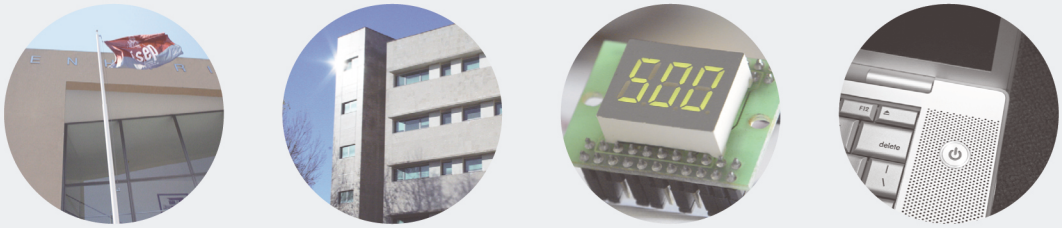
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Simulação de um Sistema Robótico de Co-transporte

FREDERICO MARQUES TEIXEIRA
Novembro de 2020

POLITÉCNICO
DO PORTO

Simulação de um Sistema Robótico de Co-transporte

Frederico Marques Teixeira

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2020

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Frederico Marques Teixeira, Nº 1150881, 1150881@isep.ipp.pt
Orientação científica: Manuel Fernando dos Santos Silva, mss@isep.ipp.pt

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
15 de Novembro de 2020

Agradecimentos

Em primeiro lugar gostaria de agradecer aos meus pais pelo apoio, incentivo e todo o esforço que depositaram na minha educação.

Agradeço ao professor Manuel Silva pela orientação e constante disponibilidade, não só durante o desenvolvimento deste projeto, mas em várias ocasiões ao longo da minha passagem pelo ensino superior.

Aos meus avós, obrigado pelo carinho e aconselhamento cheio de sabedoria que foram essenciais em todo este percurso.

Quero também deixar um obrigado à Cecília pelo apoio incondicional, paciência, motivação, encorajamento e companhia em todos os momentos, especialmente naqueles que se mostraram mais difíceis.

Por último, agradeço aos meus amigos e colegas de faculdade pois sem eles o ânimo nunca teria sido o mesmo.

Resumo

Os robôs móveis são usados para automatizar as operações de transporte na indústria desde meados do séc. XX. São cada vez mais as empresas que possuem estes sistemas, sendo que algumas já são constituídas por frotas numerosas que trabalham incansavelmente nas suas fábricas e armazéns. Com a evolução do mercado, espera-se que muitas mais possam adquirir e usufruir destas tecnologias.

Conforme o número de robôs disponíveis nas empresas aumenta, é importante que se estudem estratégias que permitam que estes cooperem entre si. Isto fará com que vários problemas sejam resolvidos mais facilmente, tal como é o caso do transporte de cargas de elevada dimensão - tarefa que ainda é um obstáculo à produtividade de algumas empresas.

No presente trabalho são estudadas três estratégias que permitem a dois robôs omnidirecionais, um *leader* e um *follower*, coordenarem o seu movimento e cooperarem no transporte de uma carga (co-transporte) que não poderia ser transportada por apenas um robô. As estratégias exploradas diferem consoante a abordagem seguida na construção mecânica de um suporte, cuja função é permitir aos robôs agarrar e transportar a carga no topo do seu chassis, e na arquitetura de controlo do robô *follower*. No que diz respeito ao suporte, este possuirá um sensor de força de modo a possibilitar a minimização das forças de contacto entre o robô *follower* e a carga. Será a partir deste conceito, implementado através de um controlador PI, que irá ser estabelecida a cooperação entre os robôs, de modo a eliminar ou reduzir o máximo possível a comunicação direta entre eles. Enquanto que o *leader* irá conduzir o sistema até ao destino, o *follower* deverá reagir às forças impostas na carga e que resultam do seu movimento.

O sistema e as potenciais estratégias de cooperação, idealizados numa primeira fase, são testados recorrendo a um simulador, mais concretamente o CoppeliaSim. Ao longo deste documento será descrito todo o processo da modelação dos robôs e dos suportes, da implementação dos algoritmos de controlo e da sua simulação no ambiente virtual.

Os resultados obtidos através da simulação são alvo de uma análise que irá avaliar qual a estratégia que melhor complementa a mobilidade omnidirecional dos constituintes do sistema e que, ao mesmo tempo, garante que menos forças são exercidas sobre a carga durante o transporte. Por fim, será também apresentada uma comparação direta entre as estratégias no seguimento de uma mesma trajetória por parte do robô *leader*.

Palavras-Chave: Co-transporte, Sistemas multi-robô, Cooperação, Transporte coletivo de cargas, Robótica móvel, Simulação, CoppeliaSim.

Abstract

Mobile robots have been used to automate transport operations in the industry since the middle of the 20th century. More and more companies have these systems, some of which are already made up of numerous fleets that work tirelessly in their factories and warehouses. With the evolution of the market, it is expected that many more will be able to acquire and benefit from the use of these technologies.

As the number of robots available in companies increases, it is important to study strategies that allow them to cooperate with each other. This will make several problems easier to solve. One good example is the transportation of a large cargo, which is a task that still presents obstacles to the productivity of some companies.

In the present work, three strategies are studied in order to allow a team of two omnidirectional robots, a leader and a follower, to coordinate their movement and cooperate in the transport of a load that could not be transported by just one robot. The strategies explored differ according to the approach followed in the mechanical construction of a support, whose function is to allow robots to grab and transport the load on the top of its chassis, and in the control architecture of the follower robot. Regarding the support, it will have a force sensor to allow the minimization of contact forces between the follower robot and the load. Based on this concept, implemented through a PI controller, the cooperation between the robots will be established, in order to eliminate or reduce as much as possible the direct communication between them. While the leader will drive the system to its destination, the follower must react to the forces imposed on the load and which result from its movement.

The system and its cooperation strategies are idealized and then tested in a simulator, more specifically, in the CoppeliaSim simulator. Throughout this document, the entire modelling process of the robots and supports, implementation of control algorithms and simulation in the virtual environment will be described.

The results obtained through the simulation are the target of an analysis that will assess which strategy best complements the omnidirectional mobility of the system's constituents and, at the same time, ensures that less forces are exerted on the load during transport. Finally, it will also be presented a direct comparison between the strategies for the same trajectory followed by the leader robot.

Keywords: Multi-Robot Systems, Cooperation, Collective load transport, Mobile robotics, Simulation, CoppeliaSim.

Índice

Agradecimentos **i**

Resumo **iii**

Abstract **v**

Índice **vii**

Índice de Figuras **xi**

Índice de Tabelas **xvii**

Acrónimos **xix**

1 Introdução **1**
1.1 Contextualização 1
1.2 Objetivos 3
1.3 Calendarização 4
1.4 Organização do Relatório 4

2 Revisão Bibliográfica **7**
2.1 Robótica Móvel e a Automatização do Transporte na Indústria . . 7
2.1.1 Veículos Automaticamente Guiados 8
2.1.2 Robôs Móveis Autónomos 9
2.1.3 Cooperação entre Robôs Móveis 10
2.2 Sistemas de Co-transporte 12
2.2.1 Conceitos Fundamentais dos Sistemas Multi-Robô 13
2.2.1.1 Arquitetura de Controlo 13
2.2.1.2 Diferenciação dos Elementos do Sistema 14
2.2.1.3 Comunicação 15
2.2.1.4 Coordenação do Sistema 17

2.2.1.5 Planeamento da Trajetória 18
2.2.2 Principais Estratégias de Transporte 19
2.2.3 Projetos de Interesse Realizados na Área 20
2.2.3.1 Considerações Acerca dos Projetos Apresentados . 26

3 Fundamentação Teórica 29
3.1 Plataformas Omnidirecionais 29
3.1.1 Tipos de Rodas Omnidirecionais 30
3.1.2 Configurações de Plataformas com Rodas de Desenho Especial 32
3.1.3 Cinemática das Plataformas com Rodas *Mecanum* 34
3.1.3.1 Modelo Cinemático de uma Roda 35
3.1.3.2 Modelo Cinemático de um Robô com Quatro Rodas 36
3.2 Sensores de Força 39
3.2.1 Tipos de Sensores de Força 39
3.2.2 Células de Carga 42
3.2.3 Sensores de Binário 44
3.2.4 Sensores de Força/Binário Multiaxiais 45
3.3 Controlo PID 47
3.3.1 Sintonia de Controladores PID 48
3.3.2 Discretização do Controlador 49

4 Escolha e Introdução ao Simulador 51
4.1 Simuladores de Robótica 51
4.1.1 *Software* para a Simulação de Sistemas Multi-Robô 52
4.1.2 Escolha do Simulador 56
4.2 Introdução ao CoppeliaSim 57
4.2.1 Interface Gráfica do Utilizador 57
4.2.2 Definição de Entidade, Modelo e Cena 60
4.2.3 Concepção de um Modelo 61
4.2.3.1 Criação das Formas Visíveis e Juntas 61
4.2.3.2 Configuração das Propriedades Dinâmicas do Modelo 62
4.2.3.3 Hierarquia, Inclusão de Sensores e Definição do Modelo 64
4.2.4 Desenvolvimento do Código 66

5 Modelação do Sistema de Co-transporte 69
5.1 Apresentação do Sistema 69
5.2 Modelação das Plataformas 72
5.2.1 Implementação do Comportamento da Roda *Mecanum* . . . 74
5.2.2 Testes Realizados à Plataforma Modelada 77

5.3 Modelação de um Suporte 81
5.3.1 Criação de uma Conexão Robô-Objeto 81
5.3.2 Inclusão do Sensor de Força 82
5.3.3 Colocação do Suporte nas Plataformas 83
5.3.3.1 Verificação da Correta Implementação do Suporte 84

6 Implementação e Simulação das Estratégias de Co-transporte 87
6.1 Princípio de Funcionamento do Sistema 87
6.2 Arquitetura de Controlo do Robô *Leader* 89
6.2.1 Controlo Manual 89
6.2.2 Seguimento de Caminhos 90
6.3 Estudo das Estratégias de Co-transporte 94
6.3.1 Controlo da Força com Controladores PID 94
6.3.1.1 Identificação do Ganho e Período de Oscilação Crítico 94
6.3.1.2 Controlador P 95
6.3.1.3 Controlador PI 96
6.3.1.4 Controlador PID 97
6.3.1.5 Conclusão 97
6.3.2 Estratégia 1 - Utilização de um Sensor de Força/Binário Multiaxial 100
6.3.2.1 Análise da Estratégia Idealizada 102
6.3.3 Estratégia 2 - Utilização de uma Célula de Carga e Sensor de Posição Angular 106
6.3.3.1 Análise da Estratégia Idealizada 108
6.3.4 Estratégia 3 - Comunicação Direta de um Parâmetro entre os Robôs 111
6.3.4.1 Análise da Estratégia Idealizada 114
6.3.5 Comparação do Desempenho das Estratégias no Seguimento de um Caminho 116

7 Conclusões 121
7.1 Considerações Finais 121
7.2 Principais Dificuldades Encontradas 123
7.3 Ideias para Trabalhos Futuros 123

Referências Bibliográficas 125

A *Scripts* em Lua Utilizados na Simulação do Sistema 133
A.1 Comportamento das Rodas *Mecanum* 133
A.2 Testes Realizados ao Modelo da Plataforma 134
A.3 Criação da Conexão entre os Robôs e o Objeto 136

A.4 *Script* do Controlo Manual do *Leader* 138
A.5 Algoritmo PID e Funções Relacionadas 140
A.6 *Script* para o Seguimento de Caminhos do *Leader* 141
A.7 *Script* do Controlo do *Follower* para a Estratégia 1 143
A.8 *Script* do Controlo do *Follower* para a Estratégia 2 144
A.9 *Script* do Controlo do *Follower* para a Estratégia 3 145

Índice de Figuras

1.1 Projeção do crescimento do mercado dos robôs de serviços [3] 2

2.1 Exemplos de equipamentos usados nas operações de transporte na indústria [9, 10] 8
2.2 Tipos de AGV existentes - veículos de carga [14], reboque [15] e garfo [16], respetivamente 9
2.3 AMR utilizados nos armazéns da Amazon [19] e em fábricas da Ford [20], respetivamente 10
2.4 Exemplos de cooperação entre robôs [21] 10
2.5 Exemplo de uma plataforma omniMove da KUKA [22] e do *tridem mode* [23], respetivamente 11
2.6 Plataforma omnidirecional da Aritex [24] e modo de acoplamento mecânico [25], respetivamente 11
2.7 Transporte cooperativo de cargas em terra [27], pelo ar [28] e na água [29] 12
2.8 Possíveis arquiteturas de controlo dos MRS 14
2.9 Grupo de robôs homogéneos [31] e heterogéneos [32], respetivamente . 15
2.10 Tipos de comunicação mais comuns num MRS 16
2.11 Tipos de comunicação mais usados nos sistemas de co-transporte [34] 17
2.12 Exemplo de uma formatura com três robôs para carregar um objeto deformável [35] e sistema com mecanismo que facilita o transporte de um objeto em formatura (adaptado de [6]), respetivamente 18
2.13 Exemplo de um sistema utilizando *pushing* (adaptado de [36]) 19
2.14 Exemplo de sistema utilizando *caging* [37] 20
2.15 Exemplos de sistemas utilizando *grasping* [25, 38] 20
2.16 Sistema desenvolvido por Pereira *et al.* e respetivo dispositivo de acoplamento ao objeto (adaptado de [39]) 21
2.17 Robôs *leader* e *follower*, respetivamente, do sistema apresentado em [38] 22
2.18 Sistema apresentado em [40] 23
2.19 Arena de testes, carga e robô usados em [21] 24

2.20 Exemplificação do comportamento do sistema apresentado em [21], quando o transporte é bem sucedido 24
2.21 Exemplificação da rotação sincronizada entre o robô *master* e o *slave* do sistema apresentado em [25] 25
2.22 Refletores usados na traseira do *slave* e sistema de dois robôs apresentado em [25], respetivamente 26

3.1 Robô com acionamento diferencial a efetuar um estacionamento paralelo e exemplificação das áreas inacessíveis de uma plataforma com geometria de Ackermann, respetivamente (adaptado de [41]) 30
3.2 Exemplo de um rodízio e de uma roda direcional, respetivamente [42] 30
3.3 Exemplo de roda universal e *mecanum*, respetivamente (adaptado de [42]) 31
3.4 Exemplo de um robô omnidirecional com três rodas universais usado nas competições RoboCup e alguns exemplos dos seus movimentos consoante a atuação de cada motor [42] 33
3.5 Exemplo de um robô omnidirecional com quatro rodas universais e alguns exemplos dos seus movimentos consoante a atuação de cada motor [42] 33
3.6 Exemplo de uma plataforma omnidirecional com quatro rodas *mecanum* e os seus movimentos consoante a atuação de cada motor [42] . . 34
3.7 Diagrama das restrições cinemáticas de uma roda *mecanum* [45] . . . 36
3.8 Configuração de uma plataforma com quatro rodas *mecanum* [45] . . . 37
3.9 Sistema de coordenadas local e global de uma plataforma com quatro rodas *mecanum* [46] 38
3.10 Exemplos da utilização de sensores de força em tarefas de teste de produtos e na monitorização das forças exercidas pela ferramenta de um robô industrial [47, 48] 39
3.11 Princípio de funcionamento de um sensor de força que possui extensómetros (adaptado de [52]) 40
3.12 Princípio de funcionamento de um sensor de força piezoelétrico [53] . . 40
3.13 Estrutura de um exemplo de um sensor capacitivo em corte [56] 41
3.14 Exemplo de um FSR e respetiva constituição [57] 42
3.15 Sensor de força ótico OptoForce [58] 42
3.16 Célula de carga do tipo *single-point* e respetivo funcionamento (adaptado de [59]) 43
3.17 Exemplos de configurações de células de carga [60] 43
3.18 Configurações *multi-point* mais comuns (adaptado de [60]) 44
3.19 Exemplificação da utilização de um sensor de binário [60] 44
3.20 Exemplos de configurações de sensores de binário [60] 45
3.21 Sensores de força/binário multiaxiais piezoelétricos da Kistler [53] . . 45

3.22 Exemplo de um sensor de força/binário de seis eixos montado num robô colaborativo para tarefas de montagem (adaptado de [61]) 46
3.23 Sensor de força/binário de seis eixos FT-AXIA da Schunk (1 - Eletrónica, 2 - Vigas metálicas com extensómetros) [62] 46
3.24 Diagrama de blocos de um controlador PID paralelo clássico 47
3.25 Principais características da onda de resposta de um sistema em malha fechada [63] 48

4.1 Simulação do KUKA youBot no Webots [66] 53
4.2 Simulação de um grupo de robôs no ARGoS [69] 54
4.3 Simulação de um grupo de robôs no USARsim [72] 54
4.4 Simulação de um drone no Gazebo [74] 55
4.5 Simulação do KUKA youbot no CoppeliaSim 56
4.6 Interface gráfica do CoppeliaSim (adaptado de [76]) 58
4.7 Barra de ferramentas nº1 58
4.8 Barra de ferramentas nº2 58
4.9 Hierarquia das cenas (adaptado de [76]) 59
4.10 Objetos existentes no CoppeliaSim (adaptado de [76]) 60
4.11 Estrutura e identificação de um modelo (adaptado de [76]) 60
4.12 Funcionalidades de criação de formas primitivas no simulador e importação de malhas, respetivamente 61
4.13 Exemplificação da constituição de um modelo de um robô articulado e do posicionamento dos seus constituintes (adaptado de [76]) 62
4.14 Propriedades dinâmicas de uma forma 63
4.15 Exemplificação da criação das formas dinâmicas (adaptado de [76]) . . 64
4.16 Exemplificação da definição das propriedades das juntas 64
4.17 Exemplificação da criação da hierarquia de um modelo (adaptado de [76]) 65
4.18 Análise do conteúdo dinâmico de um modelo e identificação do modelo (adaptado de [76]) 66
4.19 Tipos de *scripts* suportados pelo CoppeliaSim [76] 67

5.1 Robô *leader* do sistema e respetiva constituição [78] 70
5.2 Abordagens seguidas na construção do suporte do robô *follower* . . . 71
5.3 Dimensões das plataformas DiscoveryQ2 [77] 72
5.4 Junção de vários objetos e aparência final do modelo do chassis do DiscoveryQ2 73
5.5 Modelo das rodas do OmniRob e alteração da sua dimensão 73
5.6 Formas visíveis do DiscoveryQ2 e constituição de uma roda *mecanum* no CoppeliaSim 74
5.7 Propriedades das duas juntas que constituem a roda *mecanum* 75

5.8 Hierarquia do modelo das plataformas e representação das colisões entre o chassis e as rodas 75
5.9 Comportamento da roda *mecanum* antes da implementação do *script* 76
5.10 Comportamento da roda *mecanum* depois da implementação do *script* para diferentes tempos de passo da simulação 77
5.11 Comparação das velocidades máximas da plataforma com um tempo de passo de 25 e 50 ms, respetivamente 79
5.12 Comparação das velocidades transversais e rotacionais da plataforma com um tempo de passo de 25 e 50 ms, respetivamente 80
5.13 Robotiq 2F-85 e complexidade do respetivo modelo dinâmico 81
5.14 Comparação entre o modelo da ventosa e a base criada 82
5.15 Exemplificação do comportamento da ventosa 83
5.16 Inclusão do sensor de força no suporte 83
5.17 Plataforma DiscoveryQ2 e respetiva estrutura hierárquica depois da colocação do suporte 84
5.18 Propriedades *motor enabled* e *Lock motor when target velocity is zero* 84
5.19 Representação do sistema de co-transporte com a carga e da conexão criada entre os robôs e o objeto 85
5.20 Forças detetadas pelo sensor de força/binário multiaxial no robô *follower* durante movimentos longitudinais, transversais e rotacionais (com o motor de física ODE) 85

6.1 Cooperação entre humanos no transporte de um objeto [79] 88
6.2 Controlo explícito da força 89
6.3 Diagrama de blocos da arquitetura de controlo do robô *leader* para o seguimento de trajetórias 90
6.4 Fluxograma da função de *callback* responsável pela implementação da arquitetura de controlo do seguidor de caminhos 91
6.5 Fluxogramas das funções localização e PID 92
6.6 Criação e edição de *paths* no CoppeliaSim 92
6.7 Exemplificação dos *Dummies* de referência e da localização do robô . . 93
6.8 Seguimento de um caminho de teste a 0,1 m/s 93
6.9 Seguimento de um caminho de teste a 0,3 m/s 94
6.10 Exemplificação de um movimento longitudinal 95
6.11 Identificação do ganho e período de oscilação crítico para aplicação do método de Ziegler-Nichols 95
6.12 Resultados obtidos com o controlador P (K_p=0,0150) 96
6.13 Resultados obtidos com o controlador PI (K_p=0,0135; K_i=0,3240) . . 96
6.14 Resultados obtidos com o controlador PID (K_p=0,0180; K_i=0,7200; K_d=0,0001) 97
6.15 Leituras em vazio do sensor de força 98

6.16 Comparação das forças exercidas na carga com o controlador PI sintonizado com o método de Ziegler-Nichols e depois dos ajustes manuais, respetivamente 98
6.17 Comparação de duas respostas do sistema com um período de amostragem de 10 e 1 ms, respetivamente 99
6.18 Comparação das forças exercidas na carga para movimentos a diferentes velocidades, 0,1 m/s e 0,45 m/s, respetivamente 100
6.19 Comparação das forças exercidas na carga para o mesmo movimento, mas com diferentes acelerações por parte do robô *leader* (0,48 m/s^2 e 0,1 m/s^2, respetivamente) 100
6.20 Diagrama de blocos do controlo do *follower* com a utilização do sensor de força/binário multiaxial 101
6.21 Fluxograma da função de *callback* responsável pela implementação da arquitetura de controlo do *follower* para a estratégia 1 101
6.22 Forças medidas pelo sensor de força/binário multiaxial do *follower* durante um movimento longitudinal do robô *leader* (V_Y= 0,1 m/s) . . 102
6.23 Forças medidas pelo sensor de força/binário multiaxial do *follower* durante um movimento transversal do robô *leader* (V_X= 0,1 m/s) . . 102
6.24 Forças medidas pelo sensor de força/binário multiaxial do *follower* durante um movimento rotacional do robô *leader* (W= 0,1 rad/s) . . 103
6.25 Exemplificação da combinação de um movimento linear com um rotacional e forças medidas pelo sensor de força/binário multiaxial durante o movimento (V_Y= 0,1 m/s e W= 0,1 rad/s) 103
6.26 Exemplificação do desalinhamento dos robôs e forças medidas pelo sensor do *follower* durante a realização de um movimento linear (V_Y= 0,1 m/s) 104
6.27 Exemplificação do comportamento do sistema e forças medidas pelo sensor do *follower* durante a realização de uma curva por parte do *leader* a uma velocidade de 0,1 m/s 105
6.28 Erro associado à derrapagem das rodas e demora na reação às forças por parte do robô *follower* 106
6.29 Diagrama de blocos do controlo do *follower* com a utilização da célula de carga e o sensor de posição angular 107
6.30 Componentes F_Y e F_x da força medida pela célula de carga 107
6.31 Fluxograma da função de *callback* responsável pela implementação da arquitetura de controlo do *follower* para a estratégia 2 108
6.32 Forças medidas pelo sensor de força do robô *follower* durante um movimento longitudinal (V_Y= 0,1 m/s) 108
6.33 Forças medidas pelo sensor de força do robô *follower* durante um movimento transversal do *leader* (V_X= 0,1 m/s) 109

6.34 Forças medidas pelo sensor de força do robô *follower* durante um movimento oblíquo do *leader* (V_Y= 0,1 m/s e V_X= 0,1 m/s) 109
6.35 Forças medidas pelo sensor de força do robô *follower* durante um movimento transversal do *leader* a 0,05 m/s e 0,2 m/s, respetivamente 110
6.36 Exemplificação do comportamento do sistema e forças medidas pelo sensor do *follower* durante a realização de uma curva por parte do *leader* a uma velocidade de 0,1 m/s 111
6.37 Diagrama de blocos do controlo do *follower* relativo à terceira estratégia explorada 112
6.38 Ângulo de orientação dos robôs com a carga e ação do controlador de rotação 113
6.39 Ação do controlador de translação 113
6.40 Fluxograma da função de *callback* responsável pela implementação da arquitetura de controlo do *follower* para a estratégia 3 114
6.41 Forças medidas pelo sensor de força do robô *follower* durante um movimento transversal do *leader* (V_X= 0,1 m/s) 114
6.42 Forças medidas pelo sensor de força do robô *follower* durante um movimento rotacional do *leader* (W= 0,1 rad/s) 115
6.43 Exemplificação do comportamento do sistema e forças medidas pelo sensor do *follower* durante a realização de uma curva por parte do *leader* a uma velocidade de 0,1 m/s 116
6.44 Caminho implementado para a realizar a comparação entre estratégias 117
6.45 Forças detetadas pelo sensor do robô *follower* durante o movimento do sistema - Estratégia 1 117
6.46 Forças detetadas pelo sensor do robô *follower* durante o movimento do sistema - Estratégia 2 117
6.47 Forças detetadas pelo sensor do robô *follower* durante o movimento do sistema - Estratégia 3 118
6.48 Comparação entre o caminho pretendido e o efetuado pelo robô *leader* e caminho efetuado pelo robô *follower* - Estratégia 1 118
6.49 Comparação entre o caminho pretendido e o efetuado pelo robô *leader* e caminho efetuado pelo robô *follower* - Estratégia 2 119
6.50 Comparação entre o caminho pretendido e o efetuado pelo robô *leader* e caminho efetuado pelo robô *follower* - Estratégia 3 119

Índice de Tabelas

1.1 Calendarização das tarefas 4

2.1 Síntese das características dos projetos apresentados 27

3.1 Vantagens e desvantagens das rodas omnidirecionais 32
3.2 Alterações nas características da resposta do sistema, consoante o aumento de cada ganho do controlador PID 48
3.3 Sintonia dos ganhos através do método de Ziegler-Nichols em malha fechada [64] 49

Acrónimos

Acrónimo	Descrição	Página
3D	Tridimensional	56
AGV	*Automated Guided Vehicle*	1
AMR	*Autonomous Mobile Robot*	1
API	*Application Programming Interface*	55
ARGoS	*Autonomous Robots go Swarming*	53
CAD	*Computer Aided Design*	52
CSV	*Comma-Separated Values*	91
DART	*Dynamic Animation and Robotics Toolkit*	55
EPFL	École Polytechnique Fédérale de Lausanne	52
FSR	*Force Sensitive Resistor*	41
GUI	*Graphical User Interface*	52
INESC TEC	Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência	69
LED	*Light Emitting Diode*	23
LiDAR	*Light Detection and Ranging*	26
MRS	*Multi-Robot System*	2
ODE	*Open Dynamics Engine*	52
OGRE	*Object-Oriented Graphics Rendering Engine*	55
PID	Proporcional-Integral-Derivativo	29
ROS	*Robot Operating System*	52
SDF	*Simulation Description Format*	54
UI	*User Interface*	56
URDF	*Unified Robot Description Format*	52
USARsim	*Urban Search and Rescue Simulation*	53
V-REP	*Virtual-Robot Experimentation Platform*	55

Capítulo 1

Introdução

Ao longo deste capítulo é realizada uma contextualização ao tema do trabalho e apontam-se os aspetos que motivaram o estudo do mesmo. Seguidamente, são apresentados os objetivos e a calendarização que permitiu organizar e levar a cabo a execução de todas as tarefas. Por último, é feita uma descrição dos diversos capítulos que constituem este documento, de modo a facilitar a sua leitura e compreensão.

1.1 Contextualização

As operações de transporte são fundamentais à logística interna de qualquer empresa. Num ambiente fabril, tanto as matérias primas como os produtos semi-acabados necessitam de ser armazenados e, posteriormente, transportados até à linha ou célula de produção. Por outro lado, também o produto final tem de ser transportado para fora da área de trabalho ou entre armazéns. Ao longo dos últimos anos, e devido ao desenvolvimento tecnológico na área da robótica móvel, as operações de transporte têm sido alvo de diversos estudos e de uma crescente automatização.

Atualmente, a nível mundial, muitos armazéns ainda não são constituídos por nenhum meio de automação e o transporte dos produtos é efetuado manualmente ou por métodos convencionais [1]. No entanto, este cenário terá tendência a mudar rapidamente uma vez que se espera que o mercado dos robôs usados em operações logísticas apresente um crescimento acentuado [2, 3], tal como se pode verificar na Figura 1.1. Este crescimento poderá significar uma maior utilização de equipamentos robotizados, tais como os *Automated Guided Vehicles* (AGV) e os *Autonomous Mobile Robots* (AMR). Estes dois exemplos têm vindo a ser adoptados por parte das empresas para realizar o transporte de produtos, paletes,

contentores ou *trailers* em armazéns e fábricas. Adicionalmente, alguns robôs são também implementados com manipuladores, de modo a não realizarem apenas o transporte, mas também a manipulação das cargas. As suas vantagens passam por serem eficientes e permitirem a otimização do *layout* fabril, reduzindo custos e aumentando a segurança e eficácia do mesmo. A maior desvantagem à qual estão relacionados são os elevados custos associados ao investimento inicial, possíveis reestruturações de *layouts* existentes e manutenção [4, 5]. Com o aumento da competitividade entre os fabricantes destes robôs, provocado pelo crescimento do mercado, e o aparecimento de novas soluções, esta desvantagem poderá vir a ser um pouco mitigada, o que fará com que mais empresas possam adquirir exemplos destas tecnologias.

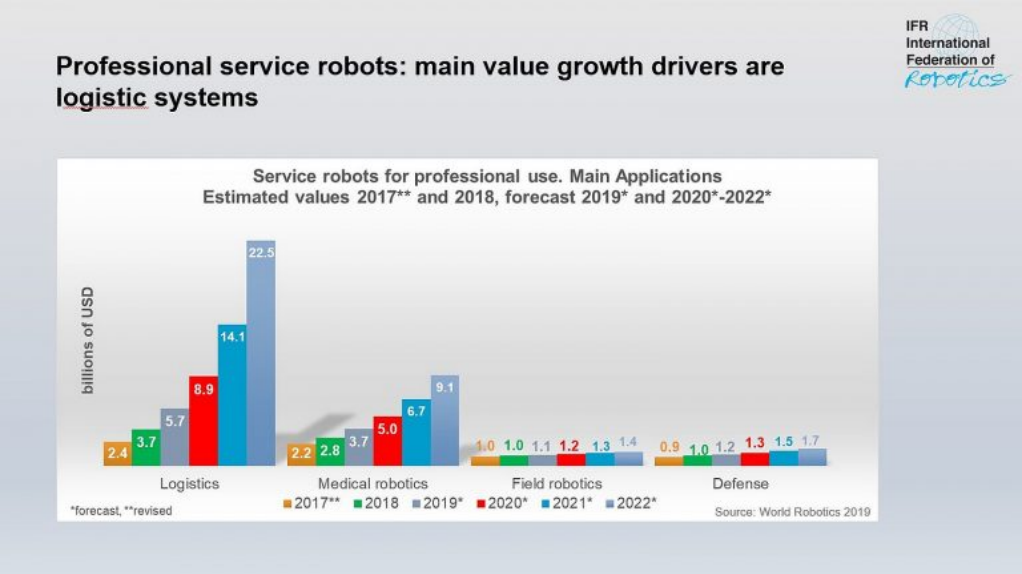


Figura 1.1: Projeção do crescimento do mercado dos robôs de serviços [3]

Constituídas por frotas de robôs móveis cada vez mais numerosas e mais "inteligentes", torna-se desejável para as empresas que sejam exploradas técnicas que permitam que as suas máquinas se comportem como uma equipa, aumentando assim a sua eficiência e flexibilidade. Por esta razão, a área dos *Multi-Robot Systems* (MRS) tem sido bastante explorada. Uma aplicação relacionada com esta área de estudo e que suscita um grande interesse por parte da indústria é o co-transporte [6, 7, 8]. Nos sistemas que recorrem a esta técnica não existe apenas um agente envolvido no transporte da carga. Em vez disso, a tarefa é realizada através de uma equipa constituída por múltiplos robôs que cooperam entre si. Tal será bastante útil em determinados setores da indústria onde o produto final apresenta dimensão e peso elevados, pelo que o uso de um único robô para realizar a sua movimentação poderá não ser a opção mais viável. Isto porque, a capacidade de carga e o tamanho do equipamento de transporte teriam de estar em conformidade

com as características do objeto a transportar. Por consequência, quanto mais pesado e volumoso este último for, menor será a manobrabilidade e maior será o tamanho e preço do robô. Poderão ainda existir outras vantagens que serão analisadas ao longo do relatório.

Posto isto, e tendo em conta os benefícios que podem advir da utilização destes sistemas, é do maior interesse que sejam estudadas estratégias que permitam aos seus constituintes coordenarem-se e movimentarem-se em conjunto.

1.2 Objetivos

Este trabalho tem como principal objetivo estudar um sistema de co-transporte constituído por duas plataformas móveis omnidirecionais. São exploradas estratégias *leader-follower* em que o *leader* é responsável por guiar o sistema pelo caminho a percorrer. O *follower*, por sua vez, irá estimar o seu movimento através da utilização de um sensor de força presente no suporte que lhe permite agarrar a carga, de modo a eliminar ou reduzir drasticamente a comunicação direta entre os dois robôs durante o transporte do objeto.

Visto não ser possível a utilização do sistema de robôs real, devido ao atual contexto de pandemia, o estudo foi realizado com recurso a uma ferramenta de simulação, o CoppeliaSim, da Coppelia Robotics.

De forma a atingir os objetivos, dividiu-se o trabalho pelas seguintes tarefas:

- estudo do co-transporte e levantamento de soluções já desenvolvidas nesta área;
- estudo dos conceitos necessários à compreensão do sistema, bem como de ferramentas que permitam a sua simulação;
- escolha do simulador e ambientação ao seu modo de funcionamento;
- análise das características do sistema de robôs reais e respetiva modelação no simulador;
- idealização das estratégias usadas na coordenação dos elementos do sistema, consoante os sensores integrados na estrutura mecânica do suporte do *follower*;
- desenvolvimento dos algoritmos de controlo;
- simulação das estratégias e análise dos resultados obtidos no CoppeliaSim.

1.3 Calendarização

Na Tabela 1.1 encontra-se representada a divisão das diferentes tarefas envolvidas na elaboração desta Tese de Mestrado, bem como a sua calendarização.

Tabela 1.1: Calendarização das tarefas

Tarefas	Início	Fim	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov
Pesquisa relativa ao estado da arte	09/02/2020	14/03/2020										
Pesquisa relativa aos conceitos teóricos	01/03/2020	09/05/2020										
Escolha e ambientação ao simulador	22/03/2020	18/04/2020										
Modelação do sistema	12/04/2020	09/05/2020										
Correção de características dos modelos	19/07/2020	01/08/2020										
Implementação dos algoritmos de controlo e simulação do sistema	03/05/2020	08/11/2020										
Redação do relatório	08/03/2020	14/11/2020										

1.4 Organização do Relatório

No primeiro capítulo - **Introdução** - é realizada a introdução do trabalho proposto. Este encontra-se dividido em quatro partes que contêm uma contextualização do trabalho, os objetivos que o motivaram, a sua calendarização e, por último, a estrutura da Dissertação.

O seguinte capítulo - **Revisão Bibliográfica** - inicia-se com a apresentação dos equipamentos que são utilizados para automatizar as operações de transporte na indústria. De seguida, são introduzidos os sistemas de co-transporte e conceitos que são fundamentais para a sua compreensão. Por último, é realizado o estudo de alguns sistemas que já foram implementados e que possuem determinadas características que vão de encontro à solução proposta neste trabalho.

Segue-se o terceiro capítulo - **Fundamentação Teórica** - onde são abordadas matérias que permitiram idealizar o sistema de co-transporte e que são cruciais para a compreensão do seu funcionamento.

No quarto capítulo - **Escolha e Introdução ao Simulador** - apresentam-se vários simuladores que podem ser usados para simular um MRS, bem como algumas das suas características mais importantes. Além disto, é ainda realizada uma introdução ao funcionamento do CoppeliaSim, de modo a facilitar a compreensão do trabalho realizado neste simulador.

O quinto capítulo - **Modelação do Sistema de Co-transporte** - descreve o sistema de robôs reais e as abordagens que poderão ser seguidas na implementação

dos suportes. Tendo isto em conta, será explicado como foi possível criar os modelos do sistema, de modo a poder realizar-se a sua simulação.

O sexto capítulo - **Implementação e Simulação das Estratégias de Co-transporte** - trata todo o processo da implementação dos algoritmos de controlo de ambos elementos do sistema no simulador. É também neste capítulo que são analisadas as diferentes estratégias exploradas e onde são apresentados todos os resultados obtidos através da sua simulação no CoppeliaSim.

No último capítulo - **Conclusões** - são expostas as considerações finais da dissertação, as principais dificuldades encontradas e algumas ideias para trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

O conteúdo deste capítulo pode ser dividido em duas grandes partes. Na primeira são abordadas as tecnologias existentes utilizadas para automatizar as operações de transporte na indústria. Relativamente à segunda, esta inclui diversos conceitos relacionados com a área do co-transporte, bem como uma análise de alguns projetos já realizados cujas características se enquadram e apresentam relevância para este trabalho.

2.1 Robótica Móvel e a Automatização do Transporte na Indústria

Inseridas em mercados altamente competitivos, muitas empresas vêem-se na necessidade de produzir produtos finais com uma qualidade cada vez maior. Simultaneamente, precisam de o fazer num intervalo de tempo reduzido e com o menor custo associado possível. Por estas razões, os processos de fabrico têm de ser alterados ou projetados de modo a eliminar qualquer atividade que não acrescente valor ao produto final e que possa significar custos adicionais desnecessários.

O transporte e manuseamento de cargas são operações essenciais para muitas empresas, porém, não acrescentam valor ao produto final. Por consequência, as empresas têm repensado os seus equipamentos e o modo como estas tarefas são realizadas nas suas fábricas e armazéns. Ao longo dos últimos anos, estas tarefas, que eram essencialmente feitas por operadores, manualmente ou com recurso a equipamentos como *trolleys*, porta-paletes, rebocadores e empilhadores (Figura 2.1), têm vindo a ser automatizadas.

Dependendo do tipo de processo de produção, a automatização do transporte é implementada com recurso, por exemplo, a tapetes de transporte e sistemas

Figura 2.1: Exemplos de equipamentos usados nas operações de transporte na indústria [9, 10]

automáticos. No que diz respeito aos últimos existem dois tipos, os AGV e os AMR, que são adotados para realizarem estas operações.

2.1.1 Veículos Automaticamente Guiados

O primeiro AGV surgiu em 1953, através da reconfiguração de um trator de reboque. O inventor, Arthur M. Barret Jr., adaptou o veículo de modo a que este pudesse ser guiado através de um cabo elétrico instalado no teto de um armazém, o que permitia que fosse operado automaticamente. Pouco tempo depois os AGV começaram a ser comercializados pela Barret Eletronics e eram guiados através da indução elétrica de um cabo soterrado no solo [11, 12].

Até aos dias de hoje esta tecnologia sofreu avanços consideráveis. Um bom exemplo é o facto da trajetória já não ter de seguir uma guia fixa (cabo elétrico, fita magnética ou linhas). Com a introdução de computadores de bordo mais potentes e tecnologia de sensorização mais avançada, foi possível que os AGV começassem a ter uma trajetória aberta. Este tipo de orientação do veículo é permitido devido à existência de sistemas mais complexos que recorrem, por exemplo, a sensores por lasers ou infravermelhos e refletores que permitem ao robô triangular a sua posição [13]. Com isto, houve um aumento da flexibilidade uma vez que os caminhos a percorrer começaram a ser implementados por *software*, sendo mais fácil alterá-los quando necessário. Outras características que melhoraram ao longo do tempo foram a comunicação, as baterias e outros componentes, aplicações de controlo e gestão de tráfego, etc [11].

As potencialidades e os benefícios que advêm da utilização dos AGV fazem com que tenham sido e sejam a aposta de muitas empresas. Conforme foram sendo aplicados em diferentes operações de transporte, começaram a surgir também vários tipos destes veículos. A sua generalidade pode ser categorizada em veículos de carga, reboque ou garfo [13]. Na Figura 2.2 pode ser observado um exemplo de cada uma destas categorias.

Figura 2.2: Tipos de AGV existentes - veículos de carga [14], reboque [15] e garfo [16], respetivamente

2.1.2 Robôs Móveis Autónomos

Ultimamente tem-se assistido a uma rápida expansão do uso dos AMR. Estes robôs, dependendo do tipo de aplicação na qual são usados, começam a ser escolhidos em detrimento dos tradicionais AGV por serem equipamentos "mais inteligentes".

Ao contrário dos AGV, os AMR são desenvolvidos de modo a orientarem-se livremente no espaço e têm a capacidade de restabelecer a sua rota. Por outras palavras, estes têm a capacidade de escolher a trajetória que pretendem percorrer sem qualquer restrição previamente definida [13, 17]. Assim sendo, isto faz com que estes robôs sejam a melhor escolha quando o transporte tem de ser feito em ambientes dinâmicos e que estejam sempre em constante mudança. São também bastante interessantes em casos em que tem de ser estabelecida uma cooperação entre os operadores e os robôs.

Um dos casos de maior sucesso da aplicação desta tecnologia foram os robôs desenvolvidos pela Kiva Systems (adquirida pela Amazon em 2012) que são usados no transporte de prateleiras em vários armazéns da Amazon, tal como se mostra na Figura 2.3. O *e-commerce* é um dos grandes impulsionadores do uso de AMR devido às características destes equipamentos [18]. No entanto, estes também são utilizados em operações idênticas às dos AGV. Tal como os últimos, podem rebocar, carregar e empilhar produtos. Na Figura 2.3 pode ser observado um exemplo de um AMR desenvolvido pela MIR, utilizado numa fábrica da Ford em Espanha para transportar materiais entre a linha de produção e o seu armazém. Como se pode verificar na figura, o robô realiza a sua operação num ambiente em que existem outros equipamentos móveis, tais como empilhadores.

Existe uma tendência para que as soluções de automatização sejam desenvolvidas de modo a permitirem uma maior flexibilidade e adaptação ao ambiente nas quais são instaladas. Tal não tem de significar o abandono de tecnologias mais antigas. Poderão existir situações em que um AGV com um sistema de trajetória fixa seja a solução ideal, não havendo a necessidade de despender dinheiro em

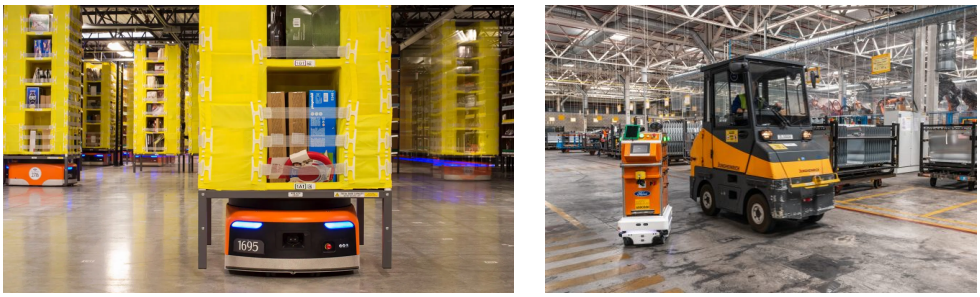


Figura 2.3: AMR utilizados nos armazéns da Amazon [19] e em fábricas da Ford [20], respetivamente

sistemas mais complexos. Porém, este será o caso de empresas cujo processo de produção não tenha tendência a alterar-se por longos períodos de tempo. Por outro lado, em empresas que tenham de repensar e alterar constantemente o seu sistema de produção será conveniente existir uma frota de robôs que se consiga adequar a estas mudanças.

2.1.3 Cooperação entre Robôs Móveis

Além das características intrínsecas a cada robô, têm sido estudadas técnicas que permitam que os robôs cooperem entre si, de modo a realizarem uma determinada operação. A possibilidade de, tal como humanos ou animais, trabalharem em conjunto para atingirem um objetivo, será importante para que se desenvolvam sistemas cada vez mais versáteis e com uma maior capacidade de resposta a adversidades. Adicionalmente, vários problemas serão resolvidos de forma mais eficiente e, por vezes, sem a necessidade de robôs tão complexos. Na Figura 2.4 são mostrados exemplos de sistemas que empregam a cooperação entre os seus elementos [21]. À esquerda os robôs recorrem à acoplação para transpor um sulco no solo, enquanto que à direita os robôs cooperam entre si para transportarem um objeto.

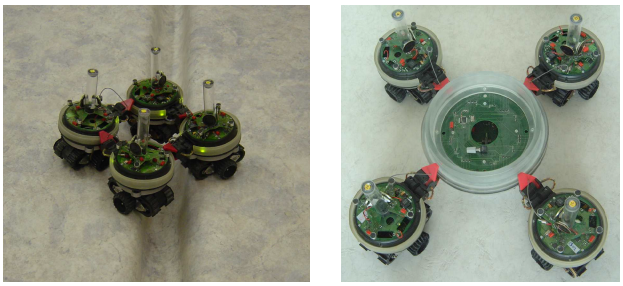


Figura 2.4: Exemplos de cooperação entre robôs [21]

Na indústria já existem soluções em que diferentes robôs móveis são capazes

de combinar esforços para realizar uma tarefa de transporte. Um bom exemplo é a plataforma omniMove da KUKA (Figura 2.5). Apesar de já terem uma capacidade de carga bastante elevada, estes robôs conseguem acoplar-se mecanicamente dando origem a uma plataforma ainda maior e com capacidade para suportar cargas mais pesadas. Segundo a empresa alemã, é possível a junção de duas (*tandem mode*) ou três plataformas (*tridem mode*), tal como se mostra na Figura 2.5 à direita [22].

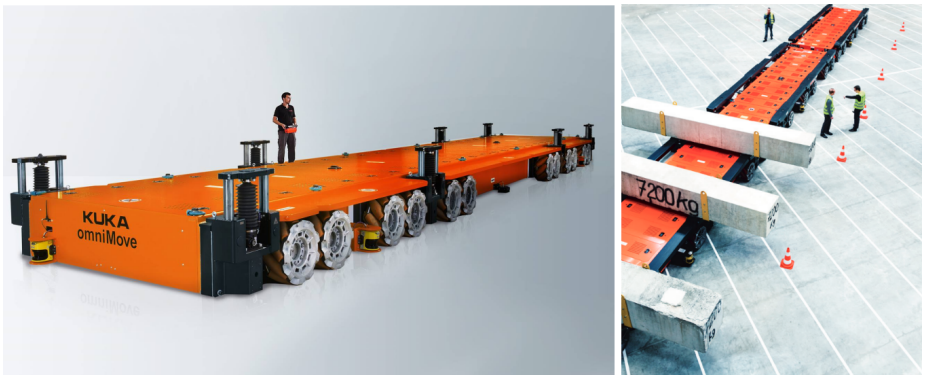


Figura 2.5: Exemplo de uma plataforma omniMove da KUKA [22] e do *tridem mode* [23], respetivamente

De forma semelhante, a Aritex também desenvolve plataformas omnidirecionais com esta propriedade [24]. Estes equipamentos, que no caso desta empresa são especialmente desenvolvidos para a indústria da aeronáutica, podem ser vistos na Figura 2.6. A aeronáutica é um dos setores que beneficia bastante da modularidade destas tecnologias, já que para construir um avião é necessário transportar estruturas com diferentes pesos e dimensões.

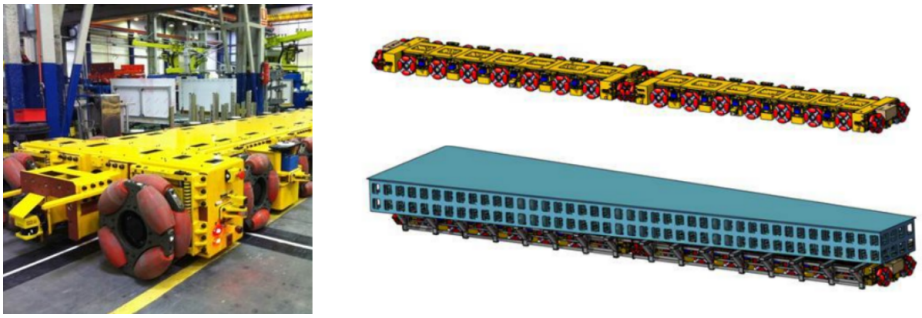


Figura 2.6: Plataforma omnidirecional da Aritex [24] e modo de acoplamento mecânico [25], respetivamente

2.2 Sistemas de Co-transporte

A cooperação entre robôs no transporte de uma carga é um tema que tem vindo a ser estudado desde os primeiros avanços na área dos sistemas multi-robô, ainda nos finais do séc. XX [26]. O interesse neste assunto surge da necessidade de criar sistemas eficientes que possam efetuar o transporte de objetos volumosos, pesados ou com formas complexas, que não poderiam ser carregados até ao destino apenas por um único robô. No entanto, num sistema deste género, nem todos os constituintes têm necessariamente de participar no transporte da carga. Estes também poderão ser responsáveis por monitorizar o ambiente em que a equipa se situa, sinalizar e até mesmo remover obstáculos para que a tarefa possa ser bem sucedida. Independentemente da função dos diferentes agentes, o sistema só será considerado cooperativo se o transporte não puder ser levado a cabo por um único robô e se forem implementados mecanismos de modo a que as ações dos agentes sejam coordenadas, para que estes se possam complementar uns aos outros [8].

Existem várias áreas em que o co-transporte pode ser usado. Neste trabalho são abordadas as operações de transporte na logística realizadas por robôs móveis (Figura 2.7 à esquerda). No entanto, também poderá ser aplicado, por exemplo, na realização do transporte de cargas pelo ar (Figura 2.7 ao centro) ou em ambientes submersos (Figura 2.7 à direita). Outras tarefas que beneficiarão do uso desta tecnologia são a recolha de resíduos ou substâncias perigosas para o ser humano, bem como o transporte de objetos em locais em que não seja possível existir a ação deste último.

