do AGV disponível;
- "State" (string): Estado do AGV que terminou a tarefa "LoadStarted", "UnloadStarted";

3.2.3 Process Flow

Neste secção é analisada a metodologia utilizada para a construção do PF 3.8, a introdução e posterior compreensão das capacidades desta ferramenta, foi um processo exigente. Dentro do modelo construído em FlexSim, o PF pode ser visto quase como um coordenador de tarefas. Nesta ferramenta do FlexSim foi construído um modelo, que vai definir não só, quando é que as mensagens *online* devem ser enviadas para o algoritmo ou lidas no caso da "TaskAllocationResponse", como também atuar como um controlador da movimentação dos AGV. É no PF que é realizado todo o processamento de troca de informações *online*, bem como a atuação dos AGV que deve ser realizada consoante as respostas que obtém do Algoritmo.

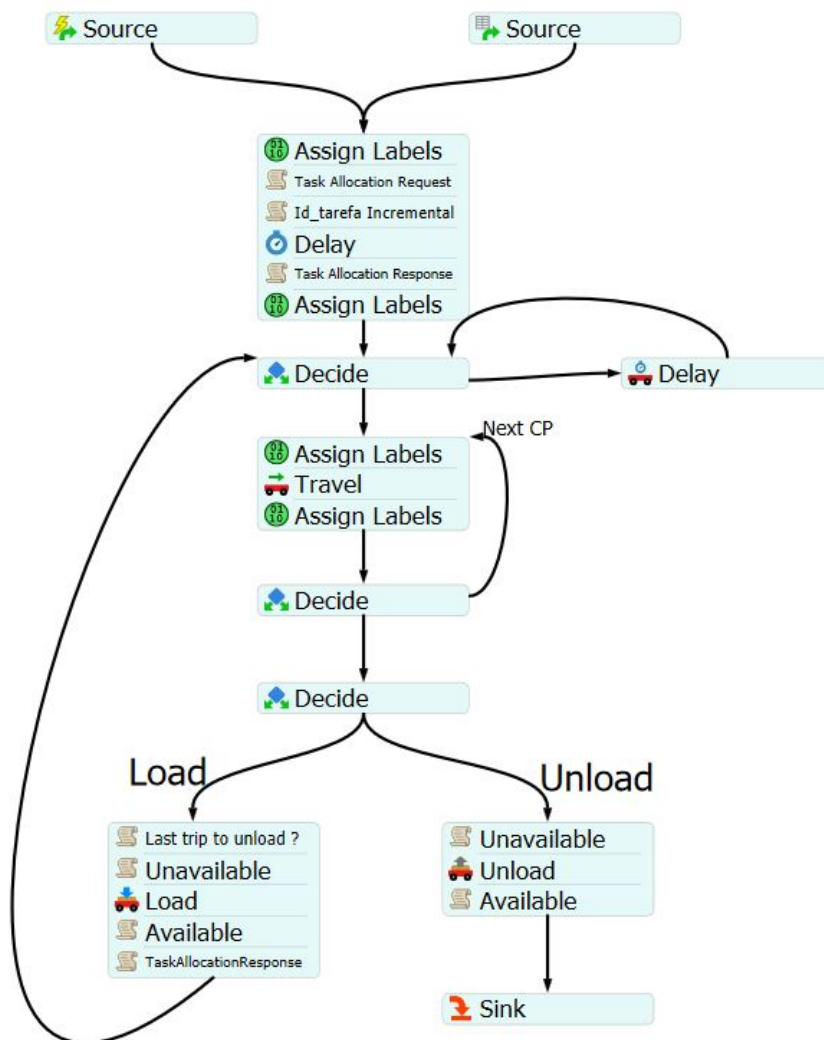


Figura 3.8: Modelo de PF de controlo do sistema multi-agv

3.2.3.1 Lógica do Process Flow

A lógica de PF acaba por ditar todo o comportamento do sistema, como referido anteriormente, o PF vai ditar as trocas de mensagem e as ordens ao modelo de simulação. O processo começa com a libertação de 3 token pelo bloco "Schedule Source" ou com o "Event Trigger Source" caso o PF já esteja ativo e haja a *sink* de um *token*, estas propriedades foram definidas em 3.10 e em 3.9, respetivamente.

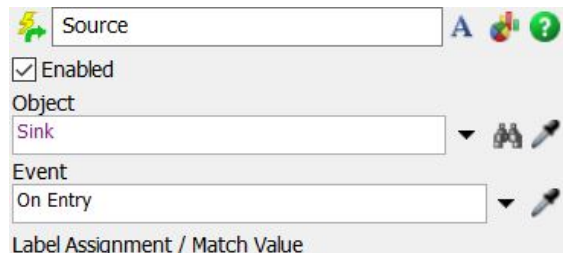


Figura 3.9: Propriedades do Event - Trigger Source

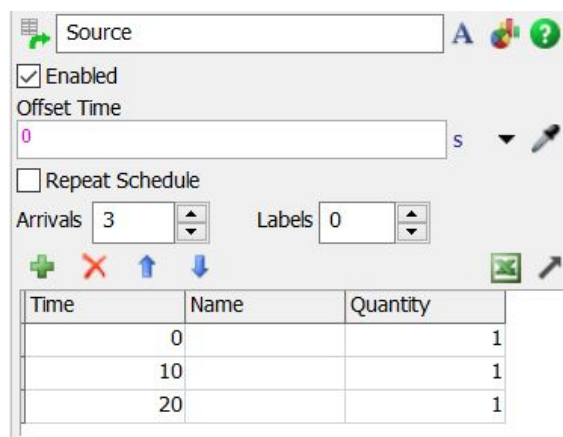


Figura 3.10: Propriedades da Schedule Source

Cada um dos blocos seguintes do PF são inicializados pela passagem do token, até ao mesmo chegar ao bloco final de "Sink" onde é extinto. O processo começa com a atribuição de uma label "Task_id" ao token, como demonstrado em 3.11, que vai definir a tarefa que o mesmo vai desencadear. Após esta atribuição é realizado a "TaskAllocationRequest", definida em A.3, que vai enviar ao algoritmo um pedido de resposta à tarefa correspondente à label do token ativou o bloco. Após ser realizada uma incrementação do valor da "Id_tarefa", o bloco "TaskAllocationResponse" (processo de leitura da mensagem em A.4) recebe do algoritmo, a atribuição da tarefa a um determinado AGV bem como a rota de CP que o mesmo deve percorrer, esta informação é atribuída à label do AGV à qual foi atribuída a tarefa, e o ID do AGV é atribuído à Label do Token, como representado em 3.12.

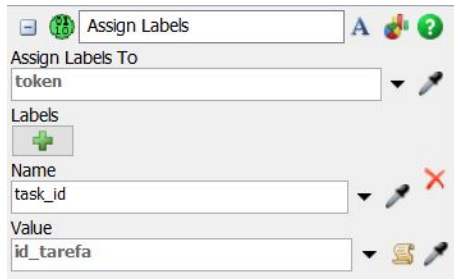


Figura 3.11: Propriedades do primeiro Assign Label

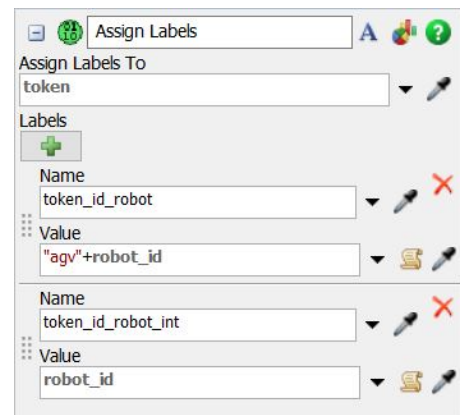


Figura 3.12: Propriedades do segundo Assign Label

O token, com as informações de ID da tarefa e ID do AGV, passa agora para o bloco "Decide" onde entra num ciclo temporal que apenas permite a movimentação dos AGV com uma periodização de 2,9 segundos, referente ao tempo de movimentação entre CP. Antes de entrar no bloco travel é feita a atribuição de uma label "is_at_node" de valor zero, que simboliza que o AGV, se encontra entre CP.

De seguida, o bloco "Travel" vai ler e retirar o primeiro valor do Array (3.14) presente na label do AGV, para o movimentar para esse CP e vai também atualizar os valores de "previous node" do AGV, esta implementação foi realizada como representado no *script* em A.5. Este processo é realizado de forma cíclica até que o *array* esteja vazio.

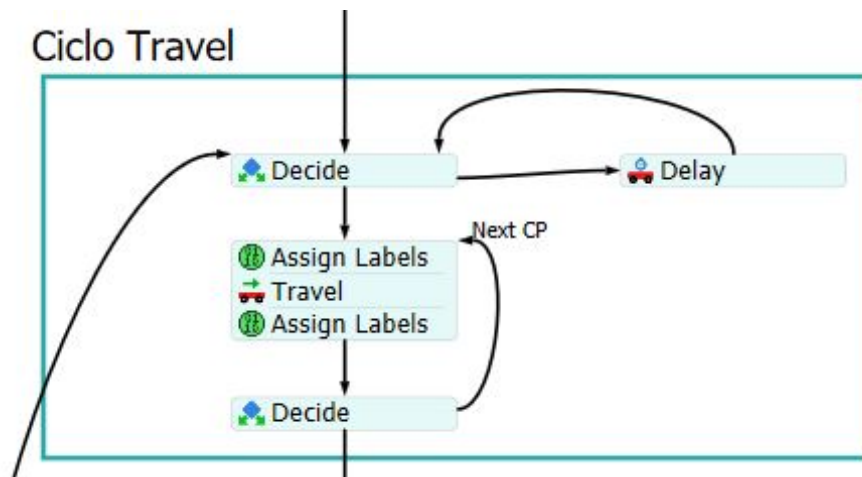


Figura 3.13: Ciclo de leitura do Array do AGV em PF

CP	58
id	3
Routes	Array[5]: {34, 77, 63, 62, 61}
prev_node	59
currentPath	19
curr_node	60

Figura 3.14: Label do agv2 - Travel de CP59->CP61

Assim que o tamanho do *array* é zero, significa que o AGV, chegou ao seu primeiro destino, a mensagem de "Unavailable" é enviada (implementação descrita em A.9) e o AGV começa o processo de "Load" da peça. Após a conclusão deste processo, o AGV esta novamente disponível para se deslocar ao destino onde deve descarregar a peça. Para isto envia a mensagem "Available" (implementação descrita em A.10), à qual o algoritmo responde com o "TaskAllocationResponse" (processo de leitura descrito em A.4). O ciclo do bloco "Travel" é novamente repetido, todavia quando o AGV, chega ao destino final, a variável global referente ao AGV "agv_x_proc" (x, igual ao valor ID do AGV, referente), vai canalizar o *token* no bloco inferior "Decide" para o bloco "Unload" onde o AGV realiza o descarregamento da peça. Consequentemente a mensagem "Available" é enviada e o *token* é suprimido no bloco "Sink". Este processo pode ocorrer de forma cíclica e paralela sempre que existe a libertação de um *token*, podendo haver até 3 *tokens* no PF, correspondente a 3 AGV a realizarem as tarefas.

3.2.4 Conclusão

O capítulo, anterior tinha como principal objetivo possibilitar, a compreensão da metodologia utilizada para implementar o FlexSim com o Algoritmo TEA*, referindo algumas das dificuldades e abordagem de integração utilizada para implementação do algoritmo com o simulador.

Capítulo 4

Caso de Estudo

Este capítulo tem como principal objetivo comparar o modelo multi AGV em que foi implementado o algoritmo TEA*, com um novo modelo em que é implementado o algoritmo A*, que está presente como ferramenta de otimização de rotas dentro do FlexSim. O algoritmo A* no FlexSim utiliza uma grelha de nós através dos quais os AGVs se movem. O algoritmo analisa os nós na direção da viagem e determina qual é a direção mais rápida, incluindo o movimento diagonal entre os nós. A grelha pode ser modificada, ou criando barreiras que restringem as opções de movimento, ou através de Mandatory Paths que definem trajetórias que os mesmos se podem deslocar.

Numa primeira parte do capítulo são descritas algumas das considerações e implementações que tiveram de ser realizadas para a construção do novo modelo em que as rotas dos AGV são definidas com recurso a um algoritmo A*, de seguida é realizada uma avaliação dos tempos de realização e eficiência das tarefas atribuídas aos dois modelos e por fim é realizada uma comparação dos resultados obtidos entre os dois modelos.

