NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

AFONSO MIGUEL GUIMARÃES SIMÕES DA ROSA ALVES

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

DIGITAL TWIN PARA CADEIRA DE RODAS ELÉTRICA

MESTRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Universidade NOVA de Lisboa
Setembro, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

DIGITAL TWIN PARA CADEIRA DE RODAS ELÉTRICA

AFONSO MIGUEL GUIMARÃES SIMÕES DA ROSA ALVES

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Orientador: Filipe de Carvalho Moutinho
Professor Auxiliar, Universidade NOVA de Lisboa

Coorientador: Luís Filipe dos Santos Gomes
Professor Associado com Agregação, Universidade NOVA de Lisboa

Júri:

Presidente: André Dionísio Bettencourt da Silva Rocha
Professor Auxiliar, Universidade NOVA de Lisboa

Arguente: João Almeida das Rosas
Professor Auxiliar, Universidade NOVA de Lisboa

Vogal: Filipe de Carvalho Moutinho
Professor Auxiliar, Universidade NOVA de Lisboa

MESTRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Universidade NOVA de Lisboa
Setembro, 2022

Digital Twin para cadeira de rodas elétrica

Dedico esta dissertação à minha família que sempre me apoiou incondicionalmente em todos os momentos do meu percurso académico e me proporcionou todos os recursos necessários para chegar a esta etapa da minha vida.

Agradecimentos

Quero começar por agradecer ao meu orientador, Professor Filipe de Carvalho Moutinho, ao meu coorientador, Luís Filipe dos Santos Gomes, e à Carolina Oliveira por todo o apoio, dedicação e disponibilidade durante o desenvolvimento deste projeto.

Em segundo, quero agradecer à minha família e aos meus amigos que estiveram sempre do meu lado durante todo o meu percurso académico, em particular à Alessia Offsas que esteve comigo em praticamente todos os trabalhos de grupo e me apoiou em todos os momentos.

Por último, um agradecimento à Maria Inês Costa que sempre me motivou e incentivou para alcançar os meus objetivos.

"Nobody can go back and start a new beginning, but anyone can start today and make a new ending." (Maria Robinson)

Resumo

Nos dias que correm, são notórias as várias limitações existentes para os utilizadores de cadeiras de rodas, nomeadamente a acessibilidade em espaços públicos, a capacidade financeira para adquirir um modelo que seja adequado às suas necessidades e a monitorização do estado dos componentes da cadeira.

Existem dezenas de milhões de utilizadores deste tipo de veículo espalhados por todo o mundo e uma grande percentagem reporta que tem, pelo menos, um acidente nos últimos três anos de utilização. A este facto, acresce a escassez de inovação no mercado do design e produção de cadeiras de rodas, o que faz com que haja uma necessidade de encontrar uma solução viável para estes problemas.

A solução proposta passa pela criação de um modelo digital, idêntico ao de uma cadeira de rodas elétrica, também conhecido por Digital Twin, cujo propósito é monitorizar o estado dos seus componentes e replicar os seus movimentos, com recurso a um simulador virtual. Para o modelo físico da cadeira, foi desenvolvido um protótipo com base no módulo EV3, da gama de robôs programáveis da LEGO, juntamente com o módulo da bússola CMPS11, integrado com o microcontrolador ESP8266. O modelo digital foi desenvolvido no ambiente ROS, com recurso ao simulador Gazebo para questões de visualização e simulação do modelo 3D, que por sua vez foi criado no software Blender. Para a monitorização, foi desenvolvida uma interface através da ferramenta Node-RED, a qual apresenta diversos gráficos com variações de inclinação e estado dos componentes da cadeira. A nível de armazenamento dos dados gerados, a interface dispõe ainda de uma ferramenta que os guarda num ficheiro CSV, para posterior análise.

Palavras-chave: Digital Twin, EV3, ev3dev, ROS, Gazebo, Node-RED, Cadeira de Rodas Elétrica, CMPS11, Monitorização

Abstract

It is notorious nowadays that there are several limitations for wheelchair users, namely accessibility in public spaces, the financial capacity to purchase a model that is suitable for their needs, and monitoring the condition of the chair's components.

There are tens of millions of wheelchair users around the world and a large percentage report having at least one accident in the last three years of use. To this fact, we can add the lack of innovation in the wheelchair design and production market, which makes it necessary to find a viable solution to these problems.

The proposed solution is the creation of a digital model, identical to an electric wheelchair, also called Digital Twin, whose purpose is to monitor the state of its components and replicate its movements, using a virtual simulator. For the physical model of the chair, a prototype was developed based on the EV3 module, from the LEGO programmable robots range, along with the CMPS11 compass module, integrated with the ESP8266 microcontroller. The digital model was developed in the ROS environment, using the Gazebo simulator for visualization and simulation of the 3D model, which was created in the Blender software. For the monitoring, an interface was developed through the Node-RED tool, which presents several graphs with variations of inclination and state of the chair components. In order to store the generated data, the interface also has a tool that saves it in a CSV file for later analysis.

Keywords: Digital Twin, EV3, ev3dev, ROS, Gazebo, Node-RED, Power Wheelchair, CMPS11, Monitoring

Índice

Índice de Figuras **xi**

Índice de Tabelas **xiv**

Siglas **xvi**

1 Introdução **1**
1.1 Contexto e Motivação 1
1.2 Objetivos 2
1.3 Estrutura 3

2 Conceitos e Tecnologias de Suporte **4**
2.1 Digital Twin 4
2.1.1 Vantagens e Aplicações 5
2.1.2 Criação e Estrutura 5
2.1.3 Entidade Física 6
2.1.4 Modelação Virtual 8
2.1.5 Gestão de Dados 10
2.1.6 Serviços 11
2.1.7 Comunicação 11
2.2 Ambiente ROS 13
2.2.1 Características 13
2.2.2 Arquitetura 14
2.2.3 Modelação e Simulação 15
2.2.4 Mapeamento e Localização 16
2.2.5 Integração com o Software Unity 17
2.3 Bases de Dados 18
2.4 Inteligência Artificial 19
2.4.1 Logística, PLN e Planeamento 19
2.4.2 Machine Learning 20

3 Sistema e Algoritmos Desenvolvidos **22**
3.1 Arquitetura 22

3.2 Entidade Física 23
3.2.1 Processo de Desenvolvimento do Protótipo 23
3.2.2 Programação do Módulo EV3 30
3.2.3 Sistema de Integração da Bússola CMPS11 42
3.3 Modelação Digital 46
3.3.1 Modelo 3D da Cadeira 46
3.3.2 Robot Operating System (ROS) 47
3.4 Interface de Monitorização 60
3.5 Comunicação entre os Elementos do Sistema 62
3.6 Armazenamento de Dados 63

4 Testes de Validação e Análise de Resultados **64**
4.1 Introdução 64
4.2 Operação do Sistema 64
4.2.1 Descrição 64
4.2.2 Resultados 65
4.3 Cenários Particulares 79
4.3.1 Aplicação de Força Externa para a Rotação das Rodas 79
4.3.2 Rotação da Cadeira Sobre os Eixos Longitudinal e Lateral 82

5 Conclusões **87**
5.1 Balanço Geral 87
5.2 Trabalho Futuro 88

Bibliografia **89**

Índice de Figuras

2.1 Características e operação de um Digital Twin 5
2.2 Estrutura de um Digital Twin proposta por Tao e Zhang (adaptado de [4]) . 6
2.3 Modelo F5 Corpus VS da Permobil e respetiva legenda (adaptado de [9]) . . 7
2.4 Cognição e controlo da entidade física 8
2.5 Tipos de modelação que podem ser feitos num Digital Twin 9
2.6 Ciclo de vida dos dados num Digital Twin 11
2.7 Funcionamento da comunicação síncrona (adaptado de [11]) 12
2.8 Funcionamento da comunicação assíncrona (adaptado de [11]) 13
2.9 Arquitetura da comunicação no ambiente ROS (adaptado de [21]) 15
2.10 Localização e mapeamento de um robô através de sensores 17
2.11 Comunicação do ambiente ROS com o software Unity (adaptado de [31]) . 18
2.12 Hierarquia conceitual da IA (adaptado de [33]) 19
2.13 Ciclo de vida de *Machine Learning* 21

3.1 Arquitetura de alto nível do sistema 23
3.2 Modelo 3D no qual se baseou a entidade física e o modelo virtual (retirado de [36]) 24
3.3 Sensores, motores e controlador remoto da gama EV3 25
3.4 Módulo programável EV3 25
3.5 Portas disponíveis no módulo programável EV3 26
3.6 Protótipo final correspondente à cadeira de rodas elétrica 27
3.7 Ligação da entidade física com os componentes da tecnologia EV3 27
3.8 Secções primárias da cadeira 28
3.9 Localização e direção rotação dos motores 28
3.10 Secção inferior do *chassis* da cadeira de rodas 29
3.11 Localização dos pesos adicionais para o equilibro da secção superior da cadeira 30
3.12 Menu apresentado no ecrã do módulo EV3 após a inicialização do sistema operativo ev3dev 32
3.13 Informação relativa ao sistema operativo instalado e à componente de hardware do EV3 32
3.14 Localização da driver do chip a ser substituída, no software WinSCP 33
3.15 Captura de ecrã do antes e depois da instalação da driver por SSH 33

3.16 Relação entre as classes MoveTank e MoveSteering (retirado de [46]) 37
3.17 Diagrama de classes UML correspondente às classes utilizadas no sistema . 37
3.18 Diagrama de atividades do programa desenvolvido 39
3.19 Algoritmos utilizados para as *threads* do motor do assento e apoio para costas 40
3.20 Algoritmo da *thread* responsável pela sincronização da posição dos motores das rodas dianteiras 41
3.21 Esquemático do sistema CMPS11 com NodeMCU V3 43
3.22 Estrutura do programa desenvolvido para o microcontrolador ESP8266 . . 44
3.23 Eixos da cadeira de rodas 45
3.24 Modelos 3D dos diferentes *links* da cadeira 47
3.25 Estrutura da "catkin" *workspace* do projeto 49
3.26 Estrutura pacote ROS do projeto 50
3.27 Estrutura do ficheiro XACRO 52
3.28 Áreas de colisão dos *links* do modelo da cadeira 52
3.29 Hierarquia dos *links* e *joints* do modelo 53
3.30 Representação gráfica das *joints* do modelo (vista lateral) 53
3.31 Tipos de controladores e respetivas *joints* utilizadas no modelo 55
3.32 Configuração dos ganhos dos controladores PID 56
3.33 Observação dos valores em diversos tópicos ROS presentes no sistema . . . 56
3.34 Envio de dados para um determinado tópico ROS do sistema 57
3.35 Envio da velocidade linear e angular para o tópico *cmd_vel* 57
3.36 Visualização dos nós e tópicos ROS do sistema 58
3.37 Modo de execução do programa de sincronização do modelo digital com a entidade física 59
3.38 Ambiente de programação Node-RED com "*flow*" de desenvolvimento da interface de monitorização 61
3.39 Interface desenvolvida para a monitorização do comportamento da cadeira 62
3.40 Estrutura da comunicação entre os elementos do sistema, através do protocolo MQTT 63

4.1 Entidades física e virtual nos seus estados iniciais 66
4.2 Interface de monitorização no estado inicial da cadeira 66
4.3 Estado das entidades física e virtual após execução do comando 1 67
4.4 Interface de monitorização durante a execução do comando 1 68
4.5 Estado das entidades física e virtual após execução do comando 2 69
4.6 Interface de monitorização durante a execução do comando 2 69
4.7 Estado das entidades física e virtual após execução do comando 3 70
4.8 Interface de monitorização durante a execução do comando 3 70
4.9 Estado das entidades física e virtual após execução do comando 4 71
4.10 Interface de monitorização durante a execução do comando 4 72
4.11 Estado das entidades física e virtual após execução do comando 5 73

4.12 Janela do *Plugin* Matplot durante a subida do assento da cadeira 73
4.13 Interface de monitorização durante a execução do comando 5 73
4.14 Janela do *Plugin* Matplot durante a descida do assento da cadeira 74
4.15 Interface de monitorização durante a execução do comando 6 74
4.16 Estado das entidades física e virtual após execução do comando 7 75
4.17 Interface de monitorização durante a execução do comando 7 75
4.18 Janela do *Plugin* Matplot durante a inclinação para trás do apoio para as costas 76
4.19 Estado das entidades física e virtual após execução do comando 8 77
4.20 Janela do *Plugin* Matplot durante a inclinação para a frente do apoio para as costas 77
4.21 Interface de monitorização durante a execução do comando 8 77
4.22 Estado das entidades virtual e física antes e depois da aplicação de uma força externa 80
4.23 Interface de monitorização durante a aplicação de uma força externa 81
4.24 Entidade fisica e interface de monitorização durante a inclinação da cadeira para trás 82
4.25 Entidade física e interface de monitorização durante a inclinação da cadeira para a frente 83
4.26 Entidade física e interface de monitorização durante a inclinação da cadeira para a direita 84
4.27 Entidade física e interface de monitorização durante a inclinação da cadeira para a esquerda 85
4.28 Notificações da passagem dos limites de rotação sobre os eixos lateral e longitudinal 86

Índice de Tabelas

2.1 NoSQL vs SQL 18

3.1 Descrição detalhada dos atributos e métodos utilizados 38
3.2 Ações associadas aos canais e botões do controlador remoto 42
3.3 Especificações da placa NodeMCU V3 43
3.4 Registos do módulo CMPS11 utilizados e respetivas funções 46
3.5 Exemplo de uma tabela gerada no formato CSV com os dados da cadeira . 63

4.1 Dados guardados no ficheiro CSV referentes aos comandos executados . . . 78
4.2 Dados guardados no ficheiro CSV durante a aplicação de uma força externa 81
4.3 Dados guardados no ficheiro CSV durante a inclinação da cadeira para trás 83
4.4 Dados guardados no ficheiro CSV durante a inclinação da cadeira para a frente 84
4.5 Dados guardados no ficheiro CSV durante a inclinação da cadeira para a direita 85
4.6 Dados guardados no ficheiro CSV durante a inclinação da cadeira para a esquerda 86

Índice de Listagens

3.1 Exemplo de importação de classes do módulo ev3dev 35
3.2 Exemplo de código para o controlo da cor dos LEDs com o sensor de toque 35
3.3 Exemplo de código para o controlo de um par de motores com MoveSteering 35
3.4 Código C++ que calibra a bússola CMPS11 com as definições de fábrica 45
3.5 Dependências do projeto incluídas no ficheiro XML package 50
3.6 Organização estrutural dos links do sistema 51
3.7 Implementação das transmissões do modelo virtual 54
3.8 Utilização do *rospy* para a ligação com os tópicos ROS 60
4.1 Posição e velocidade dos motores durante a execução do comando 1 . . 67
4.2 Posição e velocidade dos motores durante a execução do comando 2 . . 68
4.3 Posição e velocidade dos motores durante a execução do comando 3 . . 70
4.4 Posição e velocidade dos motores durante a execução do comando 4 . . 71
4.5 Posição e velocidade dos motores durante aplicação de força externa . . 80

Siglas

API Application Programming Interface

CSV Comma-separated Values

DT Digital Twin

IA Inteligência Artificial
IoT Internet of Things

JSON JavaScript Object Notation

MQTT Message Queuing Telemetry Transport

PID Proportional Integral Derivative

ROS Robot Operating System

SLAM Simultaneous localization and mapping
SSH Secure Shell

UML Unified Modeling Language
URDF Unified Robot Description Format
USB Universal Serial Bus

XML Extensible Markup Language

Capítulo 1

Introdução

Neste capítulo é introduzido o tema da dissertação, dando o contexto e motivação do problema, são descritos os objetivos pretendidos para o sistema e é apresentada a estrutura do documento.

1.1 Contexto e Motivação

Existem vários problemas associados às cadeiras de rodas elétricas e, dado que a maioria dos seus utilizadores têm capacidades físicas e/ou psicológicas limitadas, é necessário encontrar uma solução que se adapte a cada utilizador, consoante as suas dificuldades. Muitas destas dificuldades passam por questões básicas, como controlar a cadeira através do seu *joystick* ou fazer marcha-atrás. Porém o maior desafio dos utilizadores de cadeiras de rodas ocorre no contexto dos espaços públicos, nos quais existe muito pouca acessibilidade a estes veículos, como por exemplo, ir ao centro comercial, andar nos passeios, em transportes, etc [1]. Para além destes desafios, existem ainda problemas associados à componente técnica da cadeira, tais como o desgaste e necessidade de manutenção ou substituição de certos materiais, a incapacidade de compreensão de certas funcionalidades da cadeira por parte do seu utilizador, e a falta de sensibilidade que estes têm quando estão a manusear o *joystick*.