Figura 2.7: Transporte cooperativo de cargas em terra [27], pelo ar [28] e na água [29]

Seja qual for a área em que é empregue, um sistema de co-transporte permitirá a aplicação de forças em diferentes pontos do corpo de um objeto. Estas ações independentes de cada elemento possibilitam uma maior destreza do que a que seria obtida com um único robô. Inclusivamente, estes sistemas poderão ser [7, 8, 30]:

- mais eficientes em relação aos consumos energéticos;

- mais robustos e tolerantes a falhas visto que poderão apresentar redundância;
- mais baratos quando comparados com a aquisição de um único robô de maior porte;
- mais facilmente modificados para serem aplicados em novos cenários ou em operações para as quais não foram inicialmente pensados (enquanto que, por exemplo, realizar o *retroffiting* de um robô de grande dimensão poderá ser difícil).

2.2.1 Conceitos Fundamentais dos Sistemas Multi-Robô

Os sistemas de co-transporte inserem-se na área dos MRS. O facto de este ser um tópico bastante abrangente faz com que existam diversos conceitos envolvidos que têm de ser abordados, para que se tenha uma noção geral de como estes se encontram implementados e como a cooperação entre os seus constituintes é orquestrada. De seguida serão explicados aspectos como a arquitetura de controlo, diferenciação dos elementos do sistema, comunicação, coordenação e o planeamento da trajetória dos constituintes de um MRS. Estes assuntos serão abordados porque o seu conhecimento é essencial no contexto deste trabalho.

2.2.1.1 Arquitetura de Controlo

A arquitetura de controlo de um MRS poderá ser centralizada, descentralizada, hierárquica e híbrida [26], tal como se mostra na Figura 2.8.

Caso seja centralizada, um elemento central do sistema será responsável por efetuar todo o controlo. Esta estratégia tem a grande vantagem de, se a unidade central tiver uma perspetiva geral do ambiente em que se insere, poder ser mais fácil criar o plano de controlo e enviar mensagens a todos os robôs para que estes obedeçam. Contudo, existem desvantagens que são inerentes aos sistemas que têm esta arquitetura, tais como:

- não poderem ser constituídos por um grande número de robôs, caso contrário o sistema será ineficaz;
- são pouco robustos em ambientes dinâmicos e a falhas;
- são mais vulneráveis, isto porque, caso se dê algum problema com o agente responsável pelo controlo, todo o sistema terá de ser desativado até que este seja reparado ou substituído.

Relativamente às arquiteturas hierárquicas, estas são localmente centralizadas e caracterizam-se pela existência de robôs que controlam grupos de outros robôs.

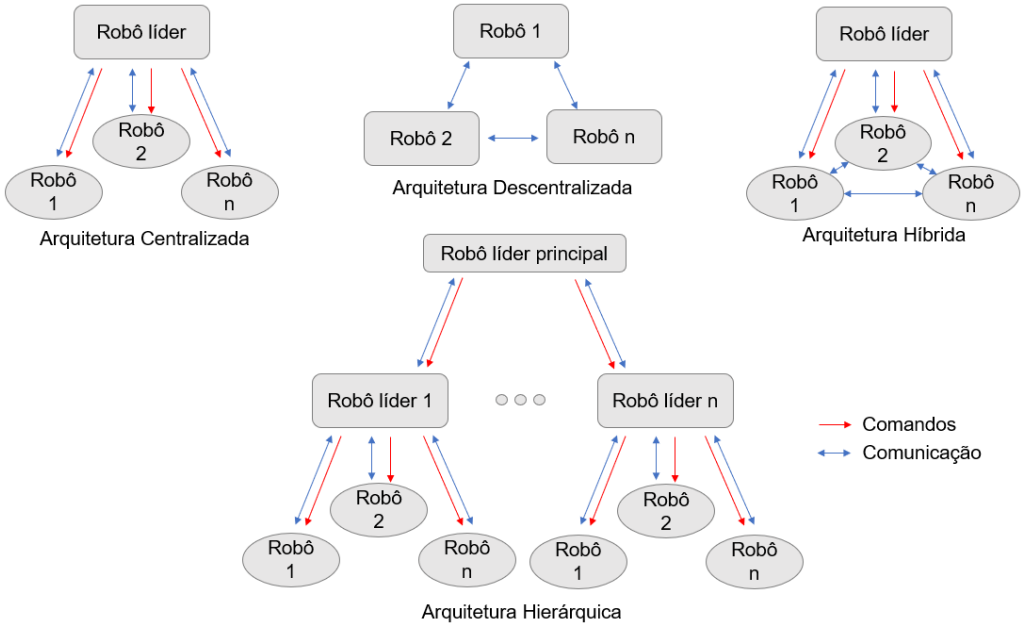


Figura 2.8: Possíveis arquiteturas de controlo dos MRS

Estes poderão ainda ser responsáveis por controlar um outro grupo, havendo assim uma delegação de autoridade por parte do principal líder. Isto faz com que estes sistemas sejam mais dinâmicos e possam ser constituídos por um maior número de robôs (quando comparados com os de arquitetura centralizada). No entanto, são também bastante vulneráveis a erros nos líderes dos níveis mais altos.

As arquiteturas descentralizadas são as mais comuns nos MRS (bem como na área do co-transporte). Estas caracterizam-se pelo facto dos robôs serem igualmente responsáveis pelo controlo da operação e autónomos quanto à tomada de decisão. Deste modo não existe um elemento que seja responsável por controlar outro, sendo que as ações são tomadas com base no conhecimento local de cada constituinte do sistema. Por outro lado, poderá ser mais complicado chegar à sincronização e coerência global entre os vários elementos.

Por último, de modo a combinar vantagens das arquiteturas explicadas anteriormente, existem arquiteturas híbridas. Nos sistemas que as adoptam pode existir um robô responsável por monitorizar as ações da equipa e enviar uma resolução geral daquilo que tem que ser feito. Os robôs irão comunicar entre si e com o líder de modo a poder executar as tarefas que lhes são impostas.

2.2.1.2 Diferenciação dos Elementos do Sistema

Um MRS poderá ser homogéneo ou heterogéneo [30]. Num sistema homogéneo, tal como o nome indica, os robôs são idênticos e possuem as mesmas capacidades,

o que não implica que sejam iguais em termos físicos ou aspetuais. As vantagens de implementar robôs análogos passam por possibilitar a existência de redundância e paralelismo, além de que, estes serão completamente intercambiáveis. Na Figura 2.9, à esquerda, é possível verificar um grupo com um grande número de robôs simples e homogéneos. Os sistemas deste género são, muitas vezes, alvo de estudo da robótica de enxame, que é uma área científica enquadrada nos MRS.

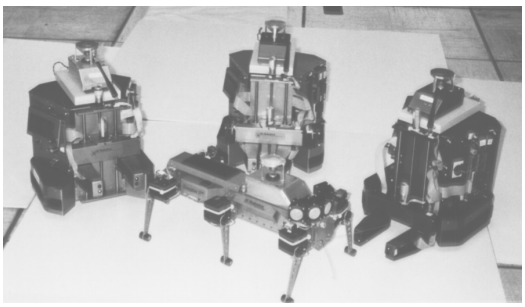



Figura 2.9: Grupo de robôs homogéneos [31] e heterogéneos [32], respetivamente

Por outro lado, existem operações que requerem que os robôs sejam dotados de um variado conjunto de tecnologias que é impossível implementar num único robô, ou então, que é muito dispendioso de replicar em todos os elementos. Estas razões levam a que sejam criados sistemas que apresentam heterogeneidade (Figura 2.9, à direita). Nestes sistemas, os robôs podem ser desenvolvidos especificamente para um tipo de operação. Porém, estão também associados a um maior grau de complexidade naquilo que diz respeito ao planeamento da alocação das tarefas, já que um único robô não conseguirá efetuar qualquer trabalho que lhe seja imposto.

Relativamente ao co-transporte, tanto a homogeneidade como a heterogeneidade têm sido exploradas em equipas de robôs [8].

2.2.1.3 Comunicação

A comunicação, como modo de interação entre os agentes de um MRS, é essencial para que exista coordenação. Através dela os robôs podem partilhar informação sobre a sua posição, sensores e o estado do ambiente em que se inserem. Isto fará com que cada elemento possa saber quais são as intenções ou os objetivos dos restantes, e que possa agir de modo a complementar as suas ações para que a tarefa que é incumbida ao sistema seja corretamente executada.

A comunicação entre robôs pode ser classificada como direta ou indireta, sendo que a última poderá ainda ser dividida em ativa (Estigmergia) e passiva (Reconhecimento de ações) [26, 30, 33], tal como se mostra na Figura 2.10. Estas três técnicas de comunicação serão explicadas de seguida:

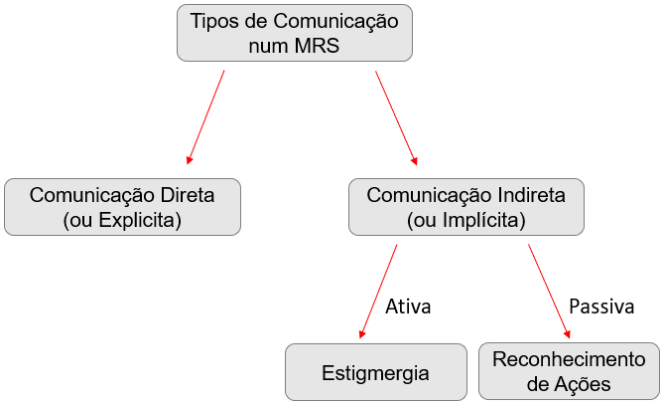


Figura 2.10: Tipos de comunicação mais comuns num MRS

- **comunicação direta:** este tipo de comunicação refere-se à troca direta de informação entre robôs, e requer que estes possuam um módulo de comunicações para que tal possa ser feito;
- **estigmergia:** este tipo de comunicação é alvo de estudo da biomimética e é inspirado no comportamento de colónias de insetos como as formigas. Um robô capaz de comunicar por estigmergia apercebe-se das ações dos outros através das alterações que estes provocam no meio em que se inserem;
- **comunicação através do reconhecimento de ações:** um robô poderá também, caso seja dotado dos sensores necessários, detetar diretamente as ações dos outros.

No co-transporte tanto a comunicação direta como a indireta são utilizadas [8, 34]. A primeira é a mais estudada e frequentemente adotada já que leva a análises matemáticas mais simples e a uma maior eficácia do sistema. Todavia, existem situações em que o ambiente no qual os robôs realizam a operação é propenso a falhas. Isto poderá originar problemas como a queda do objeto, a aplicação de forças excessivas e a diminuição do desempenho do sistema. Além disso, conforme o número de robôs aumenta, também serão necessárias redes de comunicações mais complexas e poderão surgir problemas relacionados com a largura de banda. Estas desvantagens podem ser solucionadas com o uso da comunicação indireta. Embora apresente maior complexidade a nível da análise matemática, este tipo de comunicação permitirá uma maior robustez do sistema em relação a falhas na comunicação. Ainda que a comunicação direta, quando corretamente implementada, proporcione um melhor desempenho, em certas operações não trará qualquer benefício em relação à indireta. O uso de ambos os tipos de comunicação em simultâneo também poderá ser vantajoso, dependendo da exigência e das condições em que a operação é realizada [34].

Na Figura 2.11 são apontadas as formas como a comunicação indireta é normalmente implementada nos sistemas de co-transporte. Algumas das técnicas usadas envolvem a utilização de sensores de força ou binário. Assim, o robô consegue aperceber-se das forças que estão a ser aplicadas ao objeto pelos outros robôs e pode estimar a sua trajetória. Também poderão ser usados sensores para detetar os movimentos ou ações dos outros robôs. Quanto à comunicação direta, esta é, por norma, implementada através de redes sem fios.

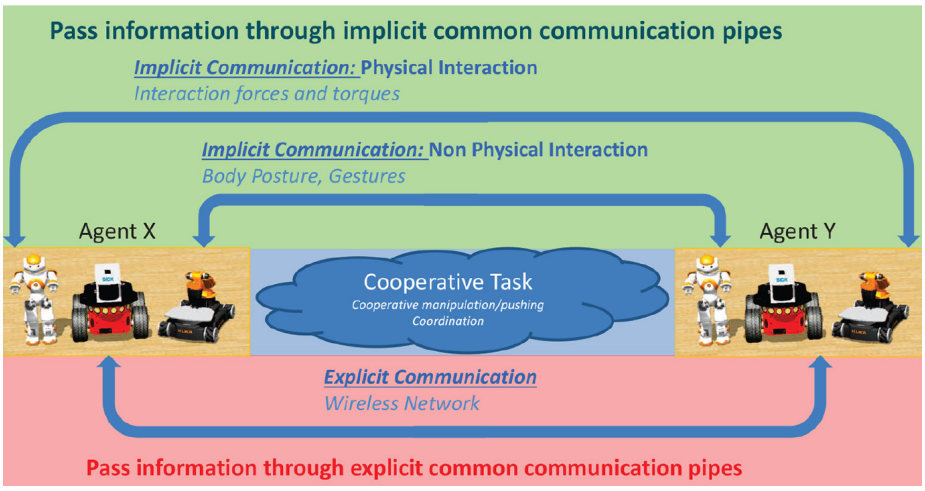


Figura 2.11: Tipos de comunicação mais usados nos sistemas de co-transporte [34]

2.2.1.4 Coordenação do Sistema

Num MRS com características cooperativas é necessário que os robôs se comportem como uma equipa, logo, a coordenação é essencial para que o sistema possa atingir o seu objetivo. Esta última poderá ser estática ou dinâmica [33].

Por coordenação estática entende-se que os robôs, antes de partir para a execução da tarefa, sabem como terão de agir em conjunto. No caso da coordenação dinâmica os agentes do sistema agem conforme a informação que adquirem (direta ou indiretamente) durante a realização de uma tarefa.

Embora possam existir sistemas que usem apenas um destes métodos de coordenação, o mais normal é que sejam empregues os dois em simultâneo [33]. Deste modo existem comportamentos que já estarão previamente definidos e outros que serão resultado daquilo que ocorre durante a execução das tarefas.

A coordenação é ainda importante para que sejam evitados conflitos de recursos inerentes à natureza destes sistemas. Estes estão normalmente relacionados com a partilha do espaço, a movimentação de objetos e a largura de banda da rede que os robôs poderão usar para comunicar entre si.

2.2.1.5 Planeamento da Trajetória

Para um robô móvel, o planeamento da sua trajetória é essencial. Este só conseguirá cumprir o seu objetivo se se mover no espaço e conseguir evitar os obstáculos. Quando se considera um conjunto de robôs este planeamento torna-se mais complicado, isto porque os vários robôs terão de evitar, não só os obstáculos presentes no ambiente, mas também os outros robôs que constituem o sistema e que se irão tornar obstáculos móveis.

Num MRS, o planeamento da trajetória poderá ser centralizado ou distribuído. No primeiro caso, uma unidade central é responsável por tomar todas as decisões necessárias para conduzir o sistema até ao destino, enquanto que, no segundo, cada robô irá planear a sua trajetória [30]. No que diz respeito aos sistemas de co-transporte ambas as abordagens poderão ser utilizadas.

O planeamento centralizado tem sido alvo de bastantes estudos devido a algumas das suas particularidades que poderão ser de grande interesse nas operações de transporte. Os sistemas que adoptam esta metodologia são normalmente constituídos por um robô líder, que sabe a trajetória a percorrer, e por um ou vários robôs que ajudarão no transporte da carga mas sem qualquer conhecimento em relação à sua posição ou ao destino (estratégia *leader-follower*).

Para aplicações como o co-transporte, poderá ser também necessário que os robôs formem um padrão para carregar o objeto até ao destino (Figura 2.12, à esquerda). Nestes casos, tal é feito através da comunicação entre os agentes e poderão ser implementados mecanismos de modo a facilitar o movimento do sistema em formatura sem que seja largado o objeto, tal como se mostra na Figura 2.12, à direita. O objeto castanho funciona como uma palete na qual o objeto é transportado, além disso cria atrito no "nariz" dos robôs e permite que estes estejam sempre orientados.

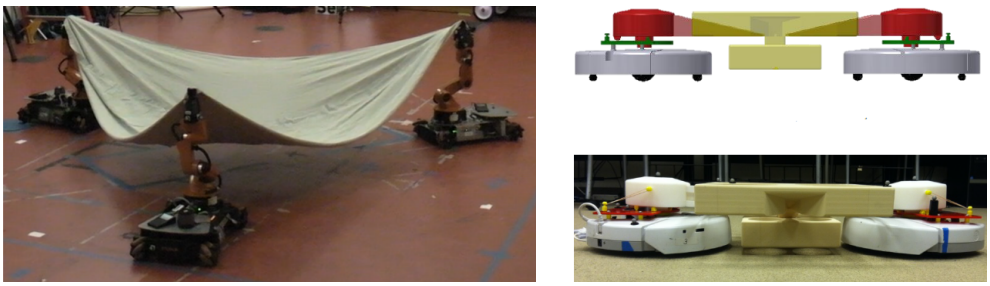


Figura 2.12: Exemplo de uma formatura com três robôs para carregar um objeto deformável [35] e sistema com mecanismo que facilita o transporte de um objeto em formatura (adaptado de [6]), respetivamente

Existem muitas outras questões que surgem conforme a complexidade e o número de robôs do sistema aumentam. Muitas delas relacionadas com aspetos

como a alocação dinâmica de tarefas aos vários robôs, a aprendizagem coletiva do sistema (*machine learning*), entre outros. Ao contrário dos conceitos apresentados anteriormente, não será feita uma análises destas características, uma vez que o sistema em estudo não apresenta tão elevada complexidade.

2.2.2 Principais Estratégias de Transporte

Num sistema de co-transporte os robôs necessitam de usar estratégias que lhes permitam, em conjunto, mover um objeto. Segundo a análise da literatura feita por Tuci, Alkilabi e Akanyeti [8] existem três modos de transporte que são geralmente usados e que serão explicados de seguida:

- ***Pushing***: nos sistemas que utilizam esta estratégia de transporte os robôs não têm, por norma, capacidades de agarrar ou pegar no objeto. Sendo assim, deverão coordenar as forças exercidas para poder empurrá-lo até à posição final (Figura 2.13). Embora o conceito aparente ser de baixa complexidade, deverá existir um grande nível de coordenação entre os robôs para iniciar a movimentação do objeto e para que se consiga manter uma trajetória estável.

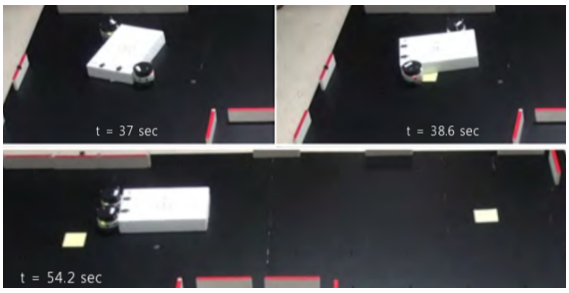


Figura 2.13: Exemplo de um sistema utilizando *pushing* (adaptado de [36])

- ***Caging***: é um caso especial da técnica apresentada anteriormente em que o objeto também é arrastado, no entanto há uma ligeira diferença. Nestes sistemas os robôs posicionam-se de maneira a envolver e prender o objeto entre eles. Para que o transporte seja feito corretamente, os vários elementos do sistema deverão manter o objeto enclausurado conforme percorrem a trajetória. A Figura 2.14 permite verificar o comportamento de um sistema que usa esta estratégia.

- ***Grasping***: os sistemas que utilizam esta técnica, tal como o que se apresenta na Figura 2.15 à esquerda, são constituídos por robôs que possuem algum mecanismo que lhes permita agarrar a carga. Quando comparada com as duas estratégias apresentadas anteriormente, esta é a que possibilita

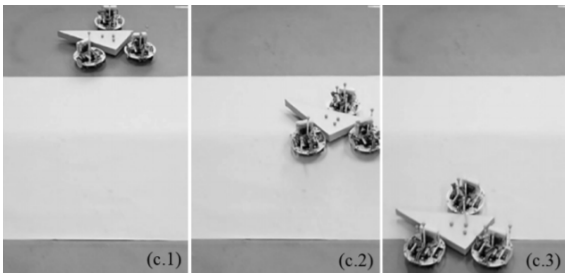


Figura 2.14: Exemplo de sistema utilizando *caging* [37]

um melhor controlo durante o transporte. Isto porque, após os robôs agarrarem o objeto, este tanto poderá ser empurrado como puxado, havendo assim uma maior manobrabilidade. Nesta categoria, os autores incluem também os sistemas em que os robôs transportam a carga no topo do seu corpo, mesmo que não seja agarrada diretamente (Figura 2.15 à direita). Isto porque, de forma análoga ao que acontece nos sistemas referidos anteriormente, os robôs têm de realizar o transporte sem perder o contacto físico com o objeto.

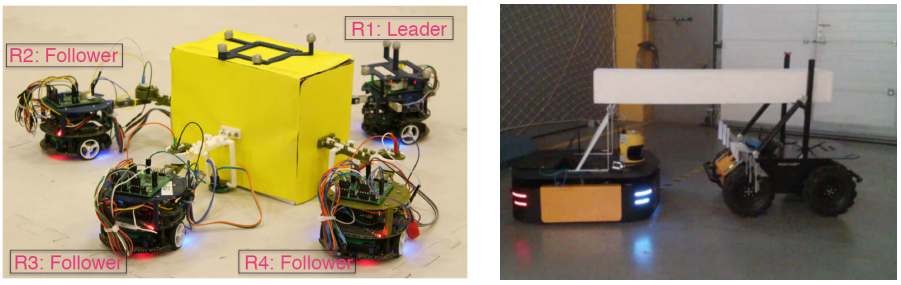


Figura 2.15: Exemplos de sistemas utilizando *grasping* [25, 38]

2.2.3 Projetos de Interesse Realizados na Área

A cooperação entre robôs no transporte de um objeto é um assunto que já foi tratado em vários estudos. Uma vez que se pretende idealizar um sistema destes é crucial que se analise o trabalho que já foi feito, para se poder tirar conclusões acerca de algumas das suas particularidades. De seguida são esmiuçados alguns projetos realizados na área e que apresentam características que são de interesse para este trabalho.

No trabalho realizado por Pereira *et al.* [39] é estudada a coordenação entre dois robôs não-holonómicos homogéneos, recorrendo ao uso da comunicação indireta. Cada plataforma é constituída por sensores de proximidade e contacto, sendo que a comunicação indireta é implementada através de dois dispositivos como o que é mostrado na Figura 2.16, e que se encontram na estrutura que é

responsável por agarrar o objeto. Cada um deles utiliza um potenciómetro rotacional e uma mola, permitindo traduzir as forças que são impostas no objeto, e consequentemente num dos robôs, num valor de resistência variável. Uma característica interessante do sistema é ter sido implementado com uma estratégia *leader-follower*, no entanto, ambos os elementos podem exercer os dois papéis consoante a dinâmica do ambiente onde se inserem o exigir. Por outro lado, apenas um deles sabe a trajetória que tem de ser percorrida, pelo que deverá ser esse a iniciar e acabar o movimento, enquanto que o outro apenas assumirá a liderança quando for necessário realizar uma manobra. Uma vez que as plataformas são não-holonómicas, isto irá facilitar quando for encontrado um obstáculo e for necessário recuar para, posteriormente, contorná-lo. Por último, convém ainda referir que a troca de líder acontece quando um robô recua, sendo que o outro se deverá aperceber do movimento e tomar a liderança.

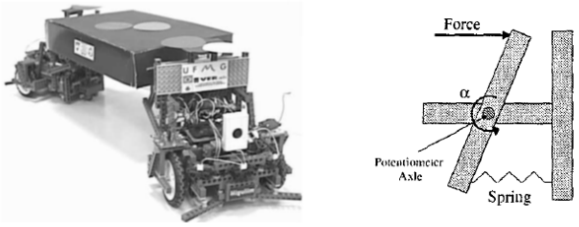


Figura 2.16: Sistema desenvolvido por Pereira *et al.* e respetivo dispositivo de acoplamento ao objeto (adaptado de [39])

Com o sistema apresentado anteriormente, os autores foram capazes de realizar vários testes, de modo a concluir sobre a eficiência da utilização da comunicação indireta. Quando comparado com o uso de comunicação direta, que permitiu que os robôs alcançassem sempre o objetivo, com a comunicação indireta tal só aconteceu em 80% dos testes. As falhas deveram-se maioritariamente à realização da troca do líder e ao facto de, depois de se dar a troca, existir uma má estimação da trajetória por parte do *follower*. Por outro lado, em nenhum caso o sistema largou o objeto. Os resultados obtidos foram bastante satisfatórios e mostram que é possível implementar o transporte de objetos usando a comunicação indireta.

No trabalho [38], Wang e Schwager estudaram um algoritmo de controlo de modo a coordenar as forças de um grupo heterogéneo de quatro robôs não-holonómicos no transporte de um objeto. Este sistema encontra-se representado na Figura 2.15. A sua arquitetura de controlo é descentralizada e nenhum dos robôs tem conhecimento da existência dos restantes. Tal como no caso anterior, o *leader* é responsável por guiar todos os *followers* pelo caminho a percorrer, cuja função será auxiliar no transporte aplicando força ao objeto. A constituição dos robôs é mostrada na Figura 2.17. Os *followers* são constituídos por uma garra que é apa-

rafusada à caixa a transportar antes de se iniciarem as experiências. Esta possui um sensor laser, para o robô poder estimar a velocidade e direção do movimento do objeto, e está conectada a uma das duas células de carga que estão montadas perpendicularmente no corpo do robô. Isto possibilita a monitorização das forças impostas no objeto num plano bidimensional, que será importante para efetuar o controlo dos motores. O robô *leader* não possui sensor laser, uma vez que irá ser ele a impor a velocidade ao sistema. É ainda de referir que, devido ao modo como foi implementado, o controlador só é eficaz durante o movimento. Sendo assim, para iniciar o movimento do sistema, os autores utilizaram um método no qual os robôs aplicam forças aleatórias. Assim que a força resultante seja maior que a força de atrito o objeto irá movimentar-se e o sistema começará a funcionar normalmente. Tudo isto é feito sem comunicação direta entre os diversos elementos.

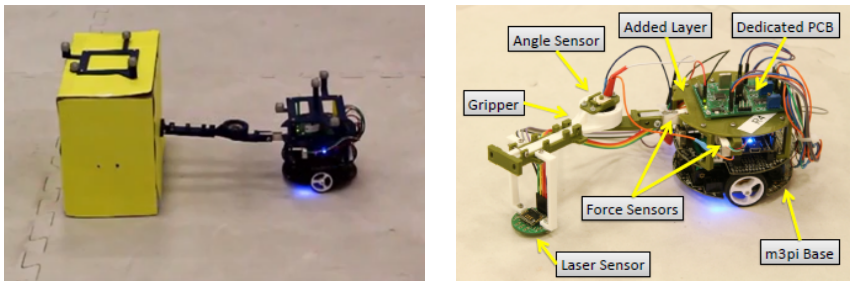


Figura 2.17: Robôs *leader* e *follower*, respetivamente, do sistema apresentado em [38]

Neste estudo os autores realizam diversos testes, entre os quais provam o funcionamento do sistema e que, inclusivamente, o *leader* poderá ser um robô autónomo, teleguiado ou, até mesmo, uma pessoa.

O estudo feito por Machado *et al.* [40] apresenta um sistema heterogéneo, com uma arquitetura descentralizada, constituído por dois robôs diferenciais, como se pode verificar na Figura 2.18. Neste sistema o *leader* irá dirigir-se até ao destino recorrendo a um sistema de visão omnidirecional que lhe permite identificar locais alvo. Por sua vez, o *follower* deverá manter uma distância apropriada entre os dois. Para este efeito, os robôs possuem um suporte com uma base constituída por duas juntas, uma prismática e outra rotacional. Conforme o *leader* se desloca, a carga irá mover-se ao longo da junta prismática do suporte do *follower*, sendo que a junta rotacional permitirá à base rodar à medida que são efetuadas mudanças de direção. Através da monitorização do *offset*, gerado pela deslocação das juntas, e da direção do movimento do *leader* (parâmetro que é comunicado diretamente), o *follower* conseguirá estimar a sua trajetória. Além do suporte mencionado, cada robô está equipado com um anel de sensores para realizar a deteção de obstáculos.

Com a realização de várias experiências, tanto em cenários reais como em si-

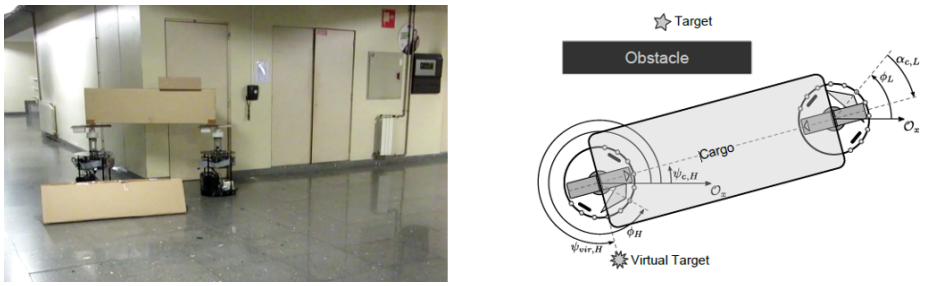


Figura 2.18: Sistema apresentado em [40]

mulação, os autores demonstram as diferentes capacidades do sistema, tais como: navegação em ambientes desconhecidos, manobrabilidade em curvas em forma de U ou L, desvio de obstáculos estáticos e dinâmicos, procura de novas trajetórias aquando da ocorrência de mudanças no *layout* e também a capacidade para transportar objetos de diferentes tamanhos. Quando comparado com os trabalhos já apresentados, este sistema possui uma vasta gama de capacidades, no entanto, os robôs que o constituem são dotados de um conjunto de tecnologias mais complexas. Por outro lado, também são usadas a comunicação direta e indireta em simultâneo, embora o uso da primeira seja minimizado ao máximo (apenas um parâmetro).

Outro projeto particularmente interessante foi desenvolvido por Tuci *et al.* [21]. Neste trabalho os autores estudam a capacidade de um grupo de robôs para transportarem um objeto, sendo que estes não necessitarão obrigatoriamente de o agarrar - também poderão acoplar-se entre si (técnica designada por *self-assembly*). Este comportamento pode ser visto em algumas colónias de insetos e é utilizado para ultrapassar adversidades presentes no ambiente ou impostas pelo próprio objeto a transportar. Nos sistemas de co-transporte poderá ser útil, entre outras aplicações, para realizar o transporte em terrenos com sulcos e pisos irregulares.

No que diz respeito ao grupo de seis robôs utilizados no estudo, este apresenta homogeneidade e uma arquitetura descentralizada. Ao contrário do que acontecia anteriormente não é implementada uma estratégia *leader-follower*, já que todos os robôs possuem meios para estimarem a trajetória a percorrer. Além disso, o objeto não se encontra acoplado no início da experiência, pelo que deverá ser localizado, agarrado e transportado até ao destino pelos robôs. Na Figura 2.19 pode ser observado o ambiente onde o sistema foi testado, um dos robôs utilizados e o objeto a transportar.

A experiência começa com o objeto posicionado tal como se ilustra na Figura 2.19 à esquerda, e os robôs são posicionados em redor e aleatoriamente (Figura 2.20, à esquerda). Numa primeira fase estes terão um anel com vários *Light Emitting Diode* (LED) com cor azul e o objeto apresentará um anel LED

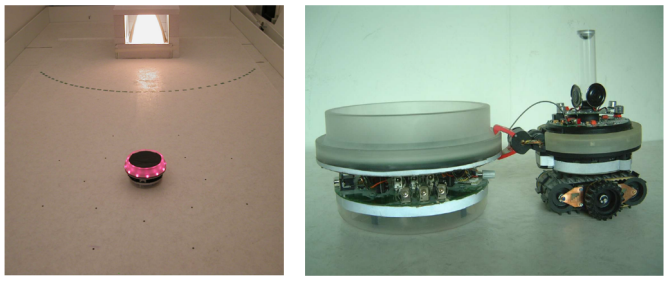


Figura 2.19: Arena de testes, carga e robô usados em [21]

com cor vermelha. Através de um sistema de visão omnidirecional cada elemento deverá identificar o objeto com cor vermelha mais próximo e acoplar-se a ele. Assim que o faça, os LED passarão a ter também cor vermelha. Isto possibilita aos robôs evitar colisões enquanto se deslocam até ao objeto, e, numa segunda fase, agarrarem-se uns aos outros. Logo que o último robô acabe de se acoplar o sistema tentará mover o objeto. Para isso, os robôs alinham o seu chassi em relação ao emissor de luz e começam a puxar nessa direção, tal é permitido devido à existência de uma junta rotacional entre o chassi e o corpo do robô. Durante o transporte, o binário no sistema de tração e o ângulo do chassi em relação ao foco de luz estão constantemente a ser monitorizados. Relativamente ao primeiro, caso seja notada alguma anomalia (valor muito alto), o robô irá estagnar e realizar um movimento de recuperação, para que o *hardware* não seja danificado. Em relação ao ângulo, se existir um desalinhamento bastante grande o robô irá parar e girar sobre si para o corrigir. Caso seja um desalinhamento leve o robô irá curvar conforme se movimenta, não havendo necessidade de parar. Na Figura 2.20 pretende-se ilustrar todo o processo descrito acima quando bem sucedido.

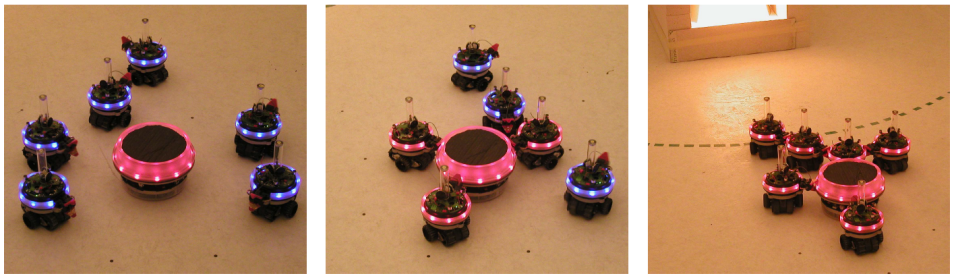


Figura 2.20: Exemplificação do comportamento do sistema apresentado em [21], quando o transporte é bem sucedido

Nas trinta experiências realizadas apenas quatro não foram bem sucedidas. Tal deveu-se ao facto de algum dos robôs não conseguir agarrar-se à carga ou a outro robô, o que fazia com que o sistema não pudesse avançar para a fase de transporte e não concluísse a tarefa dentro do tempo limite (cinco minutos). Isto

mostra que a principal falha tem a ver com o controlo do *self-assembly* e não com a estratégia de transporte implementada. Esta última permitiu, sempre que possível, que o sistema atingisse o objetivo.

O projeto GEOMOVE [25], desenvolvido por Rizzo, Lagraña e Serrano, estuda a cooperação entre dois robôs para efetuar o transporte de estruturas de grandes dimensões na indústria da aeronáutica. O objetivo deste projeto consistiu na implementação de um sistema capaz de transportar uma carga sem ser necessária uma conexão mecânica entre as plataformas (técnica que é utilizada pelas soluções que já se encontram no mercado). Para isso, os autores inspiraram-se no comportamento de rotação sincronizada que pode ser visto em alguns corpos celestes, como é o caso de Plutão e da sua lua Caronte.

O sistema utilizado é constituído por dois robôs, um *master* e um *slave*, que deverão manter a distância e o alinhamento entre si, para que a carga não seja largada ou sofra torções que possam resultar na sua danificação. No que diz respeito aos movimentos lineares, este problema é simples, já que apenas bastará aplicar as mesmas velocidades no espaço bidimensional aos dois robôs. Por outro lado, quando o *master* executa um movimento rotacional ou uma curva, o problema complica-se. Isto porque, além de ter de rodar em torno do seu eixo, o *slave* terá também que realizar um movimento de translação em torno do mesmo centro de rotação do *master*, tal como se exemplifica na Figura 2.21.

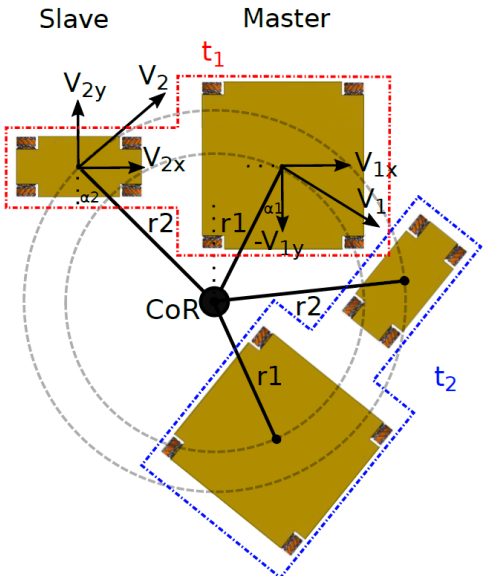


Figura 2.21: Exemplificação da rotação sincronizada entre o robô *master* e o *slave* do sistema apresentado em [25]

Através do conceito representado pela figura anterior foi possível obter as equações do movimento do sistema. Porém, para efetuar um controlo rigoroso,

estas não foram suficientes, já que o atrito, inércia, atrasos e outros fatores faziam com que o sistema perdesse a sincronia rapidamente. Sendo assim, para realizar o controlo em malha fechada, os autores aproveitaram um dos sensores *Light Detection and Ranging* (LiDAR) que já se encontrava instalado no robô *slave* e colocaram refletores na traseira do robô *master* (Figura 2.22, à esquerda). Deste modo, o *slave* consegue detetar o padrão dos refletores, compará-lo com uma referência que é guardada no início da experiência com os robôs perfeitamente alinhados e, por fim, corrigir o seu alinhamento e distância em relação ao *master*.

O sistema de dois robôs omnidirecionais desenvolvidos pode ser visto na Figura 2.22. Para validar o sistema implementado os autores controlam o robô *master* remotamente, todavia ambos os robôs possuem vários LiDAR para realizarem a deteção de obstáculos e navegação autónoma. Além disto, os robôs comunicam diretamente. O robô *master* é responsável por calcular as velocidades de todo o sistema e comunicar ao *slave*. Este, por sua vez, e tendo em consideração as correções a realizar, deverá ajustar os parâmetros da velocidade que lhe são comunicados pelo *master* de modo a garantir a sincronia.

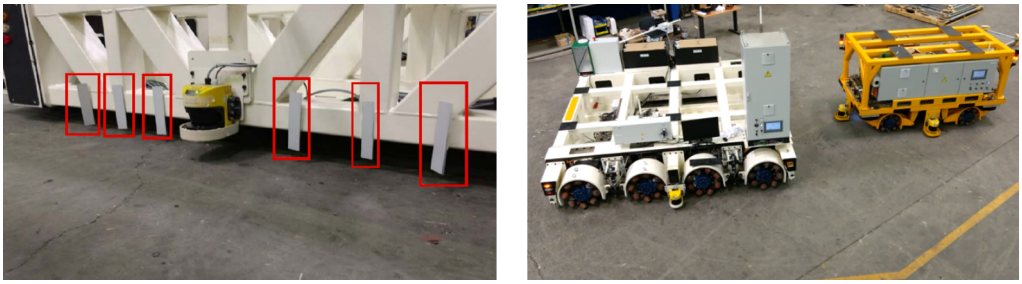


Figura 2.22: Refletores usados na traseira do *slave* e sistema de dois robôs apresentado em [25], respetivamente

Os resultados alcançados com a técnica explicada foram bastante satisfatórios, sendo que o erro angular médio entre as duas plataformas manteve-se sempre abaixo dos 0,32° e o erro linear médio entre as plataformas nunca ultrapassou os 10 mm.

2.2.3.1 Considerações Acerca dos Projetos Apresentados

Na Tabela 2.1 encontram-se sintetizadas as características e também os sensores que constituem cada um dos sistemas que foram vistos anteriormente. Através da sua análise, verifica-se que existem diferentes métodos e tecnologias que poderão ser usados para implementar um sistema de co-transporte. Ainda que apresentem disparidades, estes trabalhos permitem concluir acerca de algumas propriedades dos sistemas de co-transporte que também são utilizadas neste trabalho.

Tabela 2.1: Síntese das características dos projetos apresentados

Caraterísticas dos sistemas apresentados	**Pereira *et al.* [39]**	**Wang e Schwarger [38]**	**Machado *et al.* [40]**	**Tuci *et al.* [21]**	**Rizzo, Lagraña e Serrano [25]**
Arquitetura de controlo	Descentralizada	Descentralizada	Descentralizada	Descentralizada	Descentralizada
Número de robôs	2	4	2	6	2
Diferenciação dos elementos	Sistema homogéneo	Sistema heterogéneo	Sistema heterogéneo	Sistema homogéneo	Sistema heterogéneo
Tipos de comunicação utilizados	Comunicação indireta	Comunicação indireta	Comunicação direta e indireta	Comunicação indireta	Comunicação direta e indireta
Planeamento de trajetória	Centralizado (*leader-follower*)	Centralizado (*leader-follower*)	Centralizado (*leader-follower*)	Descentralizado	Centralizado (*leader-follower*)
Técnica de transporte	*Grasping*	*Grasping*	*Grasping*	*Grasping*	*Grasping*
Tipo de plataformas	Acionamento diferencial	Acionamento diferencial	Acionamento diferencial	Acionamento diferencial	Omnidirecional
Sensores usados para implementar o co-transporte	Sensores de proximidade e contacto, 2 sensores de força, *Encoders*	***Leader:*** Não possui sensores, recebe comandos de um controlador externo ***Follower:*** Sensor laser, 2 células de carga, sensor de ângulo	***Leader:*** Possui o mesmo equipamento do *follower* e um sistema de visão omnidirecional ***Follower:*** Anel com 11 sensores para a deteção de obstruções, suporte com 2 graus de liberdade e com monitorização do *offset* das juntas	19 sensores de proximidade, 8 sensores de luz, sensor de força, sensor de binário, sistema de visão omnidirecional	***Leader*** e ***follower:*** 4 LiDAR SICK S3000

Todos os sistemas possuem uma arquitetura de controlo descentralizada. Tal como já tinha sido mencionado, devido às suas vantagens, este é o tipo de arquitetura mais utilizada em sistemas de co-transporte e nos MRS em geral. Além disso, quando os robôs comunicam indiretamente terão de possuir a capacidade de processar a informação que recolhem do ambiente, para se contextualizarem no meio onde se enquadram ou na tarefa que estão a realizar. Isto faz com que seja necessário distribuir o controlo pelos diversos elementos do sistema.

No que diz respeito à comunicação indireta, depois da análise dos trabalhos conclui-se que existem diversas maneiras de a implementar e vários contextos em que poderá ser útil. Nos três primeiros projetos apesentados a comunicação indireta é usada pelos *followers* para poderem estimar a sua trajetória. No quarto trabalho é empregue pelos robôs para que se possam identificar durante a fase de *self-assembly*. No último, é usada pelo robô *slave* para poder corrigir o alinhamento e a distância em relação ao *master*. Apesar de ser insuficiente em algumas aplicações de maior complexidade e implique o desenvolvimento de modelos matemáticos mais complexos, esta será crucial para diminuir ao máximo a quantidade de informação que é comunicada diretamente. Isto faz com que seja poupada largura de banda e que o sistema tenha um melhor desempenho em ambientes propensos a erros. Conclui-se também que as técnicas que envolvem comunicação indireta através de sensores de força são bastante exploradas em sistemas em que não há comunicação explícita acerca das velocidades entre os

robôs.

Relativamente à estratégia *leader-follower* esta poderá simplificar o problema do planeamento da trajetória do sistema, já que apenas um robô será responsável por o fazer. Além do que, quando bem implementada, os robôs serão dotados de um comportamento análogo ao que pode ser visto nos humanos. Quando um robô não for capaz de carregar um objeto sozinho poderá ser ajudado por outro, mesmo que este não tenha noção acerca da posição final do objeto.

Convém ainda referir que, a mobilidade dos robôs é algo bastante importante para implementar um sistema de co-transporte. Com a análise do projeto GEOMOVE [25], constata-se que os robôs com mobilidade omnidirecional conferem uma maior manobrabilidade ao sistema. Esta característica será bastante importante em aplicações em que os robôs tenham de realizar operações num ambiente dinâmico.

Capítulo 3

Fundamentação Teórica

O objetivo deste capítulo passa por apresentar conceitos essenciais para a compreensão do funcionamento do sistema de co-transporte em estudo. Inicialmente serão abordadas noções acerca da mobilidade omnidirecional, dando-se mais relevância às plataformas que possuem rodas mecanum, já que são as utilizadas neste estudo. Depois disto serão analisadas algumas tecnologias que existem no mercado e que permitem realizar a medição de forças, algo que é essencial para este trabalho e que será a base da cooperação entre os robôs. Por fim, é realizada uma introdução ao controlo Proporcional-Integral-Derivativo (PID), que foi a técnica usada para realizar o controlo do sistema.

3.1 Plataformas Omnidirecionais

O termo omnidirecional é usado para descrever a capacidade de um sistema para se mover instantaneamente e em qualquer direção a partir de qualquer configuração. Sendo assim, uma plataforma omnidirecional pode controlar de forma independente todos os três graus de liberdade no espaço bidimensional (não possui restrições não-holonómicas). Quando comparadas com as plataformas convencionais que possuem acionamento diferencial ou geometria de Ackermann (Figura 3.1), estas têm uma maior manobrabilidade. Isto deve-se ao facto das rodas convencionais não permitirem movimento numa direção paralela aos seus eixos. Por isso, embora consigam atingir todas as localizações e orientações em duas dimensões, os robôs que têm restrições holonómicas necessitarão de realizar manobras mais complexas, como se pretende exemplificar na Figura 3.1 [41].

Assim sendo, a mobilidade omnidirecional torna-se a escolha mais vantajosa para soluções em que os robôs têm de se mover em espaços estreitos, evitar obstáculos e seguir trajetórias complexas.

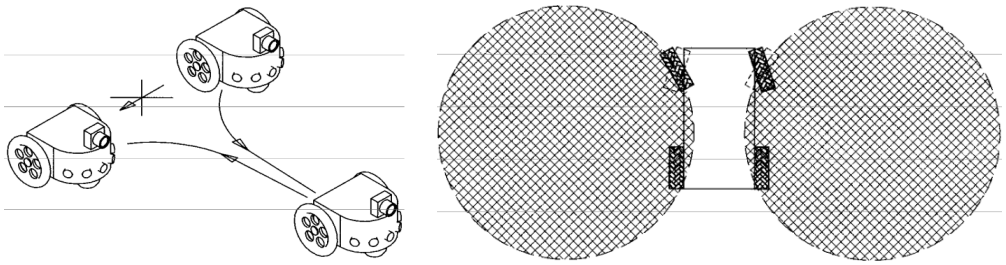


Figura 3.1: Robô com acionamento diferencial a efetuar um estacionamento paralelo e exemplificação das áreas inacessíveis de uma plataforma com geometria de Ackermann, respetivamente (adaptado de [41])

3.1.1 Tipos de Rodas Omnidirecionais

As rodas omnidirecionais são capazes de realizar tração numa determinada direção mas, ao mesmo tempo, devem permitir o deslizamento paralelo ao seu próprio eixo [41]. Tal característica permite às plataformas que as possuem efetuar movimentos laterais, girar sobre si ou até mesmo combinar movimentos de translação e rotação.

Existem duas categorias de rodas omnidirecionais: as de desenho convencional e as de desenho especial. As primeiras podem ainda ser divididas em dois tipos: os rodízios e as rodas direcionais, tal como se mostra na Figura 3.2. Comparativamente às rodas de desenho especial estas apresentam como vantagens uma maior capacidade de carga e tolerância a irregularidades do solo. A grande desvantagem prende-se com o facto de não serem verdadeiramente omnidirecionais. Por este motivo, quando é encontrada uma curva não contínua, haverá uma certa fração de tempo em que a roda terá de parar para ser reorientada. Porém, e devido a poderem rodar sobre si mesmo, têm características omnidirecionais. Convém ainda referir que a grande parte dos robôs omnidirecionais que têm este tipo de rodas têm, pelo menos, duas com um atuador de direção [41].

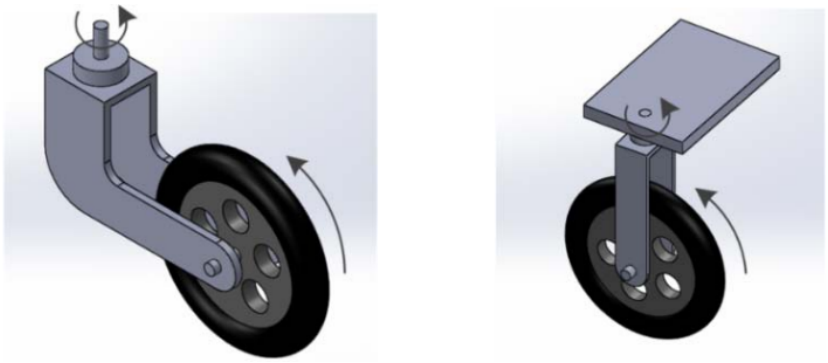


Figura 3.2: Exemplo de um rodízio e de uma roda direcional, respetivamente [42]

No que diz respeito às rodas de desenho especial o problema descrito anteriormente já não se coloca, uma vez que permitem movimento passivo paralelo ao seu eixo (não tem de existir reorientação da roda). Estas podem ser divididas em rodas universais ou *mecanum* (também conhecidas por rodas suecas) e são facilmente distinguidas pela presença dos rolos, como se pode observar na Figura 3.3. As principais vantagens destas rodas passam por não ser necessário um motor para reorientá-las, o que torna o sistema mecânico mais simples. Por outro lado, são bastante sensíveis a irregularidades do piso [41].

Figura 3.3: Exemplo de roda universal e *mecanum*, respetivamente (adaptado de [42])

Através da análise da Figura 3.3 pode-se constatar que as rodas universais são constituídas por rolos montados perpendicularmente ao eixo de rotação da roda. Por sua vez, as rodas *mecanum* são semelhantes, contudo, os rolos são colocados com um ângulo em relação ao eixo da roda. Este último é geralmente de 45°, mas poderá apresentar outros valores.