4.1 Modelo TEA*

Nesta secção foi analisado o comportamento do modelo desenvolvido, é realizada uma análise prática do comportamento do sistema para diferentes tipos de tarefas, e uma recolha de informação do tempo e espaço percorrido pelos AGV.

4.2 Análise do modelo - 1 Tarefa de *Load*

Como primeira análise prática foi avaliada a resposta do sistema para apenas uma alocação de tarefa. De forma a analisar a performance dos 3 AGV foram realizadas 3 análises de alocação de 1 tarefa de *Load*.

4.2.0.1 Load in CP55

Como primeiro caso avaliou-se a resposta do sistema para o pedido de uma tarefa que representa um carregamento no CP55. Para um pedido de tarefa no CP55, como demonstrado na figura 4.1, o algoritmo realiza uma análise de alocação de tarefa a um determinado AGV baseada na distância euclidiana, e atribui ao AGV2 a rota de CP's demonstrada na figura 4.2. A demonstração da simulação esta representada em 4.5.

```
Awaiting Client Data...
Client: {"TaskAllocationRequest":{"RobotID":0,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5}, {"Direction":89.757451555185881,"IsAtNode":1,"LinkId":19,"PrevNodeId":58,"RobotID":3,"X":-1.0141634319668762,"Y":-3.5011149738960148}, {"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}], "TaskID":0, "TaskType": "Entrada"}}
```

Figura 4.1: Pedido de alocação de tarefa para o CP55

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":2,"RobotRoutes":[{"RobotID":1,"RobotRoute":[]}, {"RobotID":2,"RobotRoute": [1,2,3,75,69,68,67,66,76,53,52,54,55]}, {"RobotID":3,"RobotRoute":[]}]}
```

Figura 4.2: Resposta ao pedido de alocação CP55

Para esta tarefa o algoritmo realizou a alocação da mesma ao AGV mais próximo, e este AGV realizou o caminho mais curto para o mesmo em 37 segundos com uma distância percorrida de cerca de 21,5 metros. Estes resultados estão demonstrados nas figuras 4.4 e 4.3.

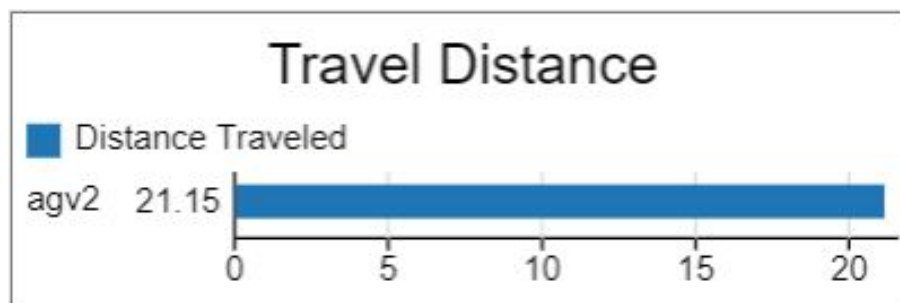


Figura 4.3: Distância percorrida pelo AGV

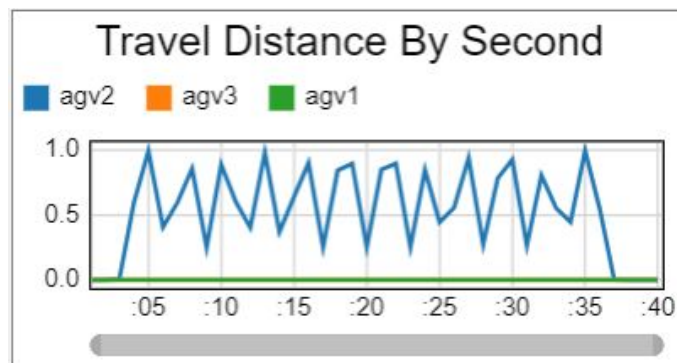


Figura 4.4: Distância percorrida p/ segundo pelo AGV2

A demonstração da simulação a decorrer está representado em 4.5, onde o AGV chega ao CP55 destino.

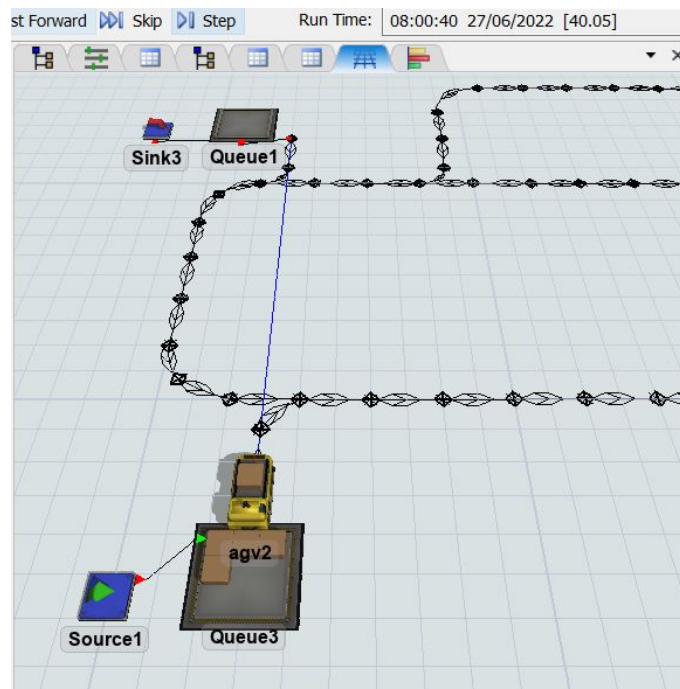


Figura 4.5: Frame de demonstração da simulação

4.2.0.2 Load in CP61

Como segundo caso avaliou-se a resposta do sistema para o pedido de uma tarefa que representa um carregamento no CP61. Para um pedido de tarefa no CP61, como demonstrado na figura 4.6, o algoritmo realiza uma análise de alocação de tarefa a um determinado AGV baseada na distância euclidiana, e atribui ao AGV2 a rota de CP demonstrada na figura 4.7. A demonstração da simulação está representada em 4.10.

```
Client: {"TaskAllocationRequest":{"RobotID":0,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5},{"Direction":89.757451555185881,"IsAtNode":1,"LinkId":19,"PrevNodeId":58,"RobotID":3,"X":-1.0141634319668762,"Y":-3.5011149738960148},{"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}], "TaskID":0,"TaskType":"Entrada"}}
```

Figura 4.6: Pedido de alocação de tarefa para o CP61

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":3,"RobotRoutes":[{"RobotID":1,"RobotRoute":[]}, {"RobotID":2,"RobotRoute":[]}, {"RobotID":3,"RobotRoute":[58,58,59,60,34,77,63,62,61]}]}
```

Figura 4.7: Resposta ao pedido de alocação CP61

Para esta tarefa o algoritmo realizou a alocação da mesma ao AGV mais próximo, e este AGV realizou o caminho mais curto para o mesmo em cerca de 21,1 segundos com uma distância percorrida de cerca de 12,85 metros. Estes resultados estão demonstrados nas figuras 4.8 e 4.9.



Figura 4.8: Distância pelo AGV3 para a deslocação até CP61

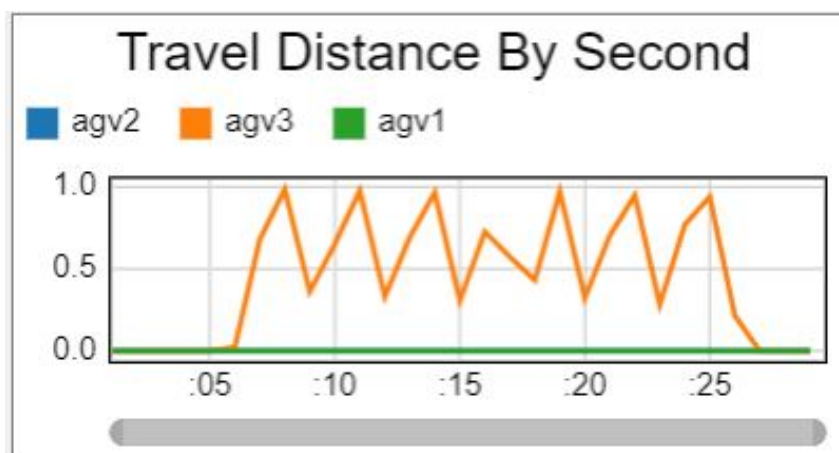


Figura 4.9: Distância percorrida p/segundo pelo AGV3 para a deslocação até CP61

A demonstração da simulação a decorrer está representado em 4.10, onde o AGV3 chega ao CP61 destino.

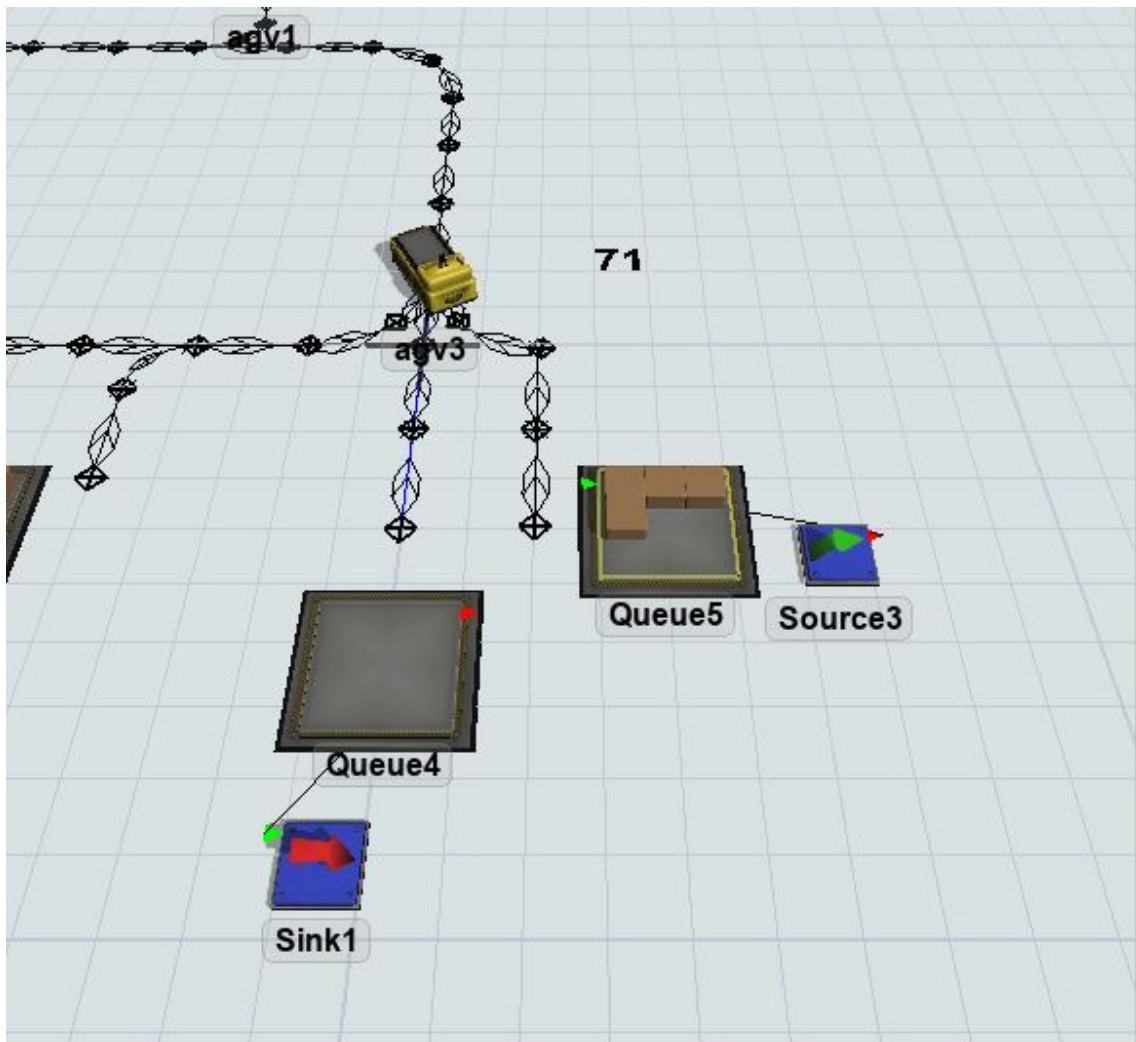


Figura 4.10: Chegada do AGV2 ao CP61

4.3 Análise do modelo - 2 Tarefas de Load

4.3.1 Load - 2 CP Diferentes

Neste caso foram pedidos duas tarefas de carregamento nos control points 61 e 57. O processo começou pelo pedido das duas tarefas ao algoritmo. Para cada um destes pedidos de tarefa (demonstrado em 4.11), o algoritmo realizou inicialmente a alocação de tarefas, sendo que atribuiu ao AGV3 a alocação ao ponto 61 e a tarefa no CP57 ao segundo robô mais disponível e mais próximo.

```
Client: {"TaskAllocationRequest":{"RobotID":0,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5},{"Direction":89.757451555185881,"IsAtNode":1,"LinkId":19,"PrevNodeId":58,"RobotID":3,"X":-1.0141634319668762,"Y":-3.5011149738960148},{"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}], "TaskID":0,"TaskType":"Entrada"}}
```

Figura 4.11: Pedido de alocação de tarefa para o CP61

```
Client: {"TaskAllocationRequest":{"RobotID":0,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5},{"Direction":89.757451555185881,"IsAtNode":1,"LinkId":19,"PrevNodeId":58,"RobotID":3,"X":-1.0141634319668762,"Y":-3.5011149738960148},{"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}], "TaskID":1,"TaskType":"Entrada"}}
```

Figura 4.12: Pedido de alocação de tarefa para o CP57

A resposta do algoritmo para a tarefa no CP61 e CP55, está demonstrada respectivamente em [4.13](#) e [4.14](#).