Todos estes problemas, para além de dificultarem a experiência da utilização da cadeira, podem resultar em acidentes e lesões graves. Cerca de 87% dos utilizadores de cadeiras de rodas reportam ter pelo menos uma queda nos últimos 3 anos, desde a primeira utilização. A cada ano, há cerca de 175 mil visitas às urgências causadas por lesões associadas a cadeiras de rodas, as quais podem surgir de pequenas elevações de 7 cm nos passeios, que são o suficiente para que uma cadeira com 126 kg caia [2]. A procura por

uma solução viável para este problema é suportada, não só pelas razões mencionadas anteriormente, mas também pelo facto de existirem cerca de 131 milhões de utilizadores de cadeiras de rodas, pelo mercado do design e produção deste produto ter vindo a estagnar ao longo das últimas décadas, pela falta de inovação, e por estarmos perante uma era em que as mais recentes tecnologias podem e devem assegurar a segurança, mobilidade e acessibilidade de todos os utilizadores. [3].

Uma das tecnologias que pode contribuir para a resolução de alguns destes problemas é o modelo Digital Twin (DT), o qual tem vindo a ser cada vez mais utilizado em diversas áreas. Este consiste na réplica digital de um objeto físico, capaz de monitorizar, prever, controlar e simular diversos processos e parâmetros do modelo real. Com o avanço das tecnologias de informação, como por exemplo Internet of Things (IoT), *cloud computing*, análise de *Big Data* e Inteligência Artificial (IA), a digitalização tem vindo crescer rapidamente, tornando-se um dos principais pilares de inovação de diversos sectores [4].

1.2 Objetivos

Dado o problema contextualizado, o objetivo desta dissertação é criar um DT de uma cadeira de rodas elétrica, nomeadamente os seguintes elementos que o constituem:

1. **Entidade Física** - Uma vez que não é possível obter uma cadeira de rodas elétrica para fins de investigação durante o período desta dissertação, um dos objetivos é a criação de um protótipo da cadeira que contenha todas as suas características básicas, como as rodas, um motor que faça subir e descer o assento e um motor para inclinar o apoio para as costas.

2. **Modelo Virtual** - Desenvolvimento de um modelo virtual, a três dimensões, da cadeira de rodas (correspondente ao protótipo), que consiga replicar o seu comportamento, em tempo real, com recurso a um simulador.

3. **Interface de Monitorização** - Pretende-se desenvolver uma interface de monitorização, que permita ao utilizador ou a um profissional acompanhar o estado dos componentes da cadeira de rodas, observar os ângulos de rotação sobre os eixos vertical, longitudinal e lateral, e alertá-lo para possíveis situações de perigo, tal como a ultrapassagem de limites predefinidos de inclinação, avarias, entre outros.

4. **Meio de Comunicação entre os Elementos do Sistema** - É imprescindível a inclusão de um meio de comunicação entre os elementos anteriormente descritos, de modo a sincronizar o comportamento da entidade física com o modelo virtual e atualizar o seu estado na interface de monitorização.

5. **Armazenamento de Dados** - O último objetivo consiste no armazenamento dos dados gerados pela entidade física para que estes sejam posteriormente processados ou pelo utilizador ou por um profissional.

1.3 Estrutura

Este documento é composto por seis capítulos, nomeadamente:

- **Capítulo 1 - Introdução:** Composto pela introdução do tema da dissertação. É dado o contexto e motivação do problema, os objetivos a cumprir e é resumida a estrutura do documento.
- **Capítulo 2 - Conceitos e Tecnologias de Suporte:** Composto pelo estado de arte do problema, nomeadamente tecnologias, ferramentas e conceitos de suporte à criação do sistema.
- **Capítulo 3 - Sistema e Algoritmos Desenvolvidos:** É abordada a arquitetura de alto nível do sistema e os algoritmos e soluções desenvolvidas para cada componente.
- **Capítulo 4 - Testes de Validação e Análise de Resultados :** São apresentados testes sobre o sistema desenvolvido, em diversos cenários, e analisados os respetivos resultados.
- **Capítulo 5 - Conclusões :** São apresentadas as conclusões da solução desenvolvida, com recurso aos testes executados.
- **Bibliografia:** São apresentadas todas as referências a documentos e *websites* utilizados para a realização do documento.

Capítulo 2

Conceitos e Tecnologias de Suporte

Neste capítulo é abordado o estado de arte do tema da dissertação, nomeadamente tecnologias e ferramentas de suporte à implementação do sistema, com o intuito de simplificar a interpretação do que será referido nos capítulos seguintes. Para além disso, são também referidas algumas tecnologias que não foram utilizadas no contexto deste projeto, no entanto podem ser uma mais valia para trabalhos futuros.

2.1 Digital Twin

Um DT é um modelo virtual dinâmico desenhado para replicar ao pormenor os aspetos necessários, num determinado contexto, de um objeto físico, processo ou serviço [4]. No contexto do tema da dissertação, foi desenvolvido um DT de uma cadeira de rodas elétrica, a partir da qual foram recolhidos dados em tempo real, através de sensores, de modo a monitorizar diversos aspetos da cadeira, tal como a posição e estado dos seus motores e a inclinação. Na figura 2.1, é possível observar a representação gráfica de um DT, incluindo as suas características e circulação dos dados.

A utilização deste tipo de modelação, com a integração de métodos de aquisição dados, análise de *Big Data*, IA e *Machine Learning*, pode permitir adquirir informação pertinente relativa a uma entidade física, podendo espelhar o seu comportamento e simulá-lo em tempo real, com o intuito de fazer levantamentos e tomadas de decisão mais precisos, melhorando o seu desempenho e eficiência [5]. Esta capacidade do DT conseguir acompanhar, em tempo real, o estado dos seus componentes, de rever dados históricos da sua utilização e de prever cenários de colisão e avarias, pode contribuir para uma monitorização contínua e possível controlo sobre os seus componentes, melhorando a experiência do utilizador.

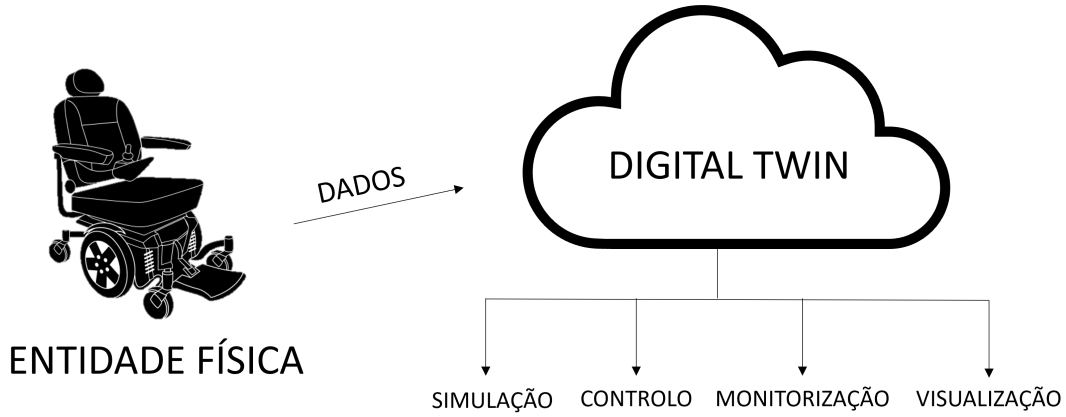


Figura 2.1: Características e operação de um Digital Twin

2.1.1 Vantagens e Aplicações

A utilização de um DT num determinado produto ou processo tem vários benefícios. Um deles assenta na qualidade e eficiência de investigação e design do mesmo, através da simulação de possíveis cenários de desempenho, suportados pelos dados recolhidos ao longo do tempo. Esta informação permite à entidade, que implementou o DT, tomar decisões, antes de colocar o produto no processo de fabrico ou até mesmo a operar. Outra vantagem é a questão da eficiência que, mesmo estando o produto em período de produção ou operação, é possível monitorizar todos os processos envolvidos nessas fases, de modo a otimizá-los.

Apesar das várias vantagens que o DT pode oferecer, nem todos os produtos ou serviços são complexos o suficiente ao ponto de requerem a sua integração, não apenas pelos custos, mas também pela falta de dados fornecidos ao modelo. No entanto, existem diversos sectores que beneficiam bastante com a implementação de um DT, como por exemplo a indústria automóvel, a produção e manufatura de diversos produtos, processos de construção e manutenção de edifícios, indústria da aviação, a medicina, o sector agrícola, etc. De um modo geral, a aplicação de um DT surge pela necessidade de regular o estado de um produto, processo, ou serviço, de modo a monitorizar o seu estado e performance. Isto é importante porque permite substituição de componentes que não estejam a corresponder às suas funções e até prever o seu desgaste a longo prazo [6].

2.1.2 Criação e Estrutura

Existem vários conceitos de Digital Twin, mas de uma forma geral, cada entidade adapta o seu conceito às suas necessidades, pelo que, podemos ter arquiteturas diferentes de modelo para modelo. A grande maioria dos investigadores defende que, para a criação de um DT, são necessárias três vertentes primárias: a física, correspondente ao objeto físico em

questão, a virtual, equivalente ao modelo digital, e a comunicações entre as duas últimas [7]. Todavia, com o avanço contínuo da tecnologia e aumento da complexidade das várias aplicações, os investigadores Tao e Zhang propuseram que um DT, para ser completo, necessita de ser composto 5 partes: a entidade física, o modelo virtual, os dados, os serviços e as comunicações entre os elas [8], como apresenta a figura 2.2. Esta estrutura abre portas à implementação de DT em áreas mais complexas, como a militar e a aeroespacial, as quais requerem soluções e modelos mais robustos e complexos.

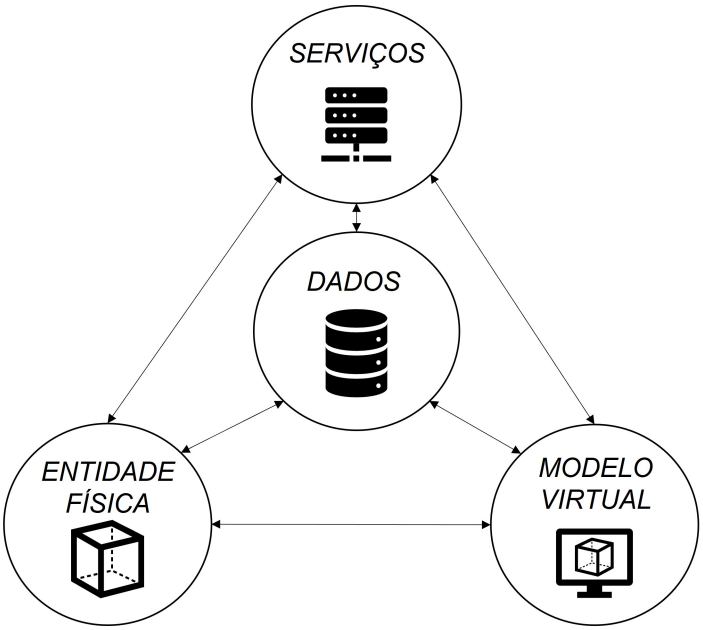


Figura 2.2: Estrutura de um Digital Twin proposta por Tao e Zhang (adaptado de [4])

Consideremos então que a base de um modelo Digital Twin é composta por cinco dimensões, de forma a abranger mais áreas tecnologias, cuja fórmula está representada na equação 2.1, onde DT significa Digital Twin, EF a entidade física, MV o modelo virtual, S os serviços, D os dados e C as comunicações entre cada componente [4].

Dado que a estrutura a cinco dimensões apresentada se adapta a várias aplicações e constitui ferramentas sólidas, dentro do que pode ser um DT, é nela que foi baseado o sistema desenvolvido sobre a cadeira de rodas elétrica. Cada componente do sistema é devidamente abordada no documento, apresentando as tecnologias utilizadas para o seu desenvolvimento e respetiva solução.

$$DT = (EF, MV, S, D, C) \tag{2.1}$$

2.1.3 Entidade Física

A entidade física representa a base de um DT, na medida em que é criado um modelo virtual sobre si, com o objetivo de simular, monitorizar e prever o seu comportamento. Esta entidade pode assumir a forma de um objeto físico, de um processo, de um sistema ou até mesmo de uma organização. De modo a obter um modelo mais preciso e próximo da

realidade, a componente física, em combinação com conhecimento, sensores e tecnologias de medição, é mapeada para o espaço virtual. No entanto, a modelação de um DT do mundo físico é deveras complexa e requer uma evolução gradual do modelo, de forma a corresponder à entidade em questão.

2.1.3.1 Cadeira de Rodas Elétrica Convencional

No contexto do tema da dissertação, a entidade física é uma cadeira de rodas elétrica. Existem diversos modelos de cadeiras de rodas elétricas e cada uma tem as suas características, no entanto, existem aspetos que todas têm em comum. Na figura 2.3, encontra-se a F5 Corpus VS, da Permobil [9], a qual serviu de referência para este projeto, por ser completa e apresentar as características e funcionalidades básicas de uma cadeira convencional. Todavia, dada a dificuldade de adquirir uma cadeira deste género, foi desenvolvido um protótipo de raiz, de menor dimensão, visualmente idêntico e com as mesmas características.

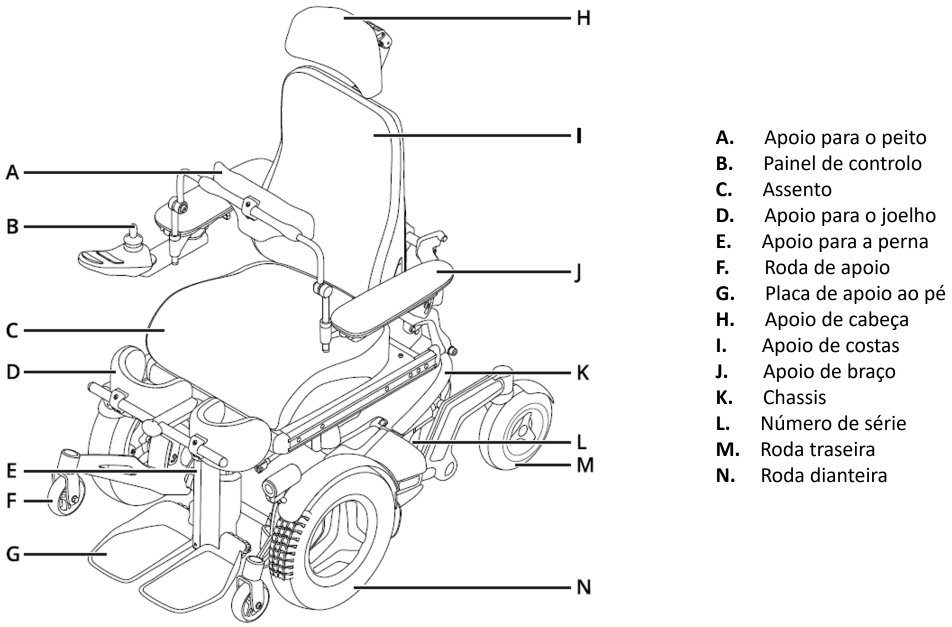


Figura 2.3: Modelo F5 Corpus VS da Permobil e respetiva legenda (adaptado de [9])

2.1.3.2 Cognição e Controlo

Para criar modelos DT fidedignos, é imprescindível a recolha de dados e a cognição do mundo físico, como ilustra a figura 2.4. O primeiro passo para o seu espelhamento passa pela medição dos seus parâmetros, tais como o tamanho, a forma, estrutura, tolerância, densidade, rigidez, posição, temperatura, humidade, etc. Algumas das tecnologias para a medição desses parâmetros incluem laser, reconhecimento de imagem, conversão e micro/nano precisão. Para além da cognição, existe ainda a componente de controlo do mundo físico. Quando é necessário que o objeto efetue determinadas funções, impostas

pelo sistema de controlo do DT, os seus atuadores são responsáveis pela execução das diversas ações. Este processo, dependendo da aplicação, envolve tecnologias de energias, como a elétrica, hidráulica e de combustíveis, de processamento, como o design, gestão e otimização, de controlo, através da rede, eletricidade e programação, e de transmissão. [4]

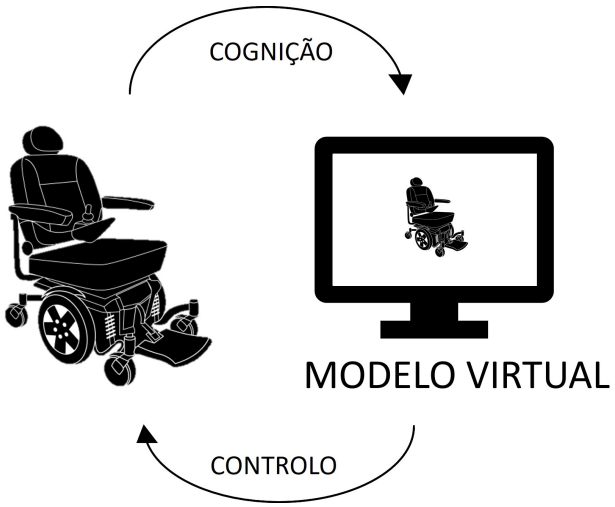


Figura 2.4: Cognição e controlo da entidade física

2.1.4 Modelação Virtual

2.1.4.1 Tipos de Modelação

Os modelos virtuais são réplicas idênticas de entidades físicas que reproduzem ao pormenor a sua geometria, comportamento, propriedades e regras, e que podem ser processadas, analisadas e geridas por computadores. A modelação geométrica, normalmente a três dimensões, caracteriza a forma, tamanho, tolerância e relação estrutural da entidade física. A modelação física, baseada nas propriedades físicas do objeto, como o tipo de material, desempenho e rigidez, reflete os fenómenos que lhe estão relacionados, como a deformação, corrosão e fracturação. A modelação comportamental representa os comportamentos e correspondentes mecanismos da entidade face às mudanças do ambiente externo. A modelação de regras fornece ao DT um conjunto de ferramentas como a avaliação, tomada de decisão e julgamento de determinados dados recolhidos por um especialista na matéria ou pelo historial de utilização da entidade. Estes processos de modelação têm por base recursos de dados, hardware e conhecimento, e podem ser observados na figura 2.5.