Na Tabela 3.1 é possível analisar com maior pormenor as vantagens e desvantagens de cada tipo de roda apresentado.

Tabela 3.1: Vantagens e desvantagens das rodas omnidirecionais

Tipos de Rodas	Vantagens	Desvantagens
Roda Direcional	• Contacto constante com o piso • Alta capacidade de carga • Lida bem com as condições do piso	• Volumosa • Pesada • Alta força de atrito durante a viragem • Mais cara e mecanicamente mais complexa devido ao motor de reorientação • Programação e controlo complexos
Rodízio	• Contacto constante com o piso • Alta capacidade de carga • Baixo atrito na viragem • Lida bem com as condições do piso	• Volumosa • Mais cara e mecanicamente mais complexa devido ao motor de reorientação • Transmissão de sinal e alimentação através de juntas rotativas • Programação e controlo complexos
Roda Universal	• Mais leve e compacta • Não necessita de motor de reorientação • Maior disponibilidade comercial • Controlo simples	• Contacto descontínuo com o piso • Variação do raio da roda (dependendo do modelo) • Sensibilidade a irregularidades do piso • Baixa capacidade de carga • Suscetível a derrapagem • Dependendo do modelo poderá ser mais complexa na odometria
Roda *Mecanum*	• Mais leve e compacta • Fácil fabrico do chassis do robô, já que podem ser colocadas em linha • Não necessita de motor de reorientação • Controlo simples • Boa capacidade de carga	• Contacto descontínuo com o piso • Sensibilidade a irregularidades do piso • Conceção mais complexa • Suscetível a derrapagem • Requer uma boa orientação do eixo

3.1.2 Configurações de Plataformas com Rodas de Desenho Especial

Existem vários tipos de plataformas omnidirecionais que podem ser obtidas com a combinação estratégica das rodas que foram estudadas. Seguidamente apontam-

se algumas das principais configurações que são implementadas com rodas de desenho especial.

Uma das configurações necessita de três rodas universais que são montadas numa estrutura triangular. Na Figura 3.4 é possível ver um exemplo de um robô com estas características e como é possível, a partir da atuação dos diferentes motores, controlar o seu movimento. Estas plataformas são mecanicamente mais simples e permitem um melhor contacto das rodas com o solo quando este possui irregularidades. Todavia, podem sofrer de problemas de estabilidade, principalmente quando o robô tiver de se mover em rampas e possuir um centro de gravidade alto [42].

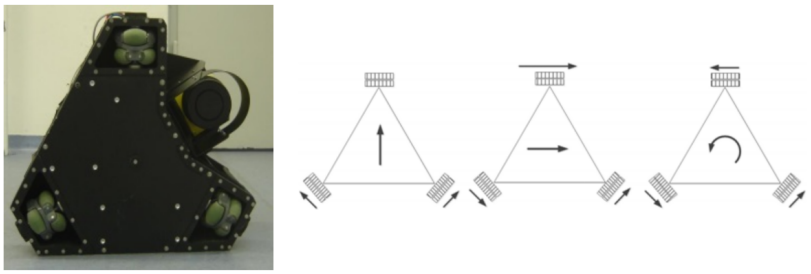


Figura 3.4: Exemplo de um robô omnidirecional com três rodas universais usado nas competições RoboCup e alguns exemplos dos seus movimentos consoante a atuação de cada motor [42]

Com a conjugação de quatro rodas universais também é possível criar plataformas omnidirecionais, tal como se mostra na Figura 3.5. Estas, quando comparadas com as de três rodas, são mais estáveis, podem atingir maior velocidade, têm maior capacidade de carga e melhor tração. Sendo que, para que a última vantagem possa ser notada, pode ser preciso montar um sistema de suspensão. Contudo, por serem constituídas por mais *hardware*, são também mais complexas mecanicamente [42, 43].

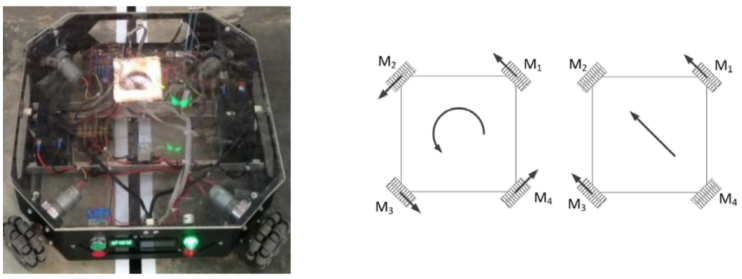


Figura 3.5: Exemplo de um robô omnidirecional com quatro rodas universais e alguns exemplos dos seus movimentos consoante a atuação de cada motor [42]

Outra das configurações muito utilizadas é a de quatro rodas *mecanum*. O

chassis destes robôs é normalmente retangular (ou quadrado de modo a simplificar o modelo matemático) e as rodas são colocadas em linha e duas a duas, tal como num carro. Deverão existir duas rodas em que os rolos têm um ângulo de 45° (*left-handed*) e outras duas em que o ângulo é de −45° (*right-handed*). Na Figura 3.6 pode ser observado um exemplo de uma plataforma destas e ainda como são conseguidos os movimentos consoante a atuação de todos os motores. Os robôs com rodas *mecanum* são utilizados em tarefas que exigem boa precisão e manobrabilidade e têm usualmente uma boa capacidade de carga. Porém, são susceptíveis a derrapagens e a falta de tração em pisos irregulares, pelo que podem necessitar de um sistema de suspensão [42, 44]. Neste trabalho serão consideradas duas plataformas com estas características.

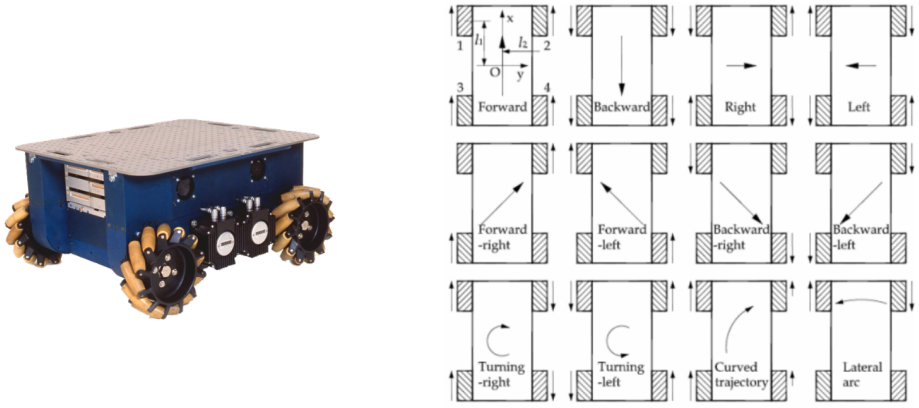


Figura 3.6: Exemplo de uma plataforma omnidirecional com quatro rodas *mecanum* e os seus movimentos consoante a atuação de cada motor [42]

3.1.3 Cinemática das Plataformas com Rodas Mecanum

O modelo cinemático de uma plataforma constituída por rodas *mecanum* pode ser obtido combinando as restrições cinemáticas de cada uma das rodas, tal como no caso dos robôs que têm rodas convencionais.

Em seguida é estudada a cinemática de uma roda *mecanum* para que subsequentemente se consiga alcançar o modelo das plataformas que são usadas neste trabalho. De modo a reduzir a complexidade da análise matemática envolvida na modelação assume-se que:

- o robô e as suas rodas são rígidos e não sofrem deformações;
- o movimento do robô está restringido ao plano bidimensional, não sendo consideradas as irregularidades do solo;
- não há a ocorrência de derrapagem nos rolos das rodas;

- o ponto de contacto entre o rolo e o chão está diretamente abaixo do centro da roda.

3.1.3.1 Modelo Cinemático de uma Roda

Na Figura 3.7 pode-se observar um diagrama com as restrições cinemáticas de uma roda *mecanum*. Os parâmetros representados têm o seguinte significado:

- r - raio da roda;
- O_i - centro da roda i;
- Y, X - sistema de coordenadas do sistema ao qual a roda pertence;
- Y_i, Xi - sistema de coordenadas da roda i;
- $\dot{\theta}$ - velocidade angular do sistema;
- α_i, l_i - ângulo e distância entre a origem do referencial da roda e do sistema à qual pertence;
- β_i - ângulo entre l_i e o eixo da roda i;
- γ_i - ângulo dos rolos da roda i;
- h_i - eixo do rolo em contacto com o solo;
- v_i - velocidade linear da roda i;
- v_{gi} - velocidade do rolo em contacto com o solo.

O diagrama permite concluir acerca da restrição do movimento da roda *mecanum*. Como se pode notar na equação 3.1 a restrição do movimento ao longo do plano da roda, quando comparada com a de uma roda convencional fixa, tem em consideração mais um parâmetro - γ_i - que diz respeito ao ângulo dos rolos. Por outro lado, não há uma restrição do movimento ortogonal da roda, devido à rotação dos rolos que a constituem [45]. Porém, se γ_i=90° este benefício perde-se, e, tal como nas rodas convencionais, volta a existir uma restrição do movimento ortogonal.

$$\dot{x}\cos(\alpha_i+\beta_i+\gamma_i)+\dot{y}\sin(\alpha_i+\beta_i+\gamma_i)+l_i\dot{\theta}\sin(\beta_i+\gamma_i)=-r\dot{\varphi}_i\sin(\gamma_i) \quad (3.1)$$

A equação anterior pode ainda ser apresentada na forma matricial, tal como se pretende exemplificar de seguida (equação 3.2):

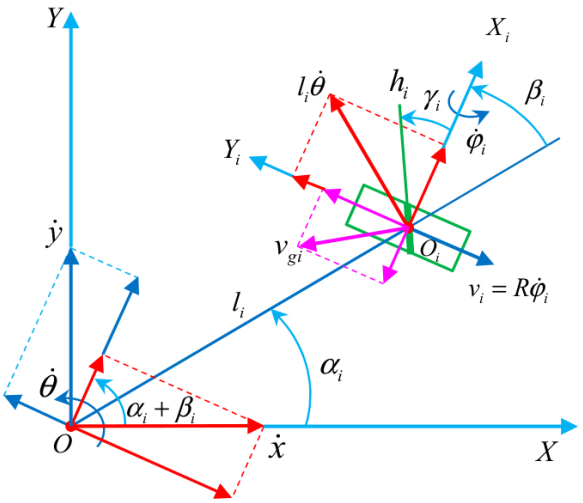


Figura 3.7: Diagrama das restrições cinemáticas de uma roda *mecanum* [45]

$$\begin{bmatrix}\cos(\alpha_i+\beta_i+\gamma_i) & \sin(\alpha_i+\beta_i+\gamma_i) & l_i\sin(\beta_i+\gamma_i)\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}+r\dot{\varphi}_i\sin(\gamma_i)=0 \tag{3.2}$$

3.1.3.2 Modelo Cinemático de um Robô com Quatro Rodas

Após a explicação relativa a uma roda, torna-se mais simples entender a cinemática de uma plataforma omnidirecional constituída por quatro rodas *mecanum*, como a que é ilustrada na Figura 3.8. Importa referir que os eixos representados a vermelho em cada roda dizem respeito ao rolo que está em contacto com o solo.

Analisando o esquema, e tendo em conta o diagrama apresentado na Figura 3.7, conclui-se que, para todas as rodas, existe a seguinte relação entre α_i e β_i:

$$\alpha_i+\beta_i=0 \tag{3.3}$$

Deste modo, a equação 3.2 relativa a cada uma das quatro rodas pode ser simplificada da seguinte maneira:

$$\begin{bmatrix}\cos(\gamma_i) & \sin(\gamma_i) & l_i\sin(\beta_i+\gamma_i)\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}+r\dot{\varphi}_i\sin(\gamma_i)=0 \tag{3.4}$$

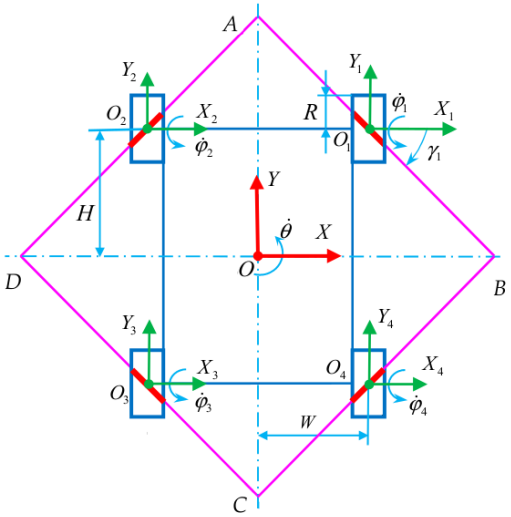


Figura 3.8: Configuração de uma plataforma com quatro rodas *mecanum* [45]

Depois disto, é possível obter a equação da cinemática inversa de cada uma das rodas:

$$\dot{\varphi_1} = -\frac{1}{r}\begin{bmatrix}\frac{1}{\tan\gamma_1} & 1 & W - \frac{H}{\tan\gamma_1}\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} \tag{3.5}$$

$$\dot{\varphi_2} = -\frac{1}{r}\begin{bmatrix}\frac{1}{\tan\gamma_2} & 1 & -W - \frac{H}{\tan\gamma_2}\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} \tag{3.6}$$

$$\dot{\varphi_3} = -\frac{1}{r}\begin{bmatrix}\frac{1}{\tan\gamma_3} & 1 & -W + \frac{H}{\tan\gamma_3}\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} \tag{3.7}$$

$$\dot{\varphi_4} = -\frac{1}{r}\begin{bmatrix}\frac{1}{\tan\gamma_4} & 1 & W + \frac{H}{\tan\gamma_4}\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} \tag{3.8}$$

Como já foi mencionado anteriormente, o modelo cinemático da plataforma será facilmente obtido combinando as equações de cada uma das rodas:

$$\begin{bmatrix}\dot{\varphi_1}\\ \dot{\varphi_2}\\ \dot{\varphi_3}\\ \dot{\varphi_4}\end{bmatrix} = -\frac{1}{r}\begin{bmatrix}\frac{1}{\tan\gamma_1} & 1 & W - \frac{H}{\tan\gamma_1}\\ \frac{1}{\tan\gamma_2} & 1 & -W - \frac{H}{\tan\gamma_2}\\ \frac{1}{\tan\gamma_3} & 1 & -W + \frac{H}{\tan\gamma_3}\\ \frac{1}{\tan\gamma_4} & 1 & W + \frac{H}{\tan\gamma_4}\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} \tag{3.9}$$

A matriz jacobiana poderá ser ainda mais simplificada já que $\gamma_1 = -\gamma_2 = -\gamma_3 = \gamma_4 = -\gamma = -45°$:

$$\mathbf{J} = -\frac{1}{r}\begin{bmatrix} \frac{1}{\tan\gamma_1} & 1 & W - \frac{H}{\tan\gamma_1} \\ \frac{1}{\tan\gamma_2} & 1 & -W - \frac{H}{\tan\gamma_2} \\ \frac{1}{\tan\gamma_3} & 1 & -W + \frac{H}{\tan\gamma_3} \\ \frac{1}{\tan\gamma_4} & 1 & W + \frac{H}{\tan\gamma_4} \end{bmatrix} = -\frac{1}{r}\begin{bmatrix} -\cot\gamma & 1 & W + Hcot\gamma \\ \cot\gamma & 1 & -W - Hcot\gamma \\ -\cot\gamma & 1 & -W - Hcot\gamma \\ \cot\gamma & 1 & W + Hcot\gamma \end{bmatrix} =$$

$$= -\frac{1}{r}\begin{bmatrix} -1 & 1 & W + H \\ 1 & 1 & -(W + H) \\ -1 & 1 & -(W + H) \\ 1 & 1 & W + H \end{bmatrix} \tag{3.10}$$

Adicionalmente, caso seja necessário realizar os cálculos atendendo a um sistema de coordenadas global (Figura 3.9), poderá ser utilizada a matriz de rotação $\mathbf{R}(\theta)$, tal como se explica de seguida:

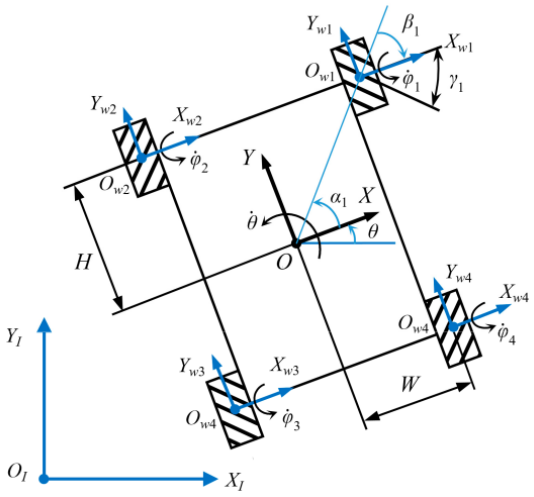


Figura 3.9: Sistema de coordenadas local e global de uma plataforma com quatro rodas *mecanum* [46]

$$\begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \dot{\varphi}_3 \\ \dot{\varphi}_4 \end{bmatrix} = \mathbf{JR}(\theta)\begin{bmatrix} \dot{x}_I \\ \dot{y}_I \\ \dot{\theta} \end{bmatrix} = -\frac{1}{r}\begin{bmatrix} -1 & 1 & W + H \\ 1 & 1 & -(W + H) \\ -1 & 1 & -(W + H) \\ 1 & 1 & W + H \end{bmatrix}\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \dot{x}_I \\ \dot{y}_I \\ \dot{\theta} \end{bmatrix} \tag{3.11}$$

3.2 Sensores de Força

Em muitas operações e processos industriais, a monitorização de forças é essencial e extremamente importante para garantir o correto funcionamento das máquinas, a qualidade do produto e a segurança dos operadores. Assim, torna-se imprescindível a instalação de um sensor de força para que esta variável possa ser quantificada. Na Figura 3.10 podem ser vistos exemplos do uso destas tecnologias.

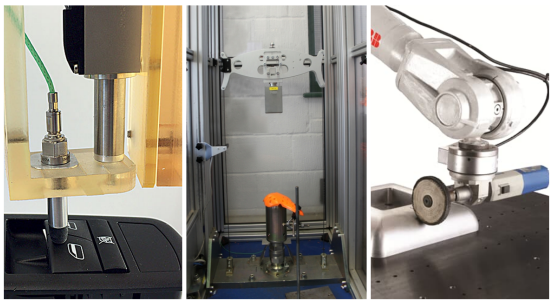


Figura 3.10: Exemplos da utilização de sensores de força em tarefas de teste de produtos e na monitorização das forças exercidas pela ferramenta de um robô industrial [47, 48]

3.2.1 Tipos de Sensores de Força

Existem diferentes tipos de sensores de força que são usados em aplicações industriais. Estes podem ser hidráulicos, pneumáticos, capacitivos, piezoelétricos, piezoresistivos, óticos, etc. Existe uma grande diversidade de tecnologias, com diferentes princípios de funcionamento e vantagens que lhes são inerentes, que permitem realizar a medição de forças. Atualmente, duas das mais utilizadas baseiam-se no uso de extensómetros ou de materiais piezoelétricos [49, 50, 51].

Os sensores que possuem extensómetros são constituídos por um corpo metálico (normalmente de aço ou alumínio) onde estes são fixados. Conforme o seu corpo é sujeito a uma força sofre uma deformação, que, por consequência, também provocará a deformação dos extensómetros resultando numa alteração da resistência destes últimos. Este fenómeno encontra-se representado na Figura 3.11. Por serem uma tecnologia mais madura, simples e barata, os sensores com extensómetros são utilizados na grande maioria das operações em que há a necessidade de monitorizar forças. Os avanços feitos na área permitem que sejam realizados sensores muito precisos e com um bom desempenho, tanto na medição de forças estáticas como também das dinâmicas.

Em relação aos sensores de força piezoelétricos, estes são constituídos por um cristal que, à medida que as forças são aplicadas ao corpo do sensor e este se de-

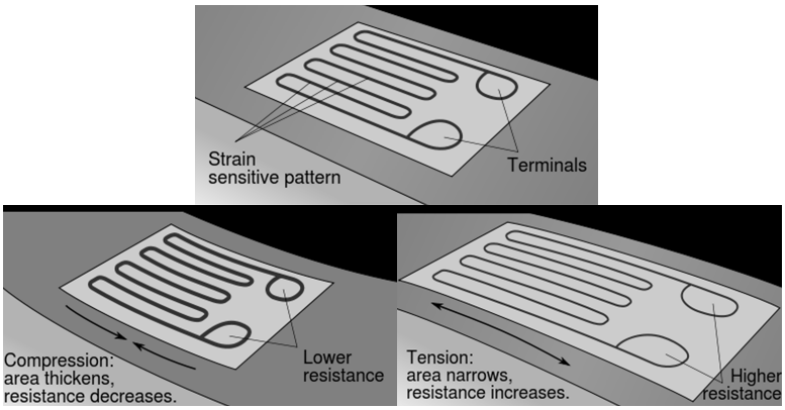


Figura 3.11: Princípio de funcionamento de um sensor de força que possui extensómetros (adaptado de [52])

forma, irá produzir uma variação de tensão, tal como se pretende exemplificar na Figura 3.12. O facto da deformação do cristal ser extremamente pequena permite que seja criada uma estrutura bastante rígida e com uma maior capacidade para capturar eventos de medição muito rápidos ou de alta frequência. Porém, devido ao seu princípio de funcionamento, estes sensores apresentam *drift* quando uma carga estática lhes é aplicada. Este fenómeno pode causar erros consideráveis, especialmente aquando da medição de pequenas forças, já que o *drift* é uma característica intrínseca do cristal piezoelétrico e não varia com a força que se pretende medir. Isto é algo que não acontece nos sensores que possuem extensómetros, o que faz com que tenham maior linearidade e estabilidade em medições que se prolongam no tempo [49, 51, 53]. No entanto, os sensores piezoelétricos também são menos susceptíveis a serem danificados por sobrecarga (devido à sua rigidez), têm dimensões mais reduzidas e uma faixa de medição maior e também detetam melhor pequenas variações quando a força a medir é bastante elevada [53].

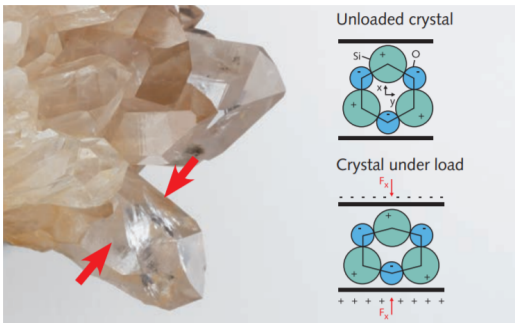


Figura 3.12: Princípio de funcionamento de um sensor de força piezoelétrico [53]

Um tipo de sensores que, embora não sejam recentes, têm vindo a ser bastante estudados são os sensores de força capacitivos. O interesse nesta tecnologia

deve-se às vantagens que tem, quando usada em sensores de força e binário multiaxiais, comparativamente aos extensómetros [54]. Alguns fabricantes, além de já possuírem exemplares, afirmam que os sensores capacitivos poderão vir a ser mais utilizados uma vez que não são tão afetados pelo ruído, necessitam de menos condicionamento de sinal e também são bastante precisos [55]. Inclusive, ao contrário do que acontece nos sensores com extensómetros, nenhum elemento tem de ser fixado com adesivos no corpo metálico, o que pode ser uma mais valia para a longevidade do equipamento e também na automação e custo da sua produção [54]. O princípio de funcionamento destes sensores tem por base o facto da capacitância de dois elementos condutores depender da distância entre eles. Quando estes se aproximam a capacitância aumenta, acontecendo o contrário quando se afastam [55]. Na Figura 3.13 é possível verificar a estrutura de um destes sensores em corte e como esta permite medir forças de compressão e cisalhamento através do conceito explicado anteriormente.

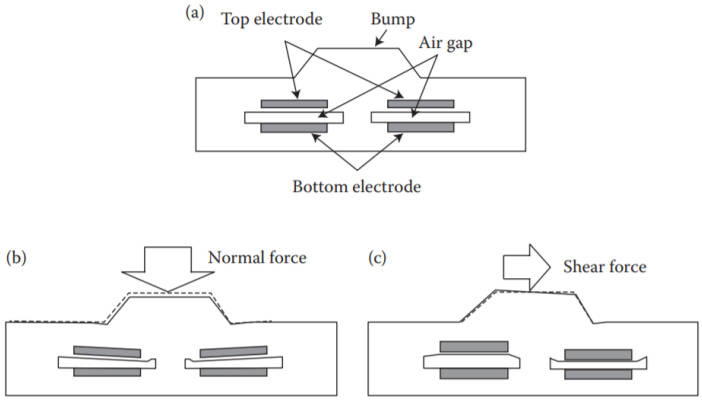


Figura 3.13: Estrutura de um exemplo de um sensor capacitivo em corte [56]

Existem muitos outros tipos de sensores, uns mais baratos, como é o caso dos *Force Sensitive Resistors* (FSR - Figura 3.14) que são usados em aplicações que não exigem uma precisão elevada, outros mais direcionados a um pequeno nicho de aplicações, tal como é o caso dos sensores óticos Optoforce (Figura 3.15), que podem, por exemplo, ser colocados nas pontas dos dedos de mãos robóticas. Não sendo o principal objetivo do trabalho, não interessa referi-los a todos, mas sim dar a conhecer algumas opções que já existem no mercado. Por outro lado, no âmbito deste projeto, interessa abordar três conceitos associados a este tópico de estudo: as células de carga, os sensores de binário e os sensores de força/binário multiaxiais. Uma vez que são os mais acessíveis e utilizados no mercado, e que poderiam perfeitamente ser usados no sistema em estudo nesta dissertação, dar-se-á mais ênfase aos sensores baseados em extensómetros; no entanto, também existem equipamentos destes que funcionam segundo os outros princípios.

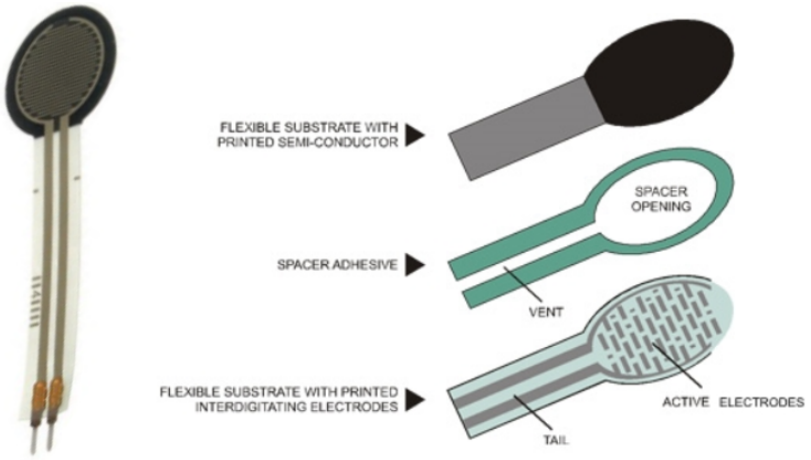


Figura 3.14: Exemplo de um FSR e respetiva constituição [57]

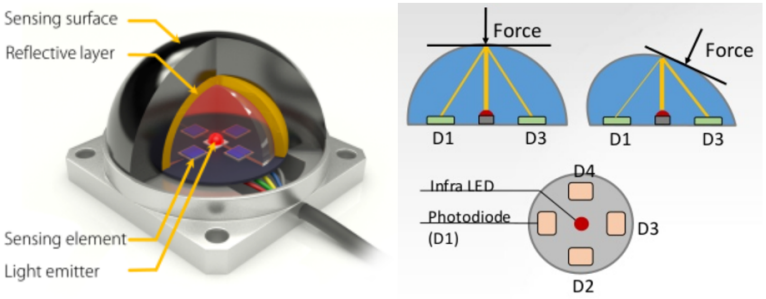


Figura 3.15: Sensor de força ótico OptoForce [58]

3.2.2 Células de Carga

As células de carga são sensores que convertem a força que lhes é aplicada num sinal elétrico. Estas são bastante usadas na indústria, principalmente em tarefas de pesagem. Todavia, existem outras áreas em que são cruciais. As células de carga também são utilizadas, por exemplo, para monitorizar as forças exercidas sobre uma determinada estrutura (hastes, vigas, rodas, etc) ou na medição indireta do nível de um tanque.

A robótica também beneficia bastante do uso destes sensores. A garra de um robô poderá ser equipada com um, para que seja possível obter *feedback* acerca da força de compressão que exerce sobre um determinado objeto. Outro exemplo são os robôs com pernas, nos quais as células de carga podem ser usadas para fornecer informação essencial ao sistema de controlo responsável pela locomoção e equilíbrio do robô.

Na Figura 3.16 está representado um destes sensores numa configuração bastante simples (*single-point*). Através da análise da figura é possível entender como

a força aplicada à célula é convertida numa variação de tensão. Os extensómetros são colocados numa ponte de *Wheatstone*. Quando não há força a ser aplicada, a ponte encontra-se equilibrada e a tensão de saída é zero (ou aproximadamente zero), isto porque todos os extensómetros apresentarão um valor de resistência bastante aproximado. Quando a força é aplicada, o corpo metálico irá deformar-se, bem como os extensómetros. Estes, por consequência, apresentarão valores de resistência diferentes, o que fará com que a ponte se desequilibre e a tensão na sua saída varie [59, 60].

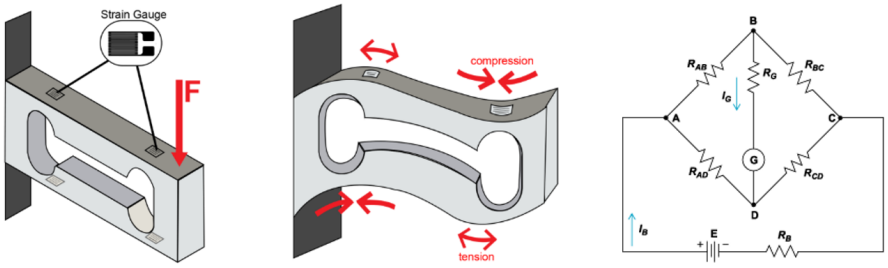


Figura 3.16: Célula de carga do tipo *single-point* e respetivo funcionamento (adaptado de [59])

Existem várias configurações de células de carga com diferentes corpos metálicos e número de extensómetros que possibilitam a monitorização de forças de tensão, compressão, cisalhamento e flexão. Na Figura 3.17 podem ser observados exemplos. Algumas destas células, ao contrário do exemplo mostrado anteriormente, poderão ter uma estrutura mais complexa do tipo *multi-point*, que resulta da junção de várias células com uma estrutura *single-point* numa só estrutura, como se pode ver na Figura 3.18.

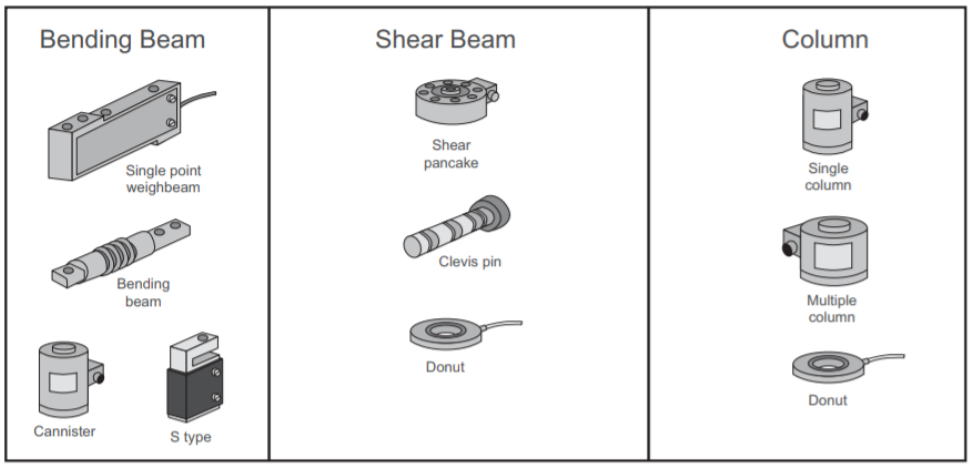


Figura 3.17: Exemplos de configurações de células de carga [60]

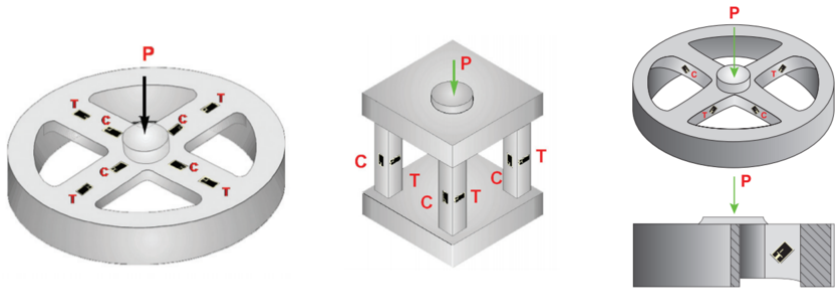


Figura 3.18: Configurações *multi-point* mais comuns (adaptado de [60])

3.2.3 Sensores de Binário

Em certos processos industriais é necessário medir forças rotacionais. Para isso, podem ser implementados sensores de binário, como se exemplifica na Figura 3.19.

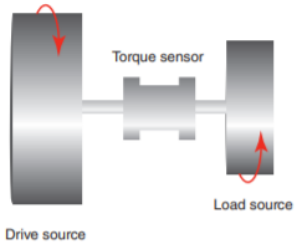


Figura 3.19: Exemplificação da utilização de um sensor de binário [60]

Existem dois tipos destes sensores, os de reação e os rotativos. A sua diferença passa por permitirem, ou não, a rotação do eixo onde se pretende medir a força. Tal como nas células de carga, a tecnologia mais usada nestes sensores são os extensómetros. O seu funcionamento também tem em conta a utilização de uma ponte de *Wheatstone* e é idêntico ao caso apresentado anteriormente, sendo que aquilo que difere é o posicionamento dos extensómetros e o corpo metálico. Na Figura 3.20 apresentam-se algumas das configurações de sensores de binário mais usadas [60].

No caso dos sensores rotativos, estes são ainda constituídos por um elemento fixo (estator) que envolve o eixo rotativo (rotor) e permite que este rode 360°. Por esta razão, tem de haver um meio que permita que o sinal de saída da ponte possa ser transferido do eixo para o estator. Normalmente, isto é conseguido através da utilização de, por exemplo, anéis coletores ou transformadores rotativos [60].

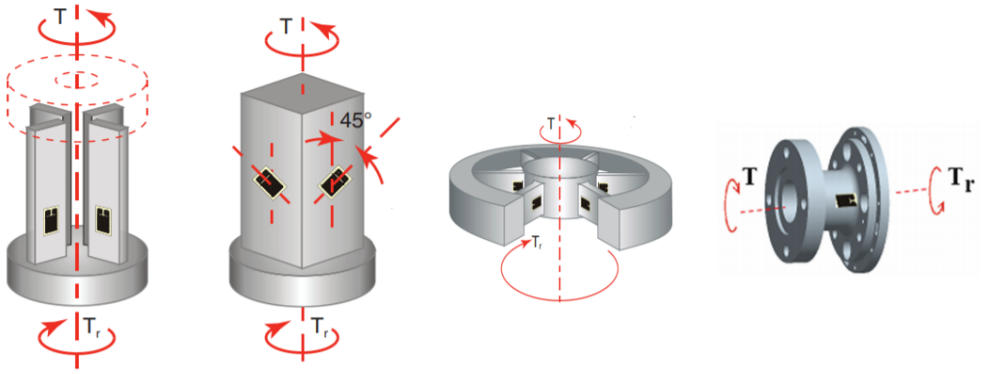


Figura 3.20: Exemplos de configurações de sensores de binário [60]

3.2.4 Sensores de Força/Binário Multiaxiais

Em alguns casos, a monitorização das forças que atuam sobre um eixo não é suficiente. Certas operações requerem que sejam efetuadas medições em vários eixos simultâneamente, por vezes, não só das forças lineares mas também das rotacionais. Isto poderá ser conseguido, dependendo da situação, através de vários sensores uniaxiais. Porém, nestas circunstâncias, o mais normal é que sejam usados sensores de força/binário multiaxiais (Figura 3.21), uma vez que são soluções mais compactas, simples de instalar e muito precisas.

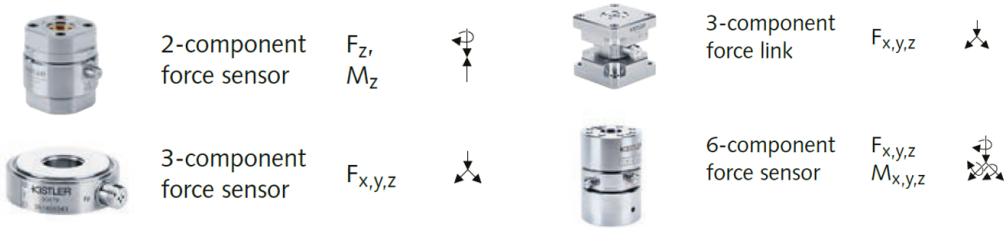


Figura 3.21: Sensores de força/binário multiaxiais piezoelétricos da Kistler [53]

Na Figura 3.22 encontra-se representado um sensor de força/binário de seis eixos instalado num robô colaborativo. Estes sensores são bastante usados em robôs industriais pelo facto de possibilitarem a medição de todas as forças e binários no espaço tridimensional, sendo essenciais em tarefas em que tem de haver um controlo minucioso das forças (rebarbagem, lixagem, polimento, montagem precisa de componentes, etc). Tal como é possível verificar na Figura 3.22, o sensor é montado entre o órgão terminal e o punho do robô, o que torna possível realizar um controlo das juntas tendo em conta as forças que estão a ser exercidas. Apesar de ser o mais comum, estes sensores não são só usados em robôs industriais. Existem outras tarefas nas quais são essenciais.

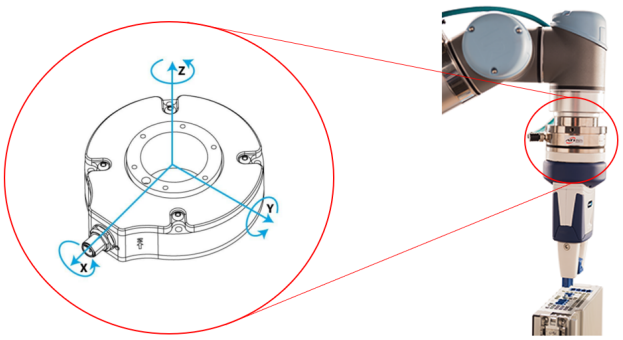


Figura 3.22: Exemplo de um sensor de força/binário de seis eixos montado num robô colaborativo para tarefas de montagem (adaptado de [61])

O funcionamento dos sensores de força/binário de seis eixos, quando constituídos por extensómetros, vai ao encontro daquilo que foi explicado para as células de carga e para os sensores de binário. A grande diferença passa pelo corpo metálico e pelo posicionamento e número de extensómetros, que têm de ser projetados de modo a permitir a medição das forças nos três eixos cartesianos (X, Y e Z) e nos três eixos rotacionais (*pitch, roll e yaw*). Na Figura 3.23 é possível observar uma configuração típica deste tipo de sensor com três vigas metálicas, cada uma com quatro extensómetros. Adicionalmente, nos sensores modernos também existe um circuito integrado responsável pela aquisição, filtragem e processamento de sinal. Isto faz com que exista à saída do sensor um sinal digital já tratado, relativo às forças nos seis eixos. Deste modo, os sensores de força/binário de seis eixos combinam as capacidades de várias células de carga e sensores de binário reativos, mas de uma forma extremamente compacta.

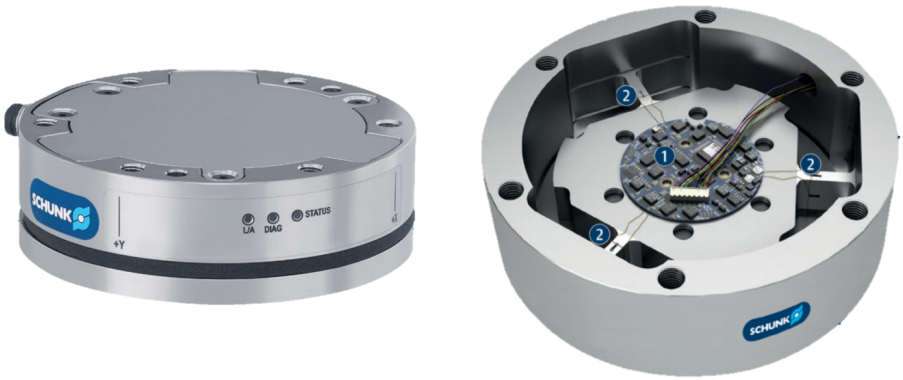


Figura 3.23: Sensor de força/binário de seis eixos FT-AXIA da Schunk (1 - Eletrónica, 2 - Vigas metálicas com extensómetros) [62]

3.3 Controlo PID

Neste trabalho será utilizado o controlo PID, ou um dos seus subconjuntos (p. ex.: P, PI, PD, etc.). A razão para se usar este tipo de controlador passa por ser simples, fácil de implementar e por proporcionar bons resultados, o que faz com que muitas vezes seja desnecessário recorrer a técnicas mais sofisticadas e matematicamente mais exigentes. Na Figura 3.24 é mostrado um diagrama de blocos que representa um controlador PID paralelo clássico em malha fechada. É possível concluir que o sinal de controlo $u(t)$ é a soma da ação dos três termos representados - proporcional, integral e derivativo - sobre o erro (equação 3.12).

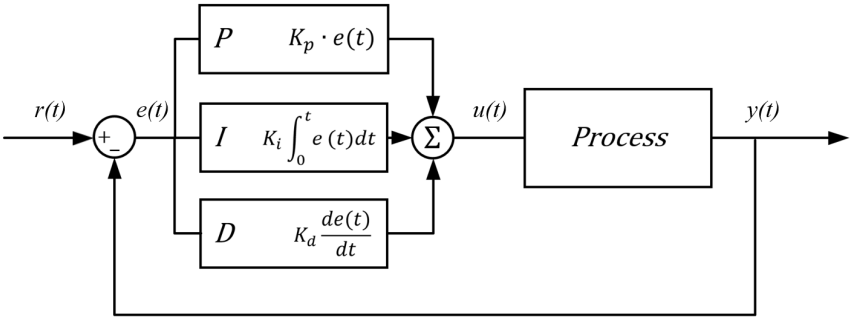


Figura 3.24: Diagrama de blocos de um controlador PID paralelo clássico

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) \cdot d\tau + K_d \cdot \dot{e}(t) =$$
$$= K_p \left[e(t) + \frac{1}{T_i} \cdot \int_0^t e(\tau) \cdot d\tau + T_d \cdot \dot{e}(t)\right] \tag{3.12}$$

A variação dos três ganhos do controlador - K_p, K_i e K_d - irá provocar alterações na resposta do sistema, e, caso seja feita uma boa sintonia, será possível aumentar o seu desempenho. Antes de se abordar as alterações provocadas pelas três componentes do controlador, convém esclarecer as principais caraterísticas da resposta de um sistema em malha fechada (Figura 3.25), sendo elas [63]:

- tempo de subida (*rise time*): é o tempo que o sistema demora a atingir a proximidade do seu valor final pela primeira vez (tempo medido entre os 10 a 90% do valor final);
- sobreelongação máxima (*overshoot*): é o valor máximo em que a variável do processo ultrapassa o valor final (é expresso como uma percentagem do valor final);
- tempo de estabelecimento (*settling time*): tempo que o sistema demora a estabilizar no seu valor final;

- erro em regime permanente (*steady-state error*): diferença entre o valor final e a referência.

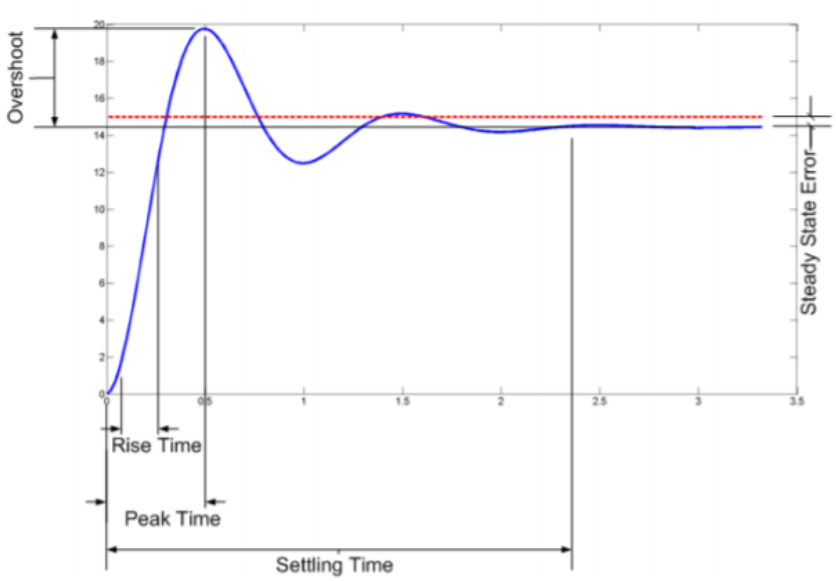


Figura 3.25: Principais características da onda de resposta de um sistema em malha fechada [63]

Tal como já foi referido anteriormente, a alteração dos ganhos irá afetar a dinâmica do sistema. Com a Tabela 3.2 procura-se apresentar as alterações que são normalmente notadas na resposta, consoante o aumento de cada um dos ganhos do controlador.

Tabela 3.2: Alterações nas características da resposta do sistema, consoante o aumento de cada ganho do controlador PID

Ganho	**Tempo de subida**	**Overshoot**	**Tempo de estabelecimento**	**Erro em regime permanente**	**Estabilidade**
$\uparrow K_p$	Diminui	Aumenta	Aumenta pouco	Diminui	Piora
$\uparrow K_i$	Diminui pouco	Aumenta	Aumenta	Diminui muito	Piora
$\uparrow K_d$	Diminui pouco	Diminui	Diminui	Pouca variação	Melhora

3.3.1 Sintonia de Controladores PID

A sintonização de um controlador PID passa por ajustar os seus parâmetros de forma a obter a melhor resposta possível. Tal poderá ser feito manualmente,

através de várias tentativas, tendo em conta as alterações que a variação de cada ganho introduz na resposta, como foi mostrado anteriormente. Porém, este processo poderá ser moroso. Por esta razão, existem métodos e heurísticas que podem ser utilizados, de forma a obter-se uma primeira aproximação aos valores do ganho dos controladores. No trabalho desenvolvido foi aplicado o método de sintonia de Ziegler-Nichols em malha fechada, sendo depois realizados ajustes manualmente, de forma a garantir o melhor funcionamento e estabilidade possível do sistema.

Para se aplicar este método implementa-se um controlador proporcional em malha fechada e aumenta-se o ganho proporcional até ser obtida uma resposta oscilatória com amplitude constante. De seguida será determinado o ganho crítico (K_c) e o período de oscilação crítico (T_c), sendo o primeiro o valor do ganho K_p que gerou a resposta oscilatória e o segundo o próprio período da onda de resposta oscilante. Por fim, com os valores de (K_c) e (T_c) obtidos, e recorrendo à Tabela 3.3, calcula-se os ganhos para o tipo de controlador desejado segundo este método de sintonia [64].

Tabela 3.3: Sintonia dos ganhos através do método de Ziegler-Nichols em malha fechada [64]

Tipo de Controlo	K_p	T_i	T_d
P	0,5 K_c	—	—
PI	0,45 K_c	$T_c/1.2$	—
PID	0,6 K_c	$T_c/2$	$T_c/8$

3.3.2 Discretização do Controlador

No presente trabalho é necessário discretizar o controlador PID para que este possa ser implementado digitalmente no simulador. Existem vários métodos para realizar a discretização, sendo um deles o método de Euler das diferenças atrasadas (equação 3.13). Seguidamente, explica-se como é possível obter a equação que representa o controlador PID digital, através deste método. Convém salientar que T é o período de amostragem e corresponderá ao incremento de tempo usado durante a simulação no CoppeliaSim.

$$\dot{x}(k) \cong \frac{x(k) - x(k-1)}{T} \tag{3.13}$$

Derivando-se ambos os membros da equação 3.12 obtém-se a seguinte equação 3.14:

$$\dot{u}(t) = K_p \cdot \dot{e}(t) + K_i \cdot e(t) + K_d \cdot \ddot{e}(t) \tag{3.14}$$

Usando-se a aproximação apresentada na equação 3.13 na equação anterior é possível chegar à equação que representa o controlador PID digital:

$$u(k) = u(k-1)+Kp\cdot(e(k)-e(k-1))+Ki\cdot T\cdot e(k)+\frac{Kd}{T}\cdot[e(k)-2e(k-1)+e(k-2)] \tag{3.15}$$

Capítulo 4

Escolha e Introdução ao Simulador

Neste capítulo são apresentadas algumas ferramentas que podem ser empregues na simulação de um MRS, apontando-se algumas das suas características mais importantes. De seguida, é explicada a razão pela qual, no âmbito do presente trabalho, se decidiu escolher o CoppeliaSim para simular o sistema em estudo. Relativamente a este simulador, são ainda introduzidos conceitos sobre o mesmo que foram essenciais ao desenvolvimento do projeto.

4.1 Simuladores de Robótica

O uso de uma ferramenta de simulação é algo que pode impactar positivamente o desenvolvimento de um projeto. Hoje em dia, existem vários simuladores que podem ser usados para recriar robôs num ambiente virtual. Isto permite ao utilizador realizar experiências naquilo que concerne, entre outros fatores, à construção do robô, aos sensores e atuadores que o constituem e também aos algoritmos que possibilitam que este execute as tarefas para o qual é pensado.

As vantagens que advêm do uso de um simulador poderão ser diversas, dependendo do caso em estudo. Contudo, existem algumas que são intrínsecas ao uso destas ferramentas, tais como:

- a possibilidade de testar diferentes cenários e hipóteses sem ser necessário possuir o robô;
- a capacidade de realizar várias iterações rapidamente (enquanto que num cenário real seria necessário preparar o sistema e o ambiente no qual este se situa);

- o aumento da liberdade e criatividade, uma vez que não existem preocupações relativas à danificação do *hardware*;
- a oportunidade de testar várias hipóteses simultaneamente (várias simulações a correr em paralelo).