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":3,"RobotRoutes":[{"RobotID":1,"RobotRoute":[]},{ "RobotID":2,"RobotRoute":[]},{ "RobotID":3,"RobotRoute": [58,58,59,60,34,77,63,62,61]}]}}
```

Figura 4.13: Resposta ao pedido de alocação de tarefa para o CP61

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":1,"RobotRoutes":[{"RobotID":1,"RobotRoute": [64,65,19,20,21,74,31,32,33,34,73,35,36,56,57]},{ "RobotID":2,"RobotRoute": []},{ "RobotID":3,"RobotRoute": [58,58,59,60,34,77,63,62,61]}]}}
```

Figura 4.14: Pedido de alocação de tarefa para o CP57

Para estas tarefas o algoritmo realizou a alocação das mesmas aos AGV mais próximos AGV3 e AGV1, que realizaram o caminho mais curto para o mesmo em cerca de 21,1 e 40 segundos respectivamente com uma distância percorrida de 12,85 metros para o AGV3 e 25,6 metros para o AGV1. Estes resultados estão demonstrados em [4.15](#) e [4.16](#).

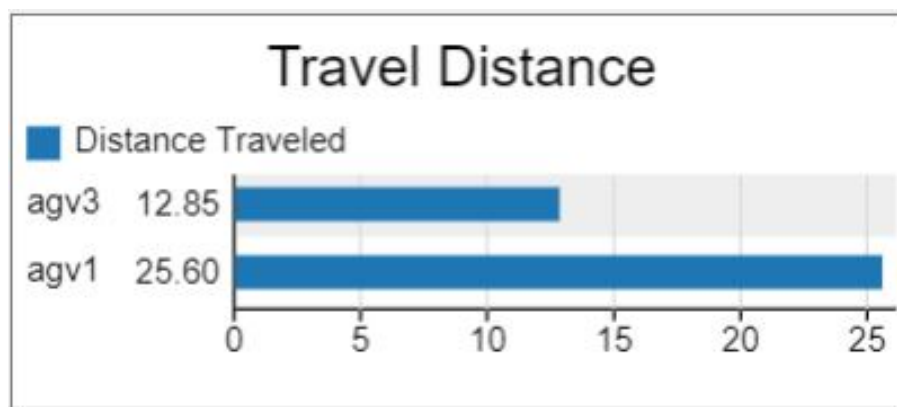


Figura 4.15: Distância percorrida pelo AGV3 e AGV1

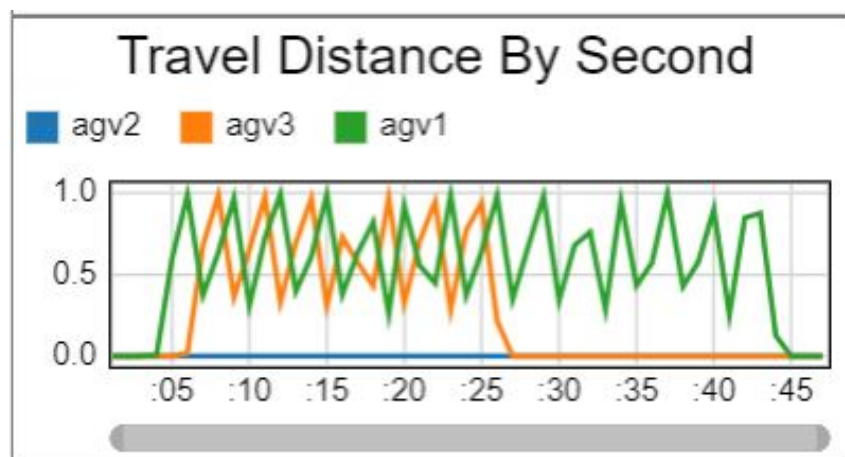


Figura 4.16: Distância percorrida por segundo pelo AGV3 e AGV1

A demonstração da simulação a decorrer está representado em 4.17, onde os AGV, 3 e 1 se deslocam para os destinos anteriormente mencionados.

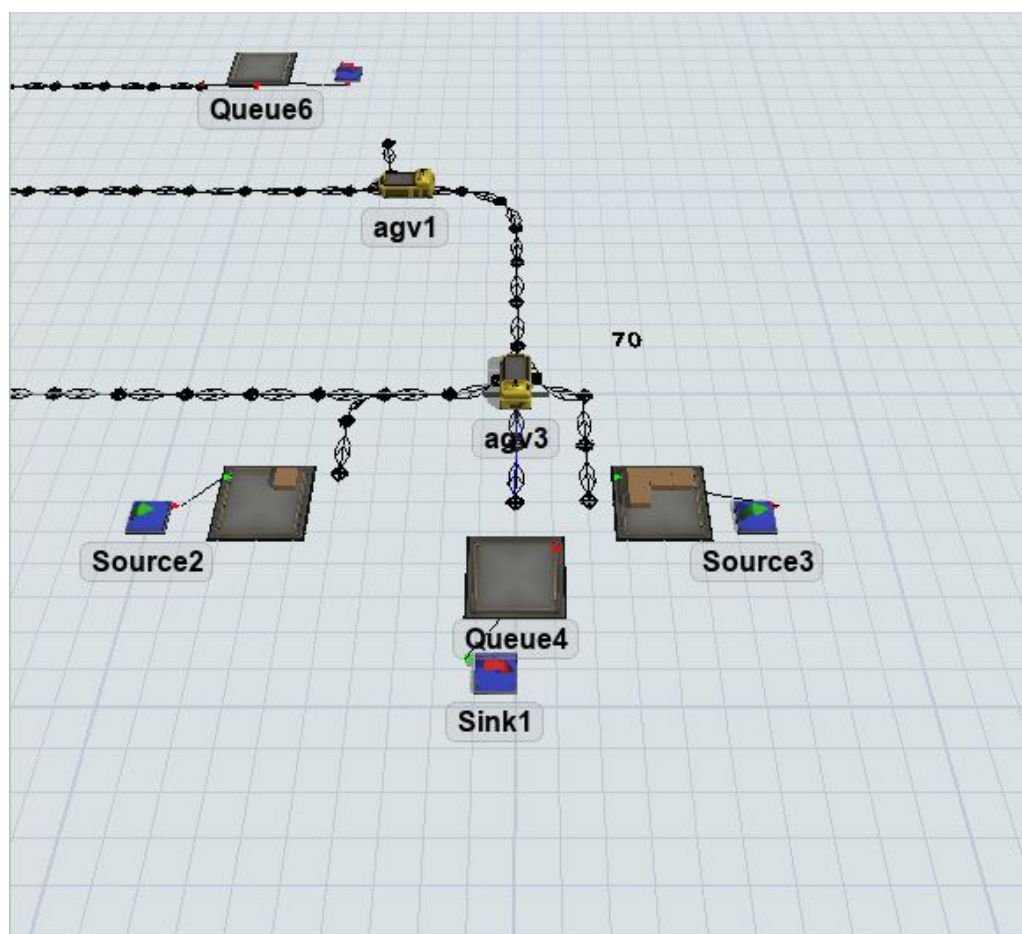


Figura 4.17: Chegada do AGV2 ao CP55

4.3.2 Load de 3 CP's

Neste caso foram pedidas 3 tarefas de carregamento nos CP 57 com TaskID=0, 61 com TaskID=1 e 55 com TaskID=2. O processo começou pelo pedido das 3 tarefas ao algoritmo, respectivamente como demonstrado nas figuras 4.18, 4.19 e 4.20. Para cada um destes pedidos de tarefa, o algoritmo realizou inicialmente a alocação de tarefas, de forma sequencial a cada um dos pedidos, sendo que atribuiu à primeira tarefa o AGV3, à segunda o AGV2 e por fim à última o restante AGV1.

```
Client: {"TaskAllocationRequest":{"RobotID":0,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5},{"Direction":89.757451555185881,"IsAtNode":1,"LinkId":19,"PrevNodeId":58,"RobotID":3,"X":-1.0141634319668762,"Y":-3.5011149738960148},{"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}], "TaskID":0,"TaskType":"Entrada"}}
```

Figura 4.18: Pedido de Alocação de Tarefa para a TaskID=0

```
Client: {"TaskAllocationRequest":{"RobotID":0,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5},{"Direction":89.757451555185881,"IsAtNode":1,"LinkId":19,"PrevNodeId":58,"RobotID":3,"X":-1.0141634319668762,"Y":-3.5011149738960148},{"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}], "TaskID":1,"TaskType":"Entrada"}}
```

Figura 4.19: Pedido de Alocação de Tarefa para a TaskID=1

```
Client: {"TaskAllocationRequest":{"RobotID":0,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5},{"Direction":89.757451555185881,"IsAtNode":1,"LinkId":19,"PrevNodeId":58,"RobotID":3,"X":-1.0141634319668762,"Y":-3.5011149738960148},{"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}], "TaskID":2,"TaskType":"Entrada"}}
```

Figura 4.20: Pedido de Alocação de Tarefa para a TaskID=2

Para cada uma destes pedidos de resposta o algoritmo retornou os CP, seguinte:

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":3,"RobotRoutes":[{"RobotID":1,"RobotRoute":[]}, {"RobotID":2,"RobotRoute":[]}, {"RobotID":3,"RobotRoute":[58,59,60,34,73,35,36,56,57]}]}}
```

Figura 4.21: Resposta ao Pedido de Alocação de Tarefa para a TaskID=0

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":1,"RobotRoutes":[{"RobotID":1,"RobotRoute":[64,65,19,20,21,74,31,32,33,34,77,63,62,61]}, {"RobotID":2,"RobotRoute":[]}, {"RobotID":3,"RobotRoute":[58,59,60,34,73,35,36,56,57]}]}}
```