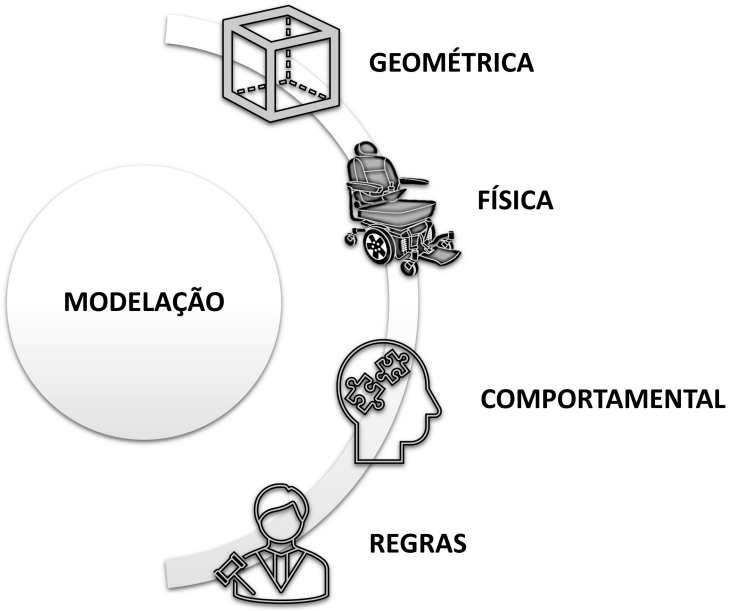


Figura 2.5: Tipos de modelação que podem ser feitos num Digital Twin

2.1.4.2 Tecnologias de Modelação

Existem várias tecnologias essenciais para a modelação do modelo virtual. A tecnologia de visualização é uma das mais importantes, uma vez que está na base da monitorização, em tempo real, da entidade física e dos processos. A precisão do modelo virtual é muito importante porque afeta diretamente a eficiência do DT e, para tal, o modelo necessita de passar por uma fase de validação, através de tecnologias de verificação, validação e acreditação (*VV&A - Verification*, *Validation & Accreditation*), e de otimização, através de algoritmos. O modelo VV&A envolve tanto métodos estáticos como dinâmicos. Os estáticos são utilizados para avaliar o aspeto estático da modelação e simulação, incluindo análises semânticas, estruturais e de controlo. Os métodos dinâmicos são utilizados para validar os aspetos dinâmicos da modelação e simulação, como a comparação de gráficos e testes de regressão. Tecnologias de evolução também são necessárias, visto que os modelos virtuais estão em constante evolução, a par da entidade física, e os dados vão sendo sempre atualizados em tempo real. As tecnologias de simulação permitem rapidamente diagnosticar falhas e outros problemas relativos à entidade física. A simulação de comportamentos físicos, por ser muito complexa, requer vários modelos, entre eles os modelos de estado, os dinâmicos, de avaliação e de problemas. Estes modelos podem ser desenvolvidos, por exemplo, com base em máquinas de estado, cadeias de Markov, redes de Petri, etc. Para a definição de regras, podem-se utilizar métodos como a notação lógica, representação orientada a objetos, semântica da web, Extensible Markup Language (XML), etc.

2.1.5 Gestão de Dados

A essência do DT passa muito pela questão dos dados, na medida em que estes são gerados pela entidade física e utilizados para diversas finalidades. O ciclo de vida dos dados é composto pelas fases de aquisição, armazenamento e análise. Na fase de aquisição, a informação pode surgir de várias fontes, como o mundo físico, incluindo os atributos estáticos e as condições dinâmicas, pelos modelos virtuais que produzem dados relativos às simulações, pelos serviços, ao invocar e executar uma determinada tarefa, e pelo conhecimento que é fornecido por especialistas na matéria. No armazenamento é necessário colocar todos os dados recolhidos numa base de dados capaz de lidar com *Big Data*, dado o enorme volume de informação a analisar. Por último, na fase de análise é feito o processamento, fusão e visualização dos dados. De seguida, constam algumas das tecnologias associadas a cada vertente do ciclo de vida dos dados:

- **Aquisição:** Os dados provenientes de hardware incluem os atributos estáticos e condições dinâmicas da entidade física. Para esta componente, várias tecnologias IoT são utilizadas, incluindo os códigos de barras, códigos QR, dispositivos de identificação por radiofrequência (RFID), câmaras, sensores, entre outros. A nível de software, os dados podem ser recolhidos através de Application Programming Interface (API) e bases de dados *open source*. Os dados da rede podem ser obtidos através de rastreadores web (*web crawlers*), motores de pesquisa e APIs públicos.

- **Armazenamento:** Para o armazenamento de dados é imprescindível a utilização de bases de dados. Estando a lidar com *Big Data*, o fluxo de informação que o modelo consome, em tempo real, requer um local onde se possa guardar todos esses dados para posterior processamento, análise e gestão. Dito isto, bases de dados como NoSQL, NewSQL, Cloud Storage e Distributed File Storage (DFS) podem ser algumas das soluções para o armazenamento, uma vez que permitem o acesso de vários utilizadores aos ficheiros e diretorias, conseguem escalar horizontalmente de forma a lidar com a vasta quantidade de dados e suportam sintaxes tradicionais de bases de dados como SQL e ACID.

- **Análise:** A análise dos dados é composta por três subpartes: o processamento, a fusão e a visualização. O processamento passa por extrair dados importantes e necessários para um dado fim, a partir da vasta quantidade de informação não processada que se tem na base de dados. Para tal, são utilizadas tecnologias que limpam, comprimem, reduzem e transformam esses dados de forma a termos um *dataset* limpo e conciso. De seguida, estes dados podem ser utilizados com a integração de IA, algoritmos de *Machine Learning*, *Deep Learning*, redes neuronais, métodos estatísticos, etc., de forma a fazer previsões, simulações e diagnósticos precisos no DT. A fusão dos dados consiste na filtragem, síntese, integração e correlação de dados provenientes de várias fontes. Alguns dos métodos e tecnologias utilizados nesta fase incluem também a IA, através de redes neuronais, *support vector machine* (SVM), teoria de

Wavelet, etc., como apresenta a figura 2.6. Por último, a visualização dos dados serve para apresentar toda a informação analisada de uma forma sucinta, intuitiva e interativa para o utilizador, a qual pode ser feita através de gráficos, histogramas, *dashboards*, entre outros.

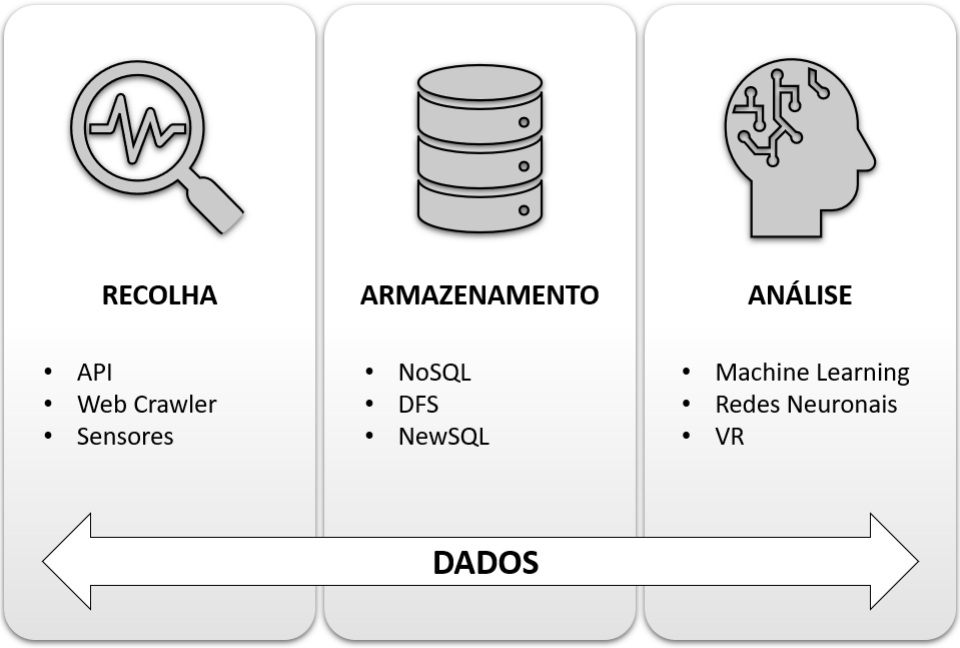


Figura 2.6: Ciclo de vida dos dados num Digital Twin

2.1.6 Serviços

A componente de serviços é fundamental para um DT, na medida em que fornece ferramentas para a simulação, verificação, monitorização, diagnóstico e previsão de dados. No contexto da cadeira de rodas, é importante perceber, por exemplo, como uma determinada pessoa utiliza a sua cadeira, perceber as dificuldades que esta tem na sua mobilização, prever qualquer falha que a cadeira venha a ter, entre outros. Esta informação, ao ser armazenada, pode ser integrada com serviços direcionados para o utilizador ou um profissional de saúde, os quais dispõem de diversas ferramentas de monitorização, simulação e previsão. A operação do DT requer o suporte contínuo dos serviços, quer sejam de software ou modelação, mantendo assim a dinâmica do modelo. Os serviços podem assumir vários formatos, sendo que uma possibilidade é o utilizador do DT ter acesso a uma aplicação com uma interface que dispõe de prognósticos, diagnósticos, simulações e monitorizações relativamente à cadeira e ao seu modo de utilização.

2.1.7 Comunicação

A última dimensão da estrutura do DT são as comunicações entre cada uma das suas vertentes, nomeadamente os dados, os serviços, a entidade física e o modelo virtual. São

as comunicações que permitem a troca de informação entre os elementos do modelo, de forma a dinamizá-lo e mantê-lo atualizado, em tempo real. No mundo digital em que vivemos, os protocolos de comunicação desempenham um papel fundamental na ligação entre os vários dispositivos IoT, na medida em que definem regras para a troca de informação entre o emissor e o recetor [10].

As comunicações envolvem tecnologias relacionadas com a *internet*, cibersegurança e com protocolos de comunicação. Estas tecnologias têm o papel de fornecer, em tempo real, a interação entre as diferentes componentes do DT e de o proteger de ataques informáticos. No caso de estarmos perante microcontroladores, se a troca de dados for feita em série, ou seja, uns a seguir aos outros, estamos perante protocolos de comunicação em série. Dentro deste tipo de protocolos, a comunicação pode ser feita de duas formas, tendo em conta o funcionamento do relógio dos dispositivos:

- **Comunicação Síncrona** - Ligação ponto a ponto de *"master"* para *"slave"*. Neste tipo de comunicação, todos os dispositivos utilizam um *bus* único do CPU (*Central Process Unit*) para partilhar dados e o relógio, como ilustra a figura 2.7, o que torna a transmissão mais rápida. Para além disso, o *baud rate*, o rácio a que os dados são transferidos entre o transmissor e o recetor, é o mesmo [11]. Alguns exemplos deste tipo de protocolo são o I2C, SPI, Universal Serial Bus (USB), CAN e Microwire.

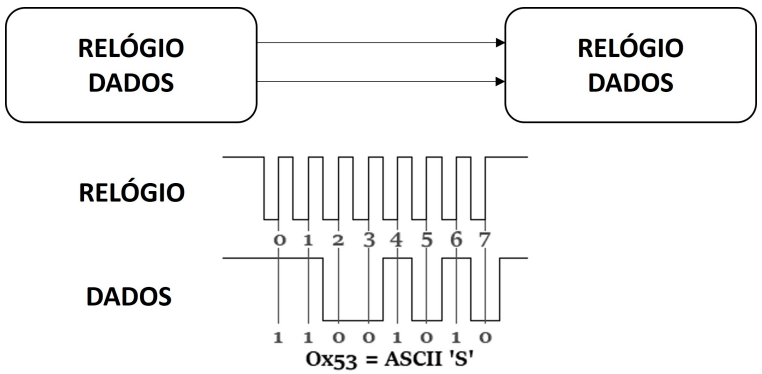


Figura 2.7: Funcionamento da comunicação síncrona (adaptado de [11])

- **Comunicação Assíncrona** - Este tipo é muito comum em aplicações de longa distância em que se quer uma ligação estável. A nível de funcionamento, o emissor envia o fluxo de dados e periodicamente insere um elemento de sinal, chamado de *flag*, para que seja possível distinguir onde começa (a informação de início da transmissão é o *start-bit*) e acaba (a informação de fim de transmissão é o *stop-bit*) o bloco de dados e qual a sua posição na sequência de dados transmitidos [12], como ilustra a figura 2.8. Exemplos deste tipo de comunicação são o RS-232, RS-422 e RS-485.

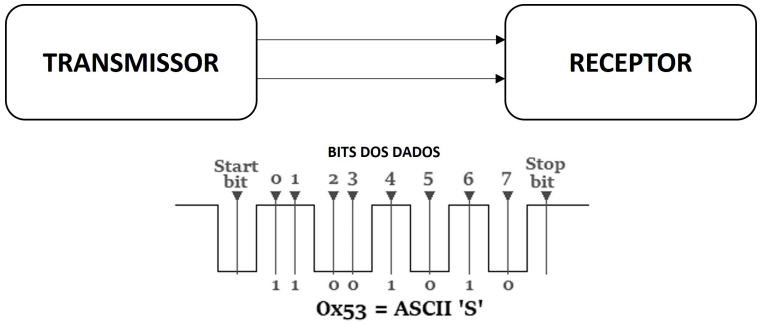


Figura 2.8: Funcionamento da comunicação assíncrona (adaptado de [11])

2.2 Ambiente ROS

Existem várias ferramentas possíveis para o desenvolvimento de um Digital Twin, nomeadamente a sua componente digital. A ferramenta utilizada nesta dissertação foi o Robot Operating System (ROS), um ambiente *open source* direcionado à programação de robôs que fornece vários serviços a diferentes tipos de projetos. O ROS integra tecnologias de controlo, desenvolvidas pela comunidade, e contém software reutilizável para a construção de um sistema de robôs, desde o controlador e modelos até à navegação autónoma. Por exemplo, algoritmos de sistemas de navegação e simuladores já foram implementados no projeto Player [13], algoritmos de visão no OpenCV [14], algoritmos de voz são suportados pelo Sphinx e algoritmos de planeamento no OpenRAVE [15]. O ROS permite que os seus programadores partilhem os seus algoritmos e módulos para que outros utilizadores possam reutilizar [16]. Na arquitetura de um Digital Twin, o ROS pode ser aplicado à modelação digital da entidade física, às comunicações entre duas ou mais vertentes do sistema, à criação de serviços e à visualização de dados.

2.2.1 Características

O ROS apresenta algumas características [17] que o distinguem face a outros ambientes de desenvolvimento:

- **Peer-to-peer:** *Peer-to-peer* ou P2P é uma arquitetura de redes de computadores onde cada um dos pontos ou nós da rede funciona tanto como cliente quanto como servidor, permitindo partilhas de serviços e dados sem a necessidade de um servidor central [18]. Um sistema desenvolvido utilizando ROS consiste em vários processos, normalmente a correr em anfitriões diferentes, ligados através de uma topologia *peer-to-peer* e que trocam mensagens entre si via tópicos/serviços.

- **Multi-lingual:** O ROS suporta várias linguagens de programação diferentes, tais como C++, Python, Octave e LISP. Cada nó comunica entre si através de mensagens escritas em XML-RPC.