Os sistemas de co-transporte, bem como todos os MRS, independentemente da área na qual são aplicados, são um tema de estudo que pode beneficiar bastante com o uso da simulação, pelas várias razões que foram mencionadas. Tendo em conta que estes implicam o uso de vários robôs, o que pode ser um impedimento para Instituições de Ensino Superior e laboratórios de investigação, os simuladores ganham ainda uma maior importância. Isto porque, mesmo que não sejam possuídos os recursos, alunos e investigadores poderão continuar a estudar estes sistemas. Também em situações como a pandemia que se vive atualmente estes serão cruciais possibilitando a continuação do estudo de uma forma segura e bastante cómoda. Por outro lado, mesmo que exista o equipamento, a simulação poderá ser útil na criação e teste dos algoritmos de controlo do sistema.

Embora seja impossível recriar virtualmente todos os fatores que podem interferir nos robôs num ambiente real, já existe *software* de simulação com funcionalidades bastante avançadas que permite tirar boas conclusões relativamente ao sistema simulado.

4.1.1 Software para a Simulação de Sistemas Multi-Robô

Neste trabalho interessa conhecer algumas das soluções que existem para efetuar a simulação de um MRS. Seguidamente, apresentam-se alguns dos simuladores *open-source* que poderão ser usados para tal efeito:

- **Webots**: trata-se de um simulador multi-plataforma que tem vindo a ser desenvolvido pela Cyberbotics Ltd., uma *spin-off* da École Polytechnique Fédérale de Lausanne (EPFL), desde 1998. Este programa é utilizado na indústria, educação e investigação. Além de ter uma biblioteca com bastantes modelos de robôs (como o youBot, representado na Figura 4.1), sensores, atuadores e objetos, o simulador permite ainda ao utilizador importar modelos *Computer Aided Design* (CAD) do Blender ou *Unified Robot Description Format* (URDF). Os modelos existentes ou importados poderão ser usados para simular uma grande gama de tipos de robôs, quer num ambiente *indoor* ou *outdoor*. Estes robôs poderão ser programados em C, C++, Python, Java, Matlab ou *Robot Operating System* (ROS). A base do Webots é construída sobre a combinação de uma *Graphical User Interface* (GUI), um motor de física, mais concretamente o *Open Dynamics Engine* (ODE), e um motor de renderização baseado no OpenGL 3.3 [65].

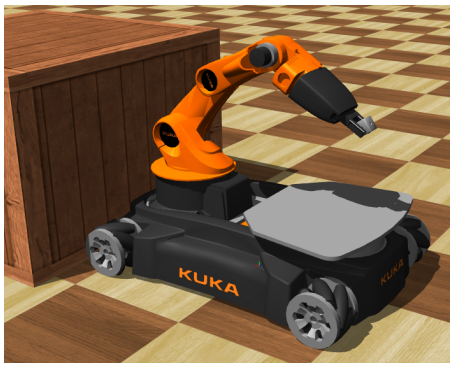


Figura 4.1: Simulação do KUKA youBot no Webots [66]

- ***Autonomous Robots go Swarming* (ARGoS)**: é um simulador desenvolvido no âmbito do projeto Swarmanoid [67]. Este programa foi pensado para as simulações na área da robótica de enxame em que é necessário simular um grande número de robôs (Figura 4.2). Algumas das suas características mais interessantes passam por:

  - possuir uma arquitetura modular que possibilita ao utilizador configurar todos os aspectos da simulação;
  - permitir o uso de vários motores de física simultâneamente numa mesma simulação (através da divisão do espaço físico virtual em várias regiões);
  - distribuir o *loop* principal da simulação por múltiplas *threads*, o que leva a um melhor aproveitamento das capacidades dos processadores *multi-core*.

  Os robôs simulados no ARGoS podem ser programados, entre outras linguagens, em C++ e Lua (*scripts*). O simulador conta com uma pequena biblioteca de modelos simples de robôs e uma das grandes desvantagens é não deixar importar outros modelos. Caso seja necessário, é necessário modelá-los com o OpenGL. A complexidade em relação a este aspeto é compensada pelo desempenho nas simulações que envolvem uma grande quantidade de robôs [68, 69].

- ***Urban Search and Rescue Simulation* (USARsim)**: este simulador multi-plataforma foi desenvolvido pela University of Pittsburg para a simulação de robôs em situações de desastre. Apesar de originalmente ter sido criado com tal intuito, também pode ser aplicado na simulação de outras operações e diversos tipos de robôs (robôs com rodas, pernas, aquáticos, aéreos, humanóides, etc.). O USARsim utiliza um motor de jogo, o Unreal Engine, para simular a física e para realizar a renderização gráfica.

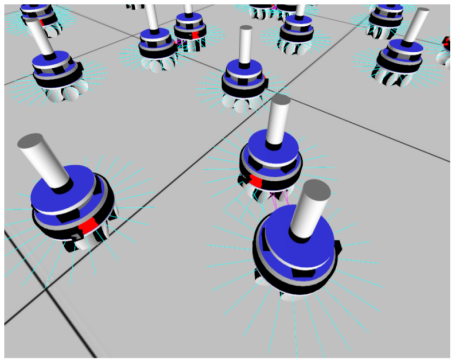


Figura 4.2: Simulação de um grupo de robôs no ARGoS [69]

Além disso, os utilizadores podem, através da aplicação UnrealEd, importar e exportar modelos de novos robôs, sensores e ambientes facilmente. Os robôs podem ser programados em diferentes linguagens, como C++ e Java. O bom desempenho deste simulador fez com que tenha sido muito usado em provas de competição virtuais da RoboCup [70, 71]. Na Figura 4.3 encontra-se ilustrado o ambiente de simulação deste programa.

Figura 4.3: Simulação de um grupo de robôs no USARsim [72]

- **Gazebo**: é uma ferramenta de simulação que começou a ser desenvolvida em 2002 na University of Southern California. O objetivo passava por construir um programa que possibilitasse a simulação de robôs em ambientes *outdoor* sob as mais diversas condições. Em 2009, o ROS foi integrado neste simulador, e, desde então, este tornou-se numa das ferramentas mais utilizadas pela comunidade ROS, que o utiliza para simular vários tipos de robôs, tanto em ambientes *outdoor* como *indoor*. O Gazebo tem uma biblioteca com modelos de robôs e sensores, sendo que os seus utilizadores podem ainda criar os seus próprios modelos recorrendo ao *Simulation Description Format* (SDF). Estes podem depois ser simulados num ambiente virtual (Figura 4.4) com gráficos renderizados pelo motor gráfico *Object-Oriented*

Graphics Rendering Engine (OGRE) e com vários motores de física, tais como: ODE, Bullet, Simbody e *Dynamic Animation and Robotics Toolkit* (DART). A programação dos modelos pode ser feita em C++ ou ROS [73].

Figura 4.4: Simulação de um drone no Gazebo [74]

- **CoppeliaSim**: foi desenvolvido pela Coppelia Robotics e é a nova versão do *Virtual-Robot Experimentation Platform* (V-REP). Este simulador multiplataforma (Figura 4.5) pode ser usado para o rápido desenvolvimento de algoritmos, simulações de ambientes fabris, prototipagem e validação, simulação de robótica para fins educacionais, entre muitas outras possibilidades. O CoppeliaSim é uma ferramenta bastante útil devido à sua versatilidade e pela liberdade que confere aos seus utilizadores, nomeadamente nos seguintes aspectos:

  – criação, edição e importação de modelos CAD: este simulador faculta as ferramentas necessárias para que se consiga criar modelos tridimensionais, ainda que simples, no próprio simulador. Adicionalmente, caso o grau de detalhe não seja o desejado, os modelos poderão ainda ser importados em vários formatos CAD, desde que estes descrevam os objetos como malhas triangulares. Por fim, as malhas também poderão ser alteradas com recurso a diversos modos de edição. Isto poderá ser útil, por exemplo, para aumentar o desempenho durante a simulação;

  – vários modos de programação: o controlo dos modelos simulados pode ser feito através de *scripts*, *plugins*, nós ROS ou BlueZero e também aplicações externas que se conectam ao simulador através de uma *Application Programming Interface* (API). Todos estes métodos podem ser usados ao mesmo tempo, o que faz com que haja um elevado grau de liberdade. Inclusivamente, várias linguagens são suportadas, tais como: C, C++, Lua, Java, Python, Matlab e Octave.

  Estas duas características são as que claramente distinguem esta ferramenta, porém, existem outras que também poderão interessar. O Coppeli-

aSim conta com quatro motores de física (Bullet, ODE, Vortex e Newton), uma biblioteca com vários robôs e sensores (visão, proximidade, acelerómetros, etc.), um *plugin* para a criação de *user interfaces* (UI) baseadas na *framework* Qt, ferramentas para a visualização e gravação de dados, entre outras funcionalidades [75].

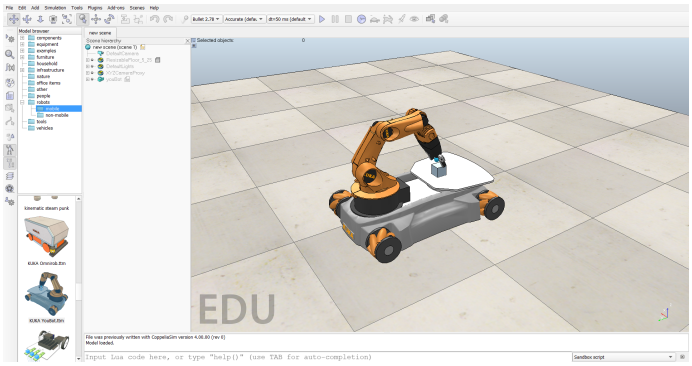


Figura 4.5: Simulação do KUKA youbot no CoppeliaSim

4.1.2 Escolha do Simulador

Para realizar a simulação do sistema em estudo, depois de feitos testes em algumas das ferramentas apresentadas, optou-se por usar o CoppeliaSim. A escolha realizada é justificada pela grande versatilidade do simulador e facilidade de adaptação ao mesmo, algo que é possível devido à simplicidade da sua interface gráfica e por toda a documentação, exemplos e tutoriais disponibilizados pela Coppelia Robotics. Naquilo que diz respeito às características intrínsecas a este projeto, há ainda outros fatores que foram levados em conta, entre outros:

- a possibilidade de criar modelos e manipular malhas no próprio simulador facilmente, o que permite colocar as ideias em prática de forma mais célere (muitas vezes sem ser necessário recorrer a um *software* CAD);
- a capacidade de se controlar os modelos através de vários *scripts* em Lua em simultâneo, levando a uma abordagem mais modular e bastante simples;
- a existência de modelos de robôs omnidirecionais na biblioteca do simulador que podem ser explorados na sua totalidade, possibilitando uma aprendizagem rápida de como é possível recriar virtualmente este tipo de plataformas;
- o facto de se poder interagir com os modelos e o ambiente tridimensional (3D), bem como monitorizar variáveis (força, posição, etc.) através da sua representação gráfica no decorrer da simulação.

4.2 Introdução ao CoppeliaSim

Tendo em vista a compreensão de todo o trabalho realizado a nível de modelação e simulação do sistema é necessário, numa primeira fase, obter-se conhecimento acerca do simulador e o modo como funciona. É preciso entender, entre outros aspectos, como se pode interagir com a aplicação e o ambiente de simulação, a estrutura hierárquica de uma cena e modelo, como são criados os modelos e como podem ser implementados os algoritmos de controlo dos robôs simulados.

4.2.1 Interface Gráfica do Utilizador

O CoppeliaSim é constituído por diversos elementos (Figura 4.6) com os quais o utilizador pode interagir de modo a efetuar a simulação desejada e para analisar os resultados, sendo os principais:

- **a janela da consola:** elemento não interativo que apenas permite mostrar informação relativa à simulação (p. ex.: *plugins* que foram carregados e o estado da sua inicialização);
- **a janela da aplicação:** janela principal onde é possível ver, editar, interagir e simular a cena e os seus respetivos modelos;
- **as caixas de diálogo:** recurso que aparece durante a interação com a janela principal e, através do qual, torna-se possível editar vários parâmetros relativos aos modelos ou à cena.

No que diz respeito à janela principal, e como se pretende mostrar na Figura 4.6, esta contém o seguinte:

- **barra da aplicação:** indica o tipo de licença, o nome do ficheiro da cena que está a ser simulada, o estado do simulador e o *frame rate*;
- **barra menu:** permite aceder às funcionalidades do simulador que, ao contrário das mais utilizadas, não podem ser acedidas através da interação com os modelos, menus *pop-up* e barras de ferramentas;
- **barras de ferramentas:** nestes elementos (Figuras 4.7 e 4.8) estão presentes, para comodidade do utilizador, as funções mais utilizadas e essenciais à interação com o simulador;

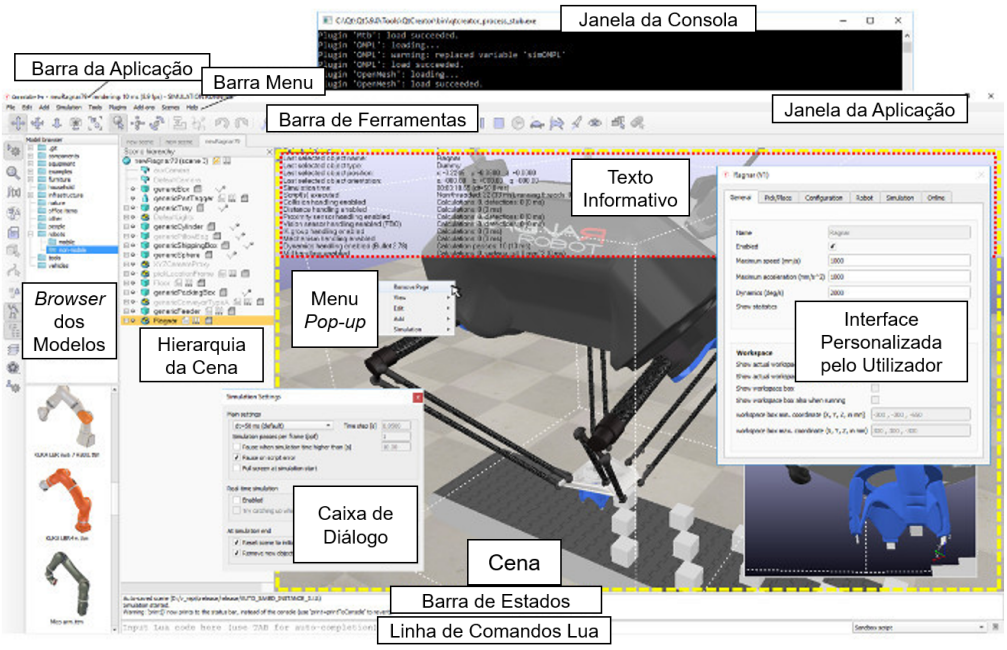


Figura 4.6: Interface gráfica do CoppeliaSim (adaptado de [76])

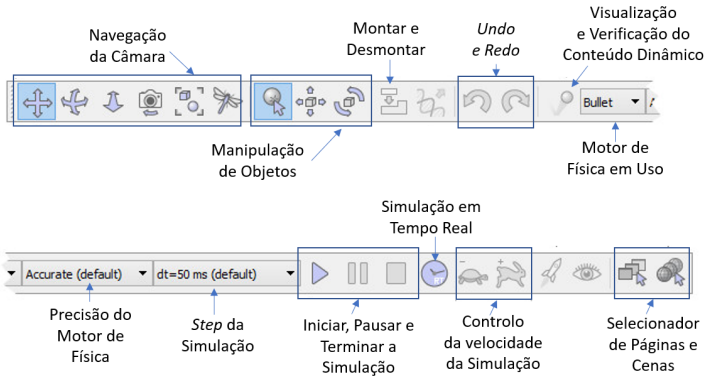


Figura 4.7: Barra de ferramentas nº1

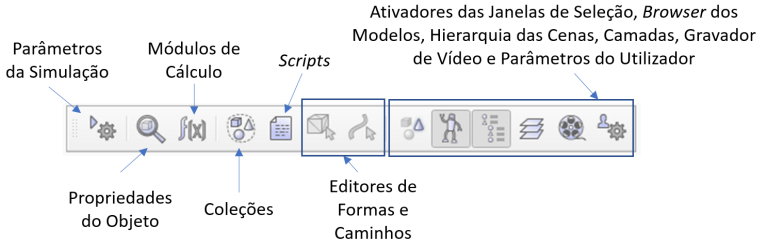


Figura 4.8: Barra de ferramentas nº2

- **browser dos modelos:** numa parte superior mostra as pastas de modelos do CoppeliaSim; por outro lado, na parte inferior, encontram-se as *thumbnails* dos modelos que podem ser incluídos na cena através da ação de *drag-and-drop* suportada pelo simulador;

- **hierarquia da cena:** aqui pode ser analisado todo o conteúdo de uma cena, ou seja, todos os objetos que a compõe. Uma vez que cada objeto é construído de forma hierárquica, é representado pela árvore da sua hierarquia, como está ilustrado na Figura 4.9. Com um duplo clique no nome de cada objeto o utilizador consegue aceder à caixa de diálogo que o permite alterar. É também com esta funcionalidade do simulador, através do *drag-and-drop*, que se criam as relações parentais entre os objetos (objetos filho são arrastados para dentro da estrutura do objeto pai);

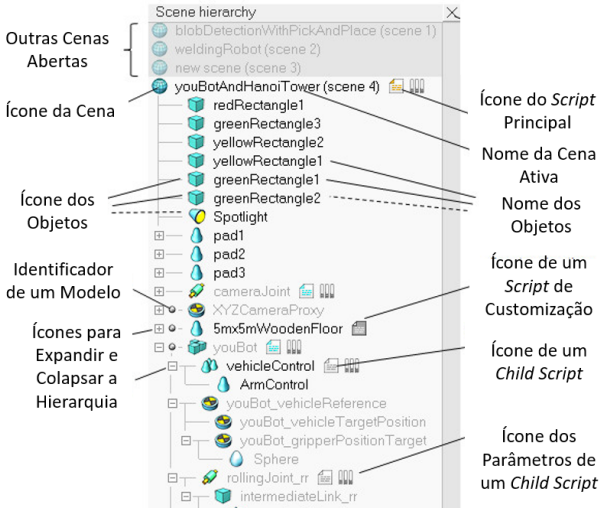


Figura 4.9: Hierarquia das cenas (adaptado de [76])

- **texto informativo:** contém informação alusiva ao objeto que se encontra selecionado num determinado momento e de parâmetros e estados da simulação;

- **barra de estados:** elemento responsável por exibir informação sobre operações, comandos e mensagens de erros. Além disto, o utilizador também poderá usá-lo para imprimir *strings* a partir de um *script*;

- **linha de comandos Lua:** é usada para introduzir e executar código em Lua.

4.2.2 Definição de Entidade, Modelo e Cena

No contexto do CoppeliaSim uma entidade refere-se a um objeto ou uma coleção de objetos virtuais 3D. Neste simulador existem diferentes objetos predefinidos, representados na Figura 4.10, que o utilizador poderá usar para construir os modelos e as cenas que pretende simular. Agrupar certos objetos em coleções também será útil caso seja necessário, em determinado aspecto, tratá-los de igual modo (p. ex.: deteção de colisões entre dois robôs constituídos por vários objetos).

Figura 4.10: Objetos existentes no CoppeliaSim (adaptado de [76])

Relativamente a um modelo, este é definido por uma seleção de objetos que pertencem à mesma estrutura hierárquica, sendo que a base da última deve ser marcada como a base do modelo (Figura 4.11). Identificar um objeto como modelo facilitará na manipulação do mesmo e dos seus constituintes, na programação (caso seja necessário duplicá-lo) e possibilitará ao utilizador guardar o seu trabalho na biblioteca de modelos do programa.

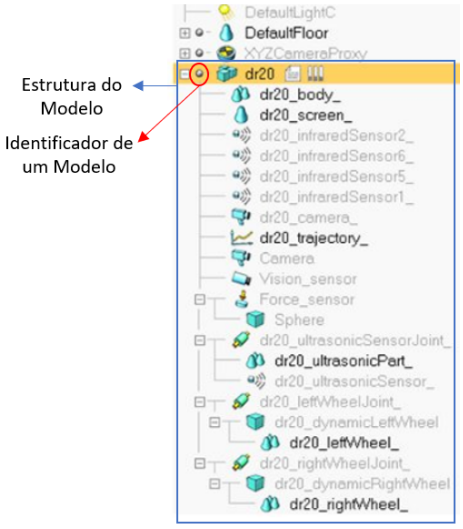


Figura 4.11: Estrutura e identificação de um modelo (adaptado de [76])

Por último, a cena, engloba todos os objetos, coleções de objetos e modelos

existentes no ambiente virtual, bem como o ambiente, que é um conjunto de parâmetros (luminosidade do ambiente, cor de fundo, etc.), páginas e vistas, que tornam possível observar o ambiente simulado em diferentes perspetivas, e ainda o *script* principal da simulação (*main script*), que contém o código essencial para poder ser feita a simulação.

4.2.3 Concepção de um Modelo

A criação de um modelo no simulador pode ser dividida por várias etapas. Seguidamente, recorrendo a um exemplo, explica-se o processo que deverá ser efetuado de modo a obter-se um modelo estável e o mais aproximado possível com o real, garantindo o desempenho da simulação.

4.2.3.1 Criação das Formas Visíveis e Juntas

O primeiro passo será criar o aspeto visual do modelo. No caso deste trabalho isto foi realizado com as ferramentas disponibilizadas pelo simulador (Figura 4.12, à esquerda) que permitem gerar formas primitivas (cubos, cilindros, esferas, etc). Contudo, caso seja necessário um maior detalhe geométrico, poderão ser importados modelos CAD, como se mostra na Figura 4.12, à direita. Se tal acontecer, será preciso analisar a complexidade da malha e recorrer às funções disponibilizadas pela Coppelia Robotics para diminuir ao máximo a sua complexidade.

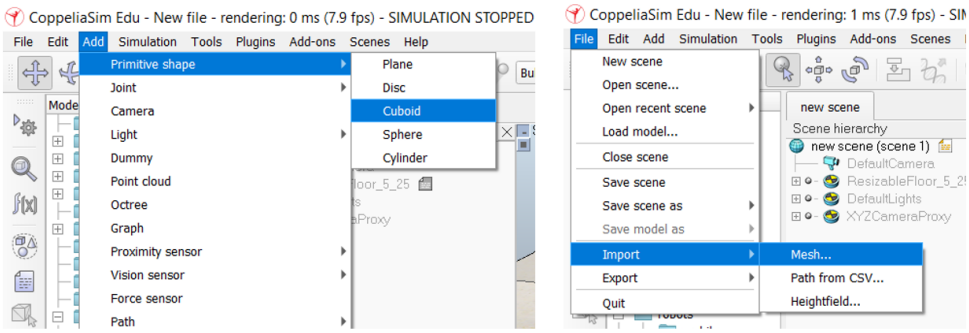


Figura 4.12: Funcionalidades de criação de formas primitivas no simulador e importação de malhas, respetivamente

Uma vez geradas as formas que representam o modelo, é importante adicionar à cena as juntas que também o irão constituir (através do mesmo menu apresentado na Figura 4.12), realizando-se posteriormente a sua configuração. Posto isto, o utilizador pode posicionar corretamente todos os objetos no ambiente virtual. Na Figura 4.13 pretende-se exemplificar o que foi descrito, recorrendo ao exemplo da modelação de um robô articulado. É possível ver que todos os objetos estão presentes na cena e já se encontram na posição correta, tal como no robô real, no

entanto, ainda é preciso rever mais alguns aspectos antes de ser possível realizar a simulação do robô.

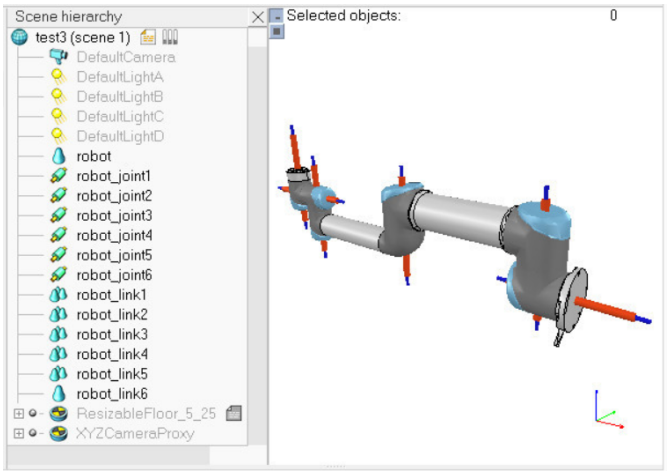


Figura 4.13: Exemplificação da constituição de um modelo de um robô articulado e do posicionamento dos seus constituintes (adaptado de [76])

4.2.3.2 Configuração das Propriedades Dinâmicas do Modelo

Para realizar a simulação dinâmica dos modelos, ou seja, a sua reação a forças externas, colisões ou até mesmo quedas, é necessário configurar as formas apropriadamente, sendo que cada uma delas poderá ser:

- ***Dynamic* ou *static***: as formas dinâmicas irão cair ou ser influenciadas por forças externas. Relativamente às estáticas, estas ficarão fixas numa determinada posição ou seguirão o movimento do objeto pai na hierarquia da cena;
- ***Respondable* ou *non-respondable***: uma forma *respondable* irá causar uma reação aquando da colisão com outras formas deste tipo; em contrapartida, tal não irá acontecer se a forma for *non-respondable*.

Dependendo daquilo que se pretende observar na simulação, deverá escolher-se estas propriedades para cada uma das formas. Isto pode ser feito na caixa de diálogo relativa às suas características dinâmicas, representada na Figura 4.14, que, para além do que foi referido, também permite alterar outros parâmetros importantes como a massa e os momentos de inércia.

Adicionalmente, deve ser tido em conta o facto de todas as formas *respondable* terem de ser o mais simples possível para garantir a estabilidade da simulação. Por esta razão, se as formas criadas anteriormente relativas ao aspeto visual do

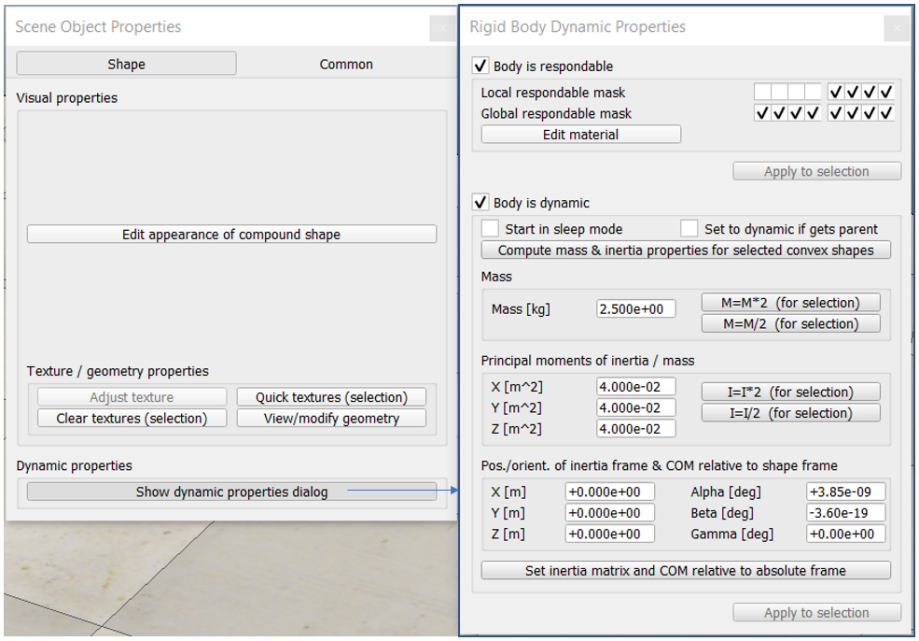


Figura 4.14: Propriedades dinâmicas de uma forma

modelo forem simples, também poderão ser usadas como as formas dinâmicas do modelo. Caso contrário, o mais correto é, por cada forma visível já existente, criar uma nova forma não visível que sirva como uma representação dinâmica e que apenas será utilizada pelo motor de física do simulador. Na Figura 4.15 é possível observar o mesmo exemplo do robô articulado já com as formas visíveis e dinâmicas criadas. Estas últimas encontram-se representadas a verde, rosa e cinza e é possível constatar que, embora mais simples, aproximam-se bastante das formas que serão visíveis. Além disso, sempre que sejam usadas formas dinâmicas diferentes das formas visíveis, as primeiras serão colocadas como o objeto pai na hierarquia da cena, como se observa na Figura 4.15, e as segundas terão de ser configuradas como estáticas. Deste modo, garante-se que ambas as formas (visível e dinâmica) coincidem em todos os momentos da simulação.

Neste caso, as formas dinâmicas apenas são utilizadas pelo motor de física, portanto todas as suas propriedades especiais podem ser desativadas (Figura 4.15, a vermelho na caixa de diálogo à direita). Inclusive, depois de efetuar o seu posicionamento desativa-se a sua visualização, o que fará com que apenas as formas visíveis sejam representadas na cena.

No presente passo da criação do modelo deverão ser também configuradas as juntas (Figura 4.16). Estas podem ter vários modos de funcionamento que são escolhidos consoante o que se procura simular [76]. No caso deste trabalho apenas interessa o modo *torque/force*, para simular as juntas atuadas por motor.

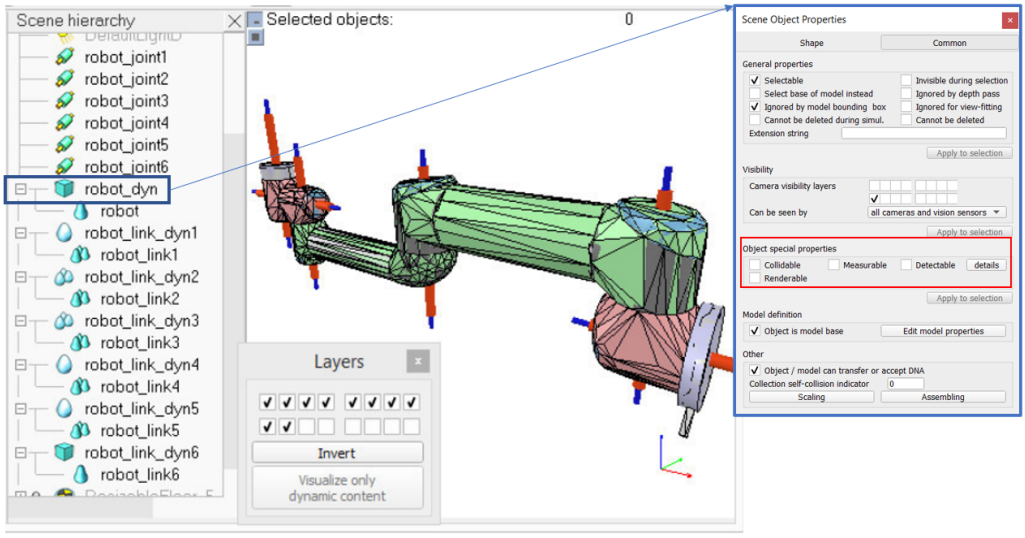


Figura 4.15: Exemplificação da criação das formas dinâmicas (adaptado de [76])

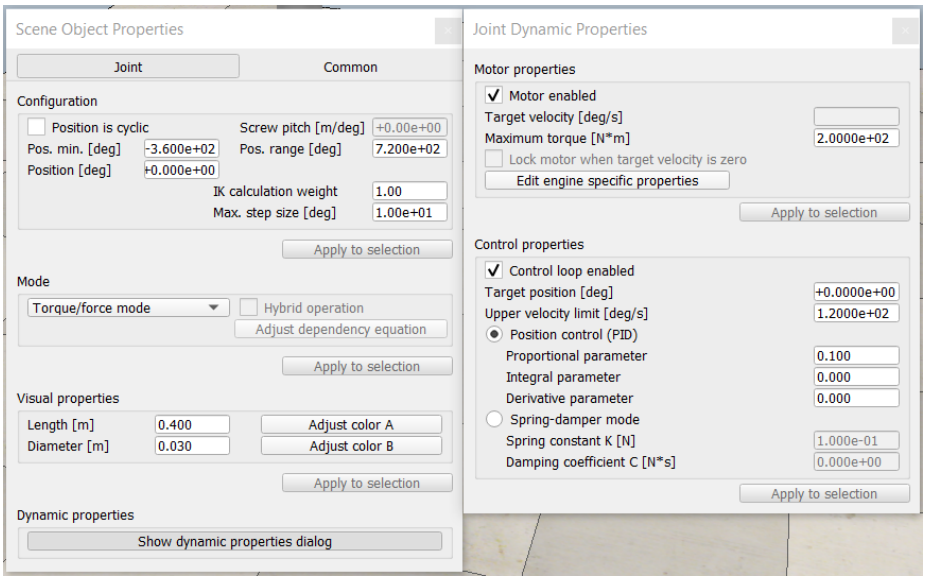


Figura 4.16: Exemplificação da definição das propriedades das juntas

4.2.3.3 Hierarquia, Inclusão de Sensores e Definição do Modelo

Para se poder realizar a simulação do robô resta criar a sua estrutura hierárquica. A hierarquia deve ser iniciada pela base do modelo, algo que será diferente dependendo daquilo que se pretende simular. No caso de um robô articulado será a base do robô, porém, num robô móvel o mais normal é que seja o chassis. A construção da hierarquia é algo que mudará de modelo para modelo mas que deve seguir a lógica de funcionamento do equipamento real. Como já foi referido

anteriormente, esta tarefa pode ser facilmente realizada através do *drag-and-drop* na janela da hierarquia da cena. É também importante referir que, conforme se constrói as relações parentais entre os objetos, deverá haver uma verificação das máscaras nas propriedades das formas dinâmicas (caixa de diálogo presente na Figura 4.14). Isto porque, duas formas do tipo *respondable* sucessivas na estrutura do modelo deverão ter máscaras locais em que nenhum dos oito *bits* seja igual, garantindo-se assim que não são processadas quaisquer colisões desnecessárias entre constituintes do mesmo modelo. Na Figura 4.17 exemplifica-se este processo para o mesmo exemplo do robô articulado.

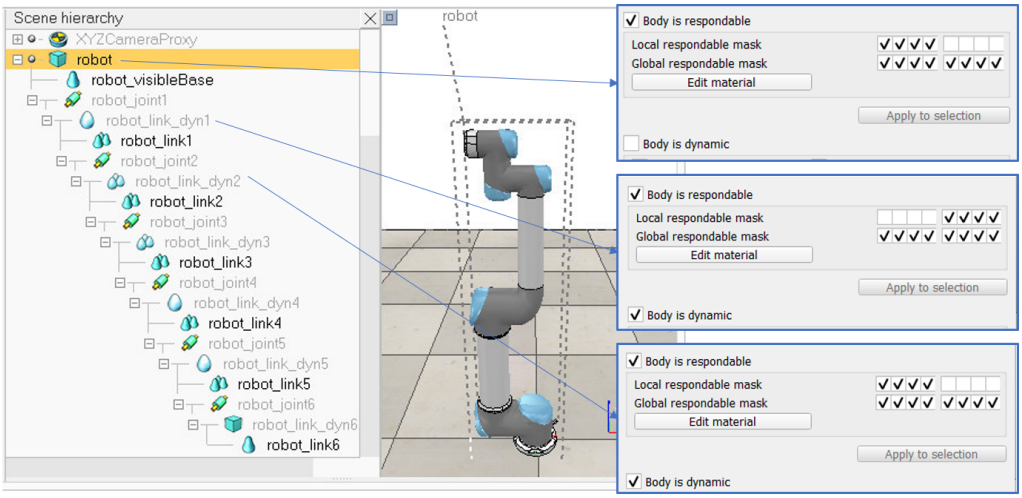


Figura 4.17: Exemplificação da criação da hierarquia de um modelo (adaptado de [76])

Efetuadas todas as operações descritas acima, o modelo deverá encontrar-se pronto para simular, contudo, poderá ser necessário realizar algumas alterações ao nível das propriedades das juntas e formas dinâmicas. Caso o modelo não tenha o comportamento esperado durante a simulação, o seu conteúdo dinâmico poderá ser analisado recorrendo à funcionalidade denotada na Figura 4.18 a vermelho, para ser possível encontrar erros na sua construção mais facilmente.

Para finalizar, o utilizador poderá incluir todos os sensores que forem pretendidos e é também uma boa prática identificar a base do modelo (Figura 4.18, à direita), pelas razões que já foram referidas na Subsecção 4.2.2, e, para evitar estragos no mesmo durante a sua manipulação, deve-se incluir as propriedades mostradas no canto inferior esquerdo da Figura 4.18. Desta maneira, quando o utilizador interagir com um objeto que pertença à hierarquia do robô, todo o modelo será selecionado e, consequentemente, manipulado como se fosse um só objeto.

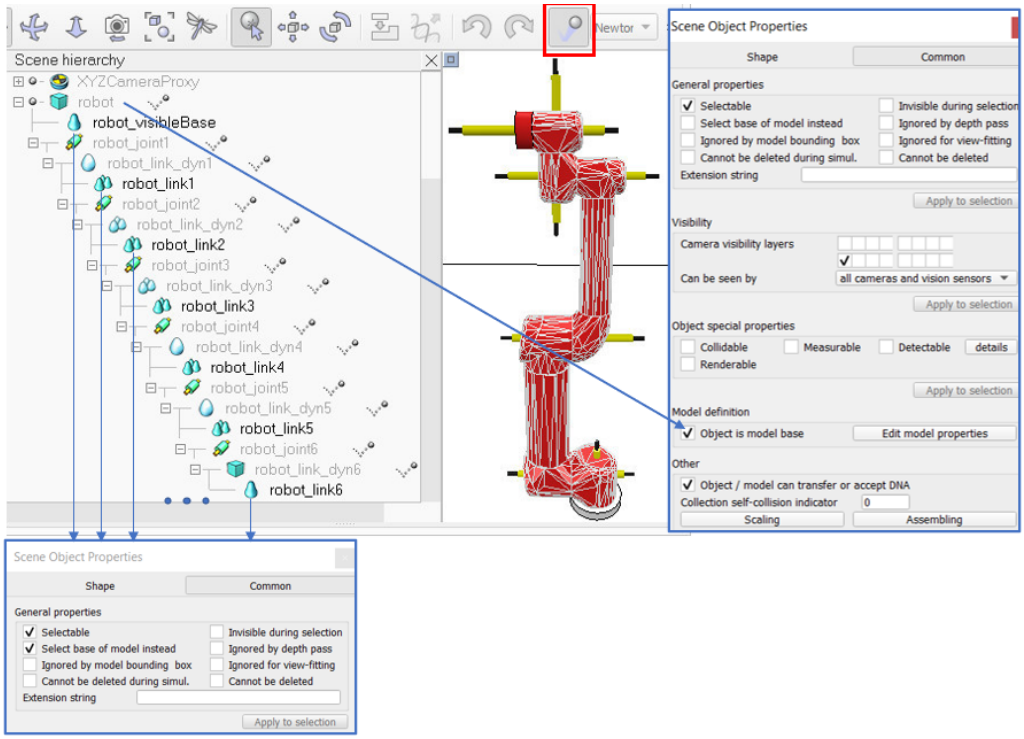


Figura 4.18: Análise do conteúdo dinâmico de um modelo e identificação do modelo (adaptado de [76])

4.2.4 Desenvolvimento do Código

Tal como foi referido anteriormente, naquilo que concerne à programação das funcionalidades e dos modelos que se pretende simular, existem várias abordagens que podem ser adotadas. Neste trabalho interessa apenas tratar os *scripts* Lua. Esta foi a metodologia pela qual se optou por ser a mais flexível, simples e compatível com todas as características do simulador.

Na Figura 4.19 observam-se os cinco tipos de *scripts* que são suportados, e, seguidamente, explica-se de forma sucinta qual o objetivo de cada um:

- ***Main script***: é o *script* principal e está sempre presente em qualquer cena, uma vez que contém o código que faz com que seja possível realizar a sua simulação, razão pela qual não deve ser modificado. Além disso, é responsável por chamar todos os outros *scripts* secundários;

- ***Child scripts***: estes *scripts* estão associados a um objeto ou modelo presente na cena e são usados para implementar os seus respetivos algoritmos de controlo. Porém, também podem ser aplicados noutros contextos.

  Existem dois tipos de *Child scripts*, os *Non-threaded* e os *Threaded*. Os primeiros são contituídos por funções que são chamadas a cada passo da

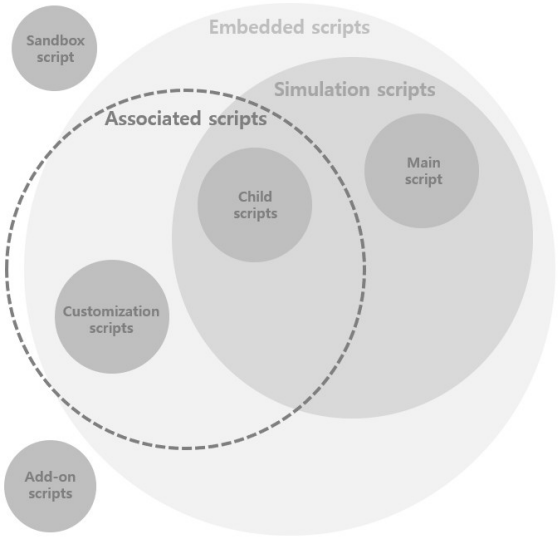


Figura 4.19: Tipos de *scripts* suportados pelo CoppeliaSim [76]

simulação e realizam uma tarefa (*callbacks*), devolvendo depois o controlo ao *script* que as chama. É essencial que isto aconteça, caso contrário, a simulação irá parar e não prosseguirá. Num *script Non-threaded* existem, entre outras, quatro funções mais importantes:

- *sysCall_init*: é a função de inicialização e também a única que não é opcional. Apenas é executada uma vez durante a simulação e contém o código de inicialização;
- *sysCall_actuation*: esta função é executada em cada passo da simulação durante a fase de atuação. Normalmente, contém o código relacionado com a atuação dos componentes do modelo;
- *sysCall_sensing*: função que também é executada em todos os passos da simulação durante a fase de sensorização. Contém, por norma, o código relativo à aquisição de dados através dos sensores do modelo;
- *sysCall_cleanup*: é a função de restauro e é executada uma vez no final da simulação ou quando o *script* é eliminado.

No que toca aos *scripts* do tipo *Threaded*, estes são executados numa nova *thread*, paralelamente à simulação. Por esta razão, não há desde logo sincronia com o passo da simulação, embora isto possa vir a ser conseguido recorrendo a um determinado conjunto de funções. Em contrapartida, a simulação não será interrompida durante a sua execução, sendo que a execução do *script* é que é interrompida no final de um passo e retomada no seguinte. Em relação à constituição de um destes *scripts* existem duas funções essenciais:

  - *sysCall_threadmain*: é a função principal do *script* e é executada desde que se inicia a nova *thread* até ao seu final. É nela que é colocado o código de inicialização e o principal *loop* do código responsável por controlar um determinado aspecto da simulação;
  - *sysCall_cleanup*: função que tem a mesma funcionalidade explicada no caso anterior.

  Recorrendo a estes *scripts* o utilizador pode implementar e testar as suas ideias de uma forma rápida e bastante flexível. Em termos de escolha de *script*, esta recai sobre o utilizador e aquilo que é pretendido simular. Contudo, segundo os desenvolvedores do CoppeliaSim, os *scripts Non-threaded* deverão ser sempre a primeira escolha visto que existem certos aspectos inerentes aos *Threaded* que poderão afetar o desempenho durante a simulação.

- ***Scripts de customização***: como acontece no exemplo apresentado anteriormente, estes *scripts* também estão associados a um objeto ou modelo, no entanto, são dedicados à customização de parâmetros do mesmo (p. ex.: altura, comprimento, etc.);
- ***Scripts Add-on* e *Sandbox***: podem ser empregues pelo utilizador para implementar certas funcionalidades mais genéricas (que não estão ligadas a um modelo ou cena em específico), permitindo, até um certo nível, personalizar o simulador.

No presente trabalho, sendo o objetivo simular o comportamento dos dois robôs omnidirecionais, a maioria do código foi implementada recorrendo a *Child Scripts*.

Capítulo 5

Modelação do Sistema de Co-transporte

No decorrer deste capítulo tratam-se aspetos relevantes acerca do sistema de co-transporte em estudo, mais concretamente, as características e constituição dos robôs e o modo como estes se deverão coordenar aquando do transporte de uma carga. Tendo tudo isto em conta, será explicado como foi feita a modelação do sistema no CoppeliaSim para, posteriormente, poder ser efetuada a sua simulação.

5.1 Apresentação do Sistema

O sistema simulado neste trabalho é constituído por duas plataformas omnidirecionais, com rodas *mecanum*, denominadas DiscoveryQ2, que são produzidas pela empresa Hangfa [77]. A escolha destas plataformas para a realização do trabalho deveu-se a serem equipamentos existentes no Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESC TEC), com os quais o sistema de co-transporte poderia ser implementado, algo que não aconteceu devido ao atual contexto de pandemia.

Na Figura 5.1, à esquerda, pode ver-se o aspecto do robô *leader* do sistema. Este robô foi implementado por um outro autor numa fase anterior a este projeto e possui [78]:

- 4 rodas *mecanum*, cada uma com um motor com caixa redutora e *encoder* incremental;
- bloco de apoio para reduzir a carga no eixo do motor e prolongar a sua vida útil, bem como para aumentar a capacidade de carga do robô;

- bateria de lítio de 12 V;
- um módulo de controlo dos motores para efetuar o controlo direto da velocidade do robô ou de cada uma das suas rodas independentemente;
- um módulo responsável pela alimentação dos restantes módulos e por permitir a sua interligação, possibilitando a comunicação entre eles;
- Raspberry Pi 3, que é o "cérebro" do sistema;
- Arduino Mega que tem como objetivo ler os *encoders* e servir de interface entre o Raspberry e o controlador dos motores;
- laser URG-04LX-UG01 da Hokuyo para, juntamente com a odometria do robô, permitir a sua localização absoluta em relação a um referencial externo.

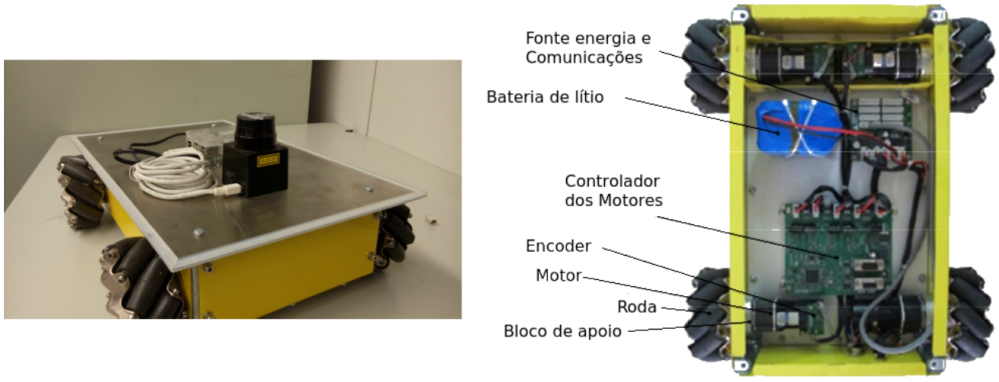


Figura 5.1: Robô *leader* do sistema e respetiva constituição [78]

Relativamente ao robô *follower*, este ainda não se encontra implementado. Porém, o pretendido é que toda a sua estrutura e componentes base, ou seja, todo o *hardware* identificado na Figura 5.1, à direita, seja análogo ao do robô *leader*. Adicionalmente, o Raspberry e o Arduino Mega também poderão ser mantidos, ou, caso não se justifique, poderá ser utilizada a placa de desenvolvimento RHF407 baseada no Cortex M4STM32F407, que vem incluída com a plataforma, ou outra solução em conta. Aquilo que irá diferir é, essencialmente, a maneira como este planeia os seus movimentos de modo a coordenar-se com o *leader*. Isto terá de ser conseguido através do uso de um sensor de força e, eventualmente, um sensor de posição angular (dependendo das capacidades do sensor de força usado).

O sistema constituído pelos dois elementos apresentados deverá possuir uma arquitetura de controlo descentralizada, em que ambos os robôs serão igualmente

responsáveis pelo controlo da tarefa. O *leader* irá planear a trajetória e definir a velocidade do sistema, enquanto que o *follower*, através da utilização do sensor (ou sensores), movimentar-se-á de modo a anular as forças exercidas na carga pelo *leader*. A comunicação direta entre os robôs deverá ser inexistente, ou, caso seja mesmo necessária, reduzida o máximo possível. Relativamente à estratégia de transporte, será implementada a técnica de *grasping*. Para isso, ambos os robôs terão de possuir um meio que lhes permita agarrar e suportar o objeto. No caso do robô *follower*, será na estrutura mecânica deste suporte que se deverão encontrar os sensores, de modo a possibilitar a monitorização das forças.

Neste trabalho serão exploradas duas abordagens no que diz respeito à construção do suporte do robô *follower*. Numa delas será simulado o uso de um sensor de força/ binário multiaxial. Este tornará possível a monitorização de três componentes de força, como ilustrado na Figura 5.2, à esquerda. Tal informação permitirá ao robô ajustar as suas velocidades lineares e rotacionais sem que a formação do sistema seja quebrada.

Na segunda abordagem será simulada uma célula de carga, que apenas permitirá monitorizar as forças exercidas segundo um eixo, porém, será também incluída uma junta rotacional que fará com que o suporte passe a ter um grau de liberdade, como se pode ver na Figura 5.2, à direita. A posição desta junta será monitorizada com recurso a um sensor de posição angular (p.ex.: potenciómetro rotativo ou *encoder*). Embora possa existir um desalinhamento entre os robôs durante o transporte, deixarão de existir forças de torção exercidas na carga e este sensor também será mais acessível naquilo que concerne aos custos do sistema.