Figura 4.22: Resposta ao Pedido de Alocação de Tarefa para a TaskID=1

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":2,"RobotRoutes":[{"RobotID":1,"RobotRoute":[64,65,19,20,21,74,31,32,33,34,77,63,62,61]}, {"RobotID":2,"RobotRoute":[1,2,3,75,69,68,67,66,76,53,52,54,55]}, {"RobotID":3,"RobotRoute":[58,59,60,34,73,35,36,56,57]}]}}
```

Figura 4.23: Resposta ao Pedido de Alocação de Tarefa para a TaskID=2

Para chegarem todos os AGV aos destinos, os AGV demoraram no total cerca de 47 segundos para cumprirem os seus deslocamentos, sendo que este foi o tempo que o AGV1 demorou a chegar ao CP61. As distâncias percorridas e a velocidade ao longo tempo de cada um estão representados nas tabelas 4.24 e 4.25.

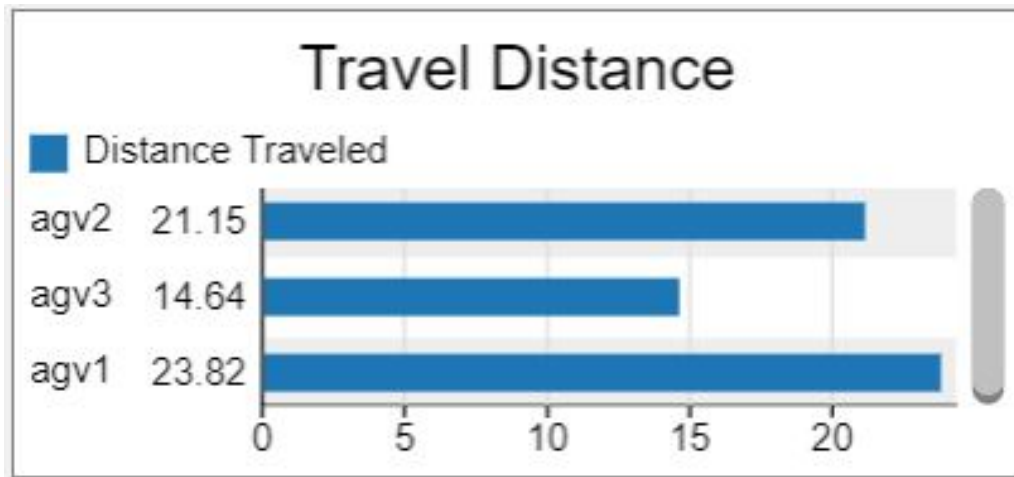


Figura 4.24: Distância percorrida pelos 3 AGV

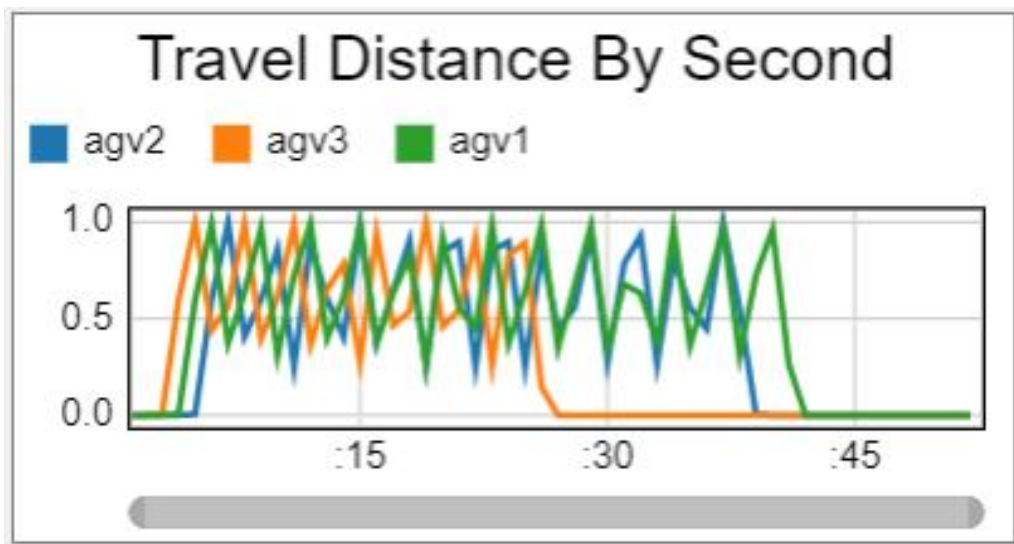


Figura 4.25: Distância percorrida por segundo pelos AGV

A demonstração da simulação a decorrer está representado em 4.26, onde os AGV 1 e 2 se deslocam para os destinos anteriormente mencionados e AGV3 já se encontra no CP55.

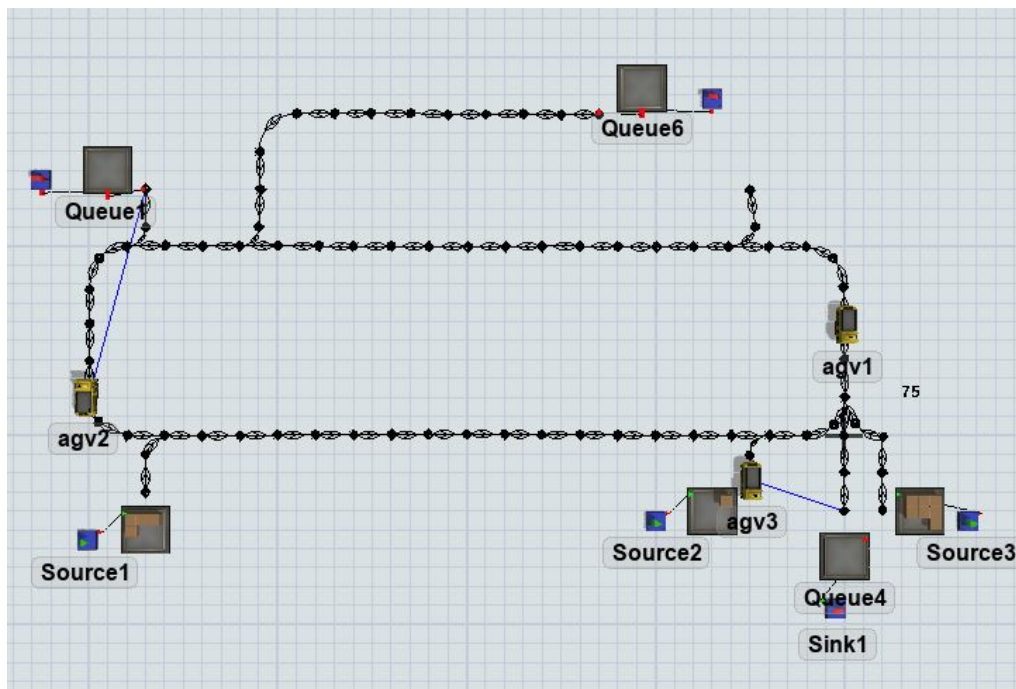


Figura 4.26: Chegada dos AGV 3 ao CP57

4.3.3 Load e Unload - Control Points

Para este caso não foi possível obter resultados, após diferentes pedidos de tarefas para os 3 pontos de *Load* e para diferente simultaneidade entre eles, o modelo de integração desenvolvido, não foi capaz de fornecer resposta uma resposta correta para os pedidos de *Unload*. Para um pedido de realização de tarefa de *Load* no CP61 e *Unload* no CP57, como demonstrado em ??, o AGV realizou a movimentação até ao CP de *Load* todavia, após ser feito o pedido de alocação de tarefa (4.27) a resposta CP61 que se obteve 4.28 não foi a esperada, visto que deveria retornar os CP's até ao ponto CP57, a paragem do AGV, está demonstrada em 4.31. Quando o AGV chega

```
Client: {"TaskAllocationRequest":{"RobotID":0,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5}, {"Direction":89.757451555185881,"IsAtNode":1,"LinkId":19,"PrevNodeId":58,"RobotID":3,"X":-1.0141634319668762,"Y":-3.5011149738960148}, {"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}], "TaskID":0, "TaskType": "Entrada"}}
```

Figura 4.27: Pedido de alocação

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":3,"RobotRoutes":[{"RobotID":1,"RobotRoute":[]}, {"RobotID":2,"RobotRoute":[]}, {"RobotID":3,"RobotRoute": [58, 58, 59, 60, 34, 77, 63, 62, 61]}]}}
```

Figura 4.28: Resposta ao pedido de alocação

ao CP61 onde deve realizar o *Load* de uma peça, após ser enviada a mensagem de *Unavailable*

e posteriormente terminar o carregamento da peça, é enviada a mensagem *Available* 4.29, que informa o algoritmo que o AGV está disponível para se deslocar para o ponto de *Unload*.

```
Client: {"RobotAvailable":{"RobotID":3,"RobotLocations":[{"Direction":-90,"IsAtNode":1,"LinkId":9,"PrevNodeId":1,"RobotID":2,"X":-38,"Y":13.5}, {"Direction":89.9999999999999901,"IsAtNode":1,"LinkId":20,"PrevNodeId":61,"RobotID":3,"X":1.0198884918819077,"Y":-3.4517056626530471}, {"Direction":-81.620216860269295,"IsAtNode":1,"LinkId":22,"PrevNodeId":64,"RobotID":1,"X":-5.9935222141940692,"Y":13.455938853686895}]}}
```

Figura 4.29: Envio da mensagem available para o algoritmo

```
Final Message to CESE: {"TaskAllocationResponse":{"RobotID":3,"RobotRoutes":[{"RobotID":1,"RobotRoute":[]}, {"RobotID":2,"RobotRoute":[]}, {"RobotID":3,"RobotRoute":[61]}]}}
```

Figura 4.30: Resposta à mensagem available

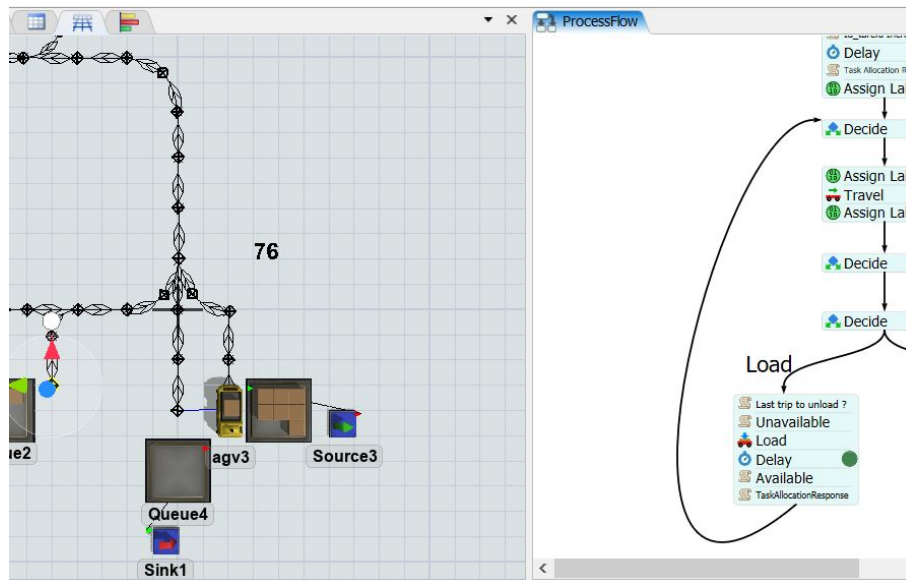


Figura 4.31: Chegada do AGV2 ao CP61

4.4 Construção do modelo A*

Para efeitos de avaliação comparativa entre os dois algoritmos na implementação do algoritmo A* ao modelo multi-AGV do FlexSim, foi utilizado o mesmo mapa de trajetos utilizado no modelo de implementação TEA* bem como o mesmo número de AGV, como representado em 4.32

- 3 AGV
- 6 CP representativos de carga/descarga

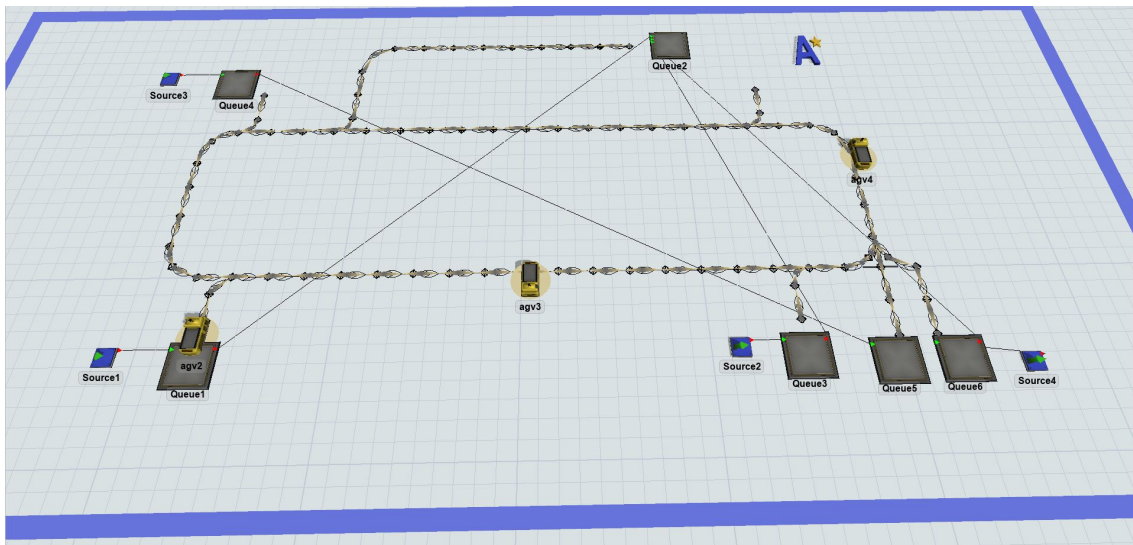


Figura 4.32: Modelo Tridimensional com recurso ao Algoritmo A*

Para a utilização do algoritmo A* presente no FlexSim, foi inicialmente necessário, utilizar as ferramentas de A* Navigation presentes na biblioteca do modelo tridimensional.