- **Variedade de Ferramentas:** De modo a lidar com a complexidade do ROS, pequenas ferramentas são utilizadas para construir e executar os vários componentes. Estas, executam diversas tarefas, nomeadamente observar as ligações *peer-to-peer*, medir a largura de banda utilizada, fazer *plot* das mensagens trocadas entre cada nó e gerar documentação automaticamente.
- **Leve:** O termo "leve" deriva do facto das bibliotecas conterem toda a complexidade e não dependerem do ROS. Para utilizar essas bibliotecas no ROS basta importá-las em executáveis de menor dimensão.
- **Gratuito:** O código fonte do ROS está publicamente disponível e é distribuído sob os termos da licença Berkeley Software Distribution (BSD), o que permite o desenvolvimento de projetos comerciais e não comerciais.

2.2.2 Arquitetura

O ROS é um software com uma estrutura baseada em Linux, cujo objetivo é desenvolver sistemas de robôs. Esta estrutura utiliza os conceitos de pacotes, nós, tópicos, mensagens e serviços. O pacote principal de um projeto em ROS, também conhecido por "catkin" *package*, contem normalmente um ficheiro XML, que fornece informação acerca do pacote, e um ficheiro de texto denominado "CMAKELists" [19], que contém instruções a descrever os ficheiros fonte do projeto. Um nó não é nada mais do que um ficheiro executável dentro de um pacote ROS que utiliza uma biblioteca ROS para comunicar com outros nós. Estes podem publicar ou subscrever certos tópicos ou até mesmo fornecer ou utilizar um determinado serviço. Nestes tópicos pode ser publicado qualquer tipo de informação com a finalidade de ser subscrita por algum nó. Esta troca de informação é feita através do envio de mensagens, do mesmo tipo, entre os diversos nós, de forma a serem reconhecidas por ambos. Por último, os serviços permitem que os nós enviem um pedido ou recebam uma resposta de outros nós [20]. Na figura 2.9, é possível observar o funcionamento das comunicações no ROS, começando pelo nó que publica uma determinada mensagem e acabando nos nós que a subscrevem.

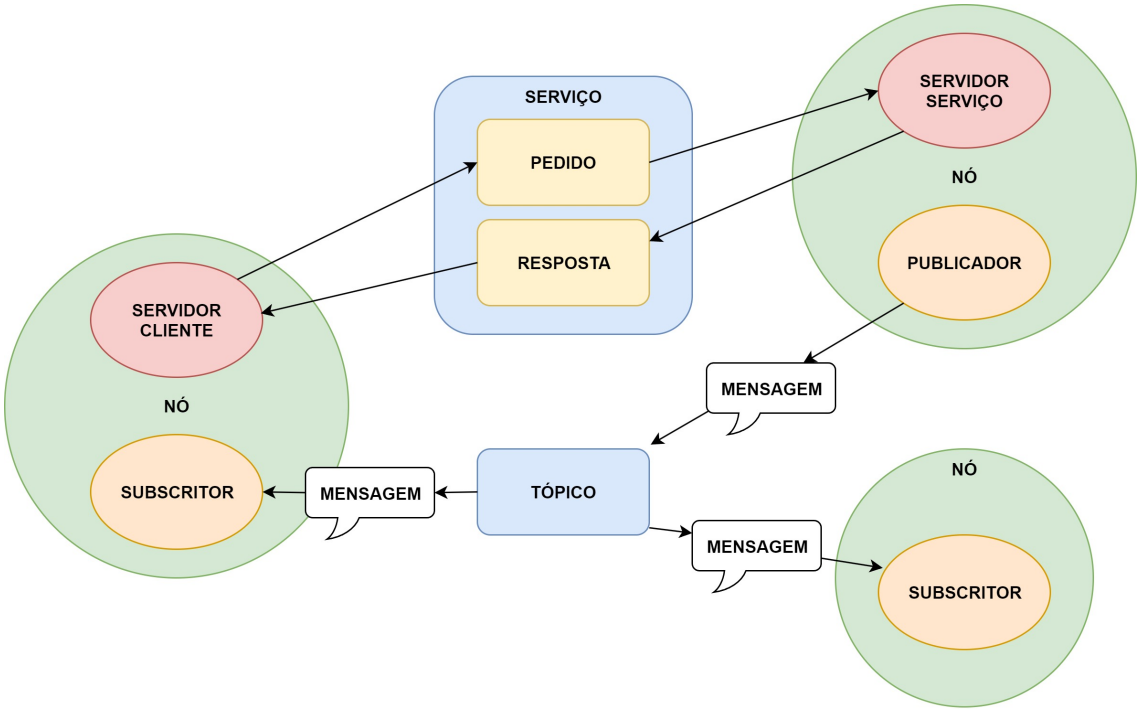


Figura 2.9: Arquitetura da comunicação no ambiente ROS (adaptado de [21])

2.2.3 Modelação e Simulação

O ROS proporciona diversas funcionalidades úteis ao desenvolvimento de robôs. A simulação é uma ferramenta essencial, visto que com ela é possível testar rapidamente algoritmos, desenhar robôs e realizar testes de regressão utilizando cenários realistas. Antes de carregar os ficheiros programados para os respetivos robôs, o ROS oferece dois simuladores, o RVIZ e o Gazebo. O RVIZ é utilizado pela sua simplicidade e ambiente 3D, enquanto o Gazebo é capaz de simular robôs em ambientes complexos, tanto no interior como no exterior. Com estas duas ferramentas, tem-se um motor de física bastante completo, gráficos de alta qualidade e ambientes de programação convenientes para realizar o projeto.

Para a criação de modelos sob a biblioteca do Gazebo, é necessário desenvolver um ficheiro no formato Unified Robot Description Format (URDF), partir do qual é feita toda a descrição física do objeto em questão. Na fase de pesquisa por projetos elaborados em ROS relacionados com cadeiras de rodas e Digital Twin, foram encontrados alguns que podem contribuir parcialmente em certos aspetos da implementação do DT, nomeadamente a modelação geométrica da cadeira através da exportação de modelos URDF já desenvolvidos.

De seguida, são enumerados os projetos considerados relevantes e a sua possível contribuição para a implementação do DT.

- **Wheelchair Automation** - Este projeto contém todos os pacotes e ficheiros necessários para a automatização de uma cadeira da PERMOBIL, incluindo o modelo

URDF, ficheiros de arranque, AMCL (um sistema de localização probabilístico para um robô a mover-se em duas dimensões), parâmetros de navegação para a cadeira e o pacote *gmapping*, com um laser baseado em Simultaneous localization and mapping (SLAM) [22].

- **Autonomous Wheelchair** - Pacote ROS para uma cadeira de rodas autónoma capaz de navegar sozinha em ambientes fechados utilizando sensores de Light Detection and Ranging (LiDAR) e Sonar. A cadeira dispõe de controlo manual, através de um *joystick*, caso a cadeira desvie do caminho pretendido [23].
- **can2RNET** - Repositório com o código e documentação necessários para a implementação de um sistema de controlo de uma cadeira de rodas elétrica do tipo R-NET, sob uma interface CAN numa Raspberry Pi [24].
- **SmartWheels** - Projeto que utiliza como base o repositório can2RNET cujo objeto é seguir alvos predefinidos e evitar obstáculos. Contém um circuito integrado na Raspberry Pi para a comunicação através de sensores ultrassónicos, os quais servem para evitar obstáculos e um escudo PiCAN2 para a comunicação com a cadeira de rodas [25].
- **Digital Twin** - Simulador de um Digital Twin em ROS de um manipulador UR10, capaz de fazer o planeamento inverso de trajetória cinemática, evitar obstáculos e ligar-se à rede [26].
- **Wheelchair URDF model** - Modelo URDF de uma cadeira de rodas exportado a partir do Solidworks, convertido para XACRO e posteriormente observado e controlado nos simuladores RVIZ e no Gazebo [27].

2.2.4 Mapeamento e Localização

Apesar de não terem sido utilizadas no âmbito deste projeto, o ROS dispõe de várias ferramentas para o planeamento de percursos e SLAM, uma abordagem que permite ao robô saber a localização de obstáculos à sua volta e planificar um percurso para os evitar. Este método é uma fusão de várias abordagens que permitem aos robôs navegarem no desconhecido. Para executar este método, o ROS possui um pacote denominado de *Navigation Stack*, que contém vários recursos para a planificação e mapeamento [28].

Um dos recursos presentes do *Navigation Stack* são os sistemas de localização, os quais permitem que um robô se localize no espaço, quer seja através de um mapa estático ou seja necessário SLAM. O AMCL, por exemplo, é uma ferramenta que permite ao robô localizar-se através de um mapa previamente criado. A desvantagem deste método é o facto do ambiente que rodeia o robô não poder sofrer qualquer alteração, uma vez que, caso isso aconteça, novos mapas teriam de ser criados, o que requer esforço e tempo. Posto isto, o *Navigation Stack* apresenta outros dois sistemas de localização: *gmapping* e *hector_mapping*.

Tanto o *gmapping* como o *hector_mapping* são baseados em SLAM, na ótica em que estão constantemente a gerar um mapa do ambiente que rodeia o robô, à medida que este se move, através da informação recolhida por sensores, tais como o LiDAR, e odometria, uma técnica de medição da distância percorrida. Desta forma, é possível obter constantes atualizações do mapa e contornar as mudanças do ambiente. A diferença entre os dois sistemas de localização é o facto do *gmapping* ter em conta a odometria para gerar e atualizar o mapa, tornando-o mais preciso. Na figura 2.10, é possível observar um diagrama de como funciona a localização e mapeamento de um robô, utilizando sensores. Para além disto, atua ainda sobre a pose do robô, a qual é estimada tirando partido da sua dinâmica, também denominada cinemática. A cinemática de um robô é influenciada por todos os componentes que garantem a sua movimentação, tais como o tipo de roda, número de rodas, posição das rodas e o ângulo em que estão dispostas.

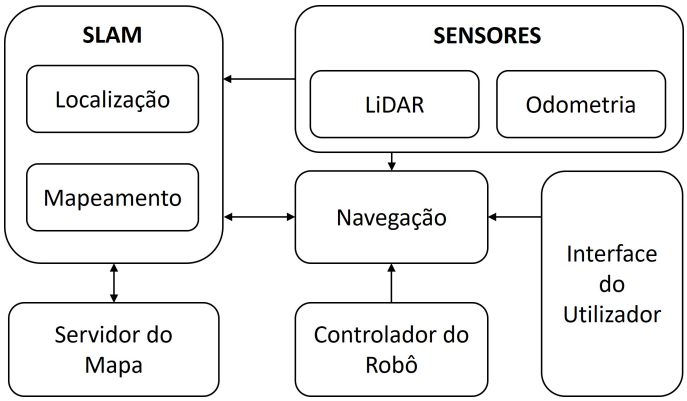


Figura 2.10: Localização e mapeamento de um robô através de sensores

2.2.5 Integração com o Software Unity

A simulação tem um papel muito importante no desenvolvimento de sistemas de robôs e ultimamente têm sido explorados diferentes ambientes para a sua execução. Recentemente, o Unity, um motor de desenvolvimento de jogos *cross-platform* [29], passou a suportar a componente de robótica para este tipo de simulações, incluindo um repositório com ferramentas, tutoriais, recursos e documentação [30]. A comunicação entre o ROS e o Unity, como apresenta a figura 2.11, é feita através de um *endpoint* Transmission Control Protocol (TCP) a correr como um nó, o qual facilita a passagem de mensagens entre os dois ambientes. Estas mensagens são serializadas através do *plugin* MessageGeneration, que é capaz de gerar classes na linguagem C#, incluindo funções de serialização e deserialização. O *plugin* ROSConnection fornece ao Unity ferramentas para publicar, subscrever ou chamar um serviço [31].

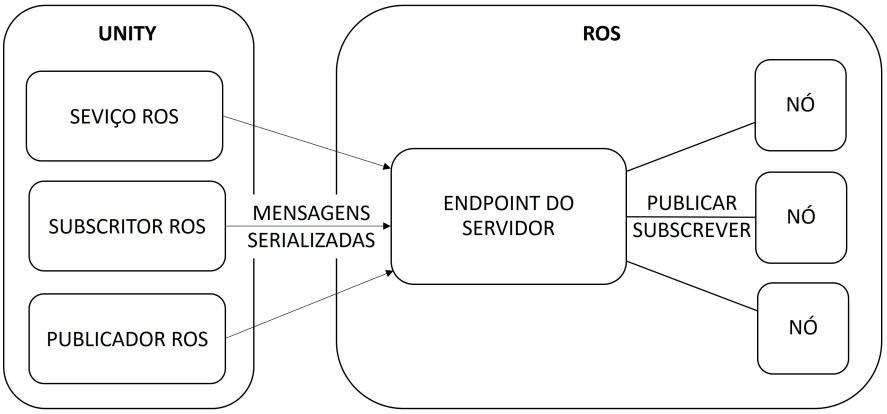


Figura 2.11: Comunicação do ambiente ROS com o software Unity (adaptado de [31])

2.3 Bases de Dados

Tal como foi dito anteriormente, é necessário armazenar a vasta quantidade de dados que são recolhidos pelos sensores, em tempo real. Para tal, existem, essencialmente, dois tipos de bases de dados: SQL e NoSQL. As primeiras foram as mais utilizadas nos últimos anos, porém, apenas suportam conjuntos de dados estruturados e com esquematização predefinida. Por outro lado, as bases de dados NoSQL são mais adequadas a conjuntos de dados não-relacionais e não estruturados, tal como *Big Data*, suportam diversas esquematizações e lidam com um elevado fluxo de dados através de *clusters* e servidores dedicados. A vantagem principal deste tipo de bases de dados é o facto de ser altamente escalável, dinâmico e consistente. Na tabela 2.1, encontra-se a comparação entre os diferentes tipos de bases de dados, consoante algumas das suas características.

Tabela 2.1: NoSQL vs SQL

Característica \ Tipo	SQL	NoSQL
Tipo	Relacional	Não Relacional
Dados	Estruturados em tabelas	Não estruturados em JSON
Esquematização	Estática	Dinâmica
Linguagem	Query estruturada	Query não estruturada
Escalabilidade	Vertical	Horizontal
Flexibilidade	Rígida	Não rígida

Dentro das bases de dados NoSQL, existem quatro tipos distintos que se destinam a diferentes finalidades. O primeiro tipo é denominado de *key-value* e é utilizado para guardar informação relativa à sessão de um utilizador, tal como o seu perfil, preferências e carrinho de compras, através de uma tabela indexada por uma chave. O segundo tipo, *document*, forma um documento, normalmente em formato JavaScript Object Notation (JSON), com múltiplos pares de chaves e valores e é utilizado para sistemas de gestão de

conteúdo, *blogs*, análise web, análise em tempo real, aplicações *e-commerce*, entre outros. O terceiro, *column-oriented*, armazena os dados em formato de colunas e cada chave é associada a múltiplos atributos e as suas aplicações são semelhantes às do tipo *document*. Por último, tem-se o tipo Graph que, como o nome indica, é direcionado para aplicações em que é necessária a modelação da estrutura dos dados, tal como redes sociais e motores de recomendações.

No contexto do modelo DT da cadeira de rodas, dado que estamos perante dispositivos IoT que adquirem um elevado fluxo de dados, em tempo real, o mais apropriado é utilizar uma base de dados NoSQL, pelo menos para a componente dos dados provenientes dos sensores. Para tal, existem duas ferramentas populares: MongoDB e CouchDB. Ambas são *open source* e desenhadas para escalar facilmente entre os diversos nós. A primeira é focada na consistência, na medida em que cada cliente tem sempre a mesma visão sobre os dados, enquanto a segunda é direcionada para a disponibilidade, permitindo aos clientes escrever e ler sobre os dados [32].

2.4 Inteligência Artificial

Uma das características principais do modelo Digital Twin é a previsão de parâmetros, situações e/ou falhas, relativas ao objeto físico. Esta previsão, bem como a otimização, tomada de decisão e processamento, podem ser feitos com base na IA, que pode ser subdividida em quatro partes: Logística, Processamento de Linguagem Natural, Planeamento e *Machine Learning*, como apresenta a figura 2.12.