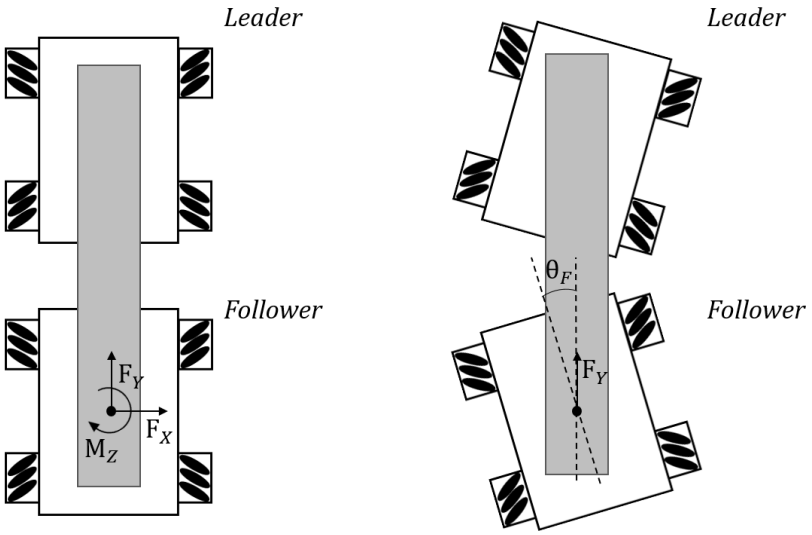


Figura 5.2: Abordagens seguidas na construção do suporte do robô *follower*

5.2 Modelação das Plataformas

Depois de apresentado o sistema e algumas das funcionalidades da ferramenta de simulação é possível iniciar a descrição do processo envolvido na criação dos modelos virtuais das plataformas. Numa primeira fase, modelou-se a estrutura base das mesmas, ou seja, o seu chassis e as quatro rodas omnidirecionais. Convém realçar que durante a modelação do sistema seguiu-se o mesmo processo exemplificado na Secção 4.2.3, do Capítulo 4.

Para efeitos de simulação existem ainda outras propriedades que são cruciais e que devem ser conhecidas, para que se possa criar um modelo 3D o mais aproximado possível ao robô real. Na Figura 5.3 podem ser verificadas as dimensões do DiscoveryQ2. Inclusivamente, segundo o fabricante, a plataforma apresenta um peso de 10 kg, uma velocidade linear máxima de 0,65 m/s e uma velocidade rotacional máxima de 140°/s.

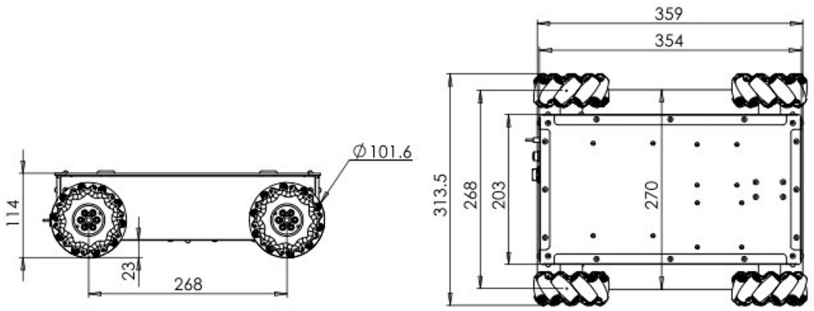


Figura 5.3: Dimensões das plataformas DiscoveryQ2 [77]

O chassis do DiscoveryQ2 pode ser facilmente recriado com as ferramentas disponibilizadas pelo simulador, já que se assemelha bastante com um simples paralelepípedo. O mesmo acontece com a tampa que fica por cima do chassis e que isola os componentes do exterior. Por esta razão, acrescentaram-se duas formas *cuboid* à cena, com os respetivos tamanhos do modelo real. Depois de se fazer isto, a cena apresentará dois objetos, que, sendo reposicionados e agrupados num só, terão um aspecto idêntico aos chassis dos robôs reais, como se pode verificar na Figura 5.4.

Seguidamente, foi necessário incluir as rodas omnidirecionais. Reproduzir o modelo exato das rodas é algo bastante complexo e que não constava no âmbito do trabalho. Por esta mesma razão, uma vez que existem no simulador modelos de rodas omnidirecionais bastante aproximados das reais, decidiu-se usar um deles, mais especificamente o das rodas da plataforma OmniRob da Kuka. Este robô pode ser facilmente encontrado no *browser* dos modelos e arrastado para o ambiente virtual. Contudo, há que referir que as rodas do robô possuem dimen-

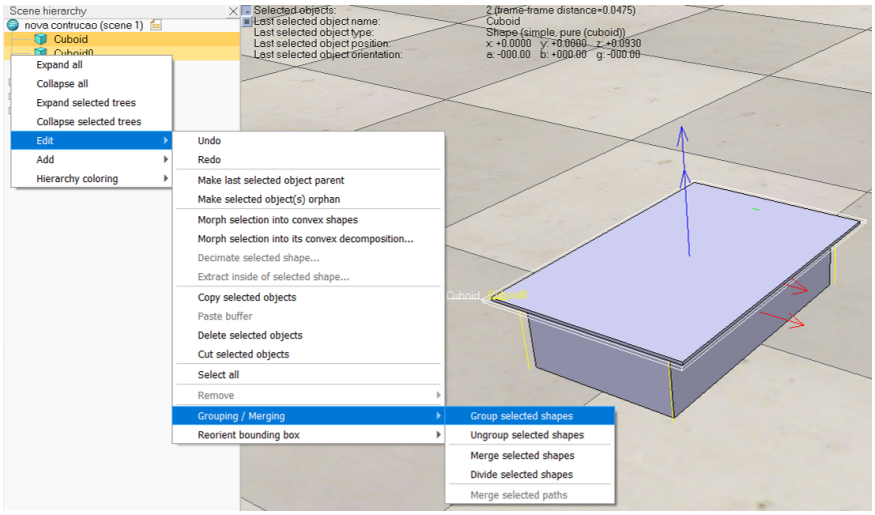


Figura 5.4: Junção de vários objetos e aparência final do modelo do chassis do DiscoveryQ2

sões consideravelmente maiores do que as do DiscoveryQ2, o que faz com que seja necessário escalar o modelo (algo que também pode ser feito no simulador, como se pode verificar na Figura 5.5). Efetuadas as alterações necessárias, é possível posicionar as formas visíveis tal como no robô real, e, com umas pequenas alterações, o aspeto fica bastante aproximado das plataformas reais, como se mostra na Figura 5.6

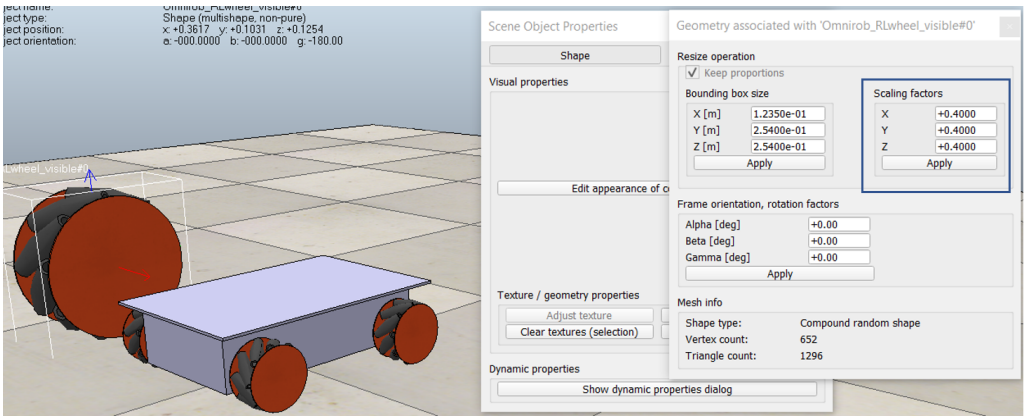


Figura 5.5: Modelo das rodas do OmniRob e alteração da sua dimensão

Já com as formas visíveis criadas, partiu-se para a criação das juntas e do modelo dinâmico da plataforma. No que toca ao chassis, sendo as formas visíveis que o constituem muito simples, usaram-se as mesmas como as formas dinâmicas. Relativamente às rodas omnidirecionais, aplicou-se a mesma técnica verificada em outros modelos de robôs presentes no simulador que possuem este tipo de

mobilidade. Cada roda *mecanum* é constituída por duas esferas, que são as formas mais básicas pelas quais esta pode ser representada dinamicamente, devido a possuir os rolos, como se pode concluir a partir da Figura 5.6. O facto de se usar as duas esferas passa por também serem necessárias duas juntas para simular o comportamento da roda, ou seja, uma junta será atuada e a outra rodará livremente, representando-se assim a atuação do motor e o movimento passivo dos rolos, respetivamente (Figura 5.6). Para que isto aconteça as juntas deverão ser configuradas no modo *torque/force* e, a junta atuada, terá ainda de possuir a propriedade *motor enabled*, como pode ser observado na Figura 5.7.

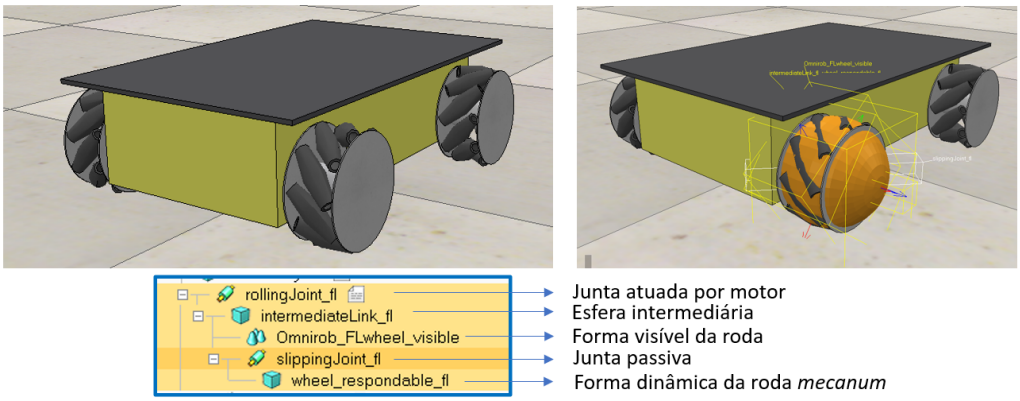


Figura 5.6: Formas visíveis do DiscoveryQ2 e constituição de uma roda *mecanum* no CoppeliaSim

Por fim, foi definida a estrutura hierárquica mostrada na Figura 5.8, tendo em conta as máscaras locais. É importante selecionar corretamente as máscaras (como explicado no capítulo anterior), visto que a forma dinâmica das rodas irá estar sempre em colisão com a do chassis por ser uma esfera.

No final de todo o processo identificou-se a base do modelo e acrescentou-se um *script* do tipo *Non-threaded* a cada junta atuada por motor. Este contém o código que possibilita recriar o comportamento da roda *mecanum* e encontra-se presente na Secção A.1 do Anexo A (tanto para as rodas *left-handed* como para as *right-handed*). De seguida, explica-se sucintamente o código e qual a sua importância na simulação.

5.2.1 Implementação do Comportamento da Roda Mecanum

Quando se analisa a estrutura hierárquica da roda (apresentada nas Figuras 5.6, 5.7 e 5.8) verifica-se que a junta passiva, que permite à esfera rodar livremente segundo o ângulo dos rolos, é "filha" da esfera intermediária, que, por sua vez, é "filha" da junta que representa o motor. Deste modo, logo que o motor seja atuado durante a simulação, todos os elementos que se encontram abaixo na estrutura

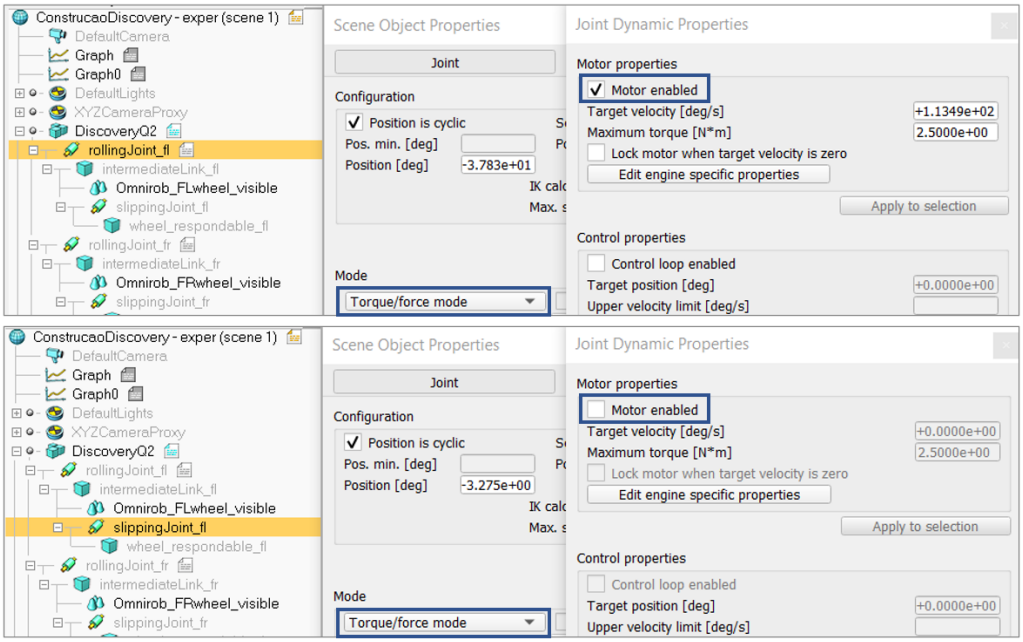


Figura 5.7: Propriedades das duas juntas que constituem a roda *mecanum*

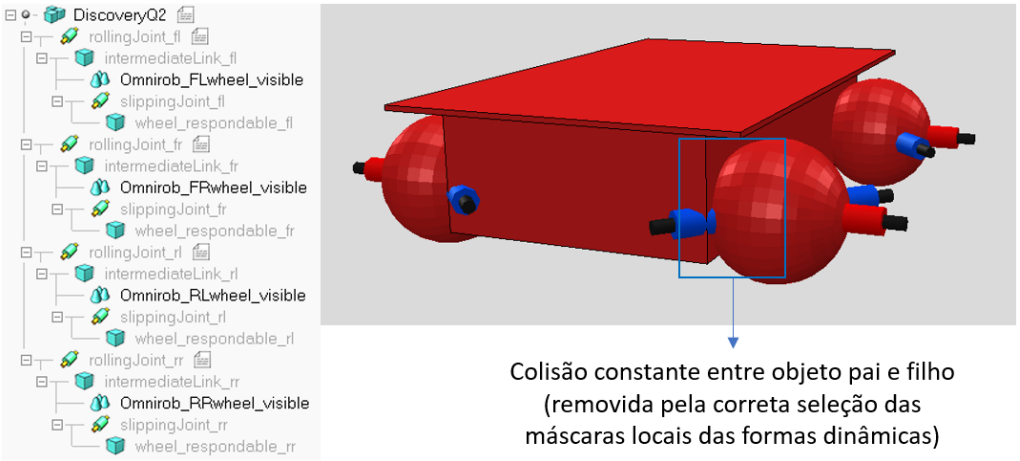


Figura 5.8: Hierarquia do modelo das plataformas e representação das colisões entre o chassis e as rodas

hierárquica irão realizar o mesmo movimento, respeitando as relações parentais. Na Figura 5.9 ilustra-se o que foi explicado.

Este comportamento não é aquilo que se pretende para a roda *mecanum*. O suposto é que a junta passiva se encontre sempre na horizontal, segundo um ângulo de 45° (ou −45°) em relação ao eixo do motor, como se pode ver na Figura 5.9, à esquerda, de modo a simular a rotação dos rolos. A única maneira de garantir que tal acontece passa por forçar a junta a ficar na mesma posição

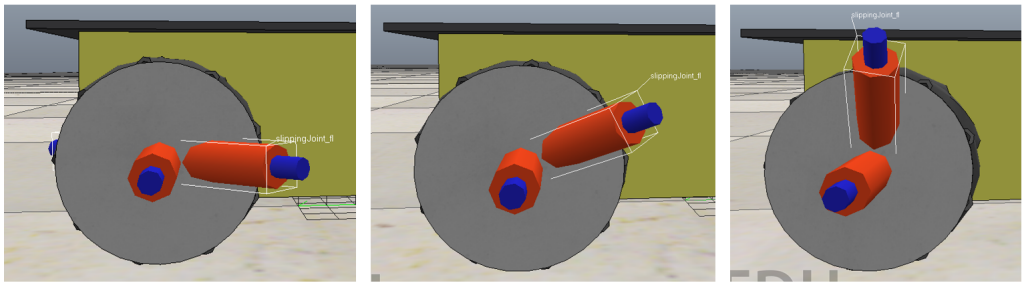


Figura 5.9: Comportamento da roda *mecanum* antes da implementação do *script*

recorrendo ao *script Non-threaded*. De seguida mostra-se um excerto do *script*, mais concretamente, a função *sysCall_actuation()*.

```
function sysCall_actuation()
  if (sim.getObjectParent(rolling)~=-1) then
        -- 1 --
        local linVel,angVel=sim.getVelocity(wheel)
        -- 2 --
        sim.resetDynamicObject(wheel)
        -- 3 --
        sim.setObjectFloatParameter(wheel,3000,linVel[1])
        sim.setObjectFloatParameter(wheel,3001,linVel[2])
        sim.setObjectFloatParameter(wheel,3002,linVel[3])
        sim.setObjectFloatParameter(wheel,3020,angVel[1])
        sim.setObjectFloatParameter(wheel,3021,angVel[2])
        sim.setObjectFloatParameter(wheel,3022,angVel[3])
        -- 4 --
        sim.setObjectPosition(slipping,rolling,{0,0,0})
        sim.setObjectOrientation(slipping,rolling,{-math.pi/4,0,0})
        sim.setObjectPosition(wheel,rolling,{0,0,0})
        sim.setObjectOrientation(wheel,rolling,{0,0,0})
    end
end
```

O objetivo desta função é, em todos os passos da simulação:

1. guardar as velocidades da forma dinâmica da roda;
2. realizar um *reset* à forma dinâmica (essencial para se poder realizar alterações à posição ou à orientação de uma forma ou junta de uma cadeia cinemática durante a simulação, programaticamente);
3. voltar a atribuir as velocidades que foram guardadas anteriormente à forma;
4. reposicionar e reorientar a junta passiva e a forma dinâmica que representa a roda *mecanum*.

Com a implementação do *script* conseguiu-se criar o comportamento desejado para a roda *mecanum*, porém, é necessário salientar que este método está diretamente dependente do passo da simulação (Figura 5.10). Se este for bastante pequeno a junta estará mais próxima da orientação desejada; conforme aumenta verifica-se que a cadência a que o *script* é executado não permite manter a orientação da junta e a roda não funcionará como é suposto. Inclusive, este fenómeno poderá agravar-se quando a velocidade angular do motor é muito elevada. Foi então necessário identificar um compromisso entre o passo da simulação e as velocidades máximas da plataforma para que o comportamento fosse o mais realístico possível, como se verá na Subsecção 5.2.2.

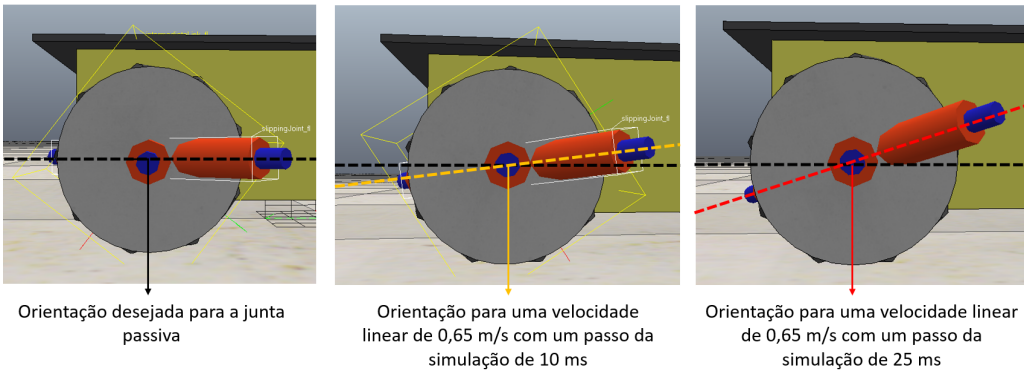


Figura 5.10: Comportamento da roda *mecanum* depois da implementação do *script* para diferentes tempos de passo da simulação

A hipótese de se realizar a roda *mecanum* de uma forma diferente, sem que dependesse do *script*, ou seja, criando juntas e formas dinâmicas por cada um dos rolos, nunca foi considerada já que seria um modelo extremamente complexo. Uma vez que as duas plataformas possuem no total oito destas rodas, o desempenho da simulação seria bastante afetado e poderiam surgir outros problemas. Esta solução é a forma mais estável de se simular a mobilidade omnidirecional das plataformas no CoppeliaSim.

5.2.2 Testes Realizados à Plataforma Modelada

Assim que a plataforma se encontrava modelada e os *scripts* relativos a cada roda implementados, fez-se um teste para observar o funcionamento do modelo e qual o melhor incremento de tempo da simulação para simular as rodas do DiscoveryQ2. O teste realizado consistiu em implementar um novo *script* do tipo *Threaded* (Secção A.2, Anexo A). De seguida mostra-se um pequeno extrato do código presente no *script*, que diz respeito ao ciclo principal do programa.

```
1 ( . . . )
```

```
2
3     while sim.getSimulationState()~=
      sim.simulation_advancing_abouttostop do
4         timepassed=sim.getSimulationTime()
5
6         if(timepassed>0 and timepassed<=2) then
7             vy=LinVel
8             vx=0
9         end
10
11        if(timepassed>2 and timepassed<=4) then
12            vy=-LinVel
13            vx=0
14        end
15
16        if(timepassed>4 and timepassed<=5) then
17            vy=0
18            vx=0
19        end
20
21        if(timepassed>5 and timepassed<=7) then
22            vy=0
23            vx=LinVel
24        end
25                                (...)
```

Facilmente se constata que a função do programa é atribuir diferentes velocidades lineares e rotacionais à plataforma consoante o tempo decorrido na simulação. Estas velocidades, recorrendo ao modelo cinemático da plataforma (equações 3.9 e 3.10, apresentadas no Capítulo 3), serão traduzidas nas velocidades angulares relativas a cada uma das rodas, como se mostra de seguida na equação 5.1.

$$\begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \dot{\varphi}_3 \\ \dot{\varphi}_4 \end{bmatrix} = -\frac{1}{0,05} \begin{bmatrix} -1 & 1 & 0,268 \\ 1 & 1 & -0,268 \\ -1 & 1 & -0,268 \\ 1 & 1 & 0,268 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{5.1}$$

Com este *script* foi possível simular movimentos lineares e rotacionais da plataforma à sua velocidade máxima (0,65 m/s e 140°/s, respetivamente) com diferentes incrementos de tempo da simulação. Na Figura 5.11 mostram-se as velocidades da plataforma durante a realização dos movimentos e é possível verificar de que forma o desalinhamento da junta passiva, explicado anteriormente, pode originar erros na simulação da plataforma.

No que diz respeito à velocidade longitudinal (Vy - a vermelho na Figura 5.11) esta não depende da rotação da junta passiva, por esta mesma razão mantém-se igual independentemente do incremento de tempo usado. Porém, no que toca

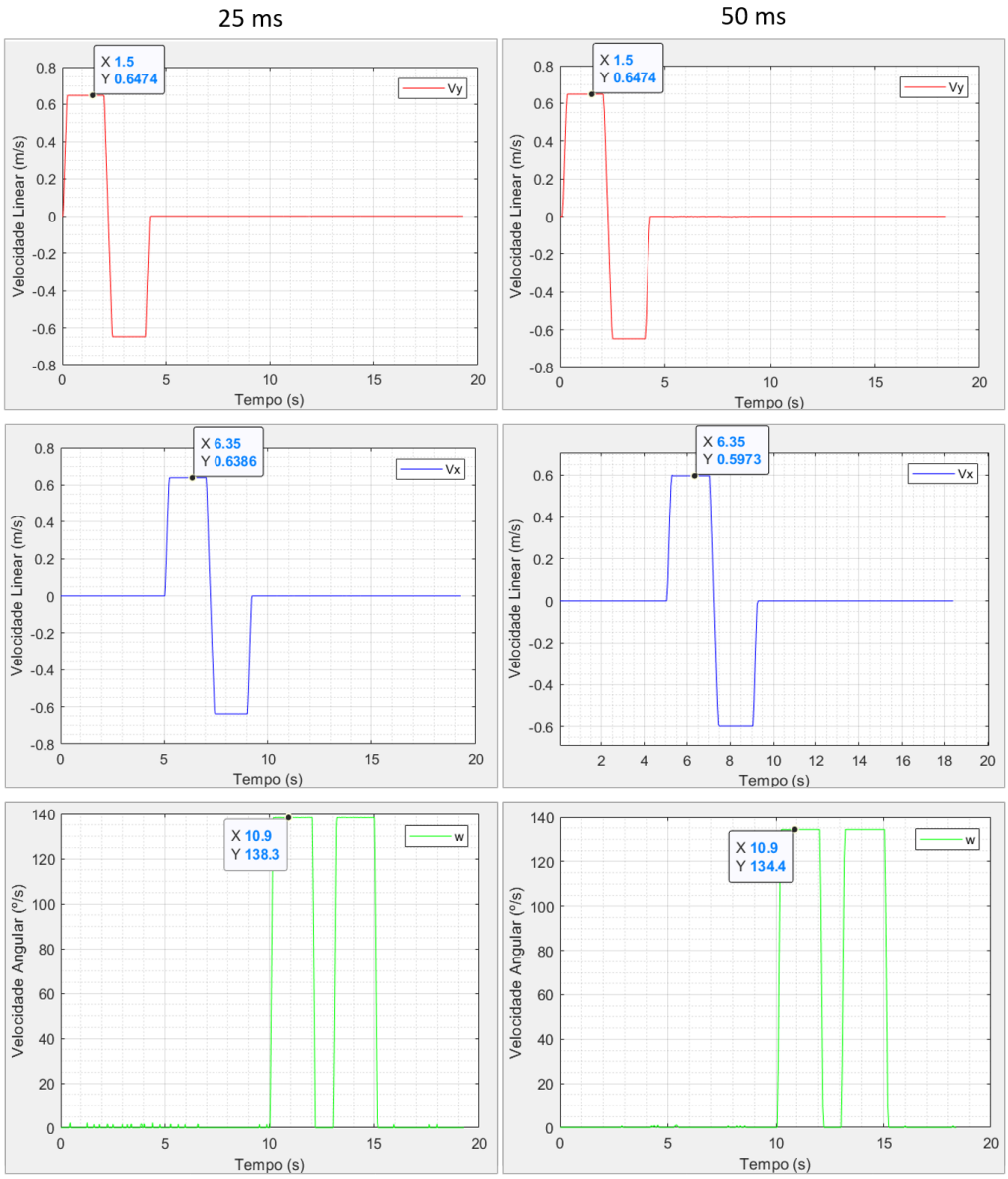


Figura 5.11: Comparação das velocidades máximas da plataforma com um tempo de passo de 25 e 50 ms, respetivamente

à velocidade transversal e rotacional (Vx e W - a azul e verde na Figura 5.11, respetivamente), observa-se que existe um erro originado pelo desalinhamento da junta. Teoricamente, caso não existisse o erro, o valor da velocidade transversal máxima teria de ser igual à velocidade longitudinal máxima, contudo, verifica-se que este é ligeiramente mais baixo. Conclui-se também que o erro irá aumentar conforme aumenta o incremento de tempo da simulação e fará com que a velocidade da plataforma seja mais baixa que o pretendido (0,65 m/s). O mesmo acontece no caso da velocidade rotacional. Por esta razão houve a necessidade

de garantir que o incremento usado originasse o menor erro possível em toda a gama de velocidades a que o DiscoveryQ2 se pode movimentar.

Na simulação do sistema de co-transporte escolheu-se usar um incremento de tempo máximo de 25 ms já que, para este valor, os erros devidos a este fenómeno eram bastante pequenos e a simulação ainda apresentava um bom desempenho. Além disso, convém referir que os resultados apresentados são relativos às velocidades máximas, e, como foi referido anteriormente, a velocidades mais baixas o erro será ainda menor. Na Figura 5.12 pode ver-se que para movimentos transversais e rotacionais a uma velocidade menor (0,3 m/s e 70°/s, respetivamente) o erro devido ao desalinhamento da junta é muito reduzido com um passo de 25 ms, e até mesmo de 50 ms. Uma vez que também não se pretende que o sistema se desloque a velocidades demasiado elevadas, o que pode por em causa o correto transporte da carga, um incremento de 25 ms é pequeno o suficiente para não existirem erros consideráveis.

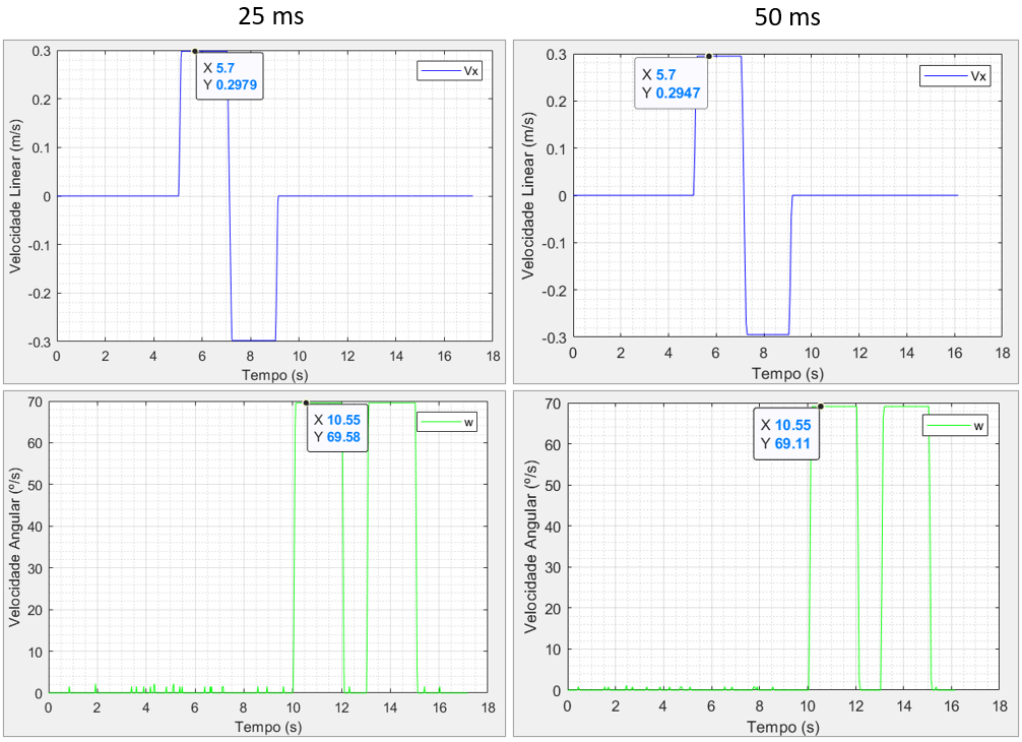


Figura 5.12: Comparação das velocidades transversais e rotacionais da plataforma com um tempo de passo de 25 e 50 ms, respetivamente

A realização deste teste permitiu:

- verificar a correta implementação da cinemática da plataforma;
- verificar o comportamento das juntas passivas;

- escolher o melhor compromisso entre o incremento de tempo da simulação e as velocidades a que as plataformas se podem movimentar, sem que o desempenho da simulação fosse afetado e houvessem erros consideráveis.

Convém realçar que se realizou este teste com os vários motores de física disponibilizados pelo simulador e que para todos eles os resultados obtidos foram idênticos, o que comprova a correta modelação do DiscoveryQ2. Embora não tenha sido possível recriar a plataforma de uma forma exata, visto que também não se conheciam diversos parâmetros relativos à mesma e foi preciso criar uma certa abstração no que diz respeito ao funcionamento das rodas omnidirecionais, fez-se por deixar o modelo o mais realístico possível.

5.3 Modelação de um Suporte

Depois da criação das plataformas foi necessário incluir na simulação o modelo do suporte que permite aos robôs agarrar e transportar o objeto no topo do seu chassis, e, ao mesmo tempo, medir as forças que nele são exercidas.

5.3.1 Criação de uma Conexão Robô-Objeto

Para facilitar o trabalho, e uma vez que a modelação de uma garra no simulador também não constava do âmbito deste projeto, pensou-se em usar um dos modelos já existentes na biblioteca do CoppeliaSim - a garra Robotiq 2F-85, ilustrada na Figura 5.13. Todavia, conforme se fizeram experiências com o modelo, foi verificado que este era extremamente complexo (Figura 5.13, à direita) e apresentava diversos comportamentos indesejados que, para serem resolvidos, levariam a ter que se efetuar um ajuste de todas as propriedades das formas dinâmicas e das juntas.

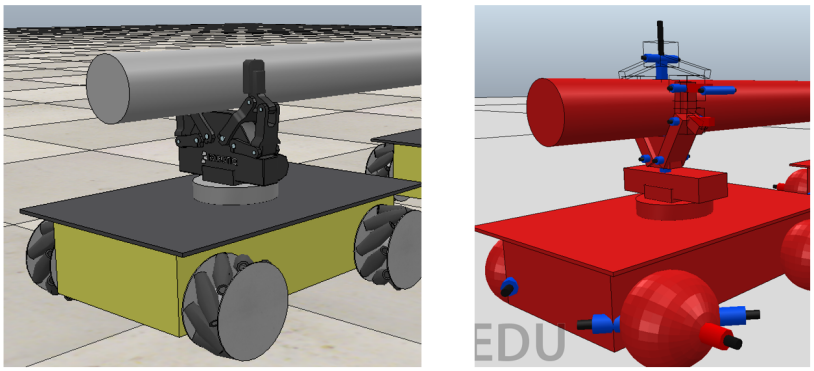


Figura 5.13: Robotiq 2F-85 e complexidade do respetivo modelo dinâmico

Posto isto, foi necessário encontrar uma alternativa para simular uma conexão entre os robôs e o objeto durante o transporte, de uma forma simples e que não estivesse sujeita às instabilidades que podem surgir quando se usa um modelo realístico de uma garra. A solução passou por implementar uma base com um comportamento idêntico a uma ventosa, que é algo que também existe no simulador. Para isso foi criada uma nova forma do tipo *cuboid* e, seguidamente, replicou-se o comportamento da ventosa na mesma, como se mostra na Figura 5.14 (o código do *script non-threaded* é idêntico ao do modelo presente no simulador e também pode ser visto no Anexo A, Secção A.3). Durante a simulação, o sensor de proximidade que existe no modelo irá detetar a forma do tipo *respondable* que está próxima da base e, recorrendo aos *dummies*, será criada uma conexão entre a base e o objeto (representada pelas setas a azul na hierarquia da cena, como se mostra na Figura 5.15). Esta abordagem provou ser muito mais estável durante a simulação.

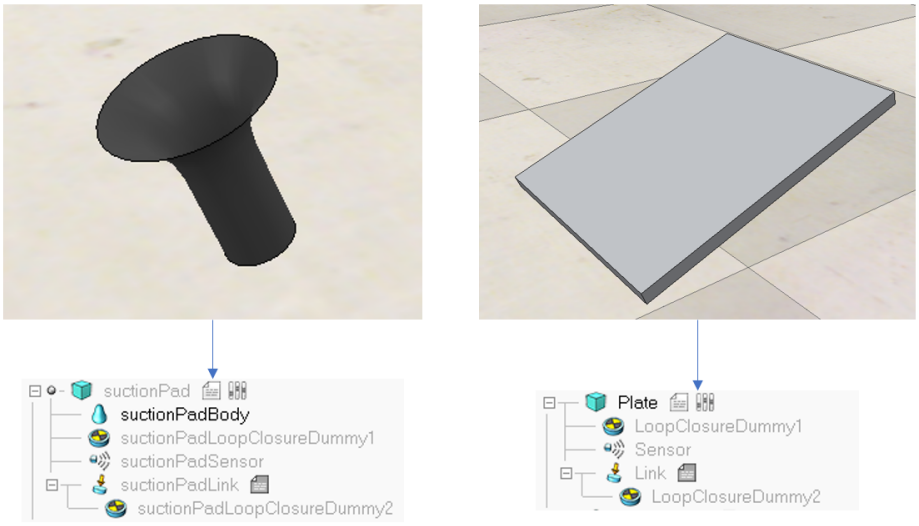


Figura 5.14: Comparação entre o modelo da ventosa e a base criada

5.3.2 Inclusão do Sensor de Força

O suporte, além de possibilitar a conexão com o objeto, tem de incluir o sensor de força para que sejam monitorizadas as forças impostas pelo robô *leader*. Sendo assim, criou-se uma nova forma do tipo *cylinder* e entre ela e a base foi colocado o sensor de força, tal como se mostra na Figura 5.16. O sensor de força no CoppeliaSim equivale a um sensor de força/binário de seis eixos, o que faz com que se possa medir todas as forças no espaço tridimensional recorrendo a esta entidade do simulador.

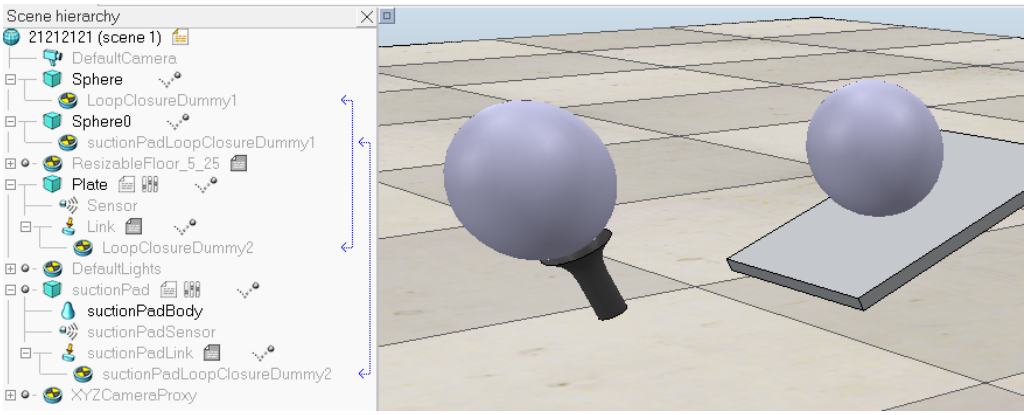


Figura 5.15: Exemplificação do comportamento da ventosa

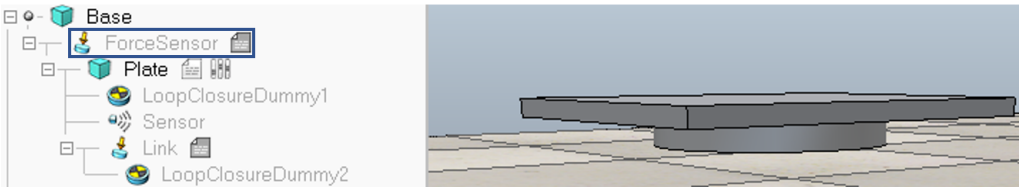


Figura 5.16: Inclusão do sensor de força no suporte

5.3.3 Colocação do Suporte nas Plataformas

O processo da modelação do suporte foi finalizado com a sua colocação na plataforma, tal como se pode verificar na Figura 5.17. A hierarquia dos dois modelos é interligada por meio de uma junta rotacional que possibilitará a simulação do sistema caso se utilize o sensor de força/binário multiaxial ou a célula de carga com o sensor de posição angular na montagem do suporte do *follower*, sem ser necessário realizar quaisquer alterações à estrutura dos modelos. Isto é conseguido através da propriedade da junta mostrada na Figura 5.18. Se for necessário um comportamento estático por parte da junta ativam-se as propriedades *motor enabled* e *Lock motor when target velocity is zero*, caso contrário mantêm-se estas propriedades desativadas.

É importante referir que embora a plataforma e o suporte sejam dois modelos diferentes, pertencem à mesma hierarquia. Por isso, as máscaras locais das formas *respondable* devem ser selecionadas, como já se explicou (diferentes entre objetos pai e filho), de modo a não existirem comportamentos estranhos por parte dos modelos durante a simulação.

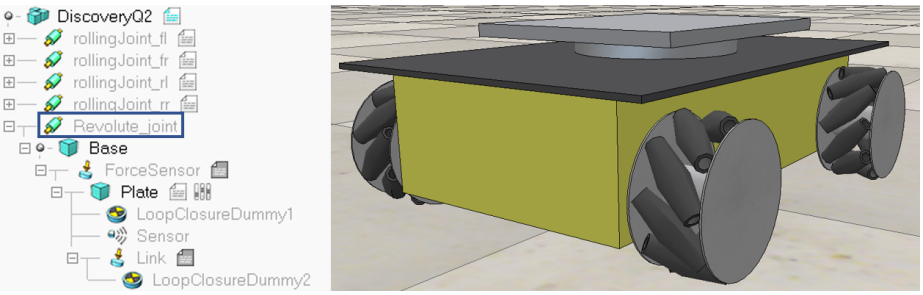


Figura 5.17: Plataforma DiscoveryQ2 e respetiva estrutura hierárquica depois da colocação do suporte

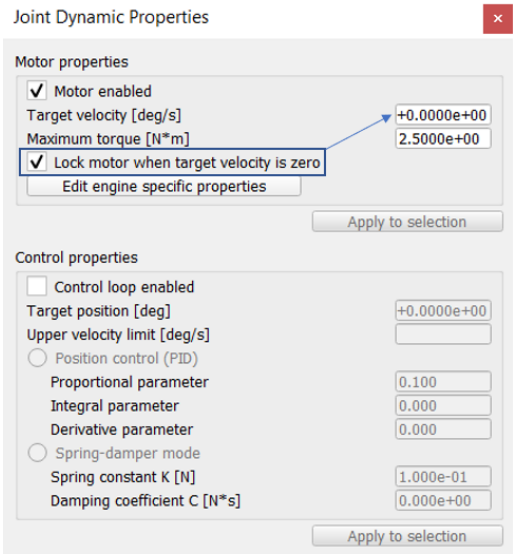


Figura 5.18: Propriedades *motor enabled* e *Lock motor when target velocity is zero*

5.3.3.1 Verificação da Correta Implementação do Suporte

Modeladas as plataformas e os suportes criou-se uma cena onde se encontram os dois robôs, *leader* e *follower*, de modo a poder simular-se o sistema de co-transporte. Além dos robôs, juntou-se à cena um objeto que representa a carga que deve ser transportada por ambos.

Depois de se posicionar o objeto sobre os robôs iniciou-se a simulação. Na Figura 5.19 é possível verificar que os suportes criam a conexão com o objeto (setas a azul na hierarquia da cena). Adicionalmente, recorrendo ao mesmo *script* implementado para testar as plataformas (Anexo A - Secção A.2), aplicaram-se velocidades lineares e rotacionais ao robô *leader*. Com isto foi possível observar que, logo que o robô *leader* realizava algum movimento, as forças resultantes da sua movimentação eram detetadas pelo sensor de força presente no suporte do

robô *follower* (Figura 5.20).

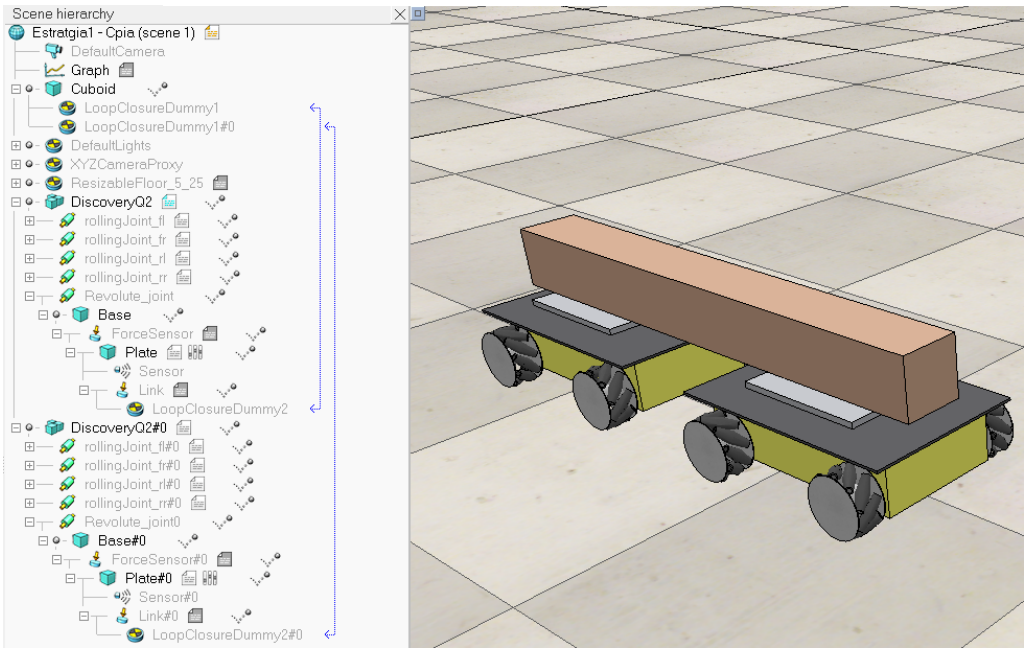


Figura 5.19: Representação do sistema de co-transporte com a carga e da conexão criada entre os robôs e o objeto

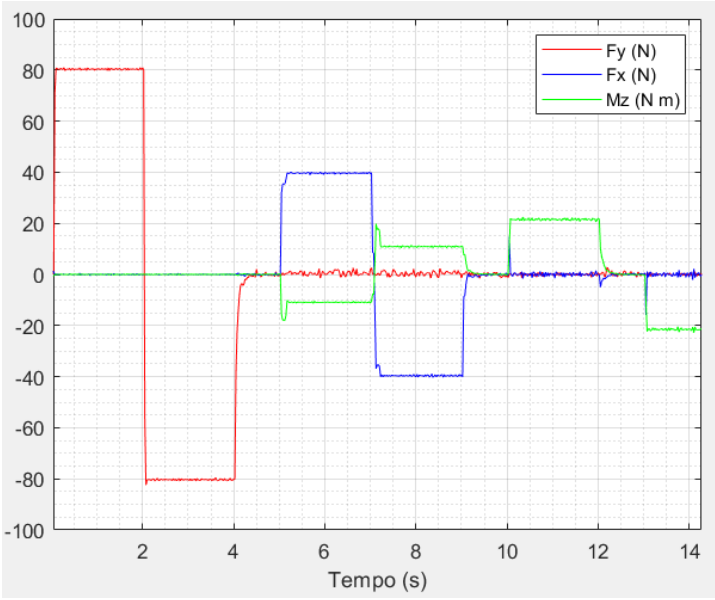


Figura 5.20: Forças detetadas pelo sensor de força/binário multiaxial no robô *follower* durante movimentos longitudinais, transversais e rotacionais (com o motor de física ODE)

Capítulo 6

Implementação e Simulação das Estratégias de Co-transporte

Este capítulo contém toda a informação acerca da implementação e simulação das estratégias de co-transporte que foram exploradas. Numa primeira fase, será introduzido o conceito que permite a existência do comportamento cooperativo entre os elementos do sistema. Posteriormente, serão abordados os modos de funcionamento implementados para o robô leader. Posto isto, parte-se para a descrição do estudo realizado acerca das diferentes estratégias de co-transporte, que irão diferir consoante a abordagem tomada na construção dos suportes, como se viu no capítulo anterior, e também na arquitetura de controlo do robô follower. Cada estratégia será analisada, de modo a poderem ser apontadas todas as suas vantagens e desvantagens. Por fim, será ainda realizada uma comparação entre as estratégias no seguimento de um mesmo caminho por parte do robô leader.

6.1 Princípio de Funcionamento do Sistema

O conceito pelo qual se rege a coordenação entre os elementos do sistema em estudo pode ser facilmente entendido quando comparado com o exemplo mostrado na Figura 6.1. Imagine-se que duas pessoas necessitam de trabalhar em conjunto para transportar um objeto pesado ou de grande dimensão, sendo que apenas uma delas (neste caso a pessoa A) sabe a posição final em que deve ser largado o objeto. Se a pessoa A se movimentar na direção representada por D irá gerar uma força de interação (F_A) com o objeto. Por sua vez, a pessoa B irá sentir uma força de igual magnitude (F_B) no local em que interage com a carga. Para que o transporte seja bem sucedido, a pessoa B deverá então movimentar-se de forma a anular a força de contacto. Repare-se também que todo o transporte poderá ser

feito sem que exista comunicação direta entre os intervenientes, sendo a pessoa B guiada apenas pelas forças sentidas no local em que agarra o objeto.

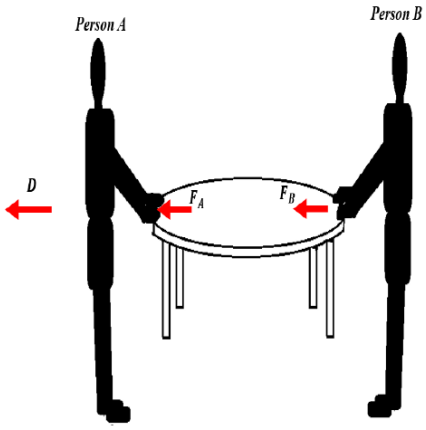


Figura 6.1: Cooperação entre humanos no transporte de um objeto [79]

O sistema em estudo neste trabalho apresenta um comportamento análogo ao que foi explicado anteriormente. O robô *leader* irá conduzir o sistema até ao destino, enquanto que o *follower* terá de possuir a capacidade de anular as forças que por ele são exercidas na carga. Deste modo, os robôs terão arquiteturas de controlo distintas, mas serão igualmente responsáveis pelo sucesso da operação de transporte.