Primeiramente começou por se definir os trajetos possíveis para os AGV, semelhante ao utilizado no modelo anterior, sendo que neste modelo ao invés de serem utilizados os CP e *paths*, foram utilizados Mandatory Paths, que como referido anteriormente definem os trajetos obrigatórios nos quais os AGV se poderão movimentar, este mapa de trajetos foi construído dentro uma área que delimita o espaço em que os task executers podem utilizar a ferramenta A* navigation, este modo de navegação é definido no menu de propriedades de cada AGV.

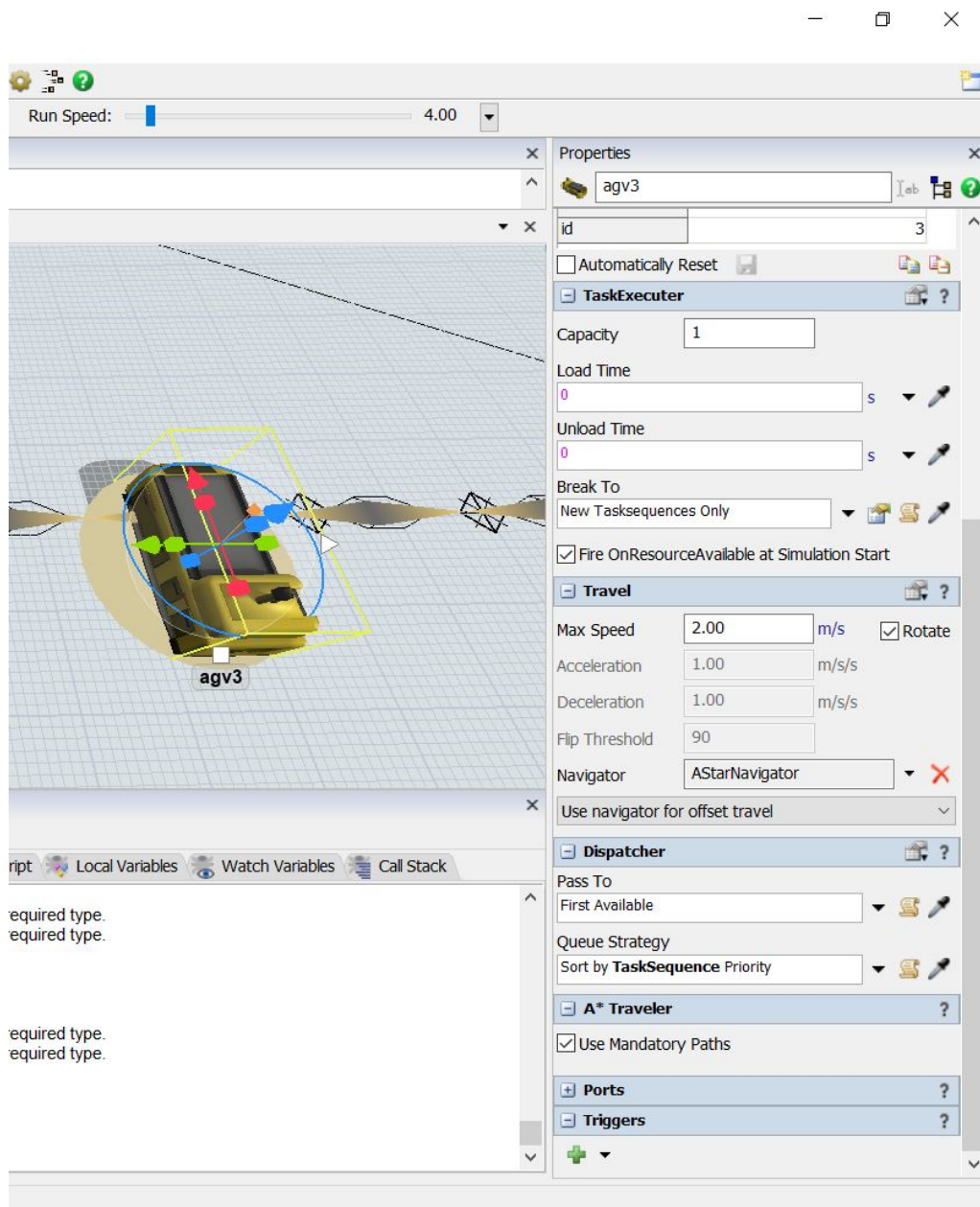


Figura 4.33: Propriedades do AGV3

4.4.1 Alocação de Tarefas

Para a atribuição de tarefas dos AGV, houve duas possibilidades de construção do modelo de simulação.

Numa das possibilidades o modelo poderia ser realizado com recurso de *source* e *sinks*, onde o primeiro libertaria peças, a um ritmo que poderia ser controlado e que deveriam ser transportadas por cada um dos AGV, todavia este tipo de modelo poderia ser considerado limitativo no momento em que fosse requerido uma análise mais escrutinada do modelo multi-AGV com re-

curso do algoritmo A*, na medida em que não havia forma de ordenar os AGVs para se dirigirem para determinados CP. Deste modo foi construído com recurso da ferramenta PF ordens de tarefas para cada um dos AGVs, este tipo de desenho, permitiu que numa comparação de resultados os dois modelos, A* e TEA*, pudessem ser submetidos a condições semelhantes de forma a serem comparados com um maior rigor.

Na metodologia utilizada para este modelo, cada um dos PF, diz respeito a cada um dos AGVs. De forma a facilitar a leitura pelos blocos "transport", utilizou se um *script* que lê e aloca a uma tabela, um array numérico, que diz respeito aos ID de diferentesCP para onde o AGV desse PF específico deve passar. Os blocos de transporte fazem a leitura de cada uma das células correspondentes dos AGV, e ordenam a movimentação dos mesmos.

4.4.1.1 Simulação com alocação de CP61

Como primeira situação foi avaliada, a resposta do modelo à ordenação do AGV3 para o CP61.

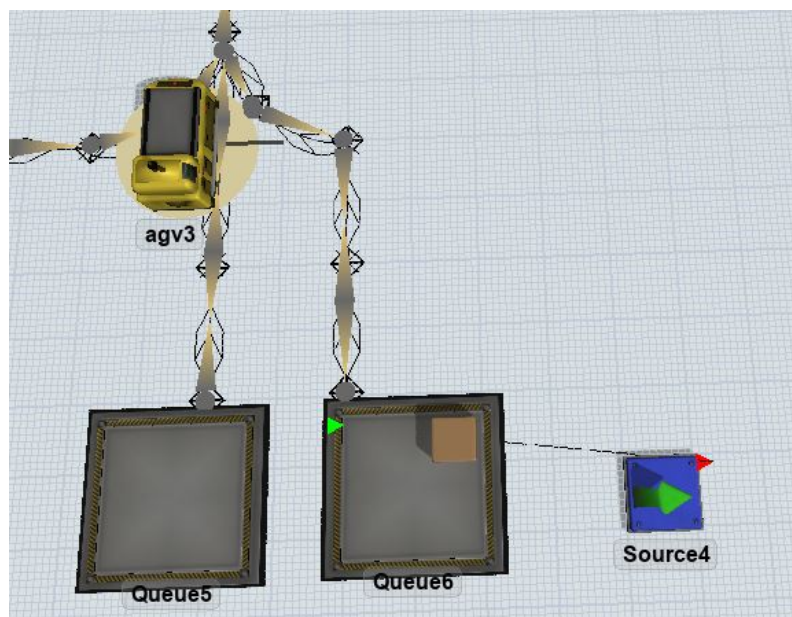


Figura 4.34: Simulação do modelo com A* com alocação de CPs - Deadlock

Da análise que se fez do modelo retirou-se que o AGV3 percorreu cerca de 15,24 metros, num espaço de 16 segundos. Como representado na figura 4.35.

4.4.1.2 Simulação com alocação de CP55 e CP61

De seguida tal como avaliado no modelo TEA*, avaliou-se a resposta do modelo à ordenação do AGV3 para o CP61 e do AGV4 para o CP57. Reparou-se que o AGV4, tomou a rota mais longa de forma a evitar o contacto com o AGV3, tendo tido uma distância percorrida de 82,65m.

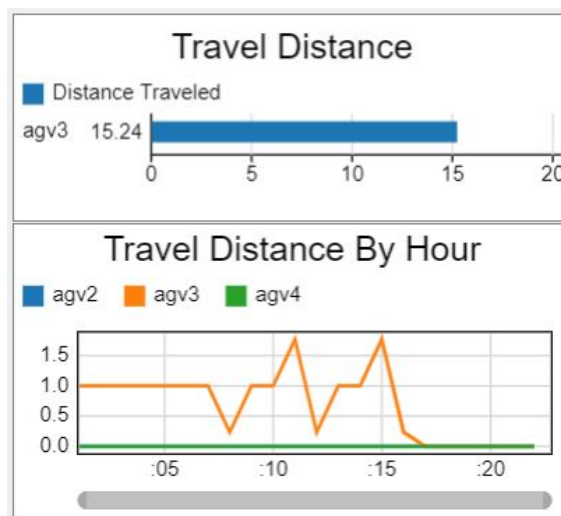


Figura 4.35: Simulação do modelo com A* com alocação de CP61

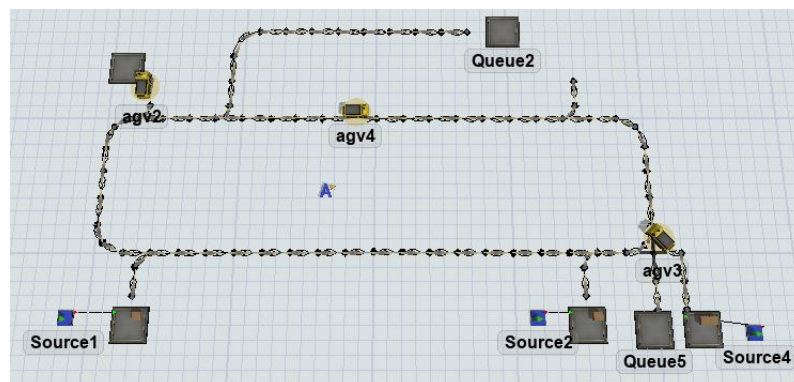


Figura 4.36: Simulação do modelo com A* com alocação de CPs - Deadlock

Da análise que se fez do modelo retirou se que o AGV3 percorreu 15,24 e o AGV4 acabou por percorrer uma distância muito superior de 82,65 metros.

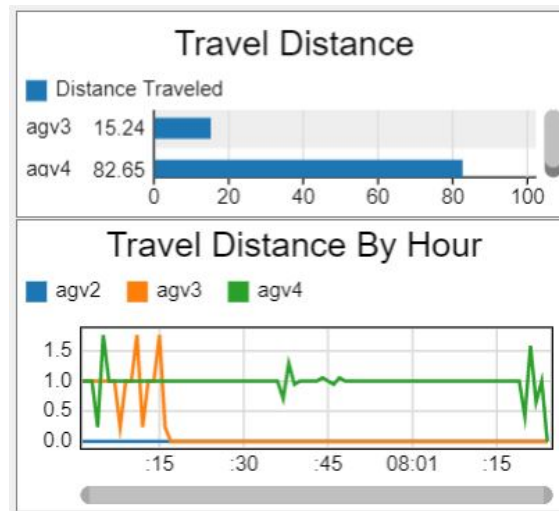


Figura 4.37: Simulação do modelo com A* com alocação de CPs - Deadlock

4.4.1.3 Simulação com alocação de CP55,CP61 e CP57

Foi também avaliado neste modelo tal como no TEA*, a resposta à ordenação do AGV3 para o CP61 e do AGV4 para o CP55 e AGV2 para CP55.