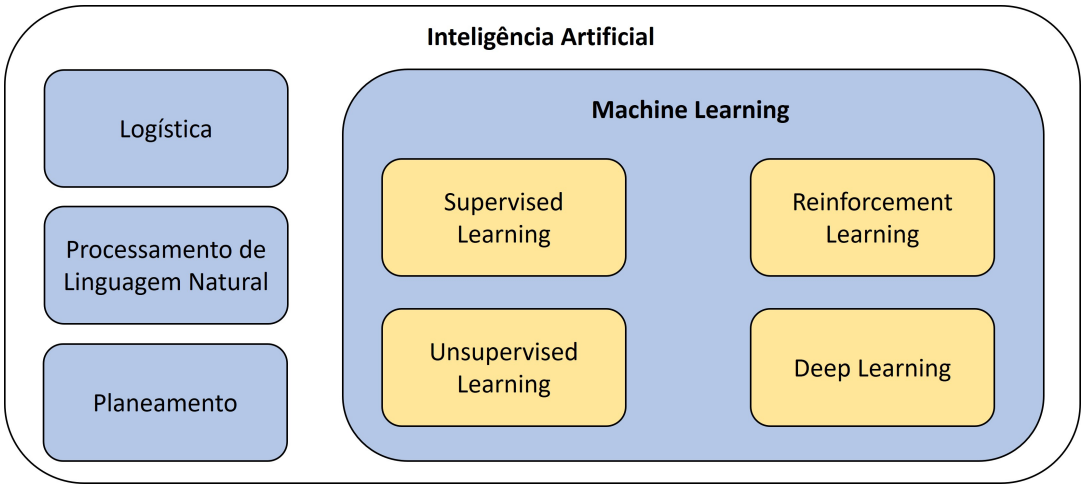


Figura 2.12: Hierarquia conceitual da IA (adaptado de [33])

2.4.1 Logística, PLN e Planeamento

A logística consiste no tratamento do *dataset*, na medida em que se completa os dados incorretos, inexistentes ou incompletos. Caso se verifique alguma destas situações com os dados e se proceda à aprendizagem, é provável que a previsão não seja precisa e,

consequentemente, a viabilidade do modelo fica comprometida. O PLN é a capacidade de treinar computadores para que estes entendam textos escritos ou o discurso humano, sendo assim uma das principais formas de comunicação entre a máquina e o utilizador. O planeamento consiste na automatização de sequências de ações, por parte do sistema inteligente, de modo a atingir um determinado objetivo. Ao contrário de utilizar *scripts* pré-programados, o planeamento automático é mais complexo e requer a adaptação do sistema face ao desafio contextualizado.

2.4.2 Machine Learning

Machine Learning é uma subcomponente da IA baseada em processos de treino e testagem de um determinado conjunto de dados, denominado *dataset*, com o objetivo de prever uma determinada variável. O seu ciclo de vida passa pela aquisição dos dados, seguido do seu processamento, pelo treino, avaliação e testagem do modelo e no final pela sua implementação, como ilustra a figura 2.13, adaptada de [34]. Os algoritmos de *Machine Learning* são utilizados para melhorar a precisão dos modelos preditivos [35] e, dada a sua aplicação, existem diversas abordagens dependendo do tipo e volume de dados. As quatro categorias de *Machine Learning* são as seguintes:

- **Supervised Learning** - Quando se está perante um cenário de *Supervised Learning*, normalmente tem-se um conjunto de dados já estabelecido e classificado. O objetivo é encontrar padrões nos dados que possam ser aplicados a processos de análise para prever um determinado resultado. Estes dados tipicamente contêm atributos que os definem e que se distinguem uns dos outros. Quando os atributos são contínuos está-se perante uma regressão e quando são finitos é uma classificação. A regressão essencialmente ajuda a compreender a correlação entre as variáveis do conjunto de dados. Os algoritmos são treinados com uma certa percentagem do *dataset* e a testagem é feita com a restante percentagem, de modo a avaliar a precisão do modelo.

- **Unsupervised Learning** - Adequado para situações em que se está perante um elevado número de dados sem atributos, como por exemplo aplicações de social media. Compreender o significado dos dados requer algoritmos que consigam classificá-los com base em padrões e *clusters*, uma vez que existem demasiadas variáveis em jogo.

- **Reinforcement Learning** - Modelo de aprendizagem com base no comportamento, na medida em que o algoritmo recebe *feedback* da análise dos dados para que o utilizador saiba qual o melhor resultado. *Reinforcement learning* difere dos dois modelos anteriores, uma vez que não é treinado com um conjunto de dados, mas sim através de tentativa e erro. O sistema aprende com sucessivas decisões, quer estejam certas ou erradas, de forma a maximizar a sua precisão e viabilidade.

- **Deep Learning** - Um subsequente método de *Machine Learning* que integra redes neuronais em camadas sucessivas, conduzindo à aprendizagem, através de dados, de uma forma interativa. Este método é utilizado particularmente quando se quer reconhecer padrões a partir de dados não estruturados. As redes neuronais são desenhadas para simular o modo como o cérebro humano funciona para que os computadores consigam ser treinados para lidar com problemas abstratos e que não sejam tão triviais.

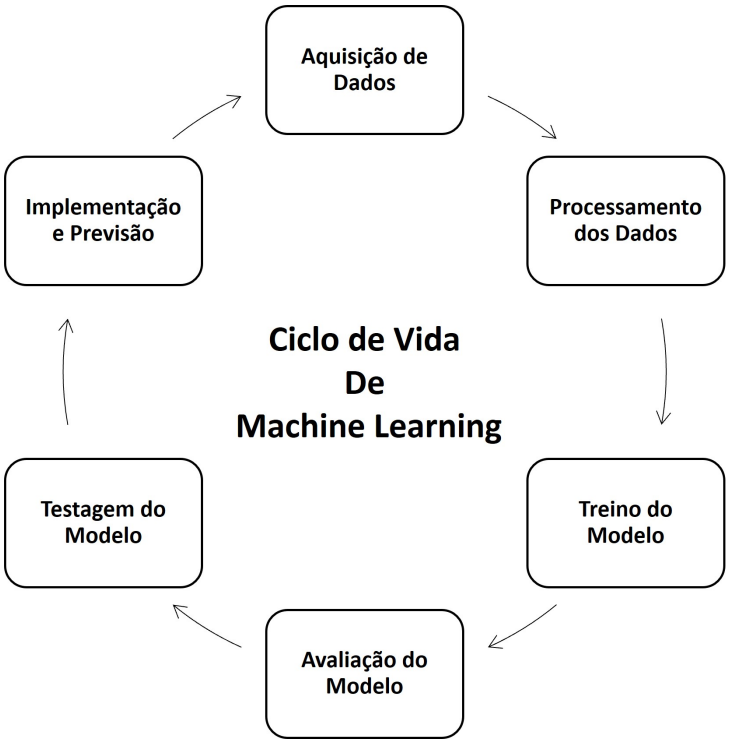


Figura 2.13: Ciclo de vida de *Machine Learning*

Capítulo 3

Sistema e Algoritmos Desenvolvidos

Neste capítulo é descrito e analisado o processo de desenvolvimento do sistema, desde as soluções propostas às tecnologias de suporte utilizadas para a sua implementação. São também apresentados os algoritmos desenvolvidos e a arquitetura geral do sistema, cujos elementos são devidamente abordados.

3.1 Arquitetura

A arquitetura de alto nível do sistema, apresentada na figura 3.1, é composta por cinco elementos principais: a entidade física, caracterizada pela cadeira de rodas elétrica; os dados que esta gera, recorrendo a sensores; o seu modelo digital, representado em três dimensões, com características físicas idênticas à cadeira real; a interface de monitorização, composta por diversas funcionalidades de processamento e monitorização de dados; e a comunicação entre todos os elementos. A nível operacional do sistema, a base está nos sensores instalados na cadeira de rodas elétrica que permitem a monitorização do comportamento dos seus componentes, em tempo real, tal como a posição dos motores. Esta informação é enviada através do protocolo de comunicação Message Queuing Telemetry Transport (MQTT), para diversos tópicos, que irão ser subscritos pelos restantes elementos do sistema. O modelo digital subscreve estes tópicos e utiliza os dados para espelhar a atividade física da cadeira, nomeadamente o seu deslocamento, o movimento vertical do assento e a inclinação do apoio para as costas. A interface de monitorização também subscreve os tópicos MQTT e apresenta os dados em formato de gráfico e permite a sua gravação em ficheiros Comma-separated Values (CSV), permitindo uma análise posterior, quer seja por um profissional ou pelo próprio utilizador da cadeira. Por último, a comunicação entre os elementos anteriormente descritos é, como já foi anteriormente dito, feita através do protocolo MQTT e na implementação do modelo digital são utilizados os nós

ROS. Na secção da comunicação são abordadas ambas as vertentes, de modo a dar uma melhor perceção do funcionamento do sistema.

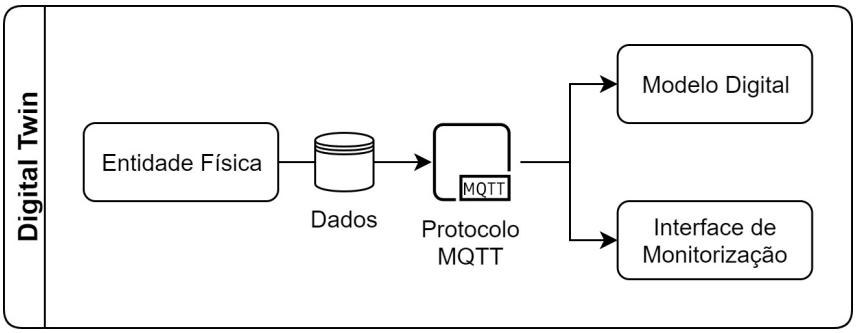


Figura 3.1: Arquitetura de alto nível do sistema

3.2 Entidade Física

Das componentes da arquitetura anteriormente referidas, a entidade física pode ser considerada como o "coração" do sistema, na medida em que está constantemente a "bombear" dados para serem processados e monitorizados pelo modelo digital e pela interface de monitorização. Para a representação da entidade física, o ideal, naturalmente, seria utilizar uma cadeira de rodas elétrica real, de modo a obter resultados mais precisos e coerentes. Não tardou muito para se chegar à conclusão que adquirir tal veículo, para fins de investigação durante intervalo de tempo desejado, seria uma tarefa muito complicada. Com este cenário em mãos, foi necessário explorar outras opções, sendo que a ideia passou sempre pela criação de um protótipo físico idêntico à cadeira de rodas elétrica, semelhante à da figura 2.3. A única dúvida seria quais os materiais a utilizar para a sua construção. O material escolhido foi, maioritariamente, peças da marca LEGO, uma vez que se tinha acesso a um vasto inventário das mesmas e já havia conhecimento prévio de algumas gamas que disponibilizassem componentes eletrónicos, tais como sensores, motores e módulos programáveis, os quais viriam a ser utilizados no desenvolvimento do protótipo. Para além disso, existe ainda uma grande comunidade que disponibiliza documentação e projetos relacionados com LEGO, algo que seria importante caso surgissem dúvidas e/ou dificuldades no decorrer do desenvolvimento do sistema.

3.2.1 Processo de Desenvolvimento do Protótipo

3.2.1.1 Design

O processo de desenvolvimento do protótipo começou pelo seu design, em que o objetivo principal seria ter um modelo convencional que se inclui todos os aspetos básicos de uma cadeira de rodas elétrica real, isto é, ser constituído por pelo menos quatro rodas, um assento que desse para subir e descer e um apoio para as costas conseguisse inclinar.

O modelo escolhido [36], no qual se baseou o protótipo e o modelo virtual, encontra-se representado, em 3D, na figura 3.2.

Figura 3.2: Modelo 3D no qual se baseou a entidade física e o modelo virtual (retirado de [36])

3.2.1.2 Gama de Robôs Programáveis Mindstorms EV3

Antes de partir para o desenvolvimento do protótipo propriamente dito, é importante salientar quais são os componentes disponíveis que a LEGO dispõe na sua gama de robôs programáveis, uma vez que será a partir destes que se vai construir o protótipo. A gama é denominada de Mindstorms e contém duas versões, a NXT e EV3, sendo que a última é a mais recente e pode ser vista como um *upgrade* da primeira, por possuir mais ferramentas e funcionalidades, o que permite o desenvolvimento de projetos mais complexos. A EV3 foi a gama escolhida para o projeto e, à semelhança da NXT, para além de disponibilizar diversos sensores, motores e um controlador remoto, como demonstra a figura 3.3, possui ainda um módulo programável, representado na figura 3.4, alimentado com seis baterias AA e com um ecrã e botões para navegar pelos programas instalados no cartão SD e alterar definições, que serve de centro de controlo e energia para o robô. As imagens dos dos componentes foram retiradas do manual de instruções [37].

(a) Sensor Infravermelho

(b) Motor Médio

(c) Motor Grande

(d) Controlador Remoto

Figura 3.3: Sensores, motores e controlador remoto da gama EV3

Figura 3.4: Módulo programável EV3

A nível de portas do módulo, tal como se pode observar pela figura 3.5, este possui uma face com quatro portas de saída para motores (portas A a D) e uma porta Mini-USB para ligação com o computador. Para além disso, dispõe ainda de: uma face com quatro portas de entrada para sensores (portas 1 a 4); uma face com um altifalante, a partir do qual saem os sons programados; uma face com uma porta para um cartão SD, o qual permite aumentar a memória disponível para a instalação de mais programas e uma porta USB Host que serve, por exemplo, para a ligação de um adaptador USB Wi-Fi, com a finalidade de conectar o módulo a uma rede sem fios.

(a) Portas de saída e porta Mini-USB

(b) Portas de entrada para sensores

(c) Altifalante

(d) Porta para cartão SD e porta USB Host

Figura 3.5: Portas disponíveis no módulo programável EV3

Os componentes anteriormente referidos, nomeadamente os sensores, motores e controlador remoto, possuem características técnicas relevantes que, numa fase mais avançada do sistema, são importantes a ter em conta. Posto isto, os detalhes técnicos de cada componente são os seguintes:

- **Motor Grande** - Motor com um sensor de rotação embutido com resolução de 1 grau para um controlo preciso. Este é otimizado para funcionar como a base de deslocamento do robô que se esteja a construir que, no caso deste protótipo, seriam as rodas dianteiras da cadeira. Este motor corre a 160-170 rpm (rotações por minuto), com um torque em funcionamento de 20 Ncm (Newton Centímetro) e um torque parado de 40 Ncm [37]. O torque em funcionamento representa a quantidade de binário que um dispositivo necessita para manter uma velocidade angular constante de um componente em rotação, quando em operação. Já o torque parado, é a quantidade de binário disponível do motor cuja velocidade rotacional de saída é zero [38].

- **Sensor Infravermelho** - Sensor digital que consegue detetar luz infravermelha refletida por objetos sólidos, nomeadamente o controlador remoto. Este sensor dispõe de três modos diferentes: o de proximidade, o guia e o modo remoto. Porém, para o protótipo só é utilizado o último, o qual comunica com o controlador.

- **Controlador Remoto** - Este controlador trabalha com duas baterias alcalinas AAA e comunica diretamente com o sensor infravermelho, na medida em que ao clicar num dos seus botões é enviado um sinal infravermelho que é lido pelo sensor. Como se pode observar pela figura 3.3d, o controlador possui dois botões vermelhos e azuis (cima e baixo), um botão central e um *switch* que muda o canal em que se encontra o comando. Existem quatro canais disponíveis, pelo que, caso se queira controlar diversos componentes com os mesmos botões, basta mudar o canal no botão *switch*.

3.2.1.3 Construção

O principal objetivo para a construção do protótipo da cadeira de rodas elétrica foi que este tivesse as mesmas funcionalidades e características face ao modelo da figura 3.2. Para tal, utilizaram-se os motores da gama EV3, sendo que dois motores grandes assumem o papel de rodas, um motor médio serve para o deslocamento vertical do assento e outro motor grande para a inclinação do apoio para as costas. Instalou-se ainda um sensor infravermelho, com a finalidade de controlar os motores, através do controlador remoto, e uma bússola CMPS11, a qual irá ser abordada com mais detalhe numa fase posterior do documento, para monitorizar possíveis inclinações e rotações da cadeira. Após um processo de construção gradual, tendo como base o módulo EV3, obteve-se o protótipo da figura 3.6. Para uma melhor perceção da sua constituição, a nível de motores e sensores da EV3, elaborou-se o esquema da figura 3.7, no qual estão representados os componentes utilizados e respetiva função.