A análise realizada ao sistema terá um maior ênfase na arquitetura de controlo do robô *follower* já que, independentemente de quais forem as capacidades e o modo de funcionamento do *leader*, o *follower* é que terá de agir de forma submissa e de modo a contrariar as forças medidas pelo sensor presente no seu suporte.

Neste trabalho serão exploradas diferentes estratégias que irão mudar consoante a estrutura mecânica do suporte (como se viu no capítulo anterior) e a arquitetura de controlo do *follower*. O objetivo será avaliar qual a estratégia que melhor complementa as características omnidirecionais do sistema e, ao mesmo tempo, que garanta a maior segurança da carga, eliminando ou reduzindo o máximo possível a comunicação direta entre os dois elementos do sistema. Embora apresentem diferenças, todas elas terão por base um controlador que possibilita realizar o controlo da força como se mostra na Figura 6.2, em que C é o controlador de força e G a planta a controlar (neste caso será a plataforma omnidirecional). O controlador irá minimizar a função de erro de força (e_f=F_{ref}-F_{med}) através da atuação dos motores da plataforma, para que a força de interação com o objeto tome o valor desejado (0 N).

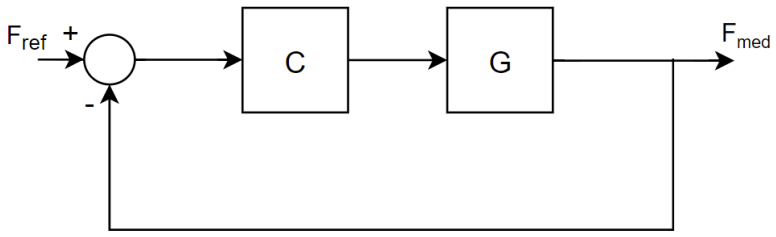


Figura 6.2: Controlo explícito da força

6.2 Arquitetura de Controlo do Robô Leader

No âmbito deste trabalho, tal como já foi referido, o modo como o robô *leader* é controlado não é a questão mais importante nem o foco do estudo. Porém, para se poder simular e analisar as várias estratégias foi necessário existir alguma forma de controlar o movimento e a velocidade deste robô.

Além de *scripts* do mesmo género do que se encontra na Secção A.2 do Anexo A (explicado no Capítulo 5), em que são atribuídas diferentes velocidades ao robô consoante o tempo que decorre na simulação, foi ainda implementado o controlo manual do robô a partir do teclado do computador e o seguimento de caminhos criados no simulador. De seguida demonstra-se como tal foi conseguido.

6.2.1 Controlo Manual

O controlo manual do modelo do robô *leader* pode ser feito através do *script* presente na Secção A.4 do Anexo A.

Cada um dos *scripts* utilizados numa cena do CoppeliaSim possui uma fila de mensagens associada (no máximo 64 mensagens). Se o utilizador tiver a janela principal do programa focada poderá enviar mensagens para o simulador através do teclado do computador. As mensagens recebidas na fila de mensagens associada ao *script* podem depois ser obtidas (e simultâneamente removidas da fila) através da função *sim.getSimulatorMessage()*, como se mostra no seguinte extrato do *script*:

```
1                                (...)
2
3     while sim.getSimulationState()~=
      sim.simulation_advancing_abouttostop do
4
5     message,auxiliaryData=sim.getSimulatorMessage()
6     --print(auxiliaryData)
7     while message~=-1 do
8         if (message==sim.message_keypress) then
```

```
9               if (auxiliaryData[1]==119) then --> W
10                  if (vy<0.65) then
11                      vy=vy+0.001
12                  end
13
14                                  (...)
```

Uma vez obtida a mensagem, será possível identificar a tecla que foi pressionada e aplicar a ação de controlo desejada. Esta funcionalidade do simulador permitiu implementar o *script* de controlo manual para que fosse possível, de forma simples, ajustar as velocidades do robô *leader*.

6.2.2 Seguimento de Caminhos

De modo a verificar-se o comportamento do sistema aquando da realização de movimentos mais complexos, bem como para se poder efetuar uma comparação direta entre as diferentes estratégias analisadas, foi necessário implementar o seguimento de caminhos para o robô *leader*. Na Figura 6.3 mostra-se o diagrama de blocos que explica a arquitetura de controlo do robô para este modo de funcionamento. Assinalados a vermelho encontram-se o controlo da velocidade dos atuadores e o sistema de localização. Estas partes do controlo do *leader* não são abordadas já que na simulação existe a garantia de que a velocidade dos atuadores é a desejada e também é possível saber qual a posição absoluta do robô no mundo virtual. No entanto, na aplicação real, estas duas partes do controlo teriam de ser incluídas. No que diz respeito à localização do robô esta poderia ser estimada a partir da odometria e do laser Hokuyo que este possui.

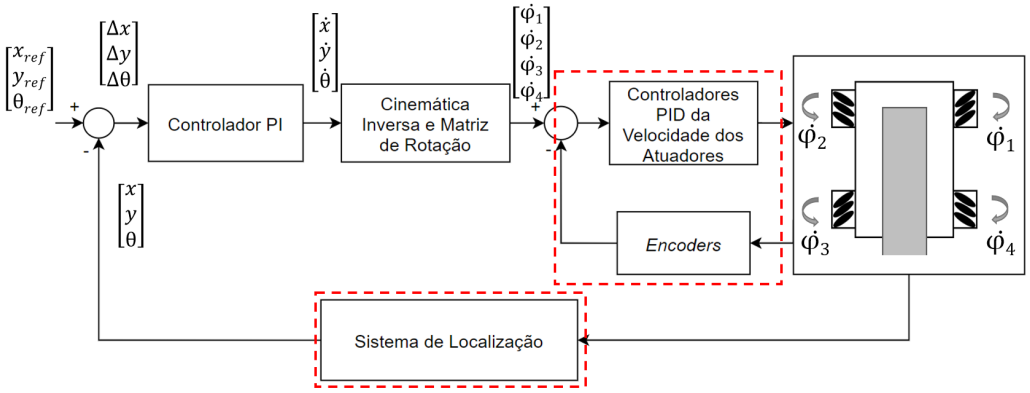


Figura 6.3: Diagrama de blocos da arquitetura de controlo do robô *leader* para o seguimento de trajetórias

Para o robô poder efetuar o seguimento de caminhos foi implementado um controlador PI que tem como entradas o erro entre a posição de referência (x_{ref}, y_{ref} e θ_{ref}) e a posição atual do robô (x, y e θ), e como saídas as velocidades do

robô em relação ao referencial global ($\dot{x}$, $\dot{y}$ e $\dot{\theta}$). Para transformar as velocidades do referencial global para o referencial do robô é usada a matriz de rotação, como foi visto na equação 3.11 no Capítulo 3, sendo depois possível calcular a velocidade de cada um dos atuadores ($\dot{\varphi_1}$, $\dot{\varphi_2}$, $\dot{\varphi_3}$ e $\dot{\varphi_4}$) a partir da cinemática inversa da plataforma. Na Figura 6.4 mostra-se também o fluxograma da função *sysCall_actuation()* presente no *script* do seguidor de caminhos e que é executada a cada passo da simulação permitindo implementar o funcionamento descrito anteriormente.

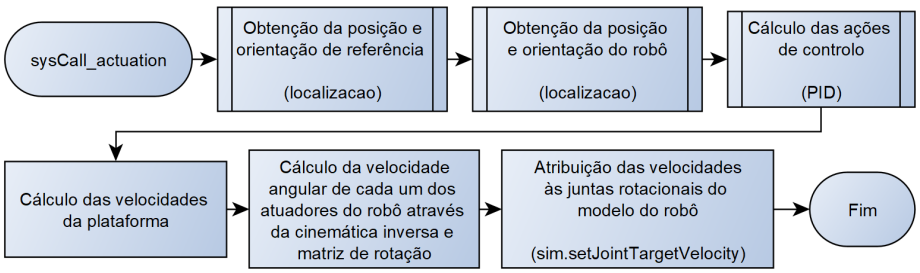


Figura 6.4: Fluxograma da função de *callback* responsável pela implementação da arquitetura de controlo do seguidor de caminhos

Tal como pode ser constatado a partir da análise da Figura 6.4, a posição de referência e a posição atual do robô são obtidas a partir da função *localizacao()*. O fluxograma desta função pode ser visualizado na Figura 6.5, à esquerda, e a sua função é, através das funções do simulador *sim.getObjectPosition()* e *sim.getObjectOrientation()*, obter a posição e orientação de um objeto no mundo virtual. Depois disto é realizada a conversão dos ângulos Euler para um ângulo entre 0 e 360°, através da função *convers()*. Por fim, é retornado um vetor com as coordenadas e orientação do objeto no espaço bidimensional (x, y e θ).

O erro entre as posições será depois alvo da ação de controlo por parte do controlador PI, que é implementado com a função *PID()* cujo funcionamento pode ser observado no fluxograma da Figura 6.5, à direita. Depois de se realizarem os cálculos necessários as velocidades poderão ser aplicadas a cada uma das juntas através da função do simulador *sim.setJointTargetVelocity()*.

A função *PID()* implementada (bem como outras funções que lhe estão associadas) e o *script* relativo ao seguidor de caminhos omnidirecional podem ser vistos nas Secções A.5 e A.6 do anexo A, respetivamente.

Os caminhos que se pretende que o robô percorra podem ser facilmente gerados no CoppeliaSim através da criação e edição de *paths* (Figura 6.6) ou então importados em formato *Comma-Separated Values* (CSV). Recorrendo ao modo de edição o utilizador poderá definir várias propriedades do caminho, tais como introduzir pontos de controlo, definir o seu posicionamento e orientação no mundo

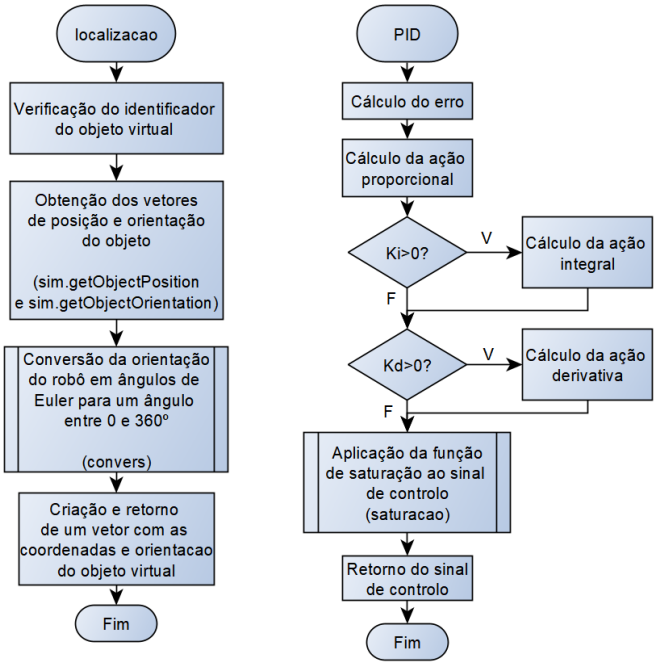


Figura 6.5: Fluxogramas das funções localização e PID

virtual, realizar a interpolação linear entre pontos de controlo através de curvas de Bézier, etc.

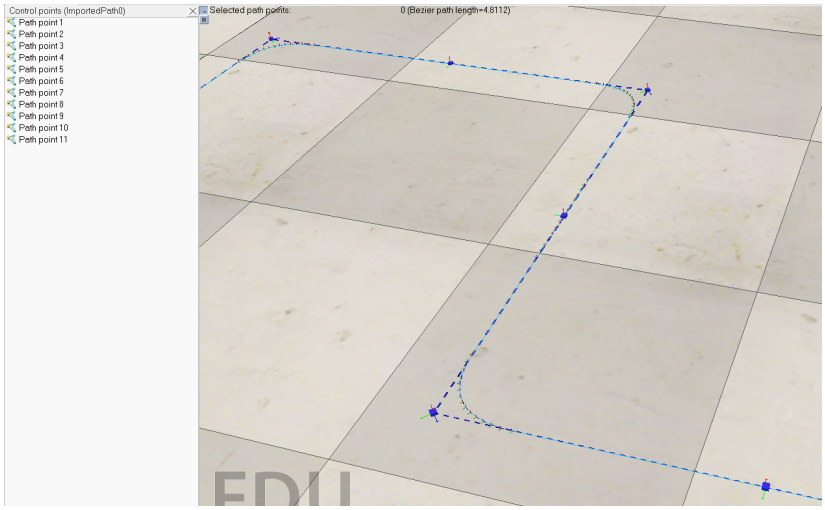


Figura 6.6: Criação e edição de *paths* no CoppeliaSim

Com o caminho já criado, adiciona-se à cena um *Dummy* cuja posição, durante a simulação, é incrementada ao longo do caminho. A posição e orientação deste *Dummy* serão as referências para o movimento do robô. Relativamente à

localização do robô, também foi feito algo deste género tendo sido colocado um novo *Dummy* no centro do chassis da plataforma que, conforme a simulação decorre, irá permitir saber qual a posição e orientação do robô no mundo virtual. A Figura 6.7 ilustra aquilo que foi explicado.

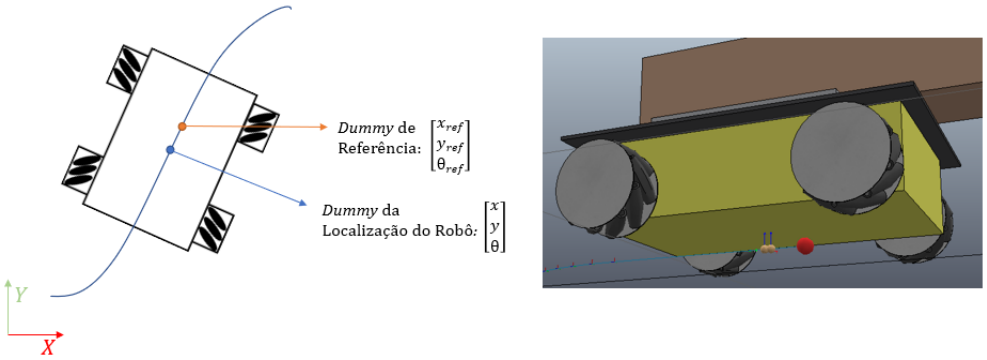


Figura 6.7: Exemplificação dos *Dummies* de referência e da localização do robô

Nas Figuras 6.8 e 6.9 apresentam-se os gráficos XY que mostram os resultados obtidos no seguimento de um caminho de teste depois de se ter ajustado manualmente os ganhos do controlador, através de várias simulações. É de realçar que na experiência da Figura 6.8 o robô moveu-se a uma velocidade de 0,1 m/s e na da Figura 6.9 a 0,3 m/s.

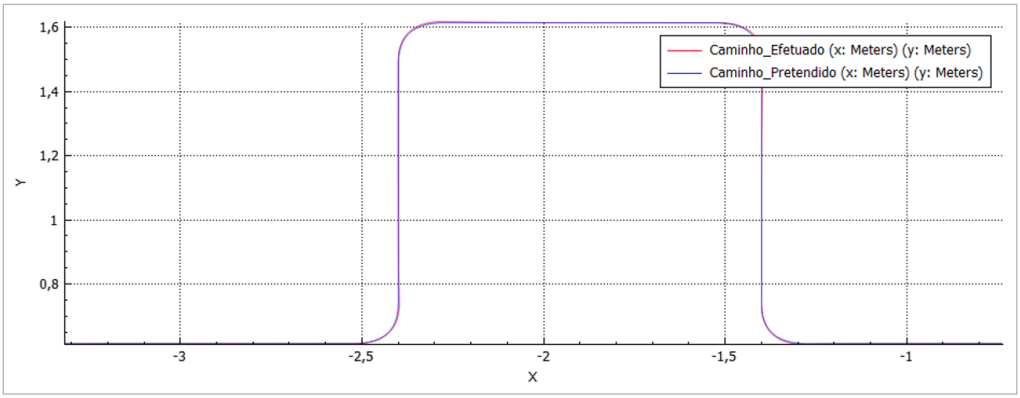


Figura 6.8: Seguimento de um caminho de teste a 0,1 m/s

Observando-se os gráficos conclui-se que o erro aumentará conforme aumenta a velocidade do robô. Contudo, e tal como é mostrado na Figura 6.9, o erro máximo no seguimento do caminho apresentado tem um valor de apenas 0,031 m, sendo que o robô já se movimenta a uma velocidade considerável (0,3 m/s) e as curvas do caminho são bastante acentuadas. Uma vez que não se pretende que

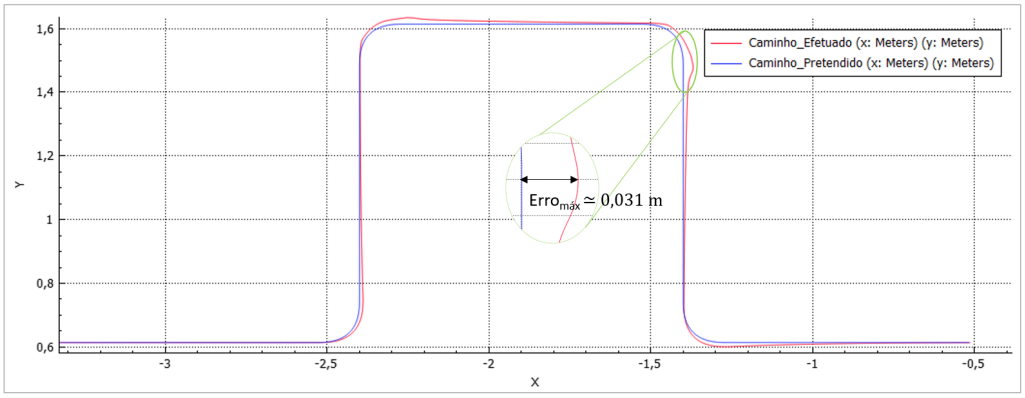


Figura 6.9: Seguimento de um caminho de teste a 0,3 m/s

o sistema se mova a altas velocidades considerou-se que este resultado já era bastante satisfatório.

6.3 Estudo das Estratégias de Co-transporte

Seguidamente explica-se todo o estudo realizado acerca das diferentes estratégias idealizadas para o funcionamento do sistema, consoante as diferenças na estrutura mecânica dos suportes. Convém salientar que o estudo incidirá sobre a arquitetura de controlo do robô *follower*, sendo que os resultados que serão mostrados foram obtidos com o robô *leader* a ser controlado pelos métodos referidos anteriormente.

6.3.1 Controlo da Força com Controladores PID

Numa primeira fase foi necessário verificar qual o tipo de controlador (P, PI ou PID) que permitia o melhor controlo das forças. Para isso realizou-se uma experiência na qual, após decorrido o primeiro segundo da simulação, o robô *leader* inicia um movimento longitudinal (direção do eixo Y, representada na Figura 6.10 a verde). Este deverá acelerar até atingir a velocidade desejada (neste caso, 0,1 m/s) e mantê-la até ao sexto segundo da simulação, no qual deverá parar. Com isto recriou-se o problema mostrado na Figura 6.1, mas no plano do sistema. Esta experiência permitiu verificar o comportamento do *follower* consoante os diferentes controladores que iam sendo usados.

6.3.1.1 Identificação do Ganho e Período de Oscilação Crítico

O primeiro passo foi identificar o ganho e período de oscilação críticos de forma a ser possível realizar a sintonia de cada um dos controladores usados. Assim sendo,

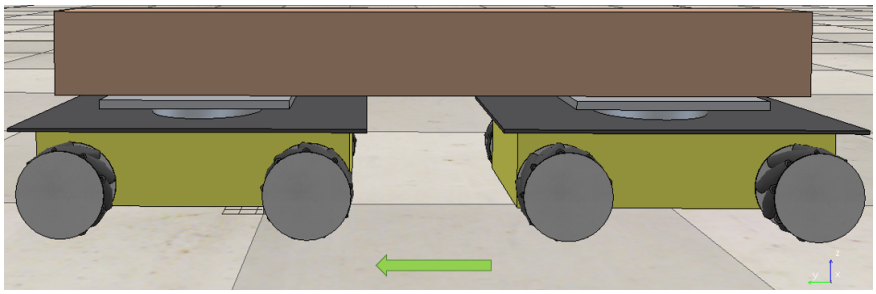


Figura 6.10: Exemplificação de um movimento longitudinal

consideraram-se os ganhos K_i e K_d nulos e aumentou-se K_p até se verificar uma resposta oscilatória como a que se encontra na Figura 6.11.

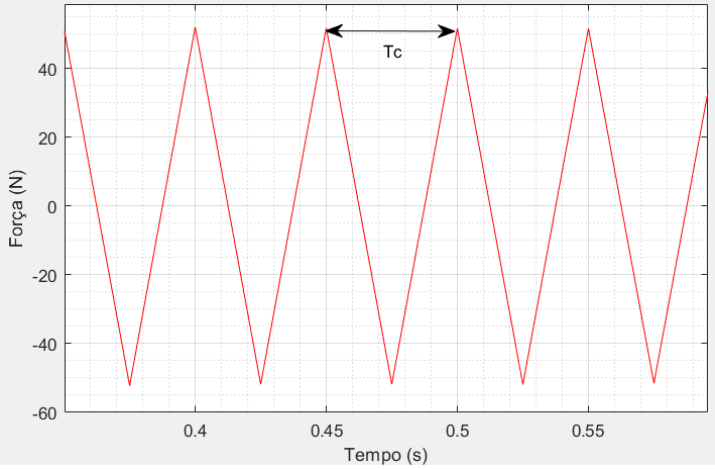


Figura 6.11: Identificação do ganho e período de oscilação crítico para aplicação do método de Ziegler-Nichols

Com isto chegou-se aos valores de K_c e T_c (0,03 e 0,05, respetivamente, para um incremento da simulação de 25 ms) com os quais, recorrendo à Tabela 3.3, foi possível aplicar o método da sintonia de Ziegler-Nichols.

6.3.1.2 Controlador P

A primeira experiência foi feita com um controlador proporcional. Depois de realizada a sintonia do mesmo, verificou-se o comportamento do robô *follower* quando o *leader* se movimentava. Na Figura 6.12 é possível observar as forças detetadas pelo sensor presente no suporte.

Através da análise do gráfico pode-se concluir que um controlador proporcional não é suficiente para se conseguir minimizar as forças de contacto. Por mais que se variasse o ganho K_p o robô não conseguia anular a força imposta na carga

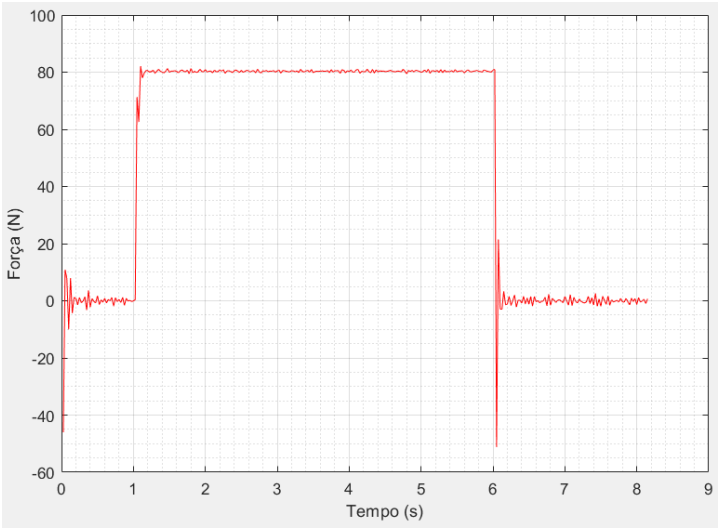


Figura 6.12: Resultados obtidos com o controlador P (K_p=0,0150)

pelo *leader*, que, por sua vez, estava a "arrastá-lo" juntamente com a carga, algo que se traduz na força constante medida pelo sensor de força durante o período em que o movimento se realizou.

6.3.1.3 Controlador PI

Depois da experiência com o controlador P foi introduzido o termo integral no controlador e calcularam-se os parâmetros K_p e K_i através do método de sintonia explicado. Posteriormente, realizou-se a mesma experiência tendo-se obtido o resultado mostrado no gráfico da Figura 6.13.

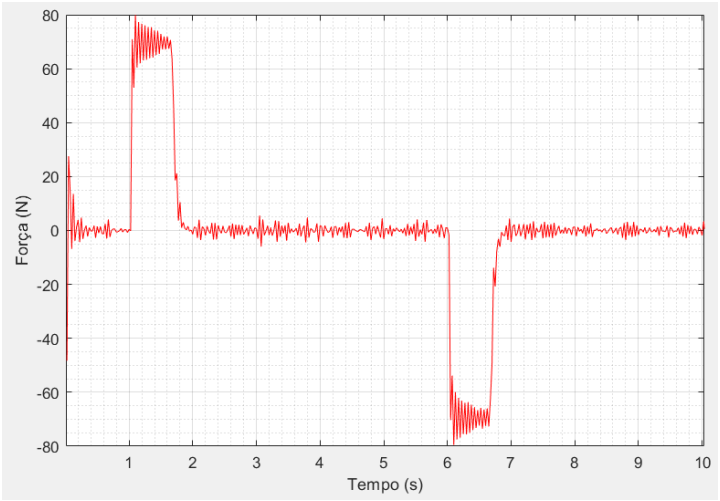


Figura 6.13: Resultados obtidos com o controlador PI (K_p=0,0135; K_i=0,3240)

Com a introdução do termo integral o erro em regime permanente é eliminado, pelo que é possível para o robô *follower* acompanhar o *leader* à mesma velocidade durante grande parte do movimento. Porém, é também de referir que no início há uma força de tensão que é exercida na carga, visto que o *leader* se encontra em aceleração e o *follower* demora para o conseguir acompanhar. Por outro lado, acontece o contrário quando o *leader* trava bruscamente, sendo exercidas forças de compressão na carga.

6.3.1.4 Controlador PID

Por último introduziu-se a componente derivativa no controlador e realizou-se o mesmo teste. O resultado pode ser visto na Figura 6.14.

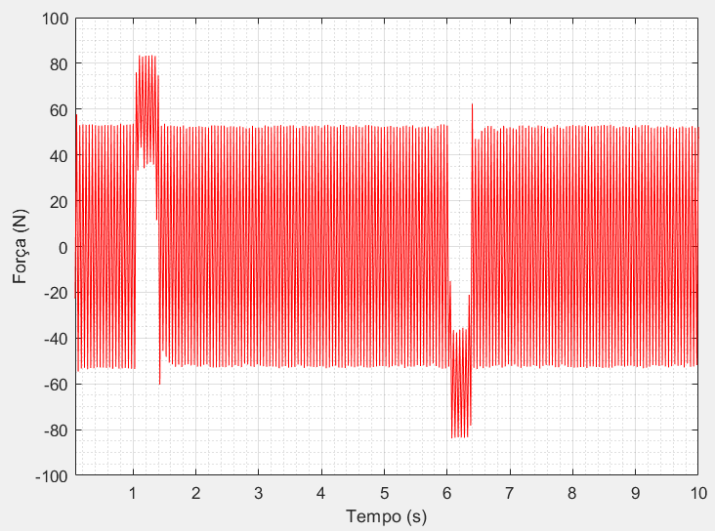


Figura 6.14: Resultados obtidos com o controlador PID (K_p=0,0180; K_i=0,7200; K_d=0,0001)

A introdução da componente derivativa, por mais baixo que seja o valor de K_d, torna impossível a obtenção de uma boa resposta ao movimento do *leader*. Isto deve-se ao ruído inerente às leituras provenientes do sensor de força, como se pode ver na Figura 6.15.

6.3.1.5 Conclusão

Depois de analisada a performance do sistema com os três controladores, chegou-se à conclusão que o melhor seria utilizar um controlador PI para realizar a minimização das forças de contacto entre o robô *follower* e a carga. Convém salientar que os resultados apresentados anteriormente foram obtidos com o método de sintonia e são apenas uma primeira aproximação, sendo que foi possível,

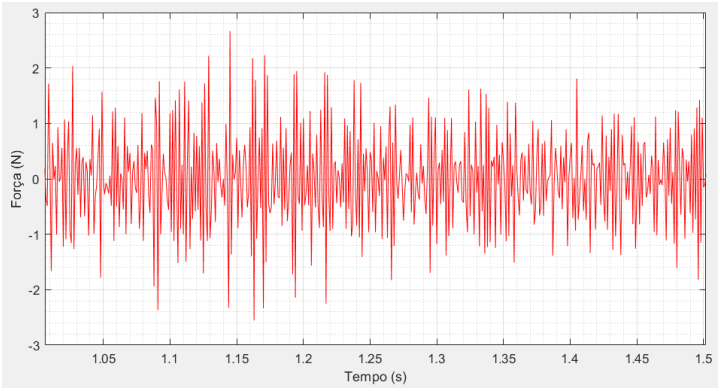


Figura 6.15: Leituras em vazio do sensor de força

através do ajuste manual dos ganhos, obter uma resposta melhor, como se pode comparar na Figura 6.16.

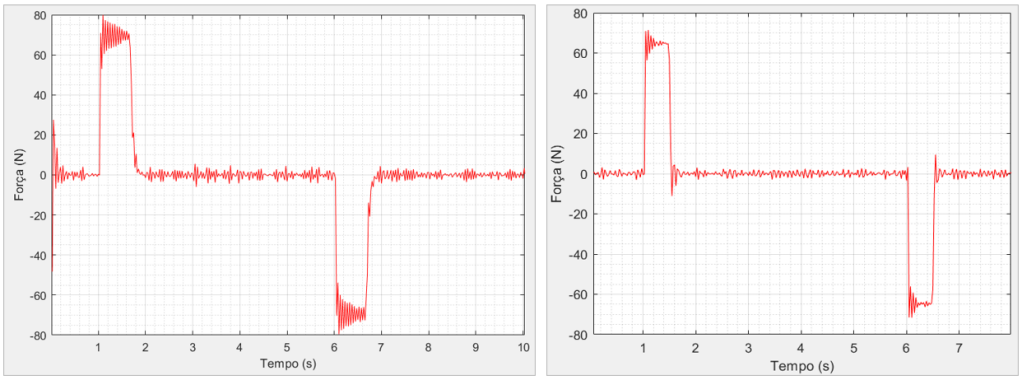


Figura 6.16: Comparação das forças exercidas na carga com o controlador PI sintonizado com o método de Ziegler-Nichols e depois dos ajustes manuais, respetivamente

Com esta experiência também foi possível apurar que, conforme se diminuía o período de amostragem era possível obter respostas mais rápidas sem um aumento considerável das oscilações, tal como se pretende mostrar com a Figura 6.17, em que se compara uma resposta obtida com um incremento de tempo da simulação de 10 ms e 1 ms, respetivamente.

Adicionalmente, verifica-se que a rapidez da resposta não é a única característica que melhora. É também notada uma grande diminuição das forças exercidas na carga durante o arranque e travagem do *leader*. Embora seja uma diferença considerável, não foi possível realizar todas as simulações do sistema com o período de amostragem de 1 ms, uma vez que a simulação ficava bastante lenta e, por vezes, acabava mesmo por bloquear. Por esta razão, decidiu-se usar o período

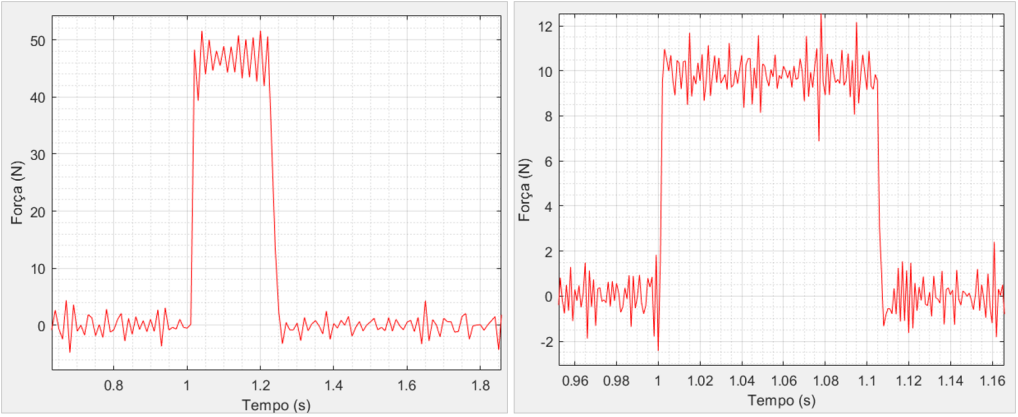


Figura 6.17: Comparação de duas respostas do sistema com um período de amostragem de 10 e 1 ms, respetivamente

de 10 ms. Contudo, é possível verificar que existiria uma melhoria notória caso se usasse um valor mais baixo.

A última questão que tem de ser referida diz respeito à aceleração/desaceleração e travagem do sistema. É possível notar que enquanto o robô *leader* acelera até atingir a velocidade desejada a carga vai-se encontrar sobre tensão, visto que o *follower* demora a reagir. O contrário acontece quando se dá uma desaceleração ou travagem, sendo que a carga será sujeita a forças de compressão. Sendo assim, quanto mais tempo o robô estiver em aceleração/desaceleração para atingir a velocidade desejada, maior será também o período de tempo em que existem forças de tensão/compressão a atuar na carga. No que diz respeito às travagens, quanto maior for a velocidade no momento em que o *leader* trava bruscamente, maior será o período de tempo em que são exercidas forças de compressão e a magnitude das mesmas.

Com a mesma experiência que vinha a ser feita é possível entender melhor o que foi explicado. Na Figura 6.18 é possível ver, à esquerda, a resposta do sistema quando o *leader* acelera ao máximo até atingir a velocidade de 0,1 m/s e depois trava bruscamente; à direita, o conceito da experiência é o mesmo, porém, a velocidade de referência para o *leader* é de 0,45 m/s. Facilmente se identifica que na experiência da direita existe um maior período de tempo em que a carga está sobre tensão ou compressão. Com isto conclui-se que é benéfico para a carga e para o sistema que não hajam grandes acelerações/desacelerações e travagens bruscas, devendo-se privilegiar um movimento mais "suavizado" e de baixa velocidade. Na Figura 6.19 mostra-se que se o arranque e a travagem do *leader* não for tão brusca (gráfico da direita), será preciso mais tempo para o último atingir a velocidade desejada mas as forças exercidas na carga terão uma menor magnitude.

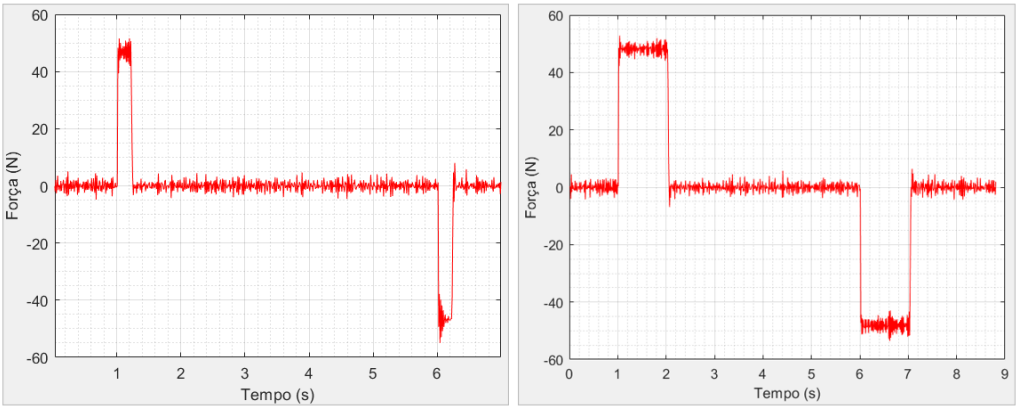


Figura 6.18: Comparação das forças exercidas na carga para movimentos a diferentes velocidades, 0,1 m/s e 0,45 m/s, respetivamente

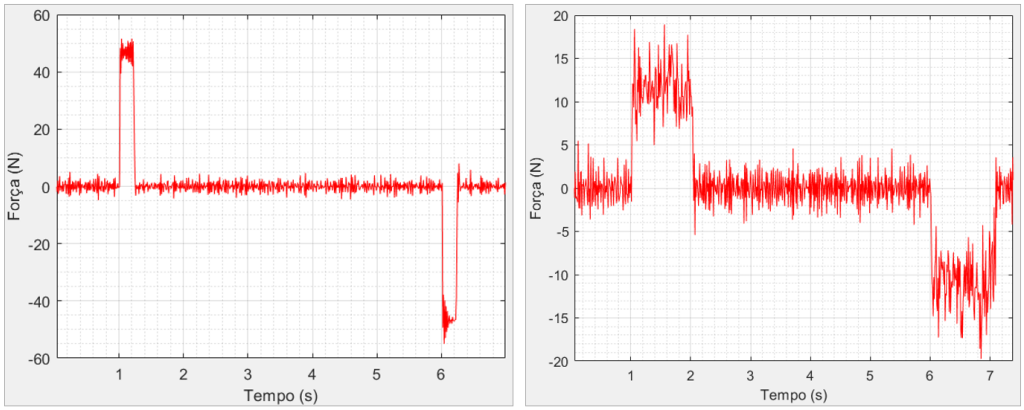


Figura 6.19: Comparação das forças exercidas na carga para o mesmo movimento, mas com diferentes acelerações por parte do robô *leader* (0,48 m/s^2 e 0,1 m/s^2, respetivamente)

Retiradas as conclusões acerca de qual seria o melhor controlador para realizar o controlo das forças exercidas na carga, bem como de alguns comportamentos do sistema derivados da sua utilização, passou-se para o estudo das diferentes estratégias de co-transporte.

6.3.2 Estratégia 1 - Utilização de um Sensor de Força/Binário Multiaxial

Na Figura 6.20 apresenta-se o diagrama de blocos que explica a arquitetura de controlo idealizada para o *follower*, numa primeira fase, caso fosse usado um sensor de força/binário multiaxial no seu suporte. Este sensor permitiria ao robô medir todas as forças e binários no espaço tridimensional, mas a monitorização

das três componentes representadas - F_X, F_Y e M_Z - seria suficiente para que o robô, através do conceito da minimização das forças de contacto explicado anteriormente, conseguisse reagir a todos os tipos de movimentos feitos pelo *leader*. Na Figura 6.20 assinalou-se também a vermelho o controlo da velocidade dos atuadores. Embora seja necessário na implementação da aplicação real, este controlo não será abordado pelas razões que já foram mencionadas.

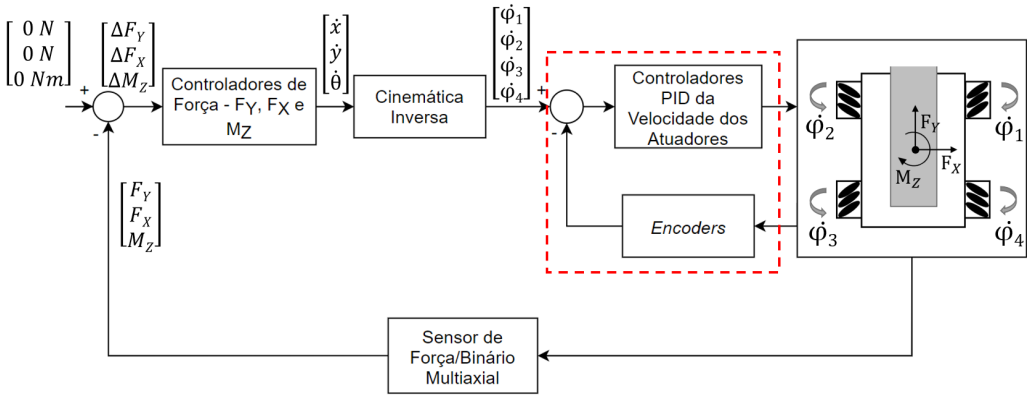


Figura 6.20: Diagrama de blocos do controlo do *follower* com a utilização do sensor de força/binário multiaxial

Desta forma, o controlo do robô *follower* pode ser feito se se implementar o controlo PI, como foi explicado anteriormente, para cada componente de força. O valor da velocidade calculado pelo controlador será depois convertido na velocidade angular de cada uma das rodas através da cinemática inversa da plataforma.

O *script non-threaded* responsável pela implementação do esquema de controlo da Figura 6.20 pode ser analisado na Secção A.7 do Anexo A. Na Figura 6.21 mostra-se também o fluxograma da sua função *sysCall_actuation()* que é executada todos os passos da simulação.

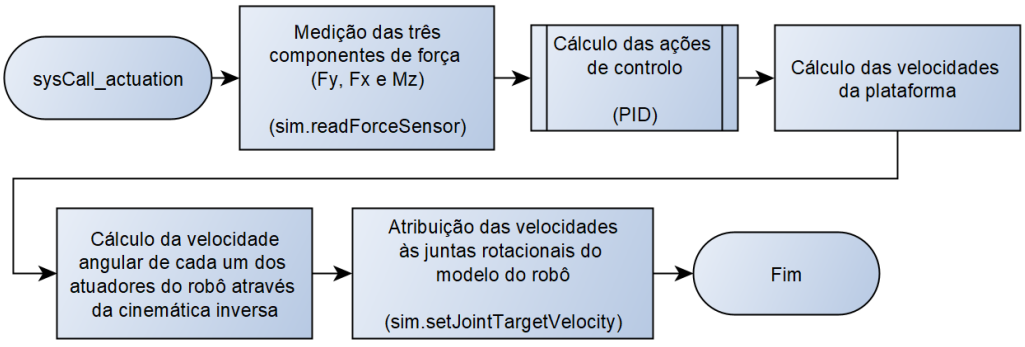


Figura 6.21: Fluxograma da função de *callback* responsável pela implementação da arquitetura de controlo do *follower* para a estratégia 1

Os valores da força medidos pelo sensor podem ser obtidos através da função do simulador *sim.readForceSensor()*, sendo o resto do fluxograma idêntico ao que já foi analisado para o *leader*.

6.3.2.1 Análise da Estratégia Idealizada

Implementado o controlador (através do mesmo processo explicado na Subsecção 6.3.1), foi possível testar o sistema. De seguida, nas Figuras 6.22, 6.23 e 6.24, e de maneira idêntica ao que foi feito anteriormente, mostram-se as forças medidas pelo sensor de força do robô *follower* durante os diferentes movimentos efetuados pelo *leader* (longitudinais, transversais e rotacionais, respetivamente).

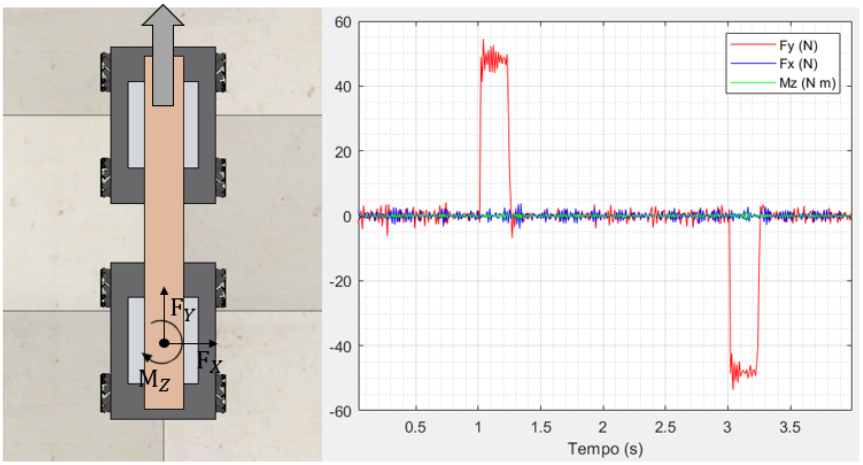


Figura 6.22: Forças medidas pelo sensor de força/binário multiaxial do *follower* durante um movimento longitudinal do robô *leader* (V_Y= 0,1 m/s)

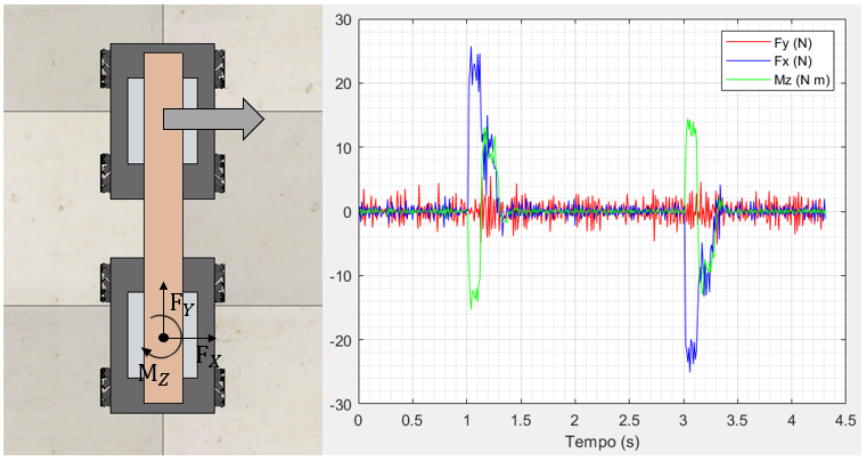


Figura 6.23: Forças medidas pelo sensor de força/binário multiaxial do *follower* durante um movimento transversal do robô *leader* (V_X= 0,1 m/s)

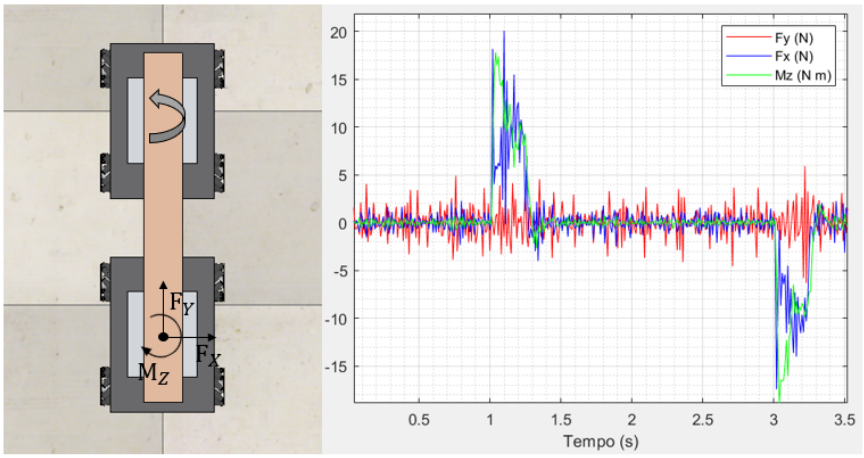


Figura 6.24: Forças medidas pelo sensor de força/binário multiaxial do *follower* durante um movimento rotacional do robô *leader* (W= 0,1 rad/s)

Analisando-se os gráficos, é possível reparar que o robô *follower* consegue seguir o *leader* minimizando as forças exercidas na carga, seja qual for o tipo de movimento realizado. Também é importante referir que poderão ser combinados vários tipos de movimentos, por exemplo, um linear com um rotacional (Figura 6.25), e o comportamento será o mesmo, a única coisa que irá diferir serão as forças a que o objeto está sujeito na fração de tempo que a resposta do sistema demora a atingir o regime permanente.

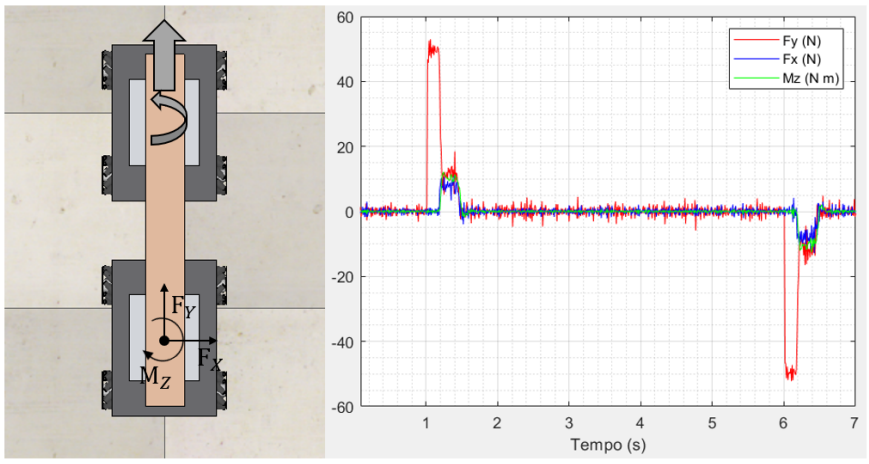


Figura 6.25: Exemplificação da combinação de um movimento linear com um rotacional e forças medidas pelo sensor de força/binário multiaxial durante o movimento (V_Y= 0,1 m/s e W= 0,1 rad/s)

Esta estratégia tem diversas particularidades que a tornam interessante. Uma delas passa por haver a garantia de que os robôs mantêm sempre a formação,

ou seja, a posição e orientação das plataformas em relação ao objeto e entre si mantém-se igual durante todo o movimento, dependendo apenas da forma como a carga é agarrada antes de ser iniciado o transporte. Isto é algo que poderá facilitar a tarefa do planeamento da trajetória do sistema. Inclusive, os robôs não terão de se encontrar alinhados com a carga para realizarem o transporte. Na Figura 6.26, pretende-se exemplificar o que foi referido. Mesmo não estando alinhados, os robôs realizam os movimentos, contudo, salienta-se que as forças a que a carga está sujeita são também diferentes. Por comparação, para um movimento linear do *leader* em que as plataformas estão alinhadas com o objeto, a força a que a carga está sujeita só tem uma componente (F_y), enquanto que no exemplo mostrado com os robôs desalinhados existem três componentes de força (F_y, F_x e M_z).