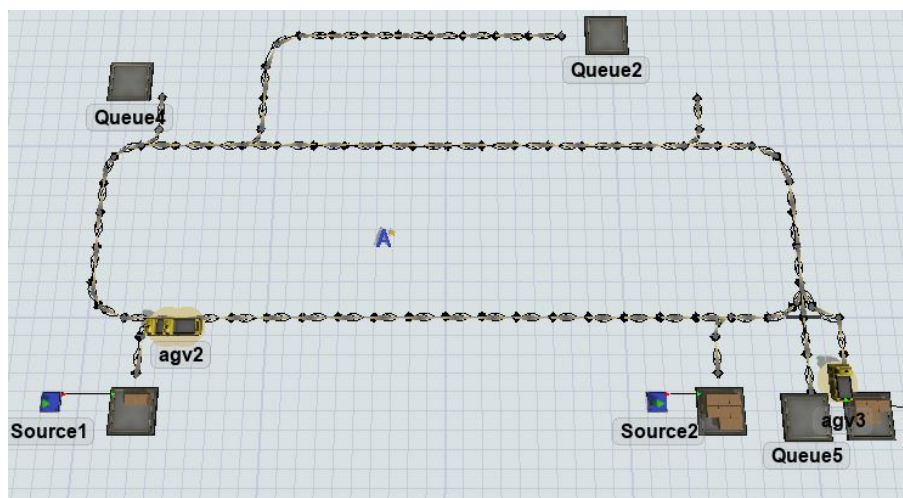


Figura 4.38: Simulação do modelo com A* com alocação de CPs - Deadlock

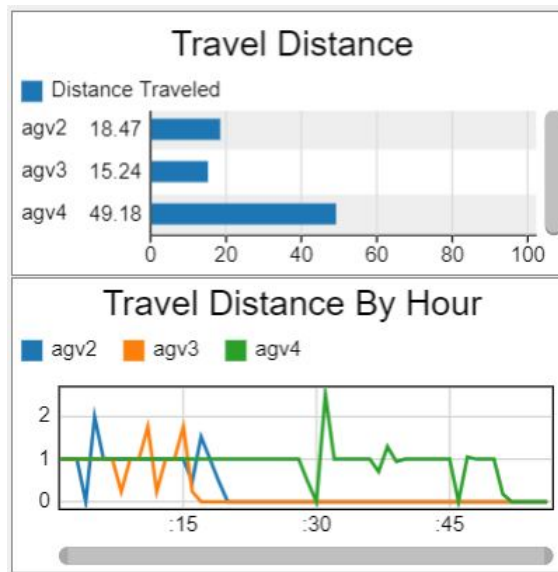


Figura 4.39: Resultados da simulação do modelo com A* com alocação de CPs - Deadlock

4.4.1.4 Alocação de vários CPS

A título exemplificativo das limitações deste Algoritmo A* realizou-se um teste onde foram ordenados a cada um dos AGVs com o ID 2, 3 e 4 que começassem nos Control Points CP55, CP44, e CP31 respetivamente e que os mesmos passassem por determinados Control Points:

- AGV2: CP23->CP40->CP50->CP45->CP23;
- AGV3: CP15->CP28->CP9->CP62->CP30;
- AGV4: CP55->CP43->CP23->CP28->CP13;

Desta simulação a uma run speed de 4 display units/s, resultou uma situação já esperada de deadlock, ao segundo 95, com uma distância média percorrida pelos AGVs de cerca de 182,5 metros.

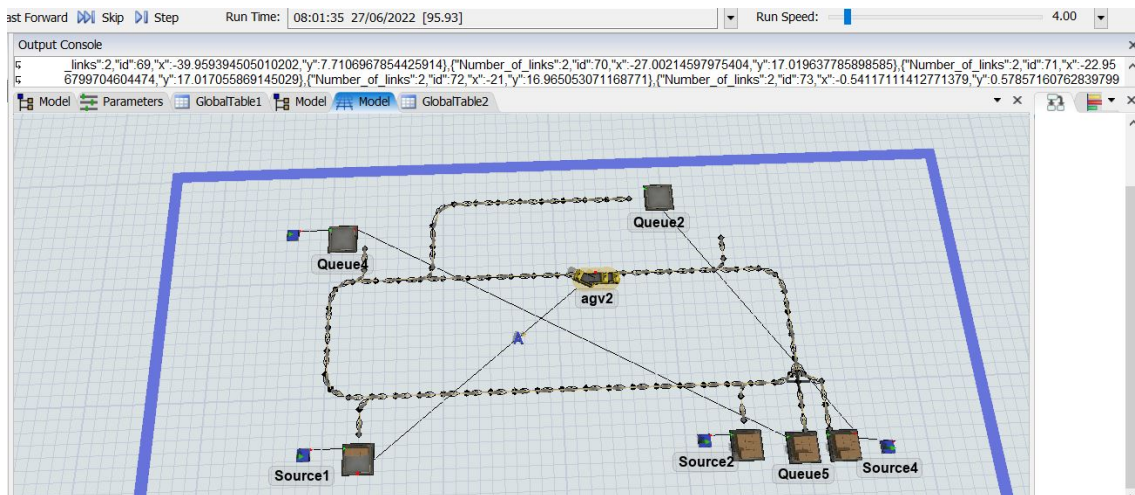


Figura 4.40: Simulação do modelo com A* com alocação de CPs - Deadlock

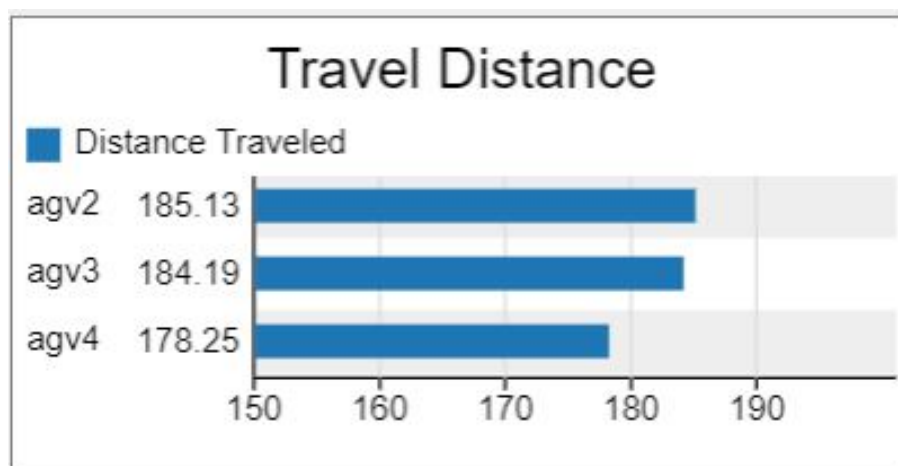


Figura 4.41: Simulação do modelo com A* - Distância percorrida pelos AGVs

4.4.1.5 Simulação com alocação de Source/Sinks

Esta simulação foi realizada com recurso a 3 source e 3 sinks, cada uma das sources tinha uma taxa de libertação de peça superior à velocidade de transporte de forma a criar uma situação de trabalho contínuo nos AGVs. Desta simulação a uma run speed de resultou também um deadlock perto de 17 horas de trabalho .

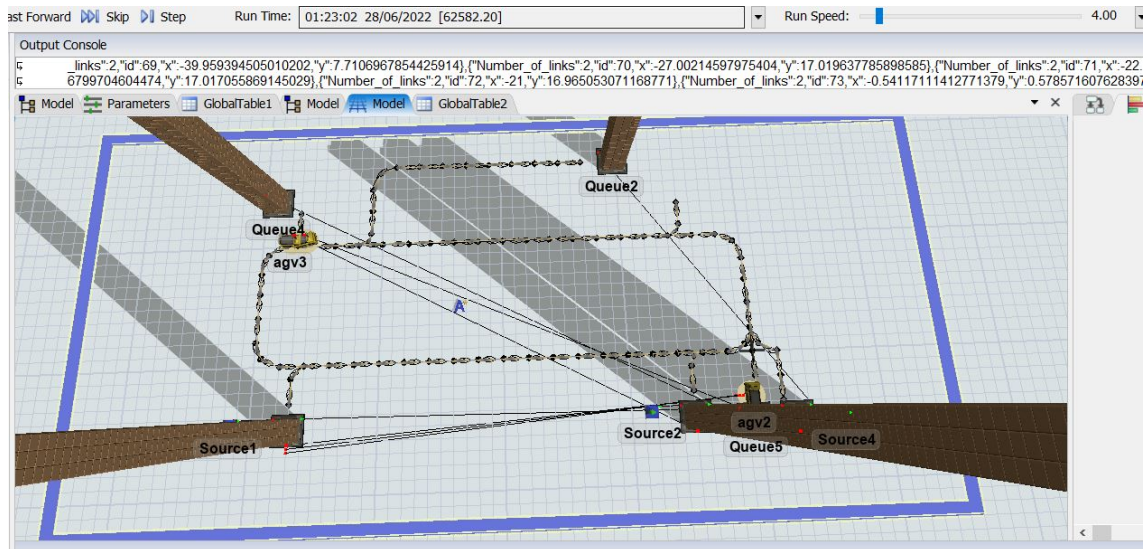


Figura 4.42: Simulação do modelo com A* com recurso de Sink/Source - Deadlock

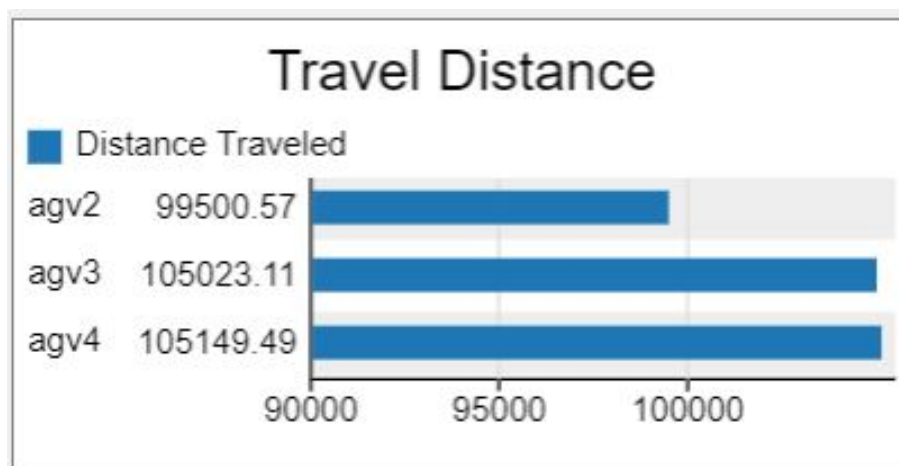


Figura 4.43: Simulação do modelo com A* com recurso de Sink/Source - Distância percorrida pelos AGVs

4.5 Comparação de Resultados

Da análise dos resultados anteriormente obtidos conseguem se comprovar algumas limitações no Algoritmo A* presente no FlexSim. No modelo desenvolvido em que se integrou o algoritmo TEA*, ainda que não se tenha conseguido realizar as operações de *Unload*, neste caso a alocação de tarefas é um processo que funciona de forma automático, processo esse que não acontece no Algoritmo A*, que necessita da construção de uma heurística que cumpra esses requisitos. No caso em que existe apenas alocação de um CP ao AGV no algoritmo A*, o mesmo acabou por se demonstrar mais vantajoso, pelo facto de ter percorrido uma distância mais curta num espaço de tempo mais curto, associada ao facto de o AGV seguir um Mandatory Path ao contrário do que acontece no algoritmo TEA* em que existe um seguimento de uma rota por cada CP que o mesmo vai passar. Por outro lado alocando mais do que uma tarefa ao sistema e mantendo os mesmos CP destino, que se alocou no modelo TEA*, o algoritmo A* demonstrou ser bastante menos eficiente. No caso em que existiu alocação de dois CP's o mesmo acabou por tomar uma rota muito mais longa quando comparado com o modelo TEA*. No casos em que existiu alocação de mais 3 ou mais tarefas ou no caso em que se realizou a alocação de tarefas pelas *Sources/Sinks*, o deadlock foi inevitável.

Capítulo 5

Conclusões e Trabalho Futuro

Este capítulo, sumariza as conclusões retiradas ao longo do trabalho desenvolvido nesta dissertação, bem como algumas considerações importantes que deverão ser tomadas para o futuro melhoramento do trabalho que foi desenvolvido neste projeto.

5.1 Conclusões do Trabalho Desenvolvido

O trabalho desenvolvido pode garantir algumas conclusões acerca da implementação do Time-Enhanced A* no software FlexSim.

Ao nível da construção do modelo tridimensional, o processo revelou-se bastante acessível e simples, as ferramentas do FlexSim proporcionam um ambiente bastante intuitivo para a construção deste modelo.

Ao nível da recolha dos dados necessários às trocas de mensagens entre o simulador e o algoritmo, este processo revelou-se um pouco mais exigente. Tanto a obtenção dos número de ligações aos CP do simulador, bem como o ID dos CP conectados a cada um dos *paths* foram processos demorados que exigiram uma inserção individual e manual, nas *labels* de cada um desses objetos. Relativamente aos *Current CP*, *Previous CP* de cada um dos AGV's, foram também dados de difícil acesso no FlexSim que obrigaram à construção de *scripts* no modelo. Ao nível dos *Current Paths* dos AGV, este dado não era possível ser obtido na versão inicial de FlexSim que estava a ser desenvolvida (22.0.4), o que obrigou a uma atualização para a versão(22.2) onde já era possível obter os *Paths* Atuais dos AGV, ainda assim este dado era obtido como string e obrigou à criação de um processo de alteração para valor inteiro em que o algoritmo estava preparado para receber. Ao nível de toda a obtenção e organização de dados no FlexSim, conclui-se que é um processo exigente que necessita um conhecimento profundo de todas a ferramentas do FlexSim.