Figura 3.6: Protótipo final correspondente à cadeira de rodas elétrica

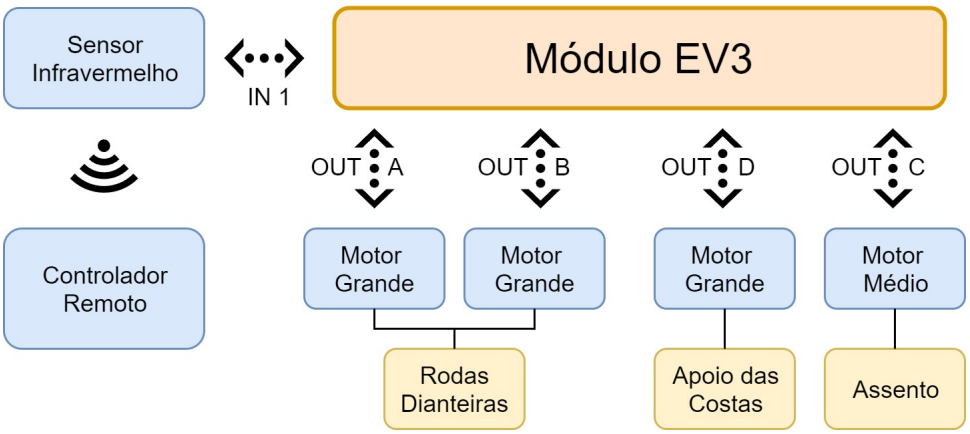
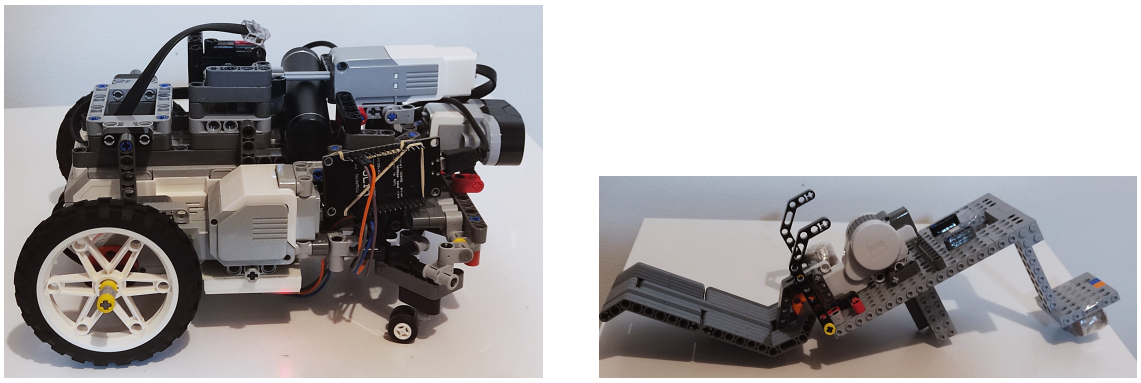


Figura 3.7: Ligação da entidade física com os componentes da tecnologia EV3

3.2.1.4 Logística Operacional

Com o intuito de compreender melhor as funcionalidades do protótipo, é importante ter conhecimento sobre a sua composição ao detalhe. Apesar de não ser distinguível na figura 3.6, a cadeira é constituída por duas secções primárias, como demonstra a figura 3.8, sendo que a figura 3.8a representa a secção inferior, que constitui as quatro rodas e o motor, e a figura 3.8b a secção superior, constituída pelo assento e apoio para as costas.



(a) Secção inferior da cadeira (b) secção superior da cadeira

Figura 3.8: Secções primárias da cadeira

Começando pela secção inferior da cadeira, esta é composta por dois motores grandes, que estão diretamente ligados às rodas e são responsáveis pela sua rotação, e um motor médio, cuja funcionalidade é rodar uma peça que faz com que a secção superior suba, até um determinado limite, ou desça até à base inferior. O terceiro motor grande encontra-se sobre o assento da cadeira, na secção superior, como é possível observar pela figura 3.8b, cuja função é inclinar o apoio das costas para trás ou para a frente, dentro dos limites definidos. Na figura 3.9 estão representados, com uma seta laranja, os eixos de rotação dos motores anteriormente descritos, os quais estão identificados por um retângulo azul. As suas direções de deslocação também estão representadas com uma seta amarela.

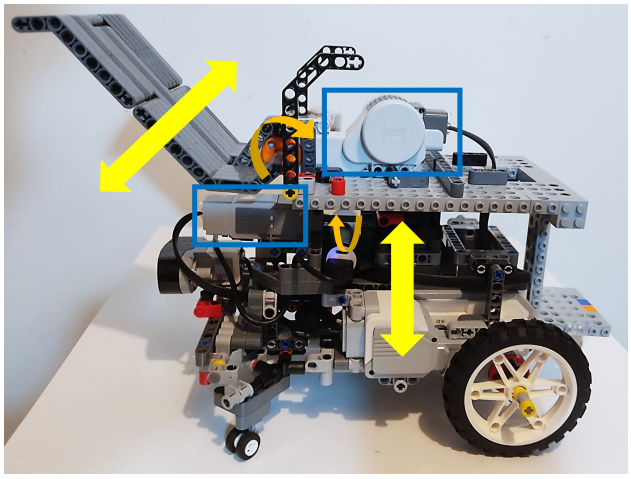


Figura 3.9: Localização e direção rotação dos motores

3.2.1.5 Extras e Compensações

Durante a fase de construção, foram ultrapassados diversos desafios, nomeadamente questões relacionadas com o peso da cadeira, a rigidez das rodas de apoio traseiras, a gestão de cabos, a logística dos motores e a posição dos componentes. Anteriormente, foi mencionada uma bússola, a qual também se encontra instalada no veículo, e que não dispensa do acompanhamento de uma placa de desenvolvimento, uma vez que é fundamental fornecer tensão para o seu funcionamento e processar os valores lidos pela bússola. A placa utilizada foi a NodeMcu V3, que integra um microcontrolador ESP8266, cuja alimentação foi feita com recurso a uma *power bank*. Este sistema constituído por três componentes (fora os cabos de ligação e uma *breadboard*) foi colocado de forma discreta no protótipo, em que a bússola, inserida numa *breadboard*, encontra-se colada na secção inferior do *chassis*, como se pode observar na figura 3.10. A placa NodeMcu foi colocada na parte lateral da cadeira, com o auxílio de elásticos, enquanto que a *power bank* foi inserida no seu interior. Estes dois últimos componentes encontram-se visivelmente instalados na figura 3.8a, correspondente à secção inferior.

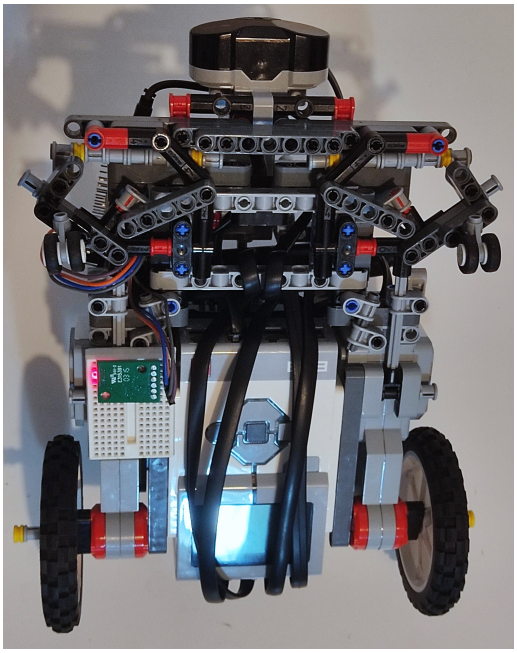


Figura 3.10: Secção inferior do *chassis* da cadeira de rodas

Um dos problemas encontrados durante a montagem do protótipo está associado ao peso da secção superior, correspondente à figura 3.8b, que se desequilibrava durante a rotação do motor médio, responsável por subir e descer o assento. Este cenário acontecia, pois existia uma discrepância no balanço do peso do assento, o que provocava a sua inclinação para um dos lados, impedindo o correto funcionamento do motor. Com o intuito de equilibrar o peso, foram colocadas duas pilhas de moedas em dois locais estratégicos, assinalados na figura 3.11 com um círculo vermelho.

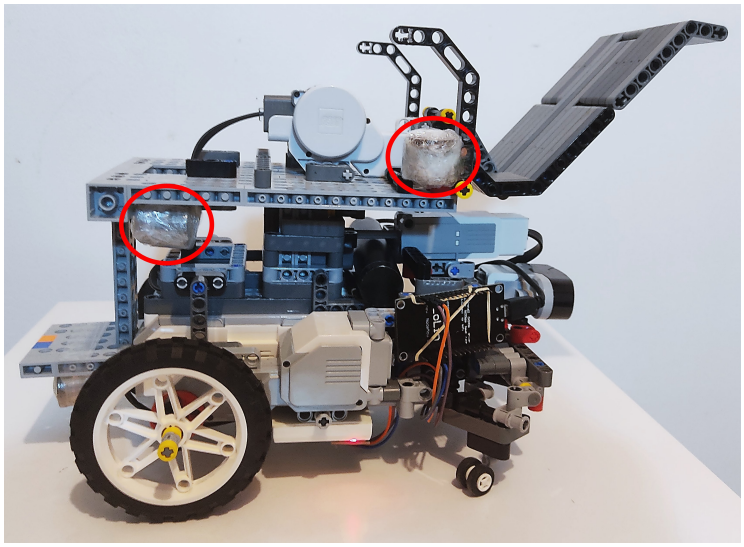


Figura 3.11: Localização dos pesos adicionais para o equilibro da secção superior da cadeira

Por último, mas não menos importante, o protótipo contém também um adaptador USB Wi-Fi EDIMAX EW-7811Un para a conexão a uma rede, sem fios. Apesar deste ser o adaptador recomendado pela LEGO para se utilizar no módulo EV3 [39] encontraram-se alguns problemas de compatibilidade que irão ser abordados na secção seguinte.

3.2.2 Programação do Módulo EV3

3.2.2.1 Sistema Operativo

Esta fase de desenvolvimento do sistema foi a que suscitou mais incertezas e obstáculos face à solução que se deveria implementar. Um dos objetivos delineados inicialmente, foi que a entidade física conseguisse comunicar de alguma forma com o modelo digital, de modo a possibilitar uma sincronização constante, em tempo real, dos valores lidos pelos sensores dos motores. Apesar da LEGO disponibilizar um software *low-code* para programar o módulo EV3, este não permite a utilização do Wi-Fi de uma forma eficaz, ou seja, que consiga fazer a ligação da entidade física com o modelo digital, estando o último desenvolvido na *framework* ROS, num computador na mesma rede. Posto isto, a solução passou sempre pela instalação de um novo sistema operativo, que permitisse a comunicação entre os dois modelos, preferencialmente através de protocolos de comunicação, recorrendo a módulos e bibliotecas disponíveis na linguagem em que se estivesse a programar. A LEGO já disponibiliza um software dessa natureza, com respetiva documentação e API, que permite o desenvolvimento de ficheiros em diversas linguagens de programação, tais como Python, Java e C++, nos quais se podem controlar diversos componentes do robô, tais como os motores e os sensores [40]. Uma particularidade interessante deste sistema operativo é o facto de ter sido baseado noutro desenvolvido pela comunidade, denominado ev3dev. No espaço temporal em que se estava a desenvolver

o protótipo, o software oficial de LEGO ainda não se encontrava disponível, pelo que se utilizou a versão não oficial: ev3dev.

3.2.2.2 Descrição

O ev3dev é um sistema operativo baseado em Debian Linux que corre em várias plataformas compatíveis com a gama de robôs programáveis Mindstorms, incluindo a EV3. O ev3dev possibilita o desenvolvimento de programas mais complexos e versáteis, face ao software *low-code* original da LEGO, uma vez que contém vários módulos disponíveis em diversas linguagens de programação e um vasto leque de projetos desenvolvidos pela comunidade [41].

3.2.2.3 Instalação

A instalação do sistema operativo ev3dev no módulo EV3 requer tanto a utilização de um cartão SD com uma capacidade mínima de 2GB, uma vez que é necessário instalar uma imagem do sistema com cerca de 2GB, como também um computador com um adaptador para o inserir. De modo a guardar vários projetos, o ideal seria um cartão de 16 ou 32GB, sendo que o último foi utilizado no projeto. Para instalar efetivamente o ev3dev no módulo EV3 foi preciso seguir os seguintes passos, por ordem, como consta em [42]:

1. **Descarregar imagem do sistema operativo ev3dev** - Descarregou-se a imagem da última versão do ev3dev a partir do *website* [42].

2. **Flash da imagem no cartão SD** - Fez-se flash da imagem descarregada no cartão SD através do software balenaEtcher. Este software estava recomendado na página de instalação do ev3dev e é especializado em fazer flash de imagens de sistemas operativos tanto em cartões SD como *pens* USB.

3. **Inicialização do ev3dev** - Ligou-se o módulo EV3, com o cartão SD inserido, e aguardou-se alguns minutos para a instalação de algumas drivers, até que o ecrã mostrasse o menu da figura 3.12. No canto superior direito encontra-se o estado da bateria do EV3, em V (volt), sendo que o máximo são 8 V e o mínimo 5 V. Os três primeiros itens do menu são os mais relevantes e os que foram utilizados no desenvolvimento do projeto. Em "File Browser", tem-se acesso a todos os ficheiros instalados, em "Device Browser" é possível ter acesso a todos os sensores e motores ligados ao módulo e em que portas se encontram e em "Wireless and Networks" é possível fazer a ligação a qualquer dispositivo ou rede, quer seja por Bluetooth, Wi-Fi ou USB.

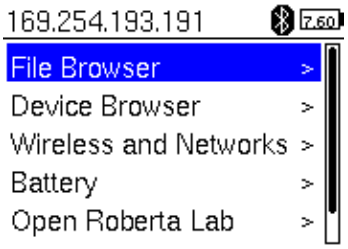


Figura 3.12: Menu apresentado no ecrã do módulo EV3 após a inicialização do sistema operativo ev3dev

4. **Ligação ao computador** - A ligação ao computador pode ser feita de diversas formas, tais como Wi-Fi, cabo USB, Ethernet, Tethering ou Bluetooth. Numa primeira instância, dado que não estava inserido nenhum adaptador Wi-Fi no EV3, optou-se por fazer a ligação através de Bluetooth. Esta ligação abre portas à conexão via Secure Shell (SSH), o que permite verificar se o módulo está corretamente ligado, correr comandos no terminal de forma segura a partir do computador, executar e instalar novos programas e editar configurações. Não foi necessário recorrer a nenhum software adicional para executar comandos no terminal para a ligação SSH, uma vez que o IDE utilizado para programar o módulo, denominado VS Code, já inclui uma extensão com essa finalidade. A informação relativa ao sistema instalado e à componente de hardware do módulo EV3 encontra-se representada na figura 3.13.

```
========== ev3dev-sysinfo ==========
Image file:         ev3dev-stretch-ev3-generic-2020-04-10
Kernel version:     4.14.117-ev3dev-2.3.5-ev3
Brickman:           0.10.3
BogoMIPS:           148.88
Bluetooth:          2.1
Board:              board0
BOARD_INFO_HW_REV=7
BOARD_INFO_MODEL=LEGO MINDSTORMS EV3
BOARD_INFO_ROM_REV=6
BOARD_INFO_SERIAL_NUM=00165342B2D3
BOARD_INFO_TYPE=main
```

Figura 3.13: Informação relativa ao sistema operativo instalado e à componente de hardware do EV3

3.2.2.4 Comunicação com o Modelo Digital

A base da ligação do módulo EV3 com o modelo digital, desenvolvido no ambiente ROS num computador, passa pela utilização de uma rede sem fios, através do Wi-Fi. Apesar ter sido adquirido o adaptador Wi-Fi recomendado pela LEGO, o Edimax EW-7811Un, este contém duas versões, com dois chips distintos. Uma delas não é, inicialmente, compatível com o módulo EV3 (a que contém o chip 8188eu), o que resultou que quando se selecionava a opção das redes, no menu do EV3, o Wi-Fi não estava ligado e não existia nenhuma rede Wi-Fi disponível, como é possível observar pela figura 3.15a. Posto isto, foi necessário recorrer à ligação SSH com o EV3 de modo a executar alguns comandos para

instalar os drivers do adaptador, para que o EV3 o reconhecesse. Para tal, seguiram-se os seguintes passos, como consta em [43]:

1. Descarregou-se a driver correspondente ao chip 8188eu no formato ZIP.
2. Ligou-se o computador ao módulo EV3 por SSH, recorrendo ao software WinSCP. Este software permite manipular o conteúdo do módulo, desde apagar, copiar e descarregar ficheiros.
3. Uma vez ligado por SSH, substituiu-se, no WinSCP, a driver previamente instalada pela nova driver descarregada, tal como demonstra a figura 3.14.

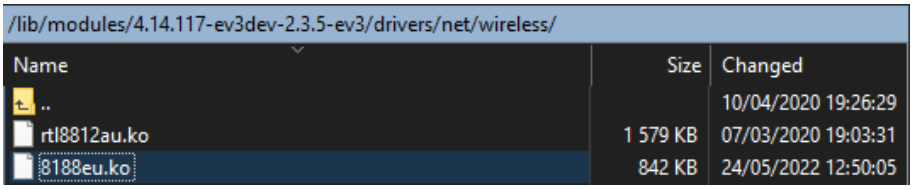
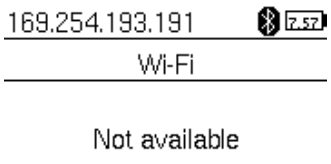
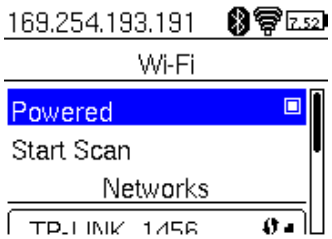


Figura 3.14: Localização da driver do chip a ser substituída, no software WinSCP

4. Reiniciou-se o módulo EV3 e executou-se, por SSH, o comando *sudo modprobe 8188eu*.
5. Ligou-se com sucesso à rede Wi-Fi desejada, como demonstra a figura 3.15b. É possível verificar no canto superior direito que, de facto, aparece um ícone com o símbolo do Wi-Fi, bem como opções como "Powered", o que significa ligado e "Start Scan" para procurar redes disponíveis.