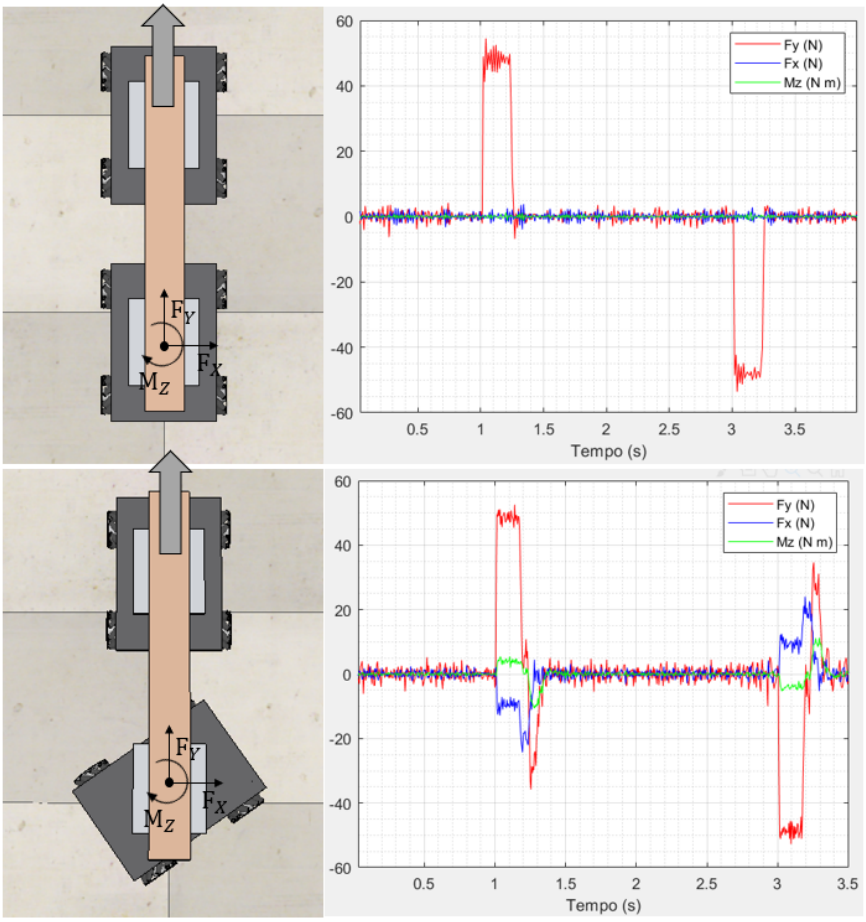


Figura 6.26: Exemplificação do desalinhamento dos robôs e forças medidas pelo sensor do *follower* durante a realização de um movimento linear (V_Y= 0,1 m/s)

Por outro lado, levando em conta o sistema de dois robôs e o tipo de controlo que se realiza, esta estratégia apresenta desvantagens que faz com que não seja

a mais ideal. O principal inconveniente passa pelas forças exercidas na carga. É possível reparar, nas figuras que foram sendo apresentadas, que todos os movimentos irão fazer com que a carga fique, mesmo que por muito curtos períodos de tempo, sobre a ação de forças. Estas forças poderão ser de tração/compressão, cisalhamento e torção. Além disso, pode-se ver que em movimentos simples como os transversais e rotacionais (Figuras 6.23 e 6.24) a carga está simultâneamente sujeita a forças de cisalhamento e torção. Este problema agrava-se se o *leader* realizar movimentos mais complexos, como a curva à direita que é mostrada na Figura 6.27. Ao percorrer o trajeto apresentado, se o *leader* ajustar constantemente a sua orientação pela da trajetória, a carga estará também mais tempo sujeita a forças de cisalhamento e torção.

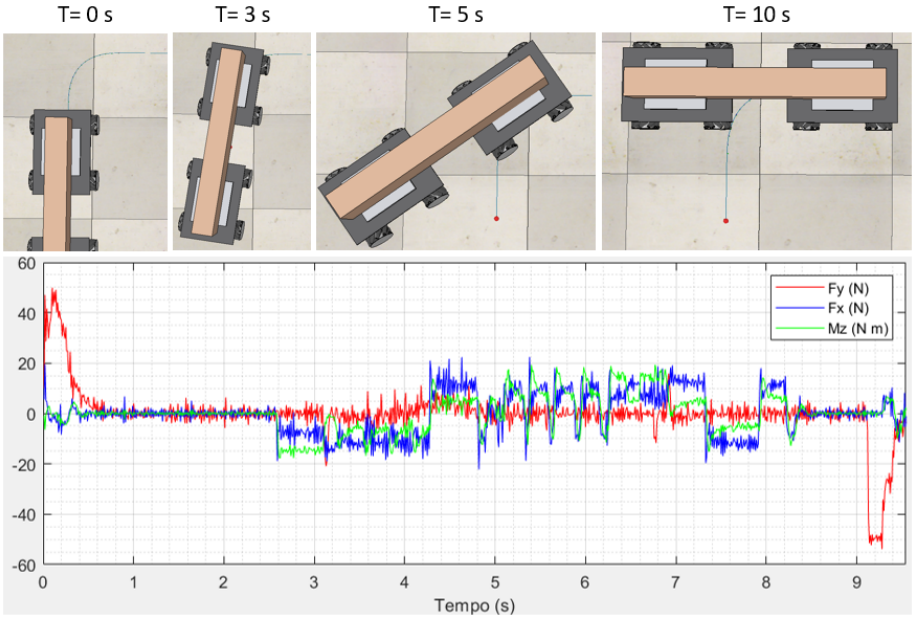


Figura 6.27: Exemplificação do comportamento do sistema e forças medidas pelo sensor do *follower* durante a realização de uma curva por parte do *leader* a uma velocidade de 0,1 m/s

Outra questão tem que ver com o fenómeno que está representado na Figura 6.28. Analisando-se a Figura 6.23 é possível ver que as forças de cisalhamento exercidas na carga têm uma magnitude consideravelmente menor que as de tração/compressão (Figura 6.22), isto deve-se ao facto das rodas omnidirecionais do robô *follower* derraparem um pouco no início dos movimentos transversais. Ora, este fenómeno, associado ao período de tempo que o robô leva a reagir às forças impostas na carga, fazem com que, caso seja realizado um movimento mais brusco, o robô *leader* se desoriente, tal como se vê na Figura 6.28. É de referir que na travagem poderá acontecer o mesmo fenómeno, devido à inércia do robô *follower* e o período de tempo que este leva para reagir. Também no final de

um movimento rotacional, e caso haja uma travagem repentina, será notado um comportamento análogo ao que foi explicado. O erro aumentará consoante a velocidade do movimento e conforme os robôs estiverem mais distanciados entre si. O facto de os robôs estarem conectados rigidamente à carga faz com que haja uma grande influência do *follower* em qualquer movimento que o *leader* execute.

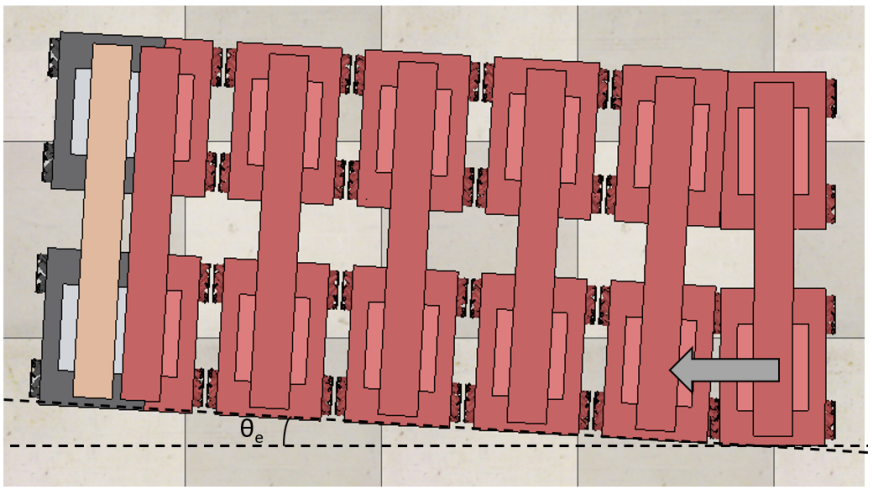


Figura 6.28: Erro associado à derrapagem das rodas e demora na reação às forças por parte do robô *follower*

As desvantagens apresentadas fazem com que esta estratégia em que se usa um sensor de força/binário multiaxial na construção dos robôs não seja a melhor. Apesar de possuir certas vantagens que poderiam ser exploradas, num sistema como o que se apresenta neste trabalho e com os algoritmos de controlo que são usados, considera-se que esta não seria a melhor opção.

6.3.3 Estratégia 2 - Utilização de uma Célula de Carga e Sensor de Posição Angular

A Figura 6.29 mostra o diagrama de blocos da arquitetura de controlo pensada para o robô *follower*, caso fossem utilizados a célula de carga e o sensor de posição angular. Através destes dois sensores o robô será capaz de estimar o movimento do *leader* e de se orientar em relação ao objeto transportado.

Para que o robô *follower* conseguisse minimizar as forças de contacto com a carga implementou-se apenas um controlador PI, de seguida explica-se o raciocínio seguido, recorrendo à Figura 6.30.

A força de contacto (F) é medida pela célula de carga instalada no suporte, sendo o erro entre a medida e a referência (0 N) transformado pelo controlador no sinal de controlo, que nada mais é que a velocidade (V) a que a plataforma tem de se deslocar para poder minimizar as forças medidas pela célula de carga.

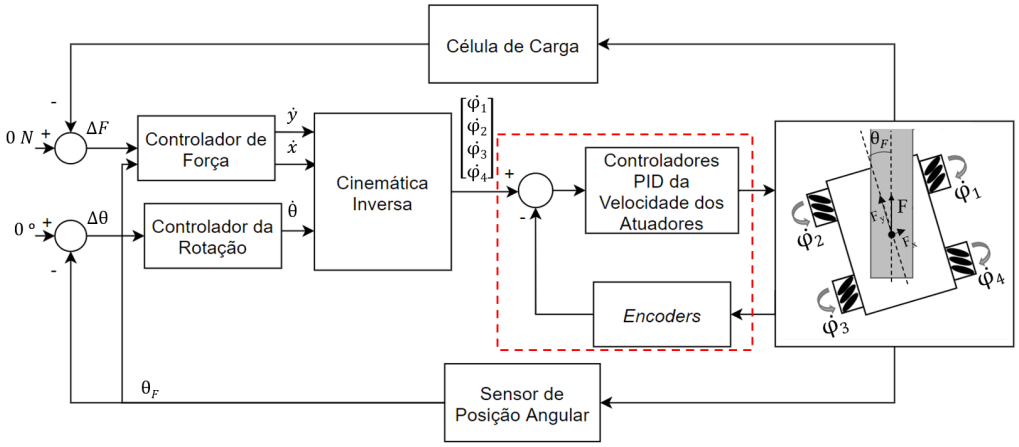


Figura 6.29: Diagrama de blocos do controlo do *follower* com a utilização da célula de carga e o sensor de posição angular

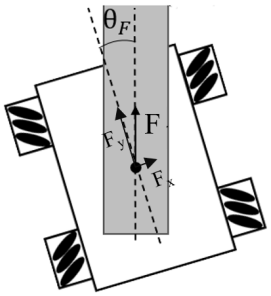


Figura 6.30: Componentes F_Y e F_x da força medida pela célula de carga

Através da posição da junta (θ_F), medida pelo sensor de posição angular, será possível calcular as componentes da velocidade ($\dot{y}$ e $\dot{x}$) que permitem minimizar as componentes da força de contacto (F_Y e F_X) entre o robô e a carga, através das equações que se seguem:

$$\dot{y} = -V \cdot sin((\pi/2) - \theta_F) \tag{6.1}$$

$$\dot{x} = V \cdot cos((\pi/2) - \theta_F) \tag{6.2}$$

Este controlador seria o suficiente para o robô *follower* cooperar com o *leader* no transporte da carga, uma vez que não possui restrições holonómicas, porém, poderá ser de interesse que o primeiro se alinhe com a carga durante o seu transporte. Para isso, foi implementado um outro controlador PI que tem como função controlar o alinhamento do robô com a carga, ou seja, manter θ_F a zero.

O *script* responsável pela implementação do esquema de controlo da Figura 6.29 pode ser analisado na Secção A.8 do Anexo A. Na Figura 6.31 mostra-se também o fluxograma da função *sysCall_actuation()*.

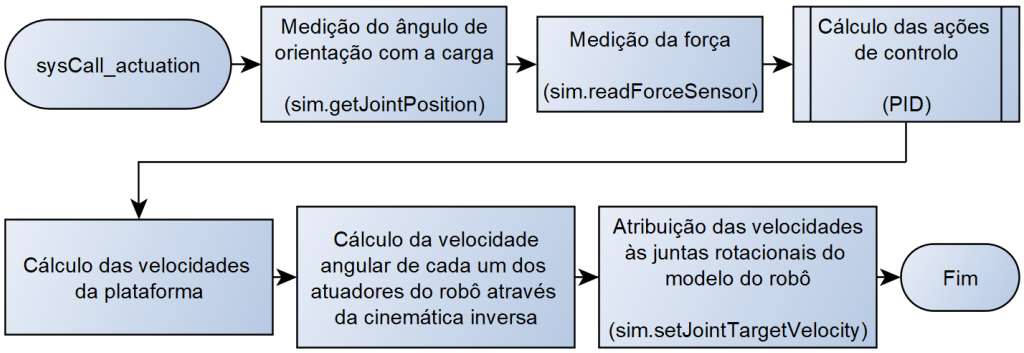


Figura 6.31: Fluxograma da função de *callback* responsável pela implementação da arquitetura de controlo do *follower* para a estratégia 2

A posição da junta (θ_F) pode ser medida com a função *sim.getJointPosition()*, sendo que as restantes funções já foram abordadas.

6.3.3.1 Análise da Estratégia Idealizada

Tal como foi feito para a outra estratégia, nas Figuras 6.32, 6.33 e 6.34 são mostradas as forças medidas pelo sensor do robô *follower* durante a realização de diferentes movimentos por parte do *leader*. Nesta estratégia não faz sentido abordar os movimentos rotacionais do *leader*, já que o seu suporte também possuirá um grau de liberdade, o que fará com que este possa rodar sobre si.

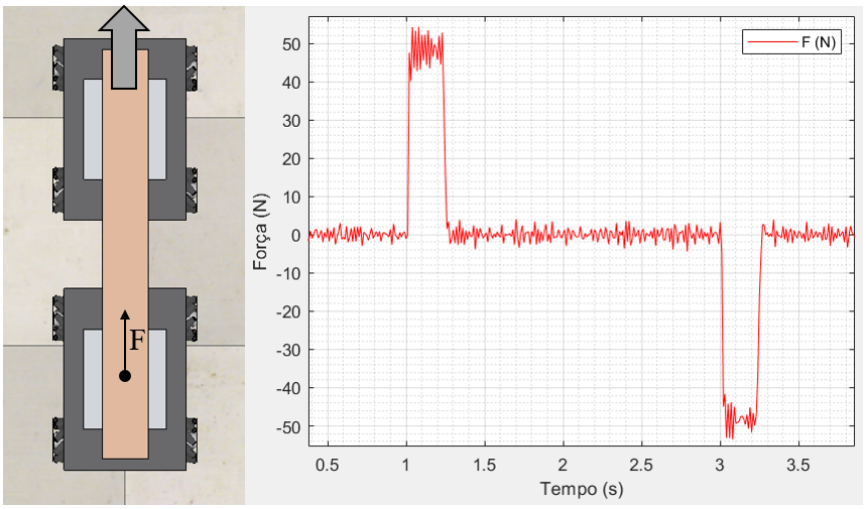


Figura 6.32: Forças medidas pelo sensor de força do robô *follower* durante um movimento longitudinal (V_Y= 0,1 m/s)

Através da análise do gráfico da Figura 6.32, e como seria de esperar, observa-se que no caso de um movimento longitudinal por parte do *leader* o comporta-

mento do sistema será igual ao que tem vindo a ser visto para este tipo de movimento. A principal mudança ocorre aquando da realização de movimentos característicos das plataformas omnidirecionais, como os transversais ou oblíquos, presentes nas Figuras 6.33 e 6.34.

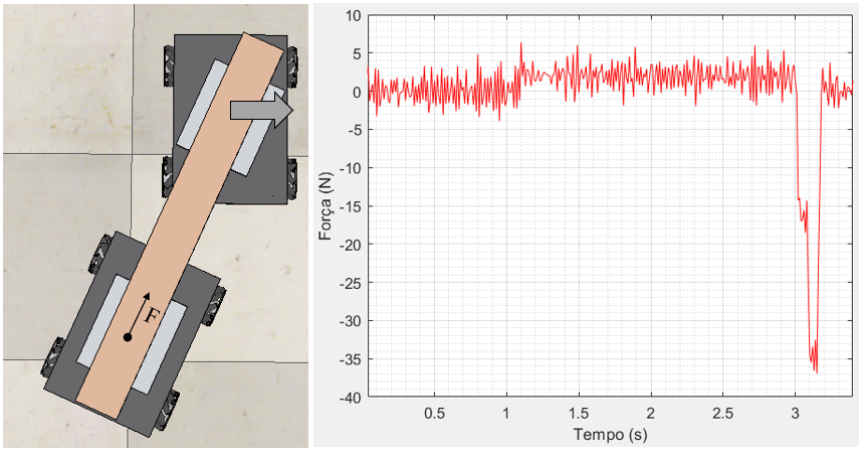


Figura 6.33: Forças medidas pelo sensor de força do robô *follower* durante um movimento transversal do *leader* (V_X= 0,1 m/s)

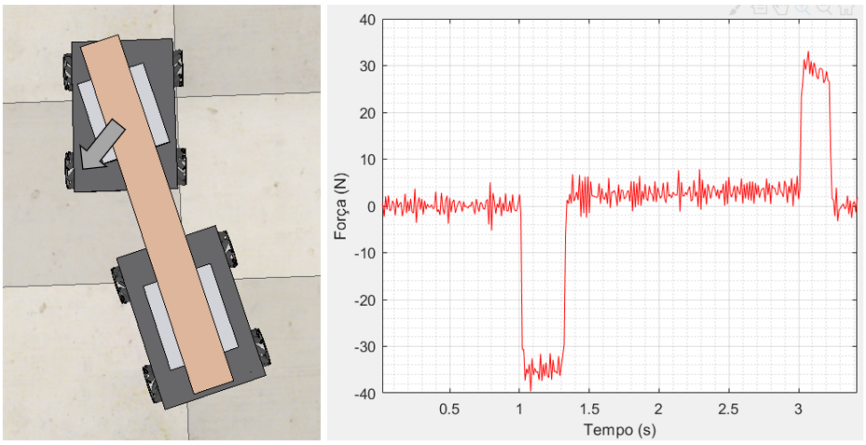


Figura 6.34: Forças medidas pelo sensor de força do robô *follower* durante um movimento oblíquo do *leader* (V_Y= 0,1 m/s e V_X= 0,1 m/s)

A primeira conclusão que pode ser retirada é que deixam de existir forças de cisalhamento e torção sobre a carga, passando esta a estar sujeita apenas a forças de tração e compressão como se mostra nas figuras. Outra das conclusões a que se chega é que, quando o *leader* realiza movimentos característicos de plataformas omnidirecionais, existe uma força de tensão ou compressão que irá ser exercida sobre a carga (devido ao tempo que o *follower* demora a reagir às forças do *leader*). Esta força será tanto maior quanto a velocidade do movimento. Contudo, se este

for feito a baixas velocidades, e não de forma brusca, a força é bastante baixa ou até mesmo nula. O movimento representado pela Figura 6.33 foi realizado a uma velocidade de 0,1 m/s, de seguida pode-se comparar os resultados obtidos para o mesmo movimento, mas a uma velocidade mais baixa (0,05 m/s - Figura 6.35, à esquerda) e outra mais alta (0,2 m/s - Figura 6.35, à direita), de maneira a exemplificar-se o que foi explicado.

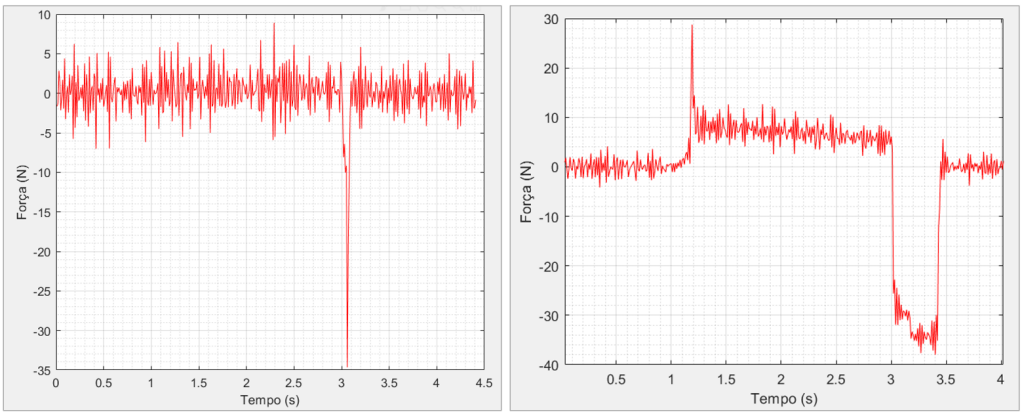


Figura 6.35: Forças medidas pelo sensor de força do robô *follower* durante um movimento transversal do *leader* a 0,05 m/s e 0,2 m/s, respetivamente

Pela análise dos resultados averigua-se que, caso o *follower* tenha tempo para reagir, as forças exercidas na carga serão mínimas durante o transporte (sem ter em consideração os arranques e travagens, cujas forças resultantes também poderão ser minimizadas se não existirem grandes acelerações/desacelerações ou travagens bruscas).

De seguida, na Figura 6.36, mostra-se também a experiência da curva como foi feito para a estratégia anterior. Repare-se que o objeto fica sujeito a muito menos forças (apenas de tração e compressão, que, excetuando os arranques e travagens, são de baixa magnitude), contudo, os robôs não mantêm a formação como se viu anteriormente.

Feita a análise dos resultados é possível concluir acerca da estratégia em si. A grande vantagem da sua utilização passa por existirem menos forças a atuar na carga. As forças de cisalhamento e torção deixam de existir, passando a notar-se apenas a existência de forças de tensão/compressão, que podem ser reduzidas se o *leader* não efetuar movimentos abruptos. A existência de menos forças significa que a carga será transportada de forma mais segura, sem se danificar (isto é algo que também irá depender das suas características como, por exemplo, a rigidez). Outro ponto forte desta abordagem passa por o equipamento usado na implementação do sistema ser mais acessível, o que significaria menos custos associados à sua implementação.

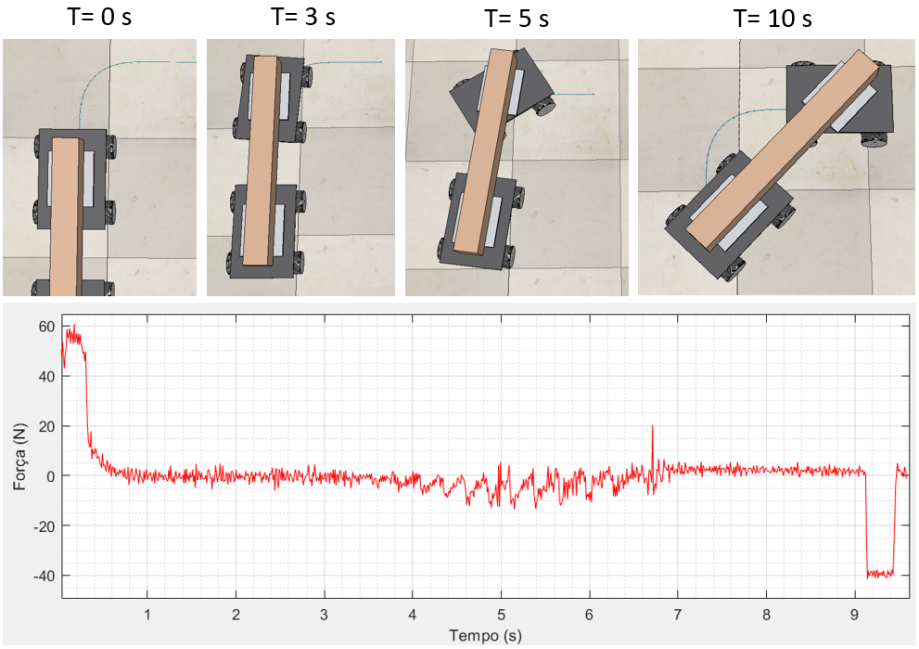


Figura 6.36: Exemplificação do comportamento do sistema e forças medidas pelo sensor do *follower* durante a realização de uma curva por parte do *leader* a uma velocidade de 0,1 m/s

Por outro lado, existe uma lacuna nesta estratégia quando comparada com a que foi vista anteriormente, que passa pelo facto de os robôs não manterem a formação. Por esta razão, não é possível tirar o máximo de proveito das capacidades omnidirecionais do sistema, sendo que grande parte do tempo irá funcionar como um sistema de robôs com tração diferencial.

6.3.4 Estratégia 3 - Comunicação Direta de um Parâmetro entre os Robôs

De modo a aproveitar o melhor de ambas as estratégias já apresentadas, passou-se ao estudo de uma terceira estratégia, que pode ser considerada como um modo de funcionamento alternativo da que foi analisada anteriormente.

Um dos principais objetivos na implementação do sistema era que este realizasse o transporte da carga sem que existisse comunicação entre os intervenientes. Com a segunda estratégia analisada viu-se que isto era possível com um custo do sistema menor e, ao mesmo tempo, sem que a carga estivesse sujeita a tantas forças como acontecia na primeira estratégia. Porém, existe um desaproveitamento das capacidades omnidirecionais do sistema. Por esta razão, procurou-se encontrar uma alternativa que permitisse superar esta desvantagem. A solução encontrada passa por existir a comunicação explícita de um parâmetro (orien-

tação relativa à carga) por parte do *leader* ao *follower*. O diagrama de blocos mostrado na Figura 6.37 representa a arquitetura de controlo do *follower* que foi idealizada, caso isto acontecesse. Seguidamente, é também explicado o raciocínio que permitiu chegar a este esquema de controlo.

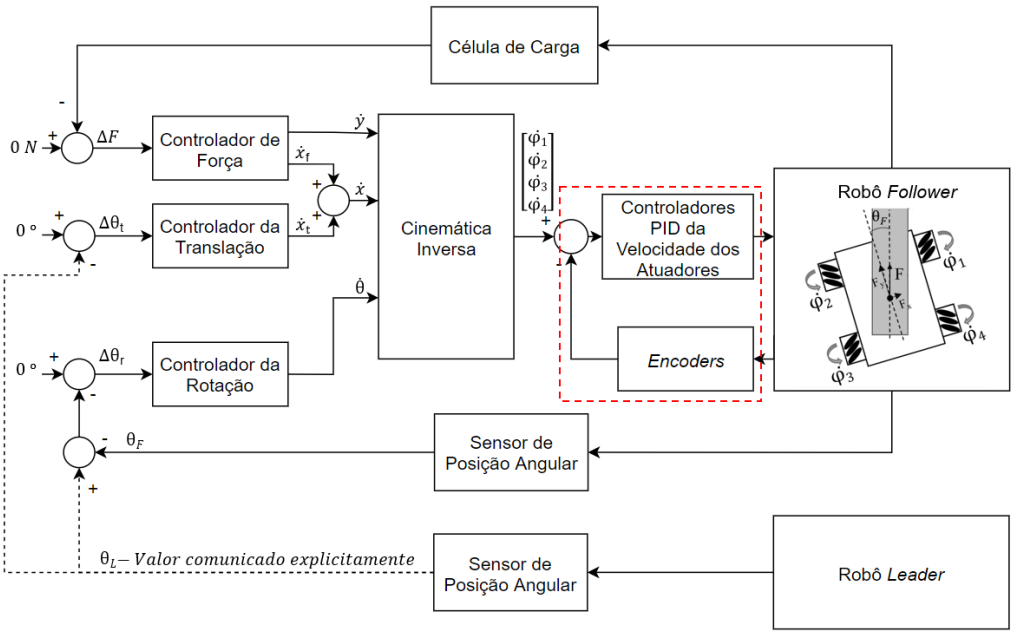


Figura 6.37: Diagrama de blocos do controlo do *follower* relativo à terceira estratégia explorada

Se o robô *follower* conhecer a orientação do *leader* em relação à carga (θ_L) poderá calcular a diferença entre esta e a sua orientação (θ_F). Por sua vez, a diferença será minimizada pelo controlador de rotação. Isto fará com que o robô *follower* tome a mesma orientação do *leader* em relação à carga, como está ilustrado na Figura 6.38.

Ainda assim, isto não é suficiente para que o sistema de dois robôs ganhe um comportamento idêntico ao que se via na estratégia 1. Para isso foi implementado o controlador de translação, cuja função pode ser vista na Figura 6.39 e será fazer com que o *follower* realize um movimento transversal de modo a alinhar-se com o *leader*. Por outras palavras, o robô *follower* tem que se alinhar com o *leader* e ao mesmo tempo alinhar o *leader* com a carga. Se isto for garantido, o funcionamento do sistema será análogo ao que foi visto na estratégia 1.

O *script* que diz respeito à implementação do esquema de controlo da Figura 6.37 pode ser analisado na Secção A.9 do Anexo A. Na Figura 6.40 mostra-se também o fluxograma da função *sysCall_actuation()*.

Uma vez que tem de existir comunicação entre os robôs, também no simula-

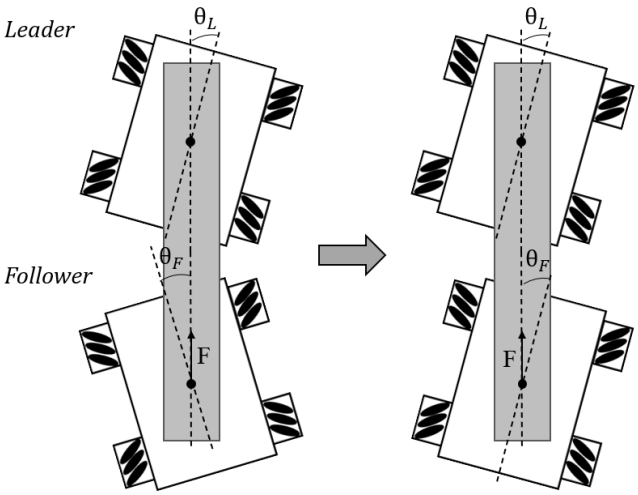


Figura 6.38: Ângulo de orientação dos robôs com a carga e ação do controlador de rotação

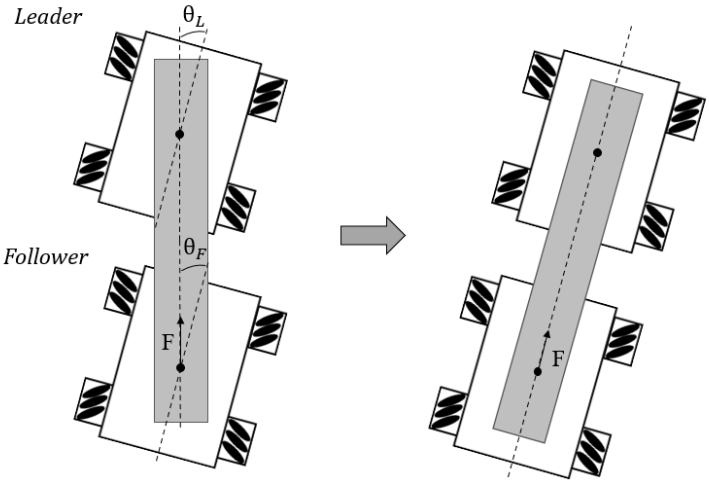


Figura 6.39: Ação do controlador de translação

dor foi necessário implementar a comunicação entre *scripts*. O valor do ângulo (θ_L) medido no *script* do *leader* necessita de ser enviado para o *follower*. Isto requer que no *script* do *leader* seja acrescentado o seguinte extrato de código (a localização exata pode ser visto na Secção A.6 do Anexo A):

```
1    angulo=sim.getJointPosition(pos)
2    sim.setFloatSignal("angulo",angulo)
```

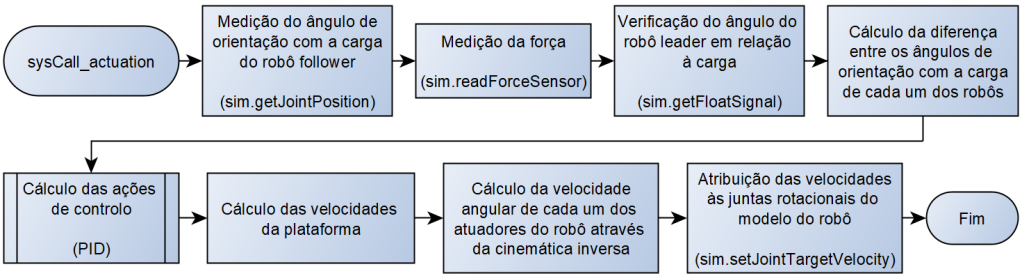


Figura 6.40: Fluxograma da função de *callback* responsável pela implementação da arquitetura de controlo do *follower* para a estratégia 3

No caso do *script* do *follower* deverá ser usada a função *sim.getFloatSignal()*, tal como se aponta na Figura 6.37, de modo a poder receber o valor comunicado.

6.3.4.1 Análise da Estratégia Idealizada

Depois de implementado o controlo do robô *follower* verificou-se o comportamento do sistema aquando da realização de vários movimentos por parte do *leader*. Para os movimentos longitudinais o comportamento foi o mesmo que foi visto nos casos anteriores, pelo que não vale a pena voltar a referenciar.

No caso dos movimentos transversais e rotacionais é possível ver o comportamento do sistema, e comparar com os resultados obtidos na estratégia 1, nas Figuras 6.41 e 6.42.

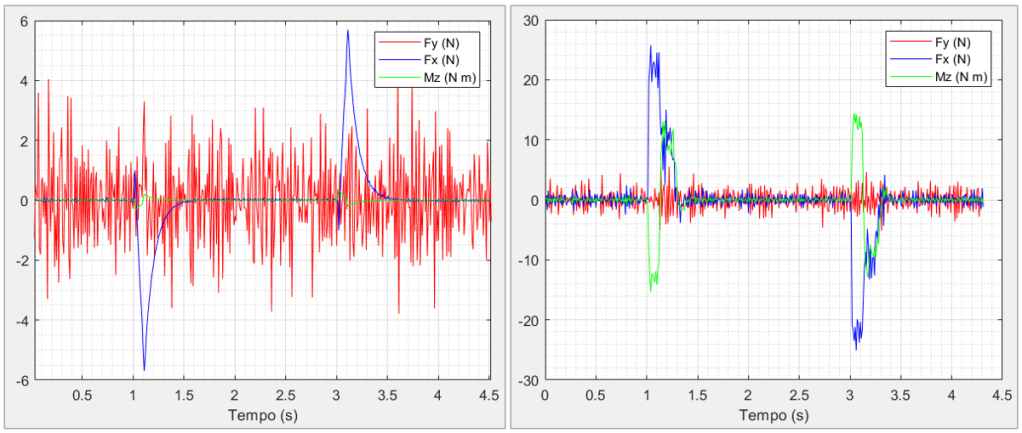


Figura 6.41: Forças medidas pelo sensor de força do robô *follower* durante um movimento transversal do *leader* (V_X= 0,1 m/s)

Para verificar se ainda existiam forças de torção e cisalhamento sobre a carga mediram-se as três componentes de força como se fosse utilizado o sensor de força/binário multiaxial (de modo a comparar diretamente aos resultados da estratégia 1), porém, convém salientar que nesta estratégia as forças são monitorizadas

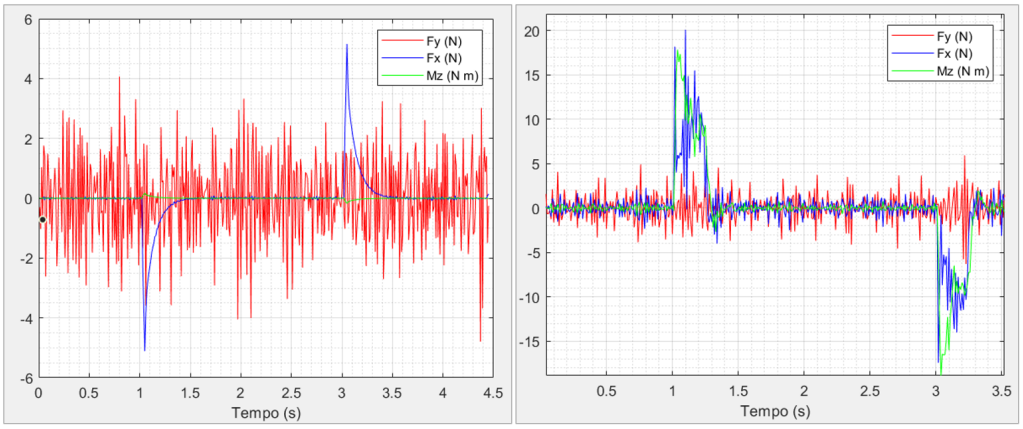


Figura 6.42: Forças medidas pelo sensor de força do robô *follower* durante um movimento rotacional do *leader* (W= 0,1 rad/s)

apenas num eixo e segundo a orientação da carga (o único valor medido seria F_Y), dado que é utilizada a célula de carga.

Com a introdução da comunicação direta do ângulo de orientação do robô *leader*, tornou-se possível para o sistema manter a formação enquanto este realizava diferentes movimentos. Foi também possível verificar que existiam algumas forças de cisalhamento no início e fim dos movimentos, porém, estas não têm qualquer relação com as que se viam na primeira estratégia e são de muito baixa magnitude. De seguida mostram-se também os resultados durante a realização da experiência da curva como se fez para as estratégias anteriores (Figura 6.43).

Observando-se o gráfico da Figura 6.43 verifica-se que, tal como acontecia durante a execução dos movimentos mais simples, são detetadas forças de baixa magnitude no eixo X. Isto significa que poderá não ser possível eliminar completamente a sua existência, porém, se o sistema não realizar movimentos bruscos e grandes variações de velocidade durante movimentos transversais e rotacionais, será garantido que as forças de cisalhamento impostas na carga são praticamente nulas.

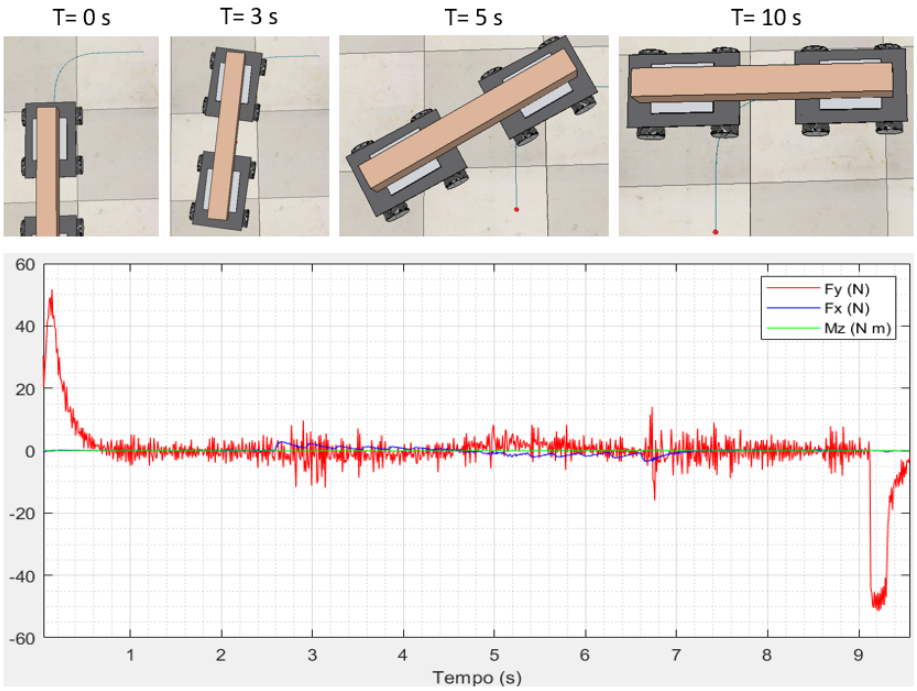


Figura 6.43: Exemplificação do comportamento do sistema e forças medidas pelo sensor do *follower* durante a realização de uma curva por parte do *leader* a uma velocidade de 0,1 m/s

6.3.5 Comparação do Desempenho das Estratégias no Seguimento de um Caminho

Considere-se o caminho mostrado na Figura 6.44. Os gráficos das Figuras 6.45, 6.46 e 6.47 dizem respeito às forças detetadas pelo sensor do robô *follower* (para a estratégia 1, 2 e 3, respetivamente) conforme este coopera com o *leader* no transporte da carga ao longo do caminho apresentado. É de salientar que se voltou a medir todas as componentes de força em qualquer estratégia, no entanto, na segunda e na terceira, a célula de carga só permitiria medir a componente F_Y. A única razão pela qual se medem todas as componentes de força passa por haver a necessidade de se saber se efetivamente as forças de torção e cisalhamento deixam de existir nestas estratégias. Convém referenciar que os resultados apresentados são para uma velocidade do *leader* de 0,1 m/s. Além das forças de interação na carga, são também apresentados os gráficos XY que mostram o caminho pretendido e percorrido pelo *leader*, bem como o caminho percorrido pelo *follower* (Figuras 6.48, 6.49 e 6.50).

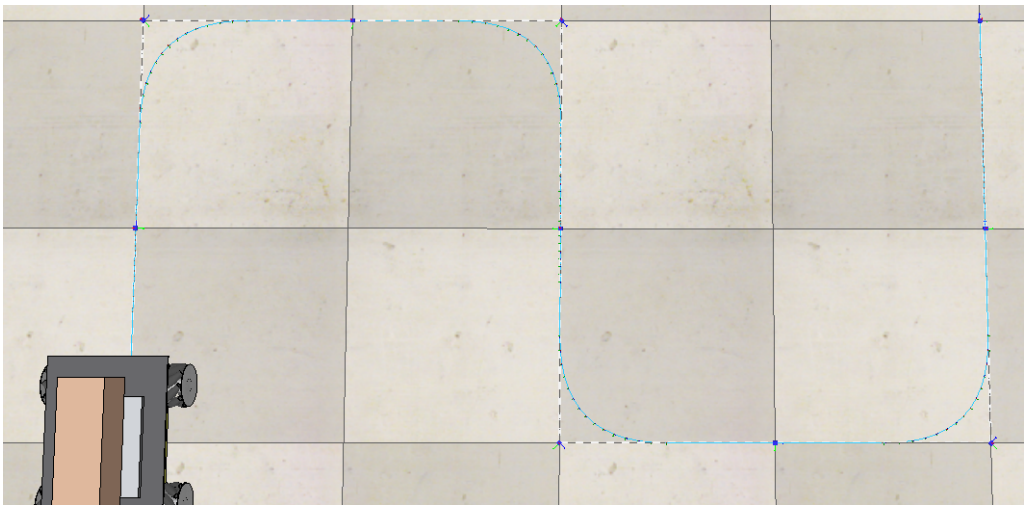


Figura 6.44: Caminho implementado para a realizar a comparação entre estratégias

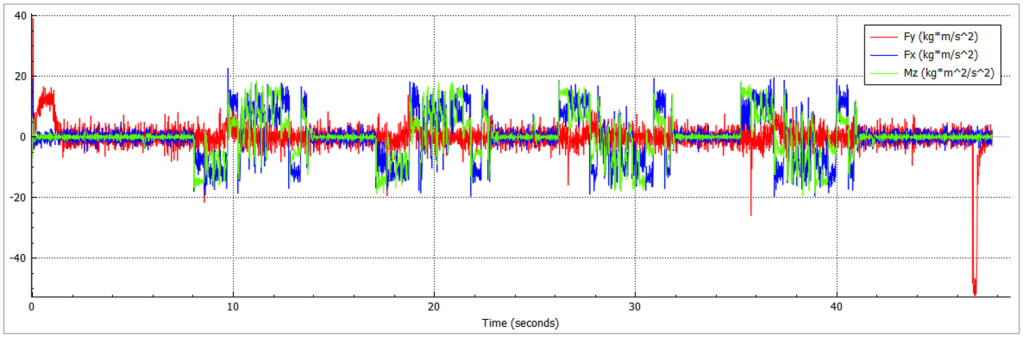


Figura 6.45: Forças detetadas pelo sensor do robô *follower* durante o movimento do sistema - Estratégia 1

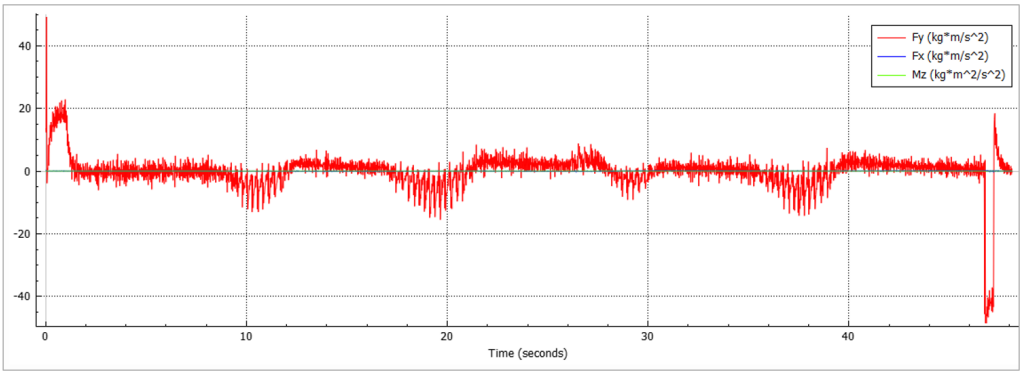


Figura 6.46: Forças detetadas pelo sensor do robô *follower* durante o movimento do sistema - Estratégia 2

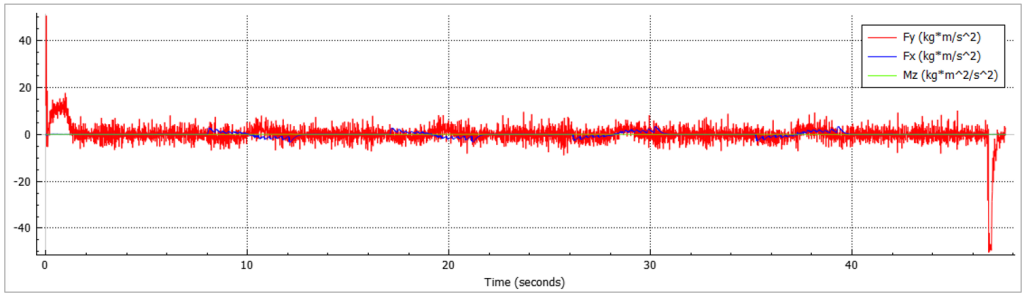


Figura 6.47: Forças detetadas pelo sensor do robô *follower* durante o movimento do sistema - Estratégia 3

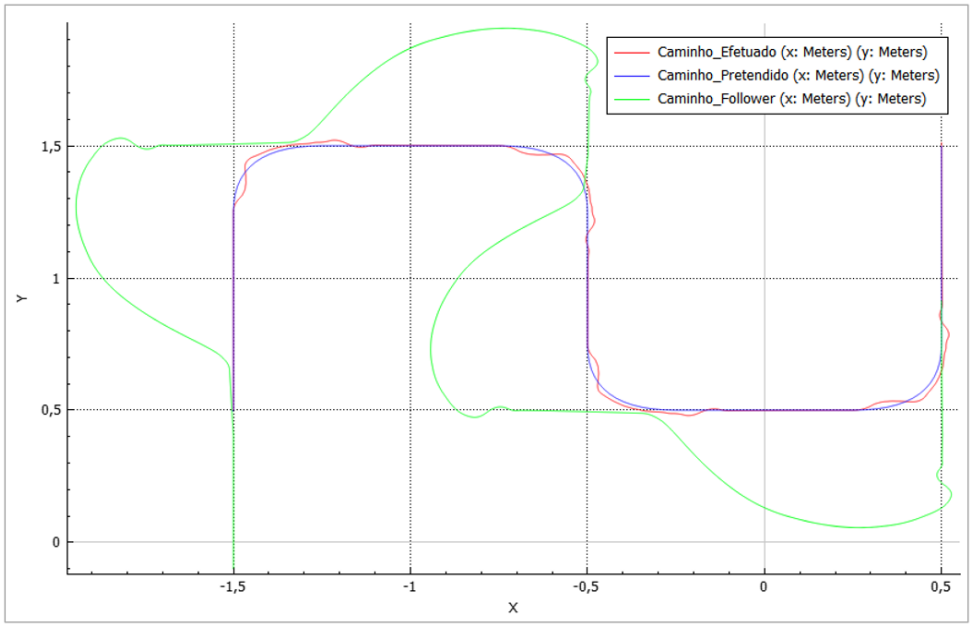


Figura 6.48: Comparação entre o caminho pretendido e o efetuado pelo robô *leader* e caminho efetuado pelo robô *follower* - Estratégia 1

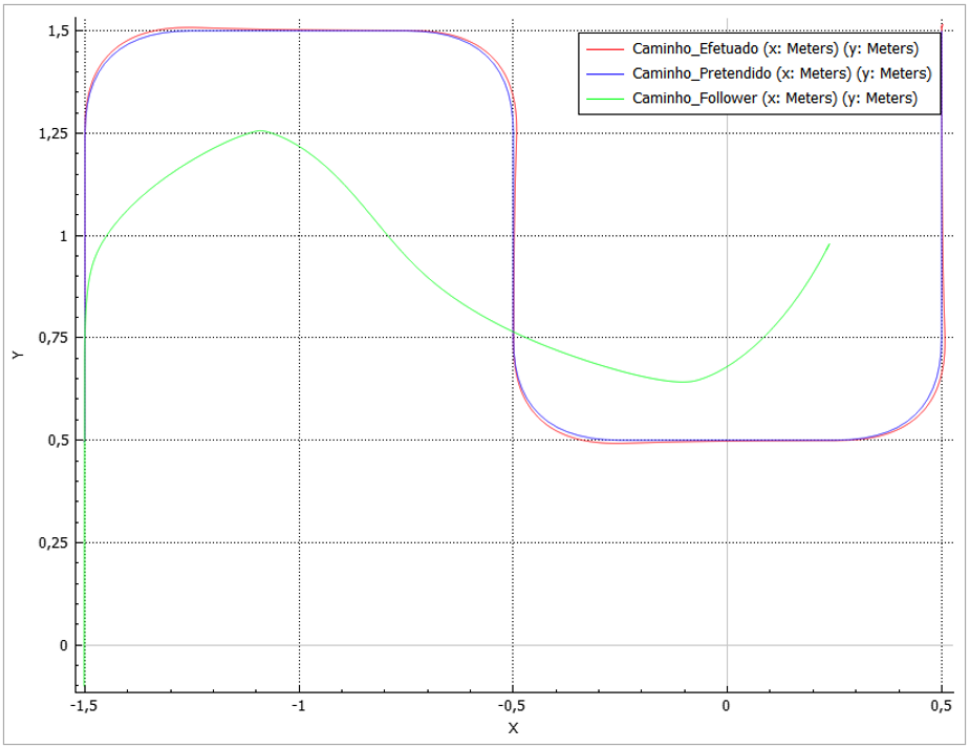


Figura 6.49: Comparação entre o caminho pretendido e o efetuado pelo robô *leader* e caminho efetuado pelo robô *follower* - Estratégia 2

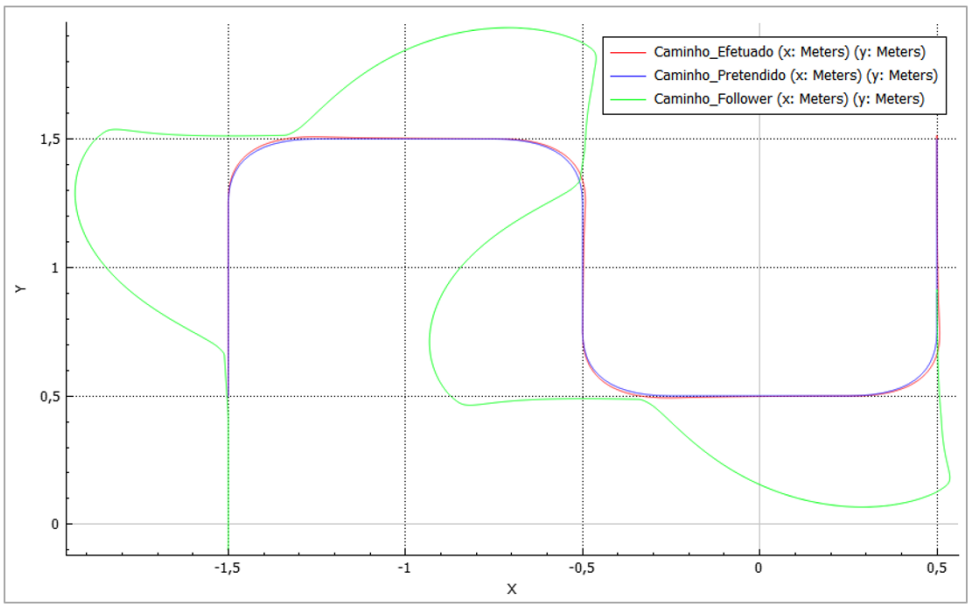


Figura 6.50: Comparação entre o caminho pretendido e o efetuado pelo robô *leader* e caminho efetuado pelo robô *follower* - Estratégia 3

Analisando os resultados é possível constatar que, de estratégia para estratégia, há uma melhoria substancial naquilo que diz respeito às forças que são exercidas na carga e também no erro relativo ao seguimento do caminho por parte do robô *leader*. Na Figura 6.48 é possível reparar que o facto dos robôs estarem conectados rigidamente ao objeto na estratégia 1 faz com que haja uma grande influência do *follower* no movimento do *leader*, o que resulta num movimento menos suavizado e com mais erros, principalmente quando há a necessidade do *leader* corrigir a sua orientação. Por outro lado, isto não se verifica nas outras estratégias uma vez que foi introduzido um grau de liberdade nos suportes. Porém, existe também uma perda relativamente à capacidade do robô *follower* conseguir estimar todos os tipos de movimentos do *leader* exclusivamente através da comunicação indireta. Inclusive, na estratégia 2, o sistema terá um comportamento análogo a um sistema de robôs diferenciais conforme percorre o caminho. Já na estratégia 3 isto não acontece, dado que se introduz a comunicação direta entre os robôs.