Comparativamente com o modelo A* já integrado no simulador como uma ferramenta, a integração de um algoritmo *outsorce* ao software obrigou a uma recolha e formatação de diferentes informações, não só relativa aos AGVs, como de todo o mapa tridimensional dentro do modelo. Esta preparação e tratamento de dados dentro do FlexSim que necessitam de ser entregues ao algoritmo respeitando certas formatações, pode ser visto como uma desvantagem ao nível da facilidade

de implementação do mesmo, quando comparado com o modelo A* já integrado no software. Por outro lado ao nível da operabilidade e comportamento do sistema multi-AGV, ainda que não tenha sido possível concretizar planeamento de tarefas Pós-carregamento a simulação com o algoritmo TEA*, apresenta se mais vantajosa pelo facto de fazer uma alocação de tarefas consoante a distância dos AGV ao destino além disso existe uma constante prevenção das situações *Deadlock* e consequentemente mais eficiente, já que o algoritmo realiza toda uma planeamento de rotas ao longo de diferente camadas temporais, quando comparado com a simulação A*, este tipo de algoritmo exige um agendamento *offline* de pré-atuação de tarefas ou uma heurísticas associadas ao mesmo que se revela ineficientes em sistemas multi-AGV quando comparadas com o algoritmo TEA*, que possui um conhecimento temporal e posicional dos AGV.

Em suma, a elevada exigência, não só na obtenção de informação assim como na formatação da mesma, para ser entregue ao algoritmo TEA*, revelaram se nesta dissertação bastante demoradas, ainda assim comparado com o modelo simulado com algoritmo A* apresentou resultados bastante mais satisfatórios.

5.2 Trabalho Futuro

Para futuras iterações ao trabalho desenvolvido nesta dissertação sugere se, ao nível da construção de um modelo de simulação, um desenvolvimento do modelo nas versões mais recentes do FlexSim, já que o software é constantemente atualizado e pode permitir uma maior facilidade em futuros processos de integração. Recomenda se também a complementação da tarefa "Unload" que não foi possível concretizar bem como a introdução de novas categorias de tarefas ao modelo como a funcionalidade de Park ou Carregamento de Baterias, que se aproximem de situações reais. Sugere se também um aumento do número de AGV e uma complexidade de trajetórias superior, de forma a reforçar a validação do algoritmo num sistema simulado mais próximo da realidade industrial.

Ao nível da complexidade e eficiência temporal do algoritmo TEA* sugere se também a possibilidade de ser capaz de poder operar com AGVs em diferentes velocidades, de maneira a reduzir o número de ajustes de rotas realizadas pelos robôs e o espaço percorrido pelos mesmos e também uma adaptação do mesmo a robôs com diferentes configurações de rodas que obriguem a ajustes de direção, e um consequente ajuste do planeamento de rotas.

Anexo A

A.1 DataGraph

```
        double client = getnodenum(node("/Tools/client",model()));
Map map;
Map map2;
Map map3;
Map map4;
Map map5;
Map map6;
Map map7;
treenode net = Model.find("AGVNetwork");
Array myArray= Array(0);
Array secArray= Array(0);
for(int i = 0; i<content(model); i++)
{
string text = rank(model,i).name;
if(text.includes("CP") == 1)
{
map["Id"]= model.subnodes[i].as(Object).id;
map["x"] = model.subnodes[i].as(Object).location.x;
map["y"] = model.subnodes[i].as(Object).location.y;
map["Number_of_Links"]= model.subnodes[i].as(Object).link;
myArray.push(JSON.parse(JSON.stringify(map)));

}
else{
continue;
}
}
for(int s = 1; s<=net.subnodes.length; s++)
```

```

{
map2["Id"]= net.subnodes[s].id;
map2["Node1_Id"] = net.subnodes[s].node_i;
map2["Node2_Id"] = net.subnodes[s].node_f;
map2["Distance"] = getvarnum(net.subnodes[s], "length");
map2["Bidirectional"] = getvarnum(net.subnodes[s], "isTwoWay");
secArray.push(JSON.parse(JSON.stringify(map2)));
//j=j+1;
}
map3["Node"]= myArray;
map4["Link"]= secArray;
map5["Nodes"]= map3;
map5["Links"]= map4;
map6["Map"]= map5;
string str=JSON.stringify(map6);
print(str);
clientsend(client,str);
return client;

```

A.2 DataMission

```

//double client = getnodenum(node("/Tools/client",model()));
//treenode op = Model.find("AGV3");
//AGV agv = AGV(op);
Map map1;
Map map2;
Map map3;
Map map4;
Map map5;
//treenode node = Model.find("AGVNetwork>variables/agvs/AGV3");
//treenode nextCP = getsdtvalue(node,"nextCPPreArrivalPoint");
treenode net = Model.find("AGVNetwork");

int j=1;
int f=1;
Array myArray= Array(0);
Array buff= Array(0);
Array rest= Array(0);
Array nod= Array(2);

```

```

for(int d = 1; d<=content(model); d++)
{
string text1 = rank(model,d).name;
if(text1.includes("CP") == 1 && (model.subnodes[d].ws)==1)
{
map2["node_id"]= model.subnodes[d].id;
map2["ws_id"] = f;
map2["isactive"] = 0;//Is active ?
map2["X"]= model.subnodes[d].as(Object).location.x;
map2["Y"]= model.subnodes[d].as(Object).location.y;
rest.push(JSON.parse(JSON.stringify(map2)));
f=f+1;

}
else{
continue;
}
}
for(int i = 1; i<=content(model); i++)
{
string text = rank(model,i).name;
if(text.includes("agv") == 1) //getname(Model.find("").subnodes[i])
{
//treenode nextCP = getsdtvalue(subnode[i],"nextCPPreArrivalPoint");
map1["robot_id"]= model.subnodes[i].id;
map1["x"] = model.subnodes[i].as(Object).location.x;
map1["y"] = model.subnodes[i].as(Object).location.y;
map1["Direction"]=0;
map1["node"]= model.subnodes[i].CP;
//map1["next"] = nextCP;
myArray.push(JSON.parse(JSON.stringify(map1)));

//print(myArray[j]);

//clientsend(client,);
}
else{
continue;
}
}
}

```

```

int rows=Table("GlobalTable1").numRows;
for(int s=1; s<=rows; s++){
map3["Task_id"]= Table("GlobalTable1")[s][5];
//map3["Robot_ID"] = Table("GlobalTable1")[s][2];
map3["priority"] = Table("GlobalTable1")[s][1];
//COMO COLOCAR OS DOIS NODES NUM SO MAP
nod[1]=(Table("GlobalTable1")[s][3]);
nod[2]=( Table("GlobalTable1")[s][4]);
map3["Nodes"]=nod;
//map3["Nodes"]= Table("GlobalTable1")[s][3] Table("GlobalTable1")[s][4];
//map3["Nodes"]=Table("GlobalTable1")[s][4];
buff.push(JSON.parse(JSON.stringify(map3)));

}
map4["Reststation"]= rest;
map4["Tasks"]= buff;
map4["Robot"]= myArray;
map5["Mission"]=map4;
print(JSON.stringify(map5));

```

A.3 TaskAllocationRequest

```

/**Custom Code*/
double client = getnodenum(node("/Tools/client",model()));
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);
Map map1;
Map map2;
Map map3;
Map map4;
Map map5;
Array myArray= Array(0);
Array secArray= Array(0);
double curr;
//map2["MessageID"]="TaskAllocationRequest";
for(int i = 1; i<=content(model); i++)
{
string text = rank(model,i).name;

```

```

if(text.includes("agv") == 1) //getname(Model.find("").subnodes[i])
{
//treenode nextCP = getsdtvalue(subnode[i],"nextCPPreArrivalPoint");
map1["RobotID"]= model.subnodes[i].id;
map1["X"] = model.subnodes[i].as(Object).location.x;
map1["Y"] = model.subnodes[i].as(Object).location.y;

Object agv = Model.find("agv" + string.fromNum(model.subnodes[i].id));

if(model.subnodes[i].id==2){
map1["Direction"]=Model.find("agv2").as(Object).rotation.z;
map1["PrevNodeId"]=Model.find("agv2").curr_node;
map1["IsAtNode"]=Model.find("agv2").is_at_node;
map1["LinkId"]=Model.find("agv2").currentPath;
}else if(model.subnodes[i].id==3){
map1["Direction"]=Model.find("agv3").as(Object).rotation.z;
map1["PrevNodeId"]= Model.find("agv3").curr_node;
map1["IsAtNode"]=Model.find("agv3").is_at_node;
map1["LinkId"]=Model.find("agv3").currentPath;//Model.find("agv3").currentPath;
}else if(model.subnodes[i].id==4){
map1["Direction"]=Model.find("agv4").as(Object).rotation.z;
map1["PrevNodeId"]=Model.find("agv4").curr_node;
map1["IsAtNode"]=Model.find("agv4").is_at_node;
map1["LinkId"]=Model.find("agv4").currentPath;
}
//AGV("agv2").
myArray.push(JSON.parse(JSON.stringify(map1)));

//print(myArray[j]);

//clientsend(client,);
}
else{
continue;
}
}

map3["TaskID"]=id_tarefa;
map3["TaskType"]="Entrada";
map3["RobotID"]=0;

```

```

map3["RobotLocations"]=myArray;
map2["TaskAllocationRequest"]=map3;
string two=JSON.stringify(map2);
//print(two);
clientsend(client,two);

```

A.4 Task Allocation Response

```

/**Custom Code*/
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);
double socknum = getnodenum(node("/Tools/client",model()));

string message = clientreceive(socknum, NULL, 2048, 0);

if(message==""){
int erro=1;
}
Map task_allocation_resp = JSON.parse(message);
Map task_all = task_allocation_resp["TaskAllocationResponse"];
robot_id = task_all["RobotID"];
Map ro= task_all["RobotRoutes"][robot_id];
Array path=ro["RobotRoute"];
if(robot_id==2){
agv_2_proc=1;
Model.find("agv2").Routes = path;
}
else if(robot_id==3){
agv_3_proc=1;
Model.find("agv3").Routes = path;
}
else if(robot_id==4){
agv_4_proc=1;
Model.find("agv4").Routes = path;
}

```

A.5 Travel - Process Flow

```

/**Custom Code*/
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode executer = param(4); //Or Task Sequence
treenode processFlow = ownerobject(activity);
//robot_id=robot_id+1;
if(token.token_id_robot=="agv2"){
int buf=Model.find("agv2").Routes.shift();
string s="CP"+buf;
int i=Model.find("agv2").curr_node;
Model.find("agv2").prev_node=i;
Model.find("agv2").curr_node=buf;
//Current Path Update
AGV agv = AGV(Model.find("agv2"));
AGV.TravelPathSection section = agv.currentTravelPathSection;
Variant path = section.path.evaluate();
string actualPath = path.name.replace("/AGVNetwork/", "");
Model.find("agv2").currentPath = actualPath.replace("Path", "").toNum();
//Travel to CP
return Model.find(s);
}
else if(token.token_id_robot=="agv3"){
int buf=Model.find("agv3").Routes.shift();
string s="CP"+buf;
int i=Model.find("agv3").curr_node;
Model.find("agv3").prev_node=i;
Model.find("agv3").curr_node=buf;
//Current Path Update
AGV agv = AGV(Model.find("agv3"));
AGV.TravelPathSection section = agv.currentTravelPathSection;
Variant path = section.path.evaluate();
string actualPath = path.name.replace("/AGVNetwork/", "");
Model.find("agv3").currentPath = actualPath.replace("Path", "").toNum();
//Travel to CP
return Model.find(s);
}

```

```

else if(token.token_id_robot=="agv4"){
int buf=Model.find("agv4").Routes.shift();
string s="CP"+buf;
int i=Model.find("agv4").curr_node;
Model.find("agv4").prev_node=i;
Model.find("agv4").curr_node=buf;
//Current Path Update
AGV agv = AGV(Model.find("agv4"));
AGV.TravelPathSection section = agv.currentTravelPathSection;
Variant path = section.path.evaluate();
string actualPath = path.name.replace("/AGVNetwork/", "");
Model.find("agv4").currentPath = actualPath.replace("Path","").toNum();
//Travel to CP
return Model.find(s);
}