(a) Opção Wi-Fi não disponível



(b) Opção Wi-Fi ligada com sucesso

Figura 3.15: Captura de ecrã do antes e depois da instalação da driver por SSH

Assim que se conectou o módulo a uma rede sem fios, surgiu a questão de qual seria a forma mais eficaz e com menos atraso de resposta para fazer a comunicação com o modelo digital. Inicialmente, a ideia foi integrar a *framework ROS* diretamente com o módulo EV3, através da instalação do *ros_comm*, um pacote de comunicação que permite que os nós ROS corram diretamente no EV3, desde que haja um *roscore* a correr noutro computador na mesma rede. O *roscore* é responsável por inicializar todos os comandos dentro do pacote *ros_comm* e é o que permite fazer a comunicação entre cada nó. Desta forma seria possível partilhar os mesmos nós com o modelo digital e sincronizar os valores

dos sensores dos motores, em tempo real. Apesar de promissora, esta solução mostrou-se bastante complexa e pouco viável, na medida em que houve falta de documentação sobre o tema, a versão do ROS era diferente daquela instalada no computador (noetic) e o facto do projeto base, a partir do qual surgiu esta ideia [44], ainda estar em processo de desenvolvimento.

A solução adotada passa pela utilização do protocolo de comunicação MQTT, para enviar e receber mensagens entre o módulo EV3 e o modelo digital, desenvolvido em ROS. O conteúdo destas mensagens consiste em dados relativos aos valores lidos pelos sensores instalados na cadeira de rodas. Portanto, sempre que existe alguma diferença no valor de algum sensor, essa informação é transmitida para o modelo digital, que por sua vez vai atualizar o estado do seu gémeo representado em três dimensões. Em paralelo com a comunicação por MQTT, o EV3 estará também a controlar os motores da cadeira e a ler os valores dos seus sensores.

3.2.2.5 IDE e Documentação para ev3dev

O desenvolvimento de ficheiros executáveis, para o módulo EV3, foi feito com base na linguagem de programação Python, com o auxílio do IDE VS Code. As vantagens deste ambiente são a sua simplicidade, a organização dos ficheiros e a grande variedade de extensões que se podem instalar de modo a facilitar a implementação do projeto que se está a desenvolver. Neste caso particular, utilizou-se uma extensão denominada de "ev3dev-browser" que permite a procura por dispositivos que tenham o sistema operativo "ev3dev-stretch" instalado, enviar ficheiros para esses mesmos dispositivos e executá-los de forma remota. Para além disso, permite também fazer capturas do ecrã do EV3, como já foi feito com as figuras 3.12 e 3.15, abrir um terminal SSH, algo que permitiu a ligação do EV3 ao Wi-Fi, e consultar informação sobre o sistema 3.13.

A nível de programação, foi utilizado um módulo, denominado de "ev3dev2", cuja documentação se encontra em [45], que contém as classes que permitem controlar motores, sensores, botões físicos, ecrãs LCD e outros componentes da gama EV3. A importação destas classes é feita como qualquer outra, como demonstra o código da listagem 3.1. Neste caso, são importadas classes como o LargeMotor, que corresponde ao motor grande da figura 3.3c, as outputs OUTPUT_A e OUTPUT_B da figura 3.5a, a SpeedPercent, que controla a velocidade dos motores, a MoveSteering que recebe como parâmetros de entrada um par de motores a controlar em simultâneo, a input INPUT_1 da figura 3.5b, o sensor TouchSensor, que é um sensível ao toque, e os Light-emitting Diode (LED) do módulo EV3.

Listagem 3.1: Exemplo de importação de classes do módulo ev3dev

```
from time import sleep

from ev3dev2.motor import LargeMotor, OUTPUT_A, SpeedPercent, MoveSteering
from ev3dev2.sensor import INPUT_1
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.led import Leds
```

Cada classe pode ser utilizada para diversas finalidades, tal como demonstra o código da listagem 3.2, com o qual é possível controlar a cor dos leds do EV3 através do sensor de toque, ou o código da listagem 3.3, com o qual se controla um par de motores utilizando a classe MoveSteering.

Listagem 3.2: Exemplo de código para o controlo da cor dos LEDs com o sensor de toque

```
# Initialize sensor and leds
ts = TouchSensor()
leds = Leds()

print("Pressiona o sensor de toque para mudar a cor do LED!")

# Infinite cycle to change colors with touch sensor
while True:
    if ts.is_pressed:
        leds.set_color("LEFT", "GREEN")
        leds.set_color("RIGHT", "GREEN")
    else:
        leds.set_color("LEFT", "RED")
        leds.set_color("RIGHT", "RED")
    sleep(0.01)
```

Listagem 3.3: Exemplo de código para o controlo de um par de motores com MoveSteering

```
# Initate steering drive with motors on output A and B
steering_drive = MoveSteering(OUTPUT_A, OUTPUT_B)

# drive in a turn for 10 rotations of the outer motor
steering_drive.on_for_rotations(-20, SpeedPercent(75), 10)

# drive with steering = 0 (straight line) and speed = 50% of max value
steering_drive.on(0, 50)
```

As classes utilizadas neste projeto são as que constam no diagrama de classes Unified Modeling Language (UML) representado na figura 3.17, o qual inclui os respetivos atributos e métodos de cada classe. Os atributos podem ser vistos como variáveis que assumem um valor num determinado momento, tal como a posição do motor, o seu estado e a sua

velocidade de rotação. Os métodos são funções que executam determinadas ações sobre o componente, tal como ligar, rodar ou parar. Cada classe pode ser descrita, conforme [46] e [47], pelo seguinte:

1. **LargeMotor** - Dispõe de vários métodos e atributos para utilizar em motores grandes com características posicionais e relacionais, tanto da gama EV3 como NXT, que vão desde o controlo à monitorização de diversos aspetos do motor. Esta classe tem como parâmetro de entrada a porta de saída onde este se encontra no EV3.

2. **MediumMotor** - Idêntico ao LargeMotor, tirando o facto de ter de ser associada ao motor médio.

3. **RemoteControl** - Esta classe pertence a uma versão anterior do módulo ev3dev e permite enviar sinais infravermelhos correspondentes a um determinado método. Possui como parâmetro de entrada o canal desejado, que vai desde o 1 ao 3, o que permite o uso dos botões do controlador para, no máximo, três diferentes situações em simultâneo.

4. **MoveSteering** - Classe que foi utilizada para os dois motores grandes, correspondentes às rodas dianteiras do protótipo. Controla um par de motores em simultâneo, assumindo um valor singular de *steering* e velocidade. O parâmetro *steering*, que significa direção, assume valores entre -100 e 100, em que -100 significa virar à esquerda (os motores rodam em direções opostas), 0 significa deslocar numa linha reta e 100 significa virar à direita (os motores rodam em direções opostas às do -100). O parâmetro *speed* assume, por predefinição, a percentagem do valor máximo da velocidade do motor. Existe ainda outra classe, semelhante ao MoveSteering, denominada MoveTank, que controla dois motores baseado na velocidade de cada um, (*left_speed* e *right_speed*). Pelo gráfico da figura 3.16, é possível observar a relação entre as duas classes, em que mostra para que valores de *left_speed* e *right_speed*, da classe MoveTank, correspondem as variáveis *steering* e *speed*, sendo que a última está fixa em 40. O parâmetro *steering* encontra-se sobre o eixo dos xx e as variáveis da velocidade no eixo dos yy. Para fins de implementação, optou-se por utilizar a MoveSteering, uma vez que tem mais margem de manobra, por depender de duas variáveis independentes, e por ter o mesmo comportamento da cadeira representada no modelo virtual.

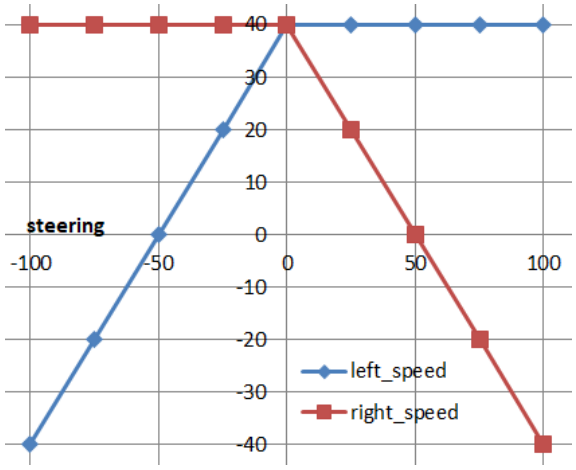


Figura 3.16: Relação entre as classes MoveTank e MoveSteering (retirado de [46])

5. **Sound** - Permite executar ficheiros WAV (ficheiro de som), converter texto para fala e soar *beeps*, tudo através do altifalante presente no EV3.

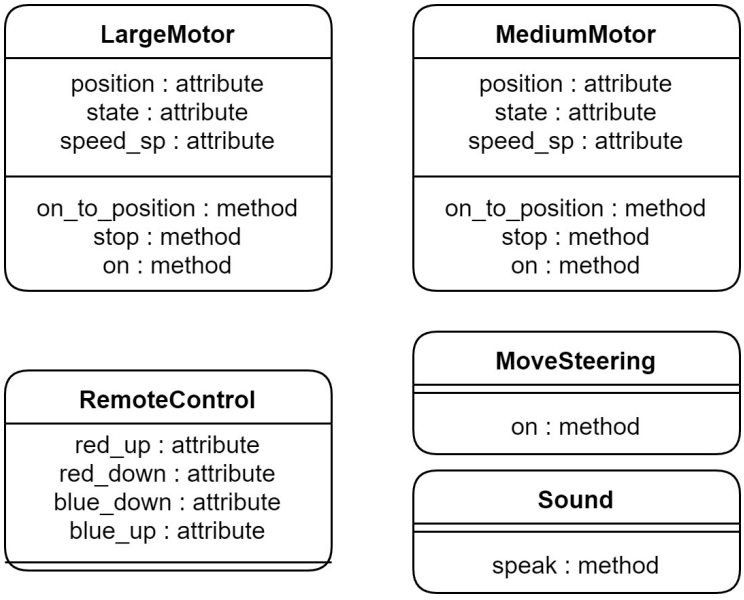


Figura 3.17: Diagrama de classes UML correspondente às classes utilizadas no sistema

Na tabela 3.1, estão descritos todos os atributos e os métodos utilizados [48] para cada classe e o seu acesso e parâmetros, respetivamente. Repare-se que no acesso, um atributo tem a possibilidade de ler ou escrever um valor e os parâmetros são obrigatórios para cada método.

Tabela 3.1: Descrição detalhada dos atributos e métodos utilizados

	Nome	Acesso	Descrição
Atributos	position	Ler/Escrever	Retorna a posição atual do motor em graus. Quando o motor roda no sentido dos ponteiros do relógio, a posição aumenta e vice-versa. Se for utilizada como escrita, o valor escolhido será o definido para a posição.
	state	Ler	Retorna uma lista com os possíveis estados do motor, que podem ser: - *running* : O motor está a rodar. - *ramping* : O motor está a rodar, porém ainda não atingiu a posição desejada. - holding : O motor não está a rodar e encontra-se "preso", numa certa posição. - *overloaded* : O motor está a rodar o mais rápido possível mas não consegue atingir o *speed_sp* - *stalled* : O motor está a tentar rodar mas não consegue.
	speed_sp	Ler/Escrever	A escrita define a velocidade do motor em graus por segundo. A leitura retorna o valor da velocidade. Um valor negativo causa o motor a rodar em modo reversivo.
	red_up / red_down blue_up / blue_down	Ler	Retorna *True* ou *False* consoante os botões estejam a ser pressionados ou não.
	Nome	**Parâmetros**	**Descrição**
Métodos	on_to_position	speed, position	Roda o motor a uma dada velocidade para uma dada posição. A variável *speed* representa a percentagem da velocidade máxima do motor.
	stop	stop_action	Pára o motor com o parâmetro stop_action, que pode assumir os seguintes valores: - *coast* : Retira a energia do motor e faz com que este pare gradualmente. - *break* : Retira a energia do motor e cria uma carga elétrica passiva. Isto provoca a absorção de energia da rotação do motor e consequentemente faz com que pare, mais rapidamente que o coast. - *hold* : O motor mantém a sua posição, independentemente de forças externas.
	on	steering, speed	Roda o motor a uma dada velocidade, infinitamente. Caso a classe seja MoveSteering, a variável *steering* é obrigatória e representa a direção. A variável *speed* é a mesma que se utiliza no método *on_to_position*.
	speak	text	Texto a ser transmitido no altifalante do EV3.

3.2.2.6 Programa e Algoritmos Desenvolvidos

O programa desenvolvido para o módulo EV3 é constituído pelos passos representados na figura 3.18 e pode ser analisado com mais detalhe no repositório GitHub criado especificamente para este projeto [49]. Inicialmente, é feita ligação do dispositivo ao MQTT *broker*, através do seu endereço de IP e respetiva porta, o qual irá ser abordado mais adiante, com recurso ao módulo *paho-mqtt*, cuja documentação, para a linguagem de programação

Python, se encontra em [50]. De seguida, são inicializadas as classes de cada componente, nomeadamente os motores, com a respetiva porta de saída, o controlador remoto, e o som. É importante salientar que as posições dos motores foram inicializadas a 0 e o motor do apoio para as costas é enviado para a posição -47 (cerca de metade do intervalo de valores permitidos para a deslocação do motor), isto porque é necessário sincronizar com a posição do gémeo digital no ROS, que se encontra precisamente nesta posição. Assim que é tudo corretamente inicializado e o EV3 está conectado, com sucesso ao MQTT *broker*, é transmitida para a consola e reproduzida pelo altifalante, uma mensagem a dizer *"Code uploaded successfully"*.

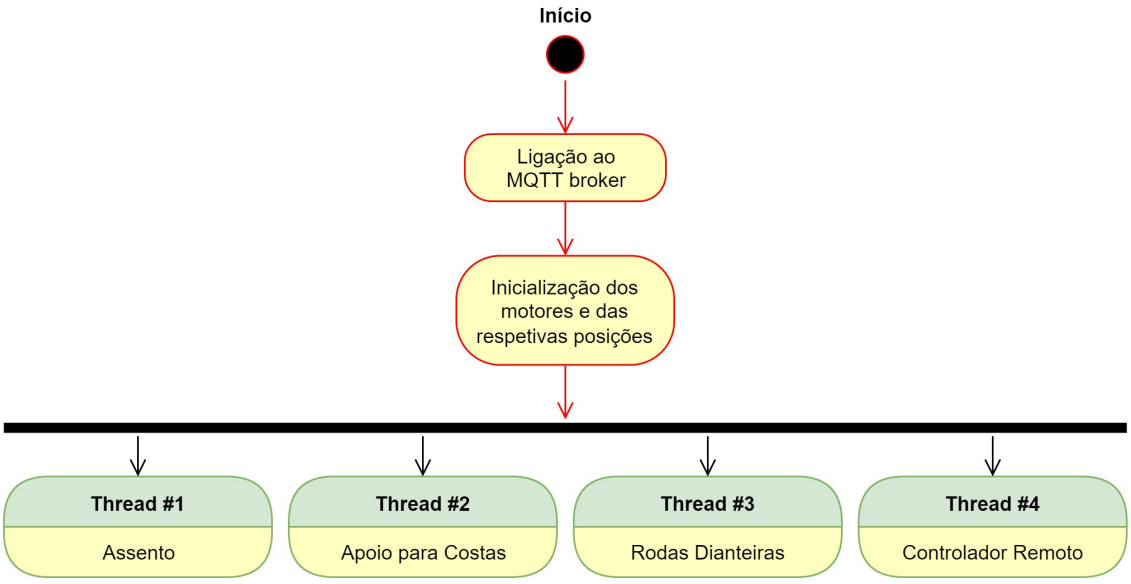
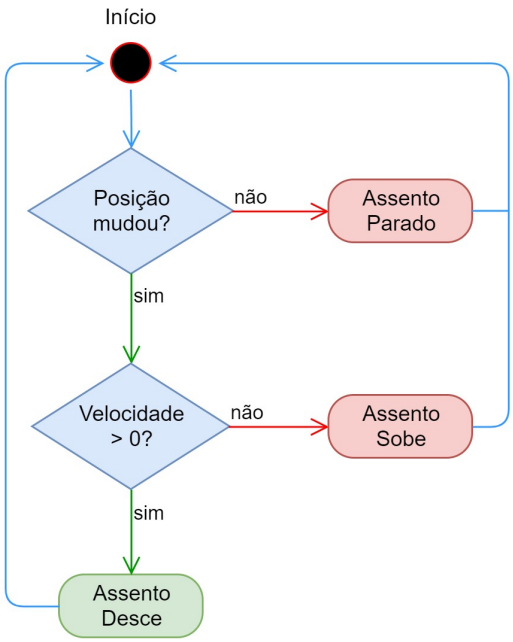
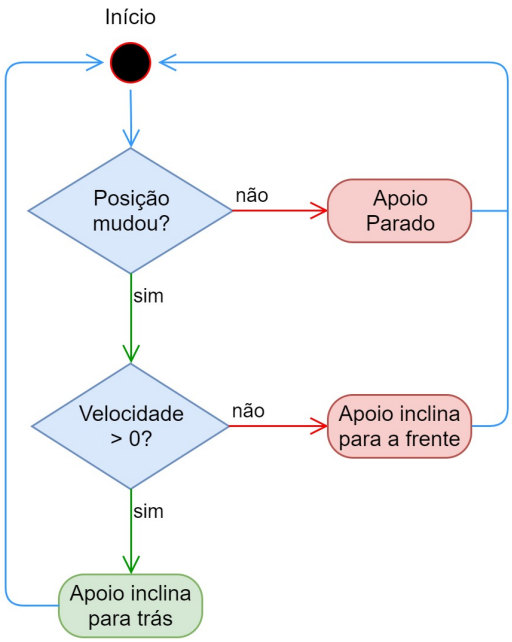


Figura 3.18: Diagrama de atividades do programa desenvolvido

A última componente do programa consiste na execução, em *loop*, de quatro *threads* distintos, correspondentes à monitorização da posição dos diversos motores e respetivo envio para o modelo digital. Não foi possível executar todos os comandos em apenas uma *thread*, uma vez que era apresentado um erro sempre que se tentava aceder às posições dos motores em simultâneo. A primeira *thread*, cuja implementação se encontra na figura 3.19a, consiste na verificação contínua da posição do motor que faz com que a cadeira suba e desça. Caso a posição mude, consoante a velocidade seja positiva ou negativa, é enviada uma mensagem, por MQTT, para o tópico "/up_down_motor_pos", a sinalizar que o motor do assento está a subir ou a descer. A segunda *thread*, representada na figura 3.19b, é semelhante à primeira, com a diferença que o motor em questão é o de apoio para as costas, ou seja, quando este apoio estiver a inclinar para a frente ou para trás, essa informação será enviada para o tópico "/back_motor_pos".