Capítulo 7

Conclusões

Neste capítulo serão apresentadas as conclusões finais relativas ao trabalho realizado, as principais dificuldades sentidas e serão perspetivadas algumas ideias para trabalhos futuros.

7.1 Considerações Finais

O objetivo deste trabalho passou por realizar um estudo acerca de estratégias que permitissem a dois robôs omnidirecionais realizar o co-transporte de uma carga que não poderia ser transportada por apenas um robô. Sendo assim, houve a necessidade de adquirir conhecimentos acerca dos sistemas multi-robô, com um maior ênfase na área do co-transporte. Além da área de estudo foi também crucial estudar características do equipamento com o qual se pretendia construir o sistema, bem como a técnica com a qual se pretendia realizar o seu controlo.

Não havendo a possibilidade de usar o sistema de dois robôs reais procurou-se uma ferramenta de simulação que permitisse continuar o estudo do sistema. Avaliadas várias opções, escolheu-se o CoppeliaSim. Depois de modelado o sistema nesta ferramenta de simulação, foi possível explorar três estratégias diferentes para estabelecer a cooperação entre os elementos do mesmo. Embora apresentem disparidades, todas elas recaem sobre uma topologia *leader-follower* e o princípio da minimização das forças de contacto. Mais importante que isso, é que todas elas permitem, com as suas vantagens e desvantagens, realizar o transporte da carga.

Numa fase inicial optou-se por seguir a abordagem que utiliza o sensor de força/binário multiaxial na construção do suporte. A ideia seria que o robô *follower* pudesse estimar todos os tipos de movimento do robô *leader* sem que houvesse a necessidade de existir comunicação direta entre eles. Como se mostrou

ao longo do relatório, o robô *follower*, através deste sensor, podia medir todas as forças no espaço tridimensional e reagir a qualquer tipo de movimento. O principal problema desta estratégia tem que ver com as forças a que a carga poderá estar sujeita e o facto de haver uma grande influência por parte do *follower* no movimento do *leader*, já que estes se encontram interligados através da carga. No entanto, verificou-se que os robôs eram capazes de se movimentar em conjunto, e as forças seriam menores conforme a velocidade do *leader* também fosse mais baixa.

Com a introdução de um grau de liberdade na segunda estratégia, e como seria de esperar, deixaram de existir forças de cisalhamento e torção a ser aplicadas na carga. Por outro lado, o sistema perdeu a capacidade de realizar movimentos omnidirecionais. Contudo, foi visto que também era possível realizar o co-transporte do objeto recorrendo a esta estratégia, mesmo que os robôs não mantivessem a formação e funcionassem na grande parte do tempo como um sistema de robôs com acionamento diferencial. Outra vantagem seria o menor custo do equipamento utilizado na implementação do sistema, visto que a célula de carga é mais acessível que o sensor de força/binário multiaxial.

Por último, tentou-se complementar a segunda estratégia de maneira a que o sistema tivesse a capacidade de realizar movimentos omnidirecionais. Isto foi conseguido através da introdução da comunicação direta de um parâmetro entre os robôs. Esta terceira estratégia foi a que apresentou melhores resultados, sendo que, fora o arranque e a travagem do sistema, as forças exercidas na carga durante um movimento a uma velocidade constante eram nulas ou extremamente reduzidas.

Todas as estratégias apresentadas têm pontos fortes e fracos que poderiam continuar a ser explorados. Porém, e caso fosse necessário implementar um sistema depois daquilo que foi analisado, a melhor opção era implementar um sistema que pudesse alternar entre os modos de funcionamento das estratégias 2 e 3. Deste modo seria possível para o sistema funcionar apenas com base na comunicação indireta, mas também poderia aproveitar ao máximo as suas propriedades omnidirecionais impondo o mínimo de forças na carga, sendo que para isso o *leader* comunicaria a sua orientação relativa à carga ao *follower*.

Posto isto, considera-se que os objetivos do trabalho foram cumpridos. Analisaram-se várias estratégias que permitem ao sistema de dois robôs omnidirecionais transportar uma carga em conjunto. Inclusive, fez-se o possível por chegar a uma que complementasse ao máximo as características do sistema, permitisse uma maior segurança da carga e reduzisse ao máximo, ou eliminasse, a comunicação direta. Além disto, ainda se implementou o seguimento de caminhos para o robô *leader* que, embora não fizesse parte do âmbito do projeto, foi essencial para se poder comparar as estratégias diretamente.

7.2 Principais Dificuldades Encontradas

A principal dificuldade sentida durante a realização deste trabalho prendeu-se com questões relativas à criação dos modelos virtuais do sistema. Embora possa parecer uma das parcelas do trabalho que é mais trivial, foi, pelo contrário, a fonte de bastantes problemas e perdas de tempo que poderia ter sido aplicado ao estudo do sistema em si. Isto deveu-se principalmente pela razão de ter de se perceber a correta implementação das cadeias dinâmicas dos modelos no simulador. A escolha da inércia e massa das formas influencia bastante o comportamento das cadeias dinâmicas. Por vezes, formas com valores de inércia e massa muito distintos poderão ser a causa de comportamentos indesejados no sistema como, por exemplo, vibrações. Estas últimas, juntamente com a criação da conexão rígida para simular o agarrar do objeto por parte dos suportes levava a que aparecesse muito ruído nas leituras do sensor de força. Para resolver isto foi necessário atribuir valores irrealistas a estas propriedades das formas de modo a garantir a estabilidade dos modelos. Além de ser necessário despender tempo a ajustar manualmente os valores, isto poderá ser um impedimento para a criação de um modelo o mais aproximado do real possível. Depois de procurar nos fóruns da Coppelia Robotics por uma solução verificou-se que o motor de física Vortex Studio lida bastante bem com estes problemas e é considerado também o mais realista entre as opções presentes no simulador. Porém, é o único que requer o pedido de uma licença. Outro problema relativo à cadeia dinâmica dos modelos é a definição das máscaras que, caso não seja feita como foi explicada no Capítulo 5, também irá provocar graves problemas durante a simulação.

7.3 Ideias para Trabalhos Futuros

A estratégia 3 foi a que permitiu obter os melhores resultados. Contudo, exige que seja realizada a comunicação de um parâmetro entre os dois robôs. Seria interessante tentar perceber de que modo o *follower* se poderia alinhar com o *leader* através da comunicação indireta. Com isto aumentar-se-ia ainda mais a capacidade do sistema para realizar tarefas em ambientes em que não é permitida a comunicação ou que são propensos a erros da mesma.

Relativamente ao controlo do sistema, este foi realizado com recurso a controladores PI. Poderia ser interessante investigar outros tipos de controlo e de que forma poderiam ou não impactar o desempenho do sistema.

O trabalho realizado passou pela simulação, mas também seria interessante implementar o sistema com as plataformas reais e verificar até que ponto é que o sistema real se comportaria como o simulado.

Referências Bibliográficas

[1] Robotic Industries Association, "Robots in Logistics and Transportation." https://www.robotics.org/blog-article.cfm/Robots-in-Logistics-and-Transportation/43, Acedido a 29 de janeiro de 2020. [citado na p. 1]

[2] Robotic Industries Association, "Logistics Robots." https://www.robotics.org/service-robots/logistics-robots, Acedido a 29 de janeiro de 2020. [citado na p. 1]

[3] Internacional Federation of Robotics, "World Robotics 2019 edition." https://ifr.org/free-downloads/, Acedido a 29 de janeiro de 2020. [citado na p. 1, 2]

[4] Conveyco, "The Advantages and Disadvantages of Automated Guided Vehicles (AGVs)." https://www.conveyco.com/advantages-disadvantages-automated-guided-vehicles-agvs/, Acedido a 29 de janeiro de 2020. [citado na p. 2]

[5] Conveyco, "Autonomous Mobile Robots (AMRS)." https://www.conveyco.com/technology/autonomous-mobile-robots-amrs/, Acedido a 29 de janeiro de 2020. [citado na p. 2]

[6] J. E. Inglett and E. J. Rodríguez-Seda, "Object transportation by cooperative robots," (United States Naval Academy Annapolis, MD 21402), 2017. [citado na p. 2, 18]

[7] I. Mas and C. Kitts, "Object manipulation using cooperative mobile multi-robot systems," in *Proceedings of the World Congress on Engineering and Computer Science 2012 Vol I*, (San Francisco, USA), 2012. [citado na p. 2, 12]

[8] E. Tuci, M. H. M. Alkilabi, and O. Akanyeti, "Cooperative object transport in multi-robot systems: A review of the state-of-the-art," in *Frontiers in Robotics and AI vol.5, article 59*, 2018. [citado na p. 2, 12, 15, 16, 19]

[9] Industry Search, "Industrial Platform Trolleys 500 kg | IT500." https://www.industrysearch.com.au/industrial-platform-trolleys-500-kg-it500/p/141730, Acedido a 18 de fevereiro de 2020. [citado na p. 8]

[10] E. Correia de Brito Lda, "Representantes em Portugal da CLARK empilhadores." https://grupoecb.pai.pt/produtos, Acedido a 18 de fevereiro de 2020. [citado na p. 8]

[11] G. Ullrich, "The history of automated guided vehicle systems," in *Automated Guided Vehicle Systems*, (Springer, Berlin, Heidelberg), pp. 1–14, 2015. [citado na p. 8]

[12] Dematic, "The history of the Automated Guided Vehicle." https://www.dematic.com/en/news-and-events/press-releases/2019/2019/12/19/14/41/2019-12-18_agv/, Acedido a 6 de março de 2020. [citado na p. 8]

[13] T. Davich, "Material handling solutions: A look into automated robotics," (Department of Industrial and Systems Engineering, University of Wisconsin-Madison), 2010. [citado na p. 8, 9]

[14] Muratec, "AGV-Unit Load Magnetic Tape Guided." https://www.muratec-usa.com/machinery/material-handling/transportation/agv-unit-load-magnetic-tape-guided/, Acedido a 18 de fevereiro de 2020. [citado na p. 9]

[15] Automatic Guided Vehicles, "Towing Vehicles." https://www.automaticguidedvehicles.com/, Acedido a 18 de fevereiro de 2020. [citado na p. 9]

[16] Kolec, "Products." https://www.kolec.co.jp/user_data/en/product/robo_fork, Acedido a 18 de fevereiro de 2020. [citado na p. 9]

[17] Fetch Robotics, "AMRs vs AGVs: The Difference Between a Robot and a Guided Vehicle." https://fetchrobotics.com/fetch-robotics-blog/amrs-vs-agvs-whats-the-difference/, Acedido a 6 de março de 2020. [citado na p. 9]

[18] R. Bogue, "Growth in e-commerce boosts innovation in the warehouse robot market," in *Industrial Robot: An International Journal, Vol. 43 Iss 6*, 2016. [citado na p. 9]

[19] Quartz, "Amazon is just beginning to use robots in its warehouses and they're already making a huge difference." https://qz.com/709541/amazon-is-just-beginning-to-use-robots-in-its-warehouses-and-theyre-already-making-a-huge-difference/, Acedido a 20 de fevereiro de 2020. [citado na p. 10]

[20] Robotic Business Review, "Case Study: Why Ford Deployed AMRs to Automate Spanish Factory." `https://www.roboticsbusinessreview.com/case_studies/case-study-why-ford-deployed-amrs-to-automate-spanish-factory/`, Acedido a 20 de fevereiro de 2020. [citado na p. 10]

[21] E. Tuci, R. Gross, V. Trianni, F. Mondada, M. Bonani, and M. Dorigo, "Cooperation through self-assembly in multi-robot systems," in *ACM Transactions on Autonomous and Adaptive Systems, Vol. 1, No. 2*, p. 115–150, 2006. [citado na p. 10, 23, 24, 27]

[22] Kuka, "KUKA omniMove (AGVs)." `https://www.kuka.com/en-us/products/mobility/mobile-platforms/kuka-omnimove`, Acedido a 23 de abril de 2020. [citado na p. 11]

[23] Kuka, "Kuka omniMove option packages." `https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kuka_omnimove_additional_options.pdf?rev=ba35b134943e483ebf75427971932d8b`, Acedido a 23 de abril de 2020. [citado na p. 11]

[24] Kuka, "Specific products for Aeronautics." `http://www.aritex-es.com/en/aeronautica/`, Acedido a 23 de abril de 2020. [citado na p. 11]

[25] C. Rizzo, A. Lagraña, and D. Serrano, "Geomove: Detached agvs for cooperative transportation of large and heavy loads in the aeronautic industry," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 126–133, 2020. [citado na p. 11, 20, 25, 26, 27, 28]

[26] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Heidelberg, Germany: Springer, 2008. [citado na p. 12, 13, 15]

[27] IEEE Spectrum, "Microdrones That Cooperate to Transport Objects Could Be Future of Warehouse Automation." `https://spectrum.ieee.org/automaton/robotics/drones/microdrones-future-of-warehouse-automation`, Acedido a 30 de janeiro de 2020. [citado na p. 12]

[28] Intelligent Agents Lab, "Projects." `http://ial.eecs.ucf.edu/projects.php`, Acedido a 30 de janeiro de 2020. [citado na p. 12]

[29] S. Heshmati-Alamdari, C. P. Bechlioulis, G. C. Karras, and K. J. Kyriakopoulos, "Decentralized impedance control for cooperative manipulation of multiple underwater vehicle manipulator systems under lean communication," arXiv:1905.04531v1 [cs.RO], 2019. [citado na p. 12]

[30] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng, "Cooperative mobile robotics: Antecedents and directions," in *Autonomous Robots 4*, (UCLA

Computer Science Department, Los Angeles, CA 90024-1596), p. 7–27, 1997. [citado na p. 12, 14, 15, 18]

[31] Diário do Nordeste, "Robôs desenvolvidos nos Estados Unidos executam trabalhos como se fossem enxames." `http://blogs.diariodonordeste.com.br/diariocientifico/robotica/robos-desenvolvidos-nos-estados-unidos-executam-trabalhos-como-se-fossem-cardumes/`, Acedido a 6 de março de 2020. [citado na p. 15]

[32] L. Parker, "L-alliance: A mechanism for adaptive action selection in heterogeneous multi-robot teams," August 1996. [citado na p. 15]

[33] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," in *International Journal of Advanced Robotic Systems*, 2013. [citado na p. 15, 17]

[34] C. Bechlioulis and K. Kyriakopoulos, "Collaborative multi-robot transportation in obstacle-cluttered environments via implicit communication," *Frontiers in Robotics and AI*, vol. 5, p. 90, 2018. [citado na p. 16, 17]

[35] J. Alonso-Mora, R. Knepper, R. Siegwart, and D. Rus, "Local motion planning for collaborative multi-robot manipulation of deformable objects," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5495–5502, 2015. [citado na p. 18]

[36] S. Moon, D. Kwak, and H. J. Kim, "Cooperative control of differential wheeled mobile robots for box pushing problem," in *2012 12th International Conference on Control, Automation and Systems*, pp. 140–144, 2012. [citado na p. 19]

[37] S. Makita and W. Wan, "A survey of robotic caging and its applications," in *Journal of the Robotics Society of Japan, Vol. 36*, pp. 316–326, 2018. [citado na p. 20]

[38] Z. Wang and M. Schwager, "Kinematic multi-robot manipulation with no communication using force feedback," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, (Stockholm, Sweden), p. 427–432, 2016. [citado na p. 20, 21, 22, 27]

[39] G. A. S. Pereira, B. S. Pimentel, L. Chaimowicz, and M. F. M. Campos, "Coordination of multiple mobile robots in an object carrying task using implicit communication," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, pp. 281–286 vol.1, 2002. [citado na p. 20, 21, 27]

[40] T. Machado, T. Malheiro, S. Monteiro, W. Erlhagen, and E. Bicho, "Multi-constrained joint transportation tasks by teams of autonomous mobile robots using a dynamical systems approach," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3111–3117, 2016. [citado na p. 22, 23, 27]

[41] I. Doroftei, V. Grosu, and V. Spinu, *Omnidirectional Mobile Robot - Design and Implementation*, pp. 512–528. 09 2007. [citado na p. 29, 30, 31]

[42] K. Kanjanawanishkul, "Omnidirectional wheeled mobile robots: Wheel types and practical applications," *International Journal of Advanced Mechatronic Systems*, vol. 6, p. 289, 02 2015. [citado na p. 30, 31, 33, 34]

[43] H. Oliveira, A. Sousa, A. Moreira, and P. Costa, *Modeling and Assessing of Omni-Directional Robots with Three and Four Wheels*. 12 2009. [citado na p. 33]

[44] F. Adascalitei and I. Doroftei, "Practical applications for mobile robots based on mecanum wheels - a systematic survey," *Romanian Review Precision Mechanics, Optics and Mechatronics*, pp. 21–29, 01 2011. [citado na p. 34]

[45] Y. Li, S. Dai, L. Zhao, X. Yan, and Y. Shi, "Topological design methods for mecanum wheel configurations of an omnidirectional mobile robot," *Symmetry*, vol. 11, p. 1268, oct 2019. [citado na p. 35, 36, 37]

[46] Y. Li, S. Ge, S. Dai, L. Zhao, X. Yan, Y. Zheng, and Y. Shi, "Kinematic modeling of a combined system of multiple mecanum-wheeled robots with velocity compensation," *Sensors*, vol. 20, p. 75, 12 2019. [citado na p. 38]

[47] AssemblyMagazine, "Miniature Force Sensors Aid in Testing Touchscreens and Consumer Electronics." `https://www.assemblymag.com/articles/93226-miniature-force-sensors-aid-in-testing-touchscreens-and-consumer-electronics?iframe=1`, Acedido a 9 de junho de 2020. [citado na p. 39]

[48] Industrial Machinery Digest, "Integrated Force Control Real-Time Feedback from ABB Robotics." `https://industrialmachinerydigest.com/imd/integrated-force-control-real-time-feedback-from-abb-robotics/`, Acedido a 9 de junho de 2020. [citado na p. 39]

[49] Kistler, "Piezo vs. strain gauge." `https://www.kistler.com/en/glossary/term/piezo-vs-strain-gauge/`, Acedido a 9 de junho de 2020. [citado na p. 39, 40]

[50] Kleckers, Thomas, "Spoilt for choice: piezoelectric or strain gauge based force transducers?." `https://www.hbm.com/fileadmin/mediapool/files/technical-articles-technotes-white-papers/Piezoelectric_or_str`

ain_gauge_based_force_transducers.pdf, Acedido a 9 de junho de 2020. [citado na p. 39]

[51] J. Chapman, "Comparative look at strain gauge and piezoelectric sensors," *Electricity + Control*, pp. 32–34, November 2013. [citado na p. 39, 40]

[52] Hoffmann-Krippner, "Spotlight: How Force Sensors Work." https://www.hoffmann-krippner.com/spotlight-how-force-sensors-work/, Acedido a 15 de junho de 2020. [citado na p. 40]

[53] Kistler, "Force Sensors." https://www.kistler.com/?type=669&fid=278&model=download, Acedido a 9 de junho de 2020. [citado na p. 40, 45]

[54] D. Lee, U. Kim, H. Jung, and H. R. Choi, "A capacitive-type novel six-axis force/torque sensor for robotic applications," *IEEE Sensors Journal*, vol. 16, no. 8, pp. 2290–2299, 2016. [citado na p. 41]

[55] Robotiq, "Force Sensors in Robotics Research." https://pt.slideshare.net/marcelochirai/force-sensors-in-robotics-research, Acedido a 27 de junho de 2020. [citado na p. 41]

[56] S. Petroni, M. Amato, and M. Vittorio, *Advanced MEMS Technologies for Tactile Sensing and Actuation*, pp. 1–30. 01 2013. [citado na p. 41]

[57] Adafruit, "Force Sensitive Resistor (FSR)." https://learn.adafruit.com/force-sensitive-resistor-fsr, Acedido a 27 de junho de 2020. [citado na p. 42]

[58] Applied Measurement, "OptoForce 3D optical sensors and six axis Force/-Torque sensors: Sensing Flexibility." https://appliedmeasurement.com.au/optoforce-3d-optical-sensors-and-six-axis-forcetorque-sensors/, Acedido a 27 de junho de 2020. [citado na p. 42]

[59] Loadstar Sensors, "What is a Load Cell? How do Load Cells Work?." https://www.loadstarsensors.com/what-is-a-load-cell.html, Acedido a 27 de julho de 2020. [citado na p. 43]

[60] Honeywell, "Effectively Using Pressure, Load, and Torque Sensors with Today's Data Acquisition Systems." https://sensing.honeywell.com/white-paper-effectivelyusingpressureloadandtorquesensorswithtodaysdataacqusitionsystems-008883-2-en.pdf, Acedido a 27 de julho de 2020. [citado na p. 43, 44, 45]

[61] Loadstar Sensors, "Force and Tactile Sensors Give Robots a Feel for the Job." https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Force-and-Tactile-Sensors-Give-

Robots-a-Feel-for-the-Job/content_id/7337, Acedido a 27 de julho de 2020. [citado na p. 46]

[62] Schunk, "Force/torque sensor FT-AXIA." https://schunk.com/fileadmin/pim/docs/IM0024734.PDF, Acedido a 27 de julho de 2020. [citado na p. 46]

[63] D. Corrigan, "Characterising the Response of a Closed Loop System." http://www.mee.tcd.ie/~corrigad/3c1/control_ho2_2012_students.pdf, Acedido a 17 de outubro de 2020. [citado na p. 47, 48]

[64] F. Haugen, "Ziegler-Nichols' Closed-Loop Method." http://techteach.no/publications/articles/zn_closed_loop_method/zn_closed_loop_method.pdf, Acedido a 17 de outubro de 2020. [citado na p. 49]

[65] Cyberbotics, "Webots Open Source Robot Simulator." https://cyberbotics.com/, Acedido a 22 de maio de 2020. [citado na p. 52]

[66] Cyberbotics, "Webots User Guide." https://cyberbotics.com/doc/guide/youbot, Acedido a 22 de maio de 2020. [citado na p. 53]

[67] European Commission, "Swarmanoid: towards humanoid robotic swarms." https://cordis.europa.eu/project/id/022888, Acedido a 22 de maio de 2020. [citado na p. 53]

[68] ARGoS, "ARGoS Large-scale robot simulations." https://www.argos-sim.info/index.php, Acedido a 22 de maio de 2020. [citado na p. 53]

[69] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, T. Stirling, A. Gutiérrez, L. M. Gambardella, and M. Dorigo, "Argos: a pluggable, multi-physics engine simulator for heterogeneous swarm robotics," 02 2011. [citado na p. 53, 54]

[70] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," pp. 1400 – 1405, 05 2007. [citado na p. 54]

[71] S. Mokaram, K. Samsudin, A. Ramli, and H. Kerdegari, "Mobile robots communication and control framework for usarsim," vol. 2, 06 2012. [citado na p. 54]

[72] E. Colombini and C. Ribeiro, "An attentive multi-sensor based system for mobile robotics," in *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pp. 1509–1514, 10 2012. [citado na p. 54]

[73] Gazebo, "Gazebo Robot simulation made easy." http://gazebosim.org/, Acedido a 22 de maio de 2020. [citado na p. 55]

[74] Dronecode, "Gazebo Simulation." https://dev.px4.io/v1.9.0/en/simulation/gazebo.html, Acedido a 22 de maio de 2020. [citado na p. 55]

[75] CoppeliaRobotics, "CoppeliaSim." https://www.coppeliarobotics.com/, Acedido a 22 de maio de 2020. [citado na p. 56]

[76] CoppeliaSim User Manual, "User Interface." https://www.coppeliarobotics.com/helpFiles/index.html, Acedido a 12 de julho de 2020. [citado na p. 58, 59, 60, 62, 63, 64, 65, 66, 67]

[77] Chengdu Hangfa Robotics, "Discovery Q2." http://www.hangfa.com/EN/robot/DiscoveryQ2.html, Acedido a 20 de agosto de 2020. [citado na p. 69, 72]

[78] F. J. M. de Sá, "Sistema de navegação para plataforma móvel omnidirecional," Master's thesis, Faculdade de Engenharia do Porto, Porto, 2016. [citado na p. 69, 70]

[79] S. Barawkar, "Collaborative transportation of a common payload using two uavs based on force feedback control," Master's thesis, University of Cincinnati, 2017. [citado na p. 88]

Anexo A

Scripts em Lua Utilizados na Simulação do Sistema

A.1 Comportamento das Rodas Mecanum

O código que se segue diz respeito à implementação do comportamento de uma roda *mecanum left-handed*. No caso das rodas *right-handed* a totalidade do código será igual, à exceção dos *handles* e da função da linha 22 que deverá ser substituída pela função comentada na linha 23.

```
1  function sysCall_init()
2      rolling=sim.getObjectHandle('rollingJoint_fl')
3      slipping=sim.getObjectHandle('slippingJoint_fl')
4      wheel=sim.getObjectHandle('wheel_respondable_fl')
5  end
6
7  function sysCall_cleanup()
8
9  end
10
11 function sysCall_actuation()
12   if (sim.getObjectParent(rolling)~=-1) then
13         local linVel,angVel=sim.getVelocity(wheel)
14         sim.resetDynamicObject(wheel)
15         sim.setObjectFloatParameter(wheel,3000,linVel[1])
16         sim.setObjectFloatParameter(wheel,3001,linVel[2])
17         sim.setObjectFloatParameter(wheel,3002,linVel[3])
18         sim.setObjectFloatParameter(wheel,3020,angVel[1])
19         sim.setObjectFloatParameter(wheel,3021,angVel[2])
20         sim.setObjectFloatParameter(wheel,3022,angVel[3])
21         sim.setObjectPosition(slipping,rolling,{0,0,0})
```

```
        sim.setObjectOrientation(slipping,rolling,{-math.pi/4,0,0})
        -- sim.setObjectOrientation(slipping,rolling,{math.pi/4,0,0}) -> roda right-handed
        sim.setObjectPosition(wheel,rolling,{0,0,0})
        sim.setObjectOrientation(wheel,rolling,{0,0,0})
    end
end
```

A.2 Testes Realizados ao Modelo da Plataforma

```
function sysCall_threadmain()

    --Inicializacoes--
    fl=sim.getObjectHandle('rollingJoint_fl')
    fr=sim.getObjectHandle('rollingJoint_fr')
    rl=sim.getObjectHandle('rollingJoint_rl')
    rr=sim.getObjectHandle('rollingJoint_rr')

    sim.setJointTargetVelocity(fl,0)
    sim.setJointTargetVelocity(fr,0)
    sim.setJointTargetVelocity(rl,0)
    sim.setJointTargetVelocity(rr,0)

    -- Parametros da Estrutura da Plataforma--
    R=0.05
    l=0.134
    L=0.134

    -- Velocidades Lineares e Rotacional da Plataforma--
    vx=0
    vy=0
    w=0

    LinVel=1
    RotVel=3.5

    while sim.getSimulationState()~=
    sim.simulation_advancing_abouttostop do
        timepassed=sim.getSimulationTime()

        if(timepassed>0 and timepassed<=2) then
            vy=LinVel
            vx=0
        end

        if(timepassed>2 and timepassed<=4) then
            vy=-LinVel
            vx=0
        end

```

```
40          if(timepassed>4 and timepassed<=5) then
41              vy=0
42              vx=0
43          end
44
45          if(timepassed>5 and timepassed<=7) then
46              vy=0
47              vx=LinVel
48          end
49
50          if(timepassed>7 and timepassed<=9) then
51              vy=0
52              vx=-LinVel
53          end
54
55          if(timepassed>9 and timepassed<=10) then
56              vy=0
57              vx=0
58          end
59
60          if(timepassed>10 and timepassed<=12) then
61              vy=0
62              vx=0
63              w=RotVel
64          end
65
66          if(timepassed>12 and timepassed<=13) then
67              vy=0
68              vx=0
69              w=0
70          end
71
72
73          if(timepassed>13 and timepassed<=15) then
74              vy=0
75              vx=0
76              w=-RotVel
77          end
78
79          if(timepassed>15) then
80              vy=0
81              vx=0
82              w=0
83          end
84
85          w1=-(1/R)*vx+(1/R)*vy+((l+L)/R)*w
86          w2=(1/R)*vx+(1/R)*vy-((l+L)/R)*w
87          w3=-(1/R)*vx+(1/R)*vy-((l+L)/R)*w
88          w4=(1/R)*vx+(1/R)*vy+((l+L)/R)*w
89
90          sim.setJointTargetVelocity(fl,w2)
```

```
        sim.setJointTargetVelocity(fr,w1)
        sim.setJointTargetVelocity(rl,w3)
        sim.setJointTargetVelocity(rr,w4)

    end
end

function sysCall_cleanup()

end
```

A.3 Criação da Conexão entre os Robôs e o Objeto

```
function sysCall_init()
    s=sim.getObjectHandle('Sensor')
    l=sim.getObjectHandle('LoopClosureDummy1')
    l2=sim.getObjectHandle('LoopClosureDummy2')
    b=sim.getObjectHandle('Plate')
    suctionPadLink=sim.getObjectHandle('Link')

    infiniteStrength=sim.getScriptSimulationParameter(
    sim.handle_self,'infiniteStrength')
    maxPullForce=sim.getScriptSimulationParameter(sim.handle_self,'
    maxPullForce')
    maxShearForce=sim.getScriptSimulationParameter(sim.handle_self,
    'maxShearForce')
    maxPeelTorque=sim.getScriptSimulationParameter(sim.handle_self,
    'maxPeelTorque')

    sim.setLinkDummy(l,-1)
    sim.setObjectParent(l,b,true)
    m=sim.getObjectMatrix(l2,-1)
    sim.setObjectMatrix(l,-1,m)
end

function sysCall_cleanup()
    sim.setLinkDummy(l,-1)
    sim.setObjectParent(l,b,true)
    m=sim.getObjectMatrix(l2,-1)
    sim.setObjectMatrix(l,-1,m)
end

function sysCall_sensing()
    parent=sim.getObjectParent(l)
    if (sim.getScriptSimulationParameter(sim.handle_self,'active')
    ==false) then
        if (parent~=b) then
            sim.setLinkDummy(l,-1)
            sim.setObjectParent(l,b,true)
            m=sim.getObjectMatrix(l2,-1)
```

```
33              sim.setObjectMatrix(l,-1,m)
34          end
35      else
36          if (parent==b) then
37              -- Here we want to detect a respondable shape, and then
     connect to it with a force sensor (via a loop closure dummy
    dummy link)
38              -- However most respondable shapes are set to "non-
    detectable", so "sim.readProximitySensor" or similar will not
    work.
39              -- But "sim.checkProximitySensor" or similar will work
    (they don't check the "detectable" flags), but we have to go
    through all shape objects!
40              index=0
41              while true do
42                  shape=sim.getObjects(index,sim.object_shape_type)
43                  if (shape==-1) then
44                      break
45                  end
46                  if (shape~=b) and (sim.getObjectInt32Parameter(
    shape,sim.shapeintparam_respondable)~=0) and (
    sim.checkProximitySensor(s,shape)==1) then
47                      -- Ok, we found a respondable shape that was
    detected
48                      -- We connect to that shape:
49                      -- Make sure the two dummies are initially
    coincident:
50                      sim.setObjectParent(l,b,true)
51                      m=sim.getObjectMatrix(l2,-1)
52                      sim.setObjectMatrix(l,-1,m)
53                      -- Do the connection:
54                      sim.setObjectParent(l,shape,true)
55                      sim.setLinkDummy(l,l2)
56                      break
57                  end
58                  index=index+1
59              end
60          else
61              -- Here we have an object attached
62              if (infiniteStrength==false) then
63                  -- We might have to conditionally beak it apart!
64                  result,force,torque=sim.readForceSensor(
    suctionPadLink) -- Here we read the median value out of 5 values
     (check the force sensor prop. dialog)
65                  if (result>0) then
66                      breakIt=false
67                      if (force[3]>maxPullForce) then breakIt=true
    end
68                      sf=math.sqrt(force[1]*force[1]+force[2]*force
    [2])
69                      if (sf>maxShearForce) then breakIt=true end
```

```
                    if (torque[1]>maxPeelTorque) then breakIt=true
    end
                    if (torque[2]>maxPeelTorque) then breakIt=true
    end
                    if (breakIt) then
                        -- We break the link:
                        sim.setLinkDummy(l,-1)
                        sim.setObjectParent(l,b,true)
                        m=sim.getObjectMatrix(l2,-1)
                        sim.setObjectMatrix(l,-1,m)
                    end
                end
            end
        end
    end
end
```

A.4 Script do Controlo Manual do Leader

```
function sysCall_threadmain()

    --Inicializacoes--
    fl=sim.getObjectHandle('rollingJoint_fl')
    fr=sim.getObjectHandle('rollingJoint_fr')
    rl=sim.getObjectHandle('rollingJoint_rl')
    rr=sim.getObjectHandle('rollingJoint_rr')

    sim.setJointTargetVelocity(fl,0)
    sim.setJointTargetVelocity(fr,0)
    sim.setJointTargetVelocity(rl,0)
    sim.setJointTargetVelocity(rr,0)

    -- Parametros da Estrutura da Plataforma--
    R=0.05
    l=0.134
    L=0.134

    -- Velocidades Lineares e Rotacional da Plataforma--
    vx=0
    vy=0
    w=0

    while sim.getSimulationState()~=
    sim.simulation_advancing_abouttostop do

    message,auxiliaryData=sim.getSimulatorMessage()
    --print(auxiliaryData)
    while message~=-1 do
        if (message==sim.message_keypress) then
            if (auxiliaryData[1]==119) then --> W
```

```
31                  if (vy<0.65) then
32                      vy=vy+0.001
33                  end
34              end
35              if (auxiliaryData[1]==97) then --> A
36                  if (vx>-0.65) then
37                      vx=vx-0.001
38                  end
39              end
40              if (auxiliaryData[1]==115) then --> S
41                  if (vy>-0.65) then
42                      vy=vy-0.001
43                  end
44              end
45              if (auxiliaryData[1]==100) then --> D
46                  if (vx<0.65) then
47                      vx=vx+0.001
48                  end
49              end
50              if (auxiliaryData[1]==101) then --> E
51                  if (w>-1) then
52                      w=w-0.001
53                  end
54              end
55              if (auxiliaryData[1]==113) then --> Q
56                  if (w<1) then
57                      w=w+0.001
58                  end
59              end
60              if (auxiliaryData[1]==32) then --> B_Espaco
61                  vx=0
62                  vy=0
63                  w=0
64              end
65
66          end
67          message,auxiliaryData=sim.getSimulatorMessage()
68      end
69
70      w1=-(1/R)*vx+(1/R)*vy+((l+L)/R)*w
71      w2=(1/R)*vx+(1/R)*vy-((l+L)/R)*w
72      w3=-(1/R)*vx+(1/R)*vy-((l+L)/R)*w
73      w4=(1/R)*vx+(1/R)*vy+((l+L)/R)*w
74
75      sim.setJointTargetVelocity(fl,w2)
76      sim.setJointTargetVelocity(fr,w1)
77      sim.setJointTargetVelocity(rl,w3)
78      sim.setJointTargetVelocity(rr,w4)
79
80      end
81  end
```

```
82
83 function sysCall_cleanup()
84
85 end
```

A.5 Algoritmo PID e Funções Relacionadas

```
1  function saturacao(x,limite)
2      if x>limite then return limite end
3      if x<-limite then return -limite end
4      return x
5  end
6
7  function criar_PID(Kp,Ki,Kd,awl)
8      pid={}
9      pid.pre_erro2=0.0
10     pid.pre_erro=0.0
11     pid.pre_output=0.0
12     pid.integral=0.0
13     pid.a_windup_limite=awl
14     pid.Kp=Kp
15     pid.Ki=Ki
16     pid.Kd=Kd
17     return pid
18 end
19
20 function PID(pid, referencia, estado_atual, h)
21     local erro=referencia-estado_atual
22
23     local output=pid.pre_output+pid.Kp*(erro-pid.pre_erro)--
       Componente P
24
25     if (pid.Ki~=0.0) then -- Componente I
26         comp_int=pid.Ki*h*erro
27         if (pid.a_windup_limite~=0) then
28             comp_int=saturacao(comp_int, pid.a_windup_limite)
29         end
30         output=output+comp_int
31     end
32
33     if (pid.Kd~=0.0) then -- Componente D
34         comp_deriv=pid.Kd/h*(erro-2*pid.pre_erro+pid.pre_erro2)
35         output=output+comp_deriv
36     end
37
38     pid.pre_output=output
39     pid.pre_erro2=pid.pre_erro
40     pid.pre_erro=erro
41     output=saturacao(output,100) --Limite de velocidade da
       plataforma
```

```
    return output
end
```

A.6 Script para o Seguimento de Caminhos do Leader

```
function convers(teta)
    if(teta>0) then
        teta=teta
    else
        teta=teta+2*math.pi
    end
    return teta
end

function localizacao(id)
    if (id==0) then
        posicao=sim.getObjectPosition(ref, -1)
        orientacao=sim.getObjectOrientation(ref, -1)
    end
    if (id==1) then
        posicao=sim.getObjectPosition(refrobo, -1)
        orientacao=sim.getObjectOrientation(refrobo, -1)
    end
    orientacao[3]=convers(orientacao[3])
    posicao_orientacao={posicao[1],posicao[2],orientacao[3]}
    return posicao_orientacao
end

function sysCall_init()
    fl=sim.getObjectHandle('rollingJoint_fl')
    fr=sim.getObjectHandle('rollingJoint_fr')
    rl=sim.getObjectHandle('rollingJoint_rl')
    rr=sim.getObjectHandle('rollingJoint_rr')
    Discovery=sim.getObjectAssociatedWithScript(sim.handle_self)
    refrobo=sim.getObjectHandle('Dummy')
    ref=sim.getObjectHandle('Dummy0')

    sim.setJointTargetVelocity(fl,0)
    sim.setJointTargetVelocity(fr,0)
    sim.setJointTargetVelocity(rl,0)
    sim.setJointTargetVelocity(rr,0)

    R=0.05
    l=0.134
    L=0.134

    pidy=criar_PID(1000,500,0,0)
    pidx=criar_PID(1000,500,0,0)
    pidw=criar_PID(1000,500,0,0)
end
```

```
46
47 function sysCall_actuation()
48
49     h=sim.getSimulationTimeStep()
50
51     --angulo=sim.getJointPosition(pos)
52     --sim.setFloatSignal("angulo",angulo)
53
54     pref=localizacao(0)
55     probo=localizacao(1)
56
57     print(pref)
58     print(probo)
59
60     if(pref[3]>=0 and pref[3]<=math.pi/2 and probo[3]>=3*math.pi/2
       and probo[3]<=2*math.pi) then
61         pref[3]=pref[3]+2*math.pi
62     end
63
64     if(probo[3]>=0 and probo[3]<=math.pi/2 and pref[3]>=3*math.pi/2
        and pref[3]<=2*math.pi) then
65         pref[3]=pref[3]-2*math.pi
66     end
67
68     conty=PID(pidy,pref[2],probo[2],h)
69     contx=PID(pidx,pref[1],probo[1],h)
70     contw=PID(pidw,pref[3],probo[3],h)
71     vy=conty/153.85
72     vx=contx/153.85
73     w=-contw/40.93
74
75     a=math.cos(probo[3])+math.sin(probo[3])
76     b=math.sin(probo[3])-math.cos(probo[3])
77     c=-math.cos(probo[3])-math.sin(probo[3])
78
79
80     w1=(a/R)*vx+(b/R)*vy-((l+L)/R)*w
81     w2=(b/R)*vx+(c/R)*vy+((l+L)/R)*w
82     w3=(a/R)*vx+(b/R)*vy+((l+L)/R)*w
83     w4=(b/R)*vx+(c/R)*vy-((l+L)/R)*w
84
85
86
87     sim.setJointTargetVelocity(fl,w2)
88     sim.setJointTargetVelocity(fr,w1)
89     sim.setJointTargetVelocity(rl,w3)
90     sim.setJointTargetVelocity(rr,w4)
91 end
92
93 function sysCall_sensing()
94
```

```
95 end
96
97 function sysCall_cleanup()
98
99 end
```

A.7 Script do Controlo do Follower para a Estratégia 1

```
1  function sysCall_init()
2      fl=sim.getObjectHandle('rollingJoint_fl#0')
3      fr=sim.getObjectHandle('rollingJoint_fr#0')
4      rl=sim.getObjectHandle('rollingJoint_rl#0')
5      rr=sim.getObjectHandle('rollingJoint_rr#0')
6      fs=sim.getObjectHandle('ForceSensor#0')
7
8      R=0.05
9      l=0.134
10     L=0.134
11
12     vforca=0
13     vbinario=0
14
15     pidy=criar_PID(0.01,1.3,0,0)
16     pidx=criar_PID(0.01,4,0,0)
17     pidw=criar_PID(0.014,1.5,0,0)
18 end
19
20 function sysCall_actuation()
21
22     h=sim.getSimulationTimeStep() --Periodo de amostragem
23
24     result,vforca,vbinario=sim.readForceSensor(fs)
25
26     if (sim.boolAnd32(result,1)~=0) then
27         conty=PID(pidy, 0, vforca[2], h)
28         contx=PID(pidx, 0, vforca[1], h)
29         contw=PID(pidw, 0, vbinario[3], h)
30
31         --print(conty)
32
33         vy=-conty/153.85
34         vx=-contx/153.85
35         w=-contw/40.93
36
37         w1=-(1/R)*vx+(1/R)*vy+((l+L)/R)*w
38         w2=(1/R)*vx+(1/R)*vy-((l+L)/R)*w
39         w3=-(1/R)*vx+(1/R)*vy-((l+L)/R)*w
40         w4=(1/R)*vx+(1/R)*vy+((l+L)/R)*w
41
42         sim.setJointTargetVelocity(fl,w2)
```

```
        sim.setJointTargetVelocity(fr,w1)
        sim.setJointTargetVelocity(rl,w3)
        sim.setJointTargetVelocity(rr,w4)
    end
end

function sysCall_sensing()
    -- put your sensing code here
end

function sysCall_cleanup()
    -- do some clean-up here
end
```

A.8 Script do Controlo do Follower para a Estratégia 2

```

function sysCall_init()
    fl=sim.getObjectHandle('rollingJoint_fl#0')
    fr=sim.getObjectHandle('rollingJoint_fr#0')
    rl=sim.getObjectHandle('rollingJoint_rl#0')
    rr=sim.getObjectHandle('rollingJoint_rr#0')
    fs=sim.getObjectHandle('ForceSensor#0')

    jf=sim.getObjectHandle('Revolute_joint#0')

    R=0.05
    l=0.134
    L=0.134

    vforca=0
    vbinario=0

    pidy=criar_PID(0.008,0.9,0,0)
    pidw=criar_PID(3500,500,0,0)
end

function sysCall_actuation()

    h=sim.getSimulationTimeStep() --Periodo de amostragem

    posicaojf=sim.getJointPosition(jf)

    result,vforca,vbinario=sim.readForceSensor(fs)

    if (sim.boolAnd32(result,1)~=0) then
        conty=PID(pidy, 0, vforca[2], h)
        contw=PID(pidw, 0, posicaojf, h)

        --print(conty)
```

```
35
36          vy=(-conty*(math.sin((math.pi/2)-posicaojf)))/153.85
37          vx=(conty*(math.cos((math.pi/2)-posicaojf)))/153.85
38          w=-contw/40.93
39
40          print(vy)
41
42          w1=-(1/R)*vx+(1/R)*vy+((l+L)/R)*w
43          w2=(1/R)*vx+(1/R)*vy-((l+L)/R)*w
44          w3=-(1/R)*vx+(1/R)*vy-((l+L)/R)*w
45          w4=(1/R)*vx+(1/R)*vy+((l+L)/R)*w
46
47          sim.setJointTargetVelocity(fl,w2)
48          sim.setJointTargetVelocity(fr,w1)
49          sim.setJointTargetVelocity(rl,w3)
50          sim.setJointTargetVelocity(rr,w4)
51      end
52  end
53
54  function sysCall_sensing()
55      -- put your sensing code here
56  end
57
58  function sysCall_cleanup()
59      -- do some clean-up here
60  end
```

A.9 Script do Controlo do Follower para a Estratégia 3

```
1   function sysCall_init()
2       fl=sim.getObjectHandle('rollingJoint_fl#0')
3       fr=sim.getObjectHandle('rollingJoint_fr#0')
4       rl=sim.getObjectHandle('rollingJoint_rl#0')
5       rr=sim.getObjectHandle('rollingJoint_rr#0')
6       fs=sim.getObjectHandle('ForceSensor#0')
7       jf=sim.getObjectHandle('Revolute_joint#0')
8
9       R=0.05
10      l=0.134
11      L=0.134
12
13      vforca=0
14      vbinario=0
15
16      pidy=criar_PID(0.00008,0.009,0,0)
17      pidx=criar_PID(500,300,1,0)
18      pidw=criar_PID(3500,500,0,0)
19  end
20
21  function sysCall_actuation()
```

```
22
23      h=sim.getSimulationTimeStep() --Periodo de amostragem
24
25      posicaojf=sim.getJointPosition(jf)
26
27      result,vforca,vbinario=sim.readForceSensor(fs)
28
29      posicaojl=sim.getFloatSignal('angulo')
30      teta=posicaojl-posicaojf
31
32      if (sim.boolAnd32(result,1)~=0) then
33          conty=PID(pidy, 0, vforca[2], h)
34          contx=PID(pidx, 0, posicaojl, h)
35          contw=PID(pidw, 0, teta, h)
36
37          --print(conty)
38
39          vy=-conty*(math.sin((math.pi/2)-posicaojf))
40          vx=conty*(math.cos((math.pi/2)-posicaojf))+contx/153.85
41          w=contw/40.93
42
43          print(vy)
44
45          w1=-(1/R)*vx+(1/R)*vy+((l+L)/R)*w
46          w2=(1/R)*vx+(1/R)*vy-((l+L)/R)*w
47          w3=-(1/R)*vx+(1/R)*vy-((l+L)/R)*w
48          w4=(1/R)*vx+(1/R)*vy+((l+L)/R)*w
49
50          sim.setJointTargetVelocity(fl,w2)
51          sim.setJointTargetVelocity(fr,w1)
52          sim.setJointTargetVelocity(rl,w3)
53          sim.setJointTargetVelocity(rr,w4)
54      end
55  end
56
57  function sysCall_sensing()
58      -- put your sensing code here
59  end
60
61  function sysCall_cleanup()
62      -- do some clean-up here
63  end
```