```

A.6 Decide Block - Process Flow

```

/**Custom Code*/
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);

//Caso a leitura do array nao tenha acabado
//retorna 1

if(((Model.find("agv2").Routes).length==0) && (token.token_id_robot=="agv2")){
return 1;
}
else if(((Model.find("agv3").Routes).length==0) && (token.token_id_robot=="agv3")){
return 1;
}
else if(((Model.find("agv4").Routes).length==0) && (token.token_id_robot=="agv4")){
return 1;
}

//Se a leitura de array acabou e chegou ao destino retorna 2

if(((Model.find("agv2").Routes).length!=0) && (token.token_id_robot=="agv2")){

```



```

return 2;
}
else if(((Model.find("agv3").Routes).length!=0) && (token.token_id_robot=="agv3"))
return 2;
}
else if(((Model.find("agv4").Routes).length!=0) && (token.token_id_robot=="agv4"))
return 2;
}

```

A.7 Decide II

```

    /**Custom Code*/
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);
if((token.token_id_robot_int==2 && agv_2_proc==1) || (token.token_id_robot_int==3 && agv_3_proc==1))
return 1;
}
if((token.token_id_robot_int==2 && agv_2_proc==2) || (token.token_id_robot_int==3 && agv_3_proc==2))
return 2;
}

```

A.8 Last Trip - Process Flow

```

    /**Custom Code*/
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);
if((Model.find("agv2").Routes).length==0 && token.token_id_robot==2){
agv_2_proc=2;
}
else if((Model.find("agv3").Routes).length==0 && token.token_id_robot==3){
agv_3_proc=2;
}
else if((Model.find("agv4").Routes).length==0 && token.token_id_robot==4){
agv_4_proc=2;
}

```

A.9 Robot Unavailable - Process Flow

```

/**Custom Code*/
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
treenode processFlow = ownerobject(activity);
double client = getnodenum(node("/Tools/client",model()));
Map map3;
Map map2;
//map3["MessageId"]="RobotUnavailable";
map3["RobotID"]=token.token_id_robot_int; //ALTERAR O VALOR DE ID DOS ROBOS;
map3["State"]="LoadStarted"; //VAI COMEÇAR A FAZER LOAD DE UMA PEÇA
map2["RobotUnavailable"]=map3;
string two=JSON.stringify(map2);
clientsend(client,two);

```

A.10 Robot Available

```

/**Custom Code*/
Object current = param(1);
treenode activity = param(2);
Token token = param(3);
double client = getnodenum(node("/Tools/client",model()));
treenode processFlow = ownerobject(activity);
Map map1;
Map map2;
Map map3;
Map map4;
Map map5;
Array myArray= Array(0);
Array secArray= Array(0);
double curr;
//map2["MessageID"]="TaskAllocationRequest";
for(int i = 1; i<=content(model); i++)
{
string text = rank(model,i).name;
if(text.includes("agv") == 1) //getname(Model.find(" ").subnodes[i])
{
//treenode nextCP = getsdtvalue(subnode[i],"nextCPPreArrivalPoint");

```

```

map1["RobotID"]= model.subnodes[i].id;
map1["X"] = model.subnodes[i].as(Object).location.x;
map1["Y"] = model.subnodes[i].as(Object).location.y;
//map1["Direction"]=0.0;
Object agv = Model.find("agv" + string.fromNum(model.subnodes[i].id));

if(model.subnodes[i].id==2){
map1["Direction"]=Model.find("agv2").as(Object).rotation.z;
map1["PrevNodeId"]=Model.find("agv2").curr_node;
map1["IsAtNode"]=Model.find("agv2").is_at_node;
//Current Path Update
AGV agv = AGV(Model.find("agv4"));
AGV.TravelPathSection section = agv.currentTravelPathSection;
Variant path = section.path.evaluate();
string actualPath = path.name.replace("/AGVNetwork/", "");
Model.find("agv4").currentPath = actualPath.replace("Path", "").toNum();
map1["LinkId"]=Model.find("agv2").currentPath;

}else if(model.subnodes[i].id==3){
map1["Direction"]=Model.find("agv3").as(Object).rotation.z;
map1["PrevNodeId"]= Model.find("agv3").curr_node;
map1["IsAtNode"]=Model.find("agv3").is_at_node;
//Current Path Update
AGV agv = AGV(Model.find("agv3"));
AGV.TravelPathSection section = agv.currentTravelPathSection;
Variant path = section.path.evaluate();
string actualPath = path.name.replace("/AGVNetwork/", "");
Model.find("agv3").currentPath = actualPath.replace("Path", "").toNum();
map1["LinkId"]=Model.find("agv3").currentPath;

}else if(model.subnodes[i].id==4){
map1["Direction"]=Model.find("agv4").as(Object).rotation.z;
map1["PrevNodeId"]=Model.find("agv4").curr_node;
map1["IsAtNode"]=Model.find("agv4").is_at_node;
//Current Path Update
AGV agv = AGV(Model.find("agv4"));
AGV.TravelPathSection section = agv.currentTravelPathSection;
Variant path = section.path.evaluate();
string actualPath = path.name.replace("/AGVNetwork/", "");
Model.find("agv4").currentPath = actualPath.replace("Path", "").toNum();

```

```

map1["LinkId"]=Model.find("agv4").currentPath;
}
//AGV("agv2").
myArray.push(JSON.parse(JSON.stringify(map1)));

//print(myArray[j]);

//clientsend(client,);
}
else{
continue;
}
}

//map3["TaskID"]=token.task_id;
//map3["MessageId"]="RobotAvailable";
map3["RobotID"]=token.token_id_robot_int;
//map3["State"]="LoadFinished";
map3["RobotLocations"]=myArray;
map2["RobotAvailable"]=map3; //adicionado
string two=JSON.stringify(map2);
//print(two);
clientsend(client,two);

```

A.11 DataMissions.JSON - teste

```

{
  "Mission":{
"Server IP": "127.0.0.1",
  "Reststation":[
    {
      "X":-37,
      "Y":13,
      "isactive":0,
      "node_id":1,
      "ws_id":1
    },
    {
      "X":-0.01418864135309671,
      "Y":-4.001114920536879,

```

```
    "isactive":0,
    "node_id":58,
    "ws_id":2
  },
  {
    "X":-5.0000127222500099,
    "Y":12.955240053962815,
    "isactive":0,
    "node_id":64,
    "ws_id":3
  },
  {
    "X":-39.953259960431396,
    "Y":5.9980795657236428,
    "isactive":0,
    "node_id":68,
    "ws_id":4
  },
  {
    "X":-39.492349075276124,
    "Y":0.6989640860239632,
    "isactive":0,
    "node_id":76,
    "ws_id":5
  }
],
"Robot":[
  {
    "Direction":0,
    "x":-38,
    "y":13.5,
    "node":1,
    "robot_id":2
  },
  {
    "Direction":0,
    "x":-1.0141886413530967,
    "y":-3.501114920536879,
    "node":58,
    "robot_id":3
  }
]
```

```
    },  
    {  
      "Direction":0,  
      "x":-6.0000127222500099,  
      "y":13.455240053962815,  
      "node":64,  
      "robot_id":4  
    }  
  ],  
  "Tasks": [  
    {  
      "Robot_ID":2,  
      "Nodes":[1,76],  
      "Task_id":1,  
      "priority":1  
    },  
    {  
      "Robot_ID":3,  
      "Nodes":[58,68],  
      "Task_id":2,  
      "priority":2  
    }  
  ]  
}
```

Referências

- [1] Günter Ullrich et al. Automated guided vehicle systems. *Springer-Verlag Berlin Heidelberg*. doi, 10:978–3, 2015.
- [2] Diana Marina de Sousa Neves. *Modelo e Protótipos, à escala, de AGVs para o transporte de material numa fábrica*. Tese de doutoramento, UNIVERSIDADE DO PORTO, 2015.
- [3] Ksenia Shabalina, Artur Sagitov, e Evgeni Magid. Comparative analysis of mobile robot wheels design. Em *2018 11th International Conference on Developments in eSystems Engineering (DeSE)*, páginas 175–179, 2018. doi:10.1109/DeSE.2018.00041.
- [4] M. Bala Subramanian, K. Sudhagar, e G. Rajarajeswari. Design of navigation control architecture for an autonomous mobile robot agent. *Indian journal of science and technology*, 9, 2016.
- [5] Jean-Claude Latombe. *Introduction and Overview*, páginas 1–57. Springer US, Boston, MA, 1991. URL: https://doi.org/10.1007/978-1-4615-4022-9_1, doi:10.1007/978-1-4615-4022-9_1.
- [6] Pedro L. C. Gomes da Costa. Planeamento cooperativo de tarefas e trajectórias em múltiplos robôs, 2011. Faculdade de Engenharia da Universidade do Porto, disponível em <https://repositorio-aberto.up.pt/bitstream/10216/62107/1/000148893.pdf>.
- [7] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. doi:10.1017/CBO9780511546877.
- [8] Joana Santos, Paulo M. Rebelo, Luis F. Rocha, Pedro Costa, e Germano Veiga. A* based routing and scheduling modules for multiple agvs in an industrial scenario. *Robotics*, 10(2), 2021. URL: <https://www.mdpi.com/2218-6581/10/2/72>, doi:10.3390/robotics10020072.
- [9] Grand View Resarch. Automated guided vehicle market size report, 2020. Disponível em [grandviewresearch.com/industry-analysis/automated-guided-vehicle-agv-market](https://www.grandviewresearch.com/industry-analysis/automated-guided-vehicle-agv-market).
- [10] Eduardo António da Silva Santos. Logística baseada em agvs. 2013.
- [11] Wenjie Wang e Wooi-Boon Goh. Multi-robot path planning with the spatio-temporal a* algorithm and its variants. Em Francien Dechesne, Hiromitsu Hattori, Adriaan ter Mors, Jose Miguel Such, Danny Weyns, e Frank Dignum, editores, *Advanced Agent Technology*, páginas 313–329, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. Disponível em https://doi.org/10.1007/978-3-642-27216-5_22.

- [12] Roberto Olmi, Cristian Secchi, e Cesare Fantuzzi. Coordination of multiple agvs in an industrial application. Em *2008 IEEE International Conference on Robotics and Automation*, páginas 1916–1921, 2008. doi:10.1109/ROBOT.2008.4543487.
- [13] Nenad Smolic-Rocak, Stjepan Bogdan, Zdenko Kovacic, e Tamara Petrovic. Time windows based dynamic routing in multi-agv systems. *IEEE Transactions on Automation Science and Engineering*, 7(1):151–155, 2010. doi:10.1109/TASE.2009.2016350.
- [14] Ivica Draganjac, Damjan Miklič, Zdenko Kovačić, Goran Vasiljević, e Stjepan Bogdan. Decentralized control of multi-agv systems in autonomous warehousing applications. *IEEE Transactions on Automation Science and Engineering*, 13(4):1433–1447, 2016.
- [15] Ana Sofia Poças da Silva Cruz. Multi agv communication failuretolerant industrial supervisory system, 2021. Faculdade de Engenharia da Universidade do Porto.
- [16] Pedro Henrique Fernandes Moura. Coordenação de multi-robots numambiente industrial, 2018. Faculdade de Engenharia da Universidade do Porto.
- [17] Stefan Edelkamp, Stefan Schroedl, e Sven Koenig. *Heuristic Search: Theory and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.
- [18] Joana Santos, Pedro Costa, Luís F. Rocha, A. Paulo Moreira, e Germano Veiga. Time enhanced a: Towards the development of a new approach for multi-robot coordination. Em *2015 IEEE International Conference on Industrial Technology (ICIT)*, páginas 3314–3319, 2015.
- [19] K. Petrinc e Z. Kovacic. The application of spline functions and bezier curves to agv path planning. Em *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.*, volume 4, páginas 1453–1458, 2005. doi:10.1109/ISIE.2005.1529146.
- [20] What are the differences between simulation software: Discrete, continuous, and agent-based?
- [21] Miguel Marques Pereira. *Transporte Robotizado de Bagagens em Aeroportos*. Tese de doutoramento. Instituto Superior de Engenharia do Porto.
- [22] Talumis. flexsim agv simulation software module <https://www.talumis.com/agv-simulation-software/>.
- [23] Tutorials:task 1.1 - build a 3d model <https://docs.flexsim.com/en/22.2/Tutorials/FlexSimBasics/1-1Build3DModel/1-1Build3DModel.html>.
- [24] FlexSim <https://www.flexsim.com/pt/geral/next-generation-simulation-modeling-with-pro>
- [25] Tutorials:task 1.3 - build a process flow model <https://docs.flexsim.com/en/20.0/Tutorials/FlexSimBasics/1-3BuildProcessFlow/1-3BuildProcessFlow.html>.