(a) Algoritmo da *thread* responsável pela sincronização da posição do motor do assento



(b) Algoritmo da *thread* responsável pela sincronização da posição do motor das costas

Figura 3.19: Algoritmos utilizados para as *threads* do motor do assento e apoio para costas

A terceira *thread*, representada na figura 3.20, é responsável por monitorizar a posição e velocidade dos motores das rodas dianteiras. Consoante a mudança da posição de ambos os motores, existem quatro cenários possíveis para o comportamento da cadeira: virar à esquerda, a virar à direita, a deslocar-se para a frente ou a deslocar-se para trás. Caso a posição do motor de ambas as rodas seja maior, no instante de tempo lido, face à posição guardada no instante de tempo anterior, então a cadeira está a mover-se para a frente. O mesmo acontece quando a cadeira se está a mover para trás, mas com a posição registada no instante de tempo sendo menor que a no instante anterior. Caso a posição do motor esquerdo seja maior que a anterior e a posição do motor direito seja menor que a anterior, então a cadeira está a virar para a direita. Igualmente, quando a cadeira está a virar à esquerda, a posição do motor esquerdo é menor que a anterior e a do motor direito é maior. A informação relativa à posição e velocidade dos motores das rodas dianteiras é enviada para o tópico "/steering".

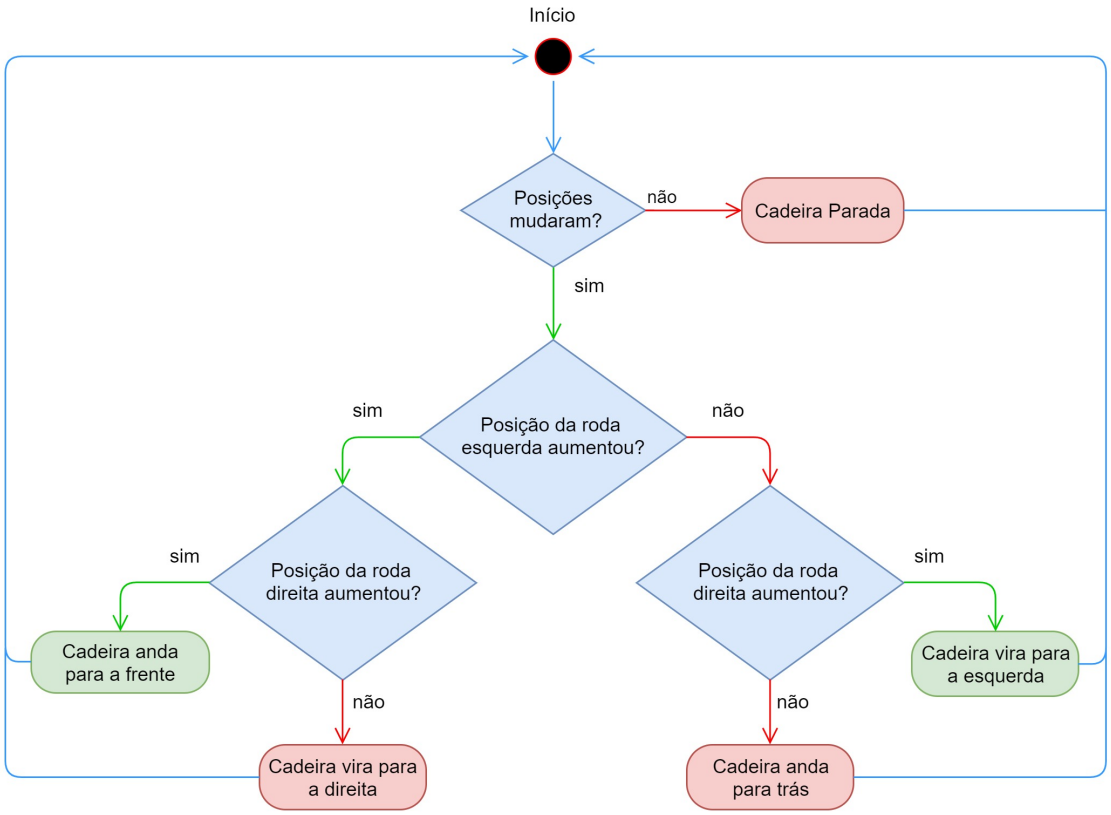


Figura 3.20: Algoritmo da *thread* responsável pela sincronização da posição dos motores das rodas dianteiras

A quarta e última *thread* consiste no reconhecimento do pressionar dos botões no controlador remoto e no desencadear de diferentes ações, associadas à cadeira. O controlador dispõe de três canais, os quais se podem intercalar, de modo a poder utilizar os mesmos botões para cenários diferentes. Na tabela 3.2, encontram-se os comandos executados ao pressionar determinados botões nos três diferentes canais. No canal 1, é feito o controlo da condução da cadeira, ou seja, controlam-se os motores grandes que constituem a classe MoveSteering. Esta, dispõe do comando on, que por sua vez tem dois parâmetros de entrada obrigatórios, o *steering* e o *speed*. Tal como foi anteriormente explicado, o *steering* está associado à direção e o *speed* corresponde à velocidade a que giram os motores. Estando no canal 1, ao pressionar os botões vermelhos, a cadeira move-se para frente ou para trás, a uma velocidade linear constante de 20% (percentagem da velocidade máxima do motor) enquanto que ao pressionar os botões azuis a cadeira roda a uma velocidade angular constante de 9%. No canal 2, o controlo é feito sobre o motor médio do assento da cadeira. Os botões vermelhos estão associados à sua subida e descida até aos limites definidos e os botões azuis têm uma função semelhante, com a diferença que, quando se deixa de pressionar, o motor pára, podendo não chegar às posições limite. Por último, no canal 3 é controlado o motor do apoio para as costas, o qual possui um modo operacional idêntico ao do canal 2.

Quando nenhuma das situações da tabela se confirma, ou seja, quando nenhum botão

está a ser pressionado, foi programado para que os motores grandes, correspondentes às rodas, estivessem parados. O mesmo acontece para o motor do apoio para as costas e para o motor do assento, com a diferença de que são forçados a manter a sua posição atual, independentemente se houver uma força externa a tentar rodá-los.

Tabela 3.2: Ações associadas aos canais e botões do controlador remoto

Canal	**Motores**	**Botão**			
		Vermelho		**Azul**	
		Baixo	**Cima**	**Baixo**	**Cima**
1	Rodas Dianteiras (MoveSteering)	Desloca para trás	Desloca para a frente	Vira à direita	Vira à esquerda
2	Assento (MediumMotor)	Desce o assento até à mínima posição	Sobe o assento até à máxima posição	Desce o assento (nunca ultrapassando a posição mínima)	Sobe o assento (nunca ultrapassando a posição máxima)
3	Apoio para Costas (LargeMotor)	Inclina o apoio para trás até à posição minima	Inclina o apoio para a frente até à posição máxima	Inclina o apoio para trás (nunca ultrapassando a posição mínima)	Inclina o apoio para a frente (nunca ultrapassando a posição máxima)

3.2.3 Sistema de Integração da Bússola CMPS11

Do ponto de vista do modelo digital, dois aspetos relevantes a monitorizar são a orientação e inclinação da cadeira. Apesar da gama Mindstorms disponibilizar um sensor rotacional e direcional (giroscópio), não foi possível obtê-lo, pelo que foi necessário optar por outra solução.

3.2.3.1 Módulo CMPS11

A solução passou pela utilização do módulo CMPS11, o qual funciona como bússola de compensação inclinada, cuja documentação se encontra em [51]. Este módulo constitui um magnetómetro, um giroscópio e um acelerómetro, em que todos medem as componentes x, y e z em torno do seu campo magnético. Possui também um filtro Kalman que combina o giroscópio e o acelerômetro para retirar erros causados pela inclinação da placa PCB. Para a alimentação, o módulo requer uma tensão entre os 3.6 e os 5 V a uma corrente nominal de 25 mA. A nível de comunicação com outro dispositivo, pode ser conectado por I2C ou serial.

3.2.3.2 Placa NodeMCU V3 com Microcontrolador ESP8266

De modo a fornecer tensão e processar os valores lidos pelo CMPS11, foi necessário utilizar uma placa com um microcontrolador, preferencialmente que disponibilizasse pinos para a comunicação por I2C e que incluísse um módulo Wi-Fi integrado, para o envio dos dados através do protocolo MQTT. A placa utilizada foi a NodeMCU V3, baseada no microcontrolador ESP8266, que permite a instalação e execução de ficheiros na linguagem C++, no ambiente Arduino, e cumpre com os requisitos inicialmente delineados, uma vez que tem Wi-Fi integrado IEEE 802.11 b/g/n e dois pinos para a comunicação I2C, um

para data e outro para *clock*. Na tabela 3.3, encontram-se outras especificações relevantes relativas à placa, tais como a tensão de operação, o intervalo de tensão de entrada e a resolução do ADC.

Tabela 3.3: Especificações da placa NodeMCU V3

Especificação	Valor
Tensão de Operação	3.3V
VIN	5V...12V
Resolução ADC	10 bits (0...1023)
Velocidade do relógio	80/160 MHz
Tensão Máxima de Entrada	15V
Corrente Máxima de Saída	1A

3.2.3.3 Esquemático do Sistema

Este sistema, constituído essencialmente pela placa NodeMCU V3 e pelo módulo CMPS11, tem o esquemático representado na figura 3.21, na qual podemos observar que os pinos de comunicação I2C do CMPS11, SCL e SDA, estão ligados aos pinos D1(SCL) e D2(SDA) da placa, respetivamente. Os pinos G (*ground*) dos dois dispositivos estão também ligados e o pino da alimentação do CMPS11 está ligado ao pino VU da NodeMCU, uma vez que este tem uma tensão de saída de 5V, quando o microcontrolador está alimentado via USB, que neste caso é uma *power bank* que fornece 5V DC. Não seria possível utilizar outro pino para alimentar o CMPS11 visto que este requer uma tensão mínima de 3.6V.

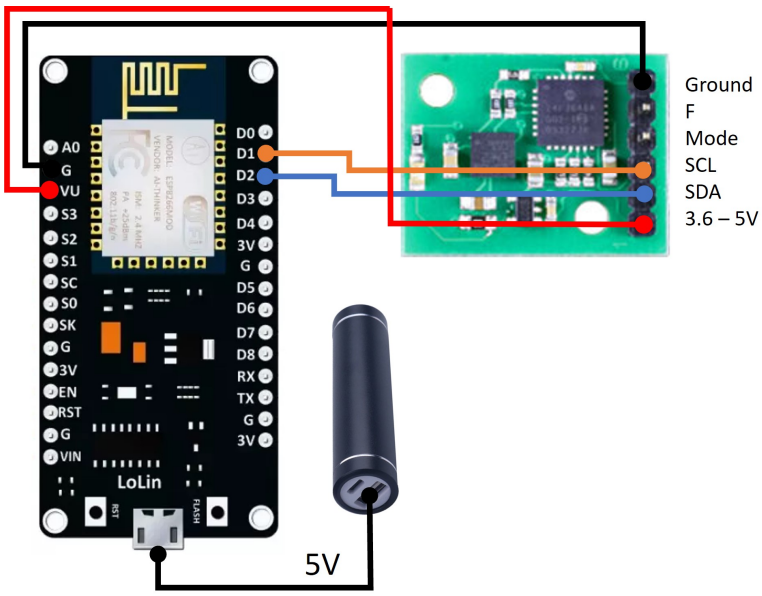


Figura 3.21: Esquemático do sistema CMPS11 com NodeMCU V3

3.2.3.4 Programa e Algoritmos Desenvolvidos

O programa a ser executado no microcontrolador ESP8266 foi desenvolvido na linguagem de programação C++, no ambiente Arduino, com o intuito de processar os valores lidos pelo CMPS11 e enviá-los por MQTT para o tópico "/gyro" que, por sua vez, é subscrito pelo sistema do modelo digital e pela interface de monitorização. A estrutura de alto nível do programa encontra-se representada na figura 3.22 e é constituída pela fase de *setup*, ou seja, a fase inicial do programa, onde é feita a calibração do CMPS11 e a ligação ao Wi-Fi e o *broker* MQTT, e pela fase *loop*, na qual é feita a leitura e processamento dos valores lidos pelo módulo até que o programa seja interrompido. Mais detalhes sobre o código em si podem ser consultados no repositório GitHub [52].

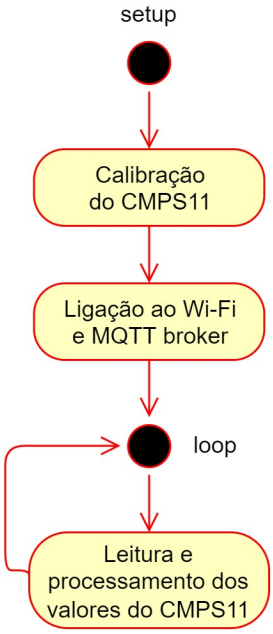


Figura 3.22: Estrutura do programa desenvolvido para o microcontrolador ESP8266

Relativamente à calibração do módulo CMPS11, é importante compreender quais são os eixos sobre os quais a cadeira pode rodar. Estes estão representados na figura 3.23, na qual se pode observar o eixo lateral, longitudinal e vertical. É também importante perceber como funciona a comunicação através do protocolo I2C, no microcontrolador ESP8266, a qual foi suportada pela biblioteca Wire [53]. A comunicação começa com o envio de um bit inicial, correspondente ao endereço do módulo que, neste caso, é o 0x60, seguido da escrita do número do registo que se quer ler ou escrever. O CMPS11 dispõe de 28 bytes de registos (1 byte para cada registo), cuja função daqueles utilizados se encontra na tabela 3.4. A calibração dos três eixos foi feita com base nas configurações de fábrica, segundo [51], através da escrita dos comandos 0x20, 0x2A e 0x60, em três transmissões diferentes, no registo de comandos que se encontra, pela tabela, no registo 0. Na listagem 3.4, é possível observar o código na linguagem C++ com o auxílio da biblioteca Wire que calibra a bússola CMPS11 com as configurações descritas anteriormente.

Listagem 3.4: Código C++ que calibra a bússola CMPS11 com as definições de fábrica

```
#include <Wire.h>
#define CMPS11_ADDRESS 0x60 // Address of CMPS11 shifted right one bit for
    arduino wire library

Serial.println("Default Calibration Mode");
delay(2000); //2 second before starting
Serial.println("Start");

Wire.beginTransmission(CMPS11_ADDRESS);
Wire.write(0); //command register
Wire.write(0x20);
Wire.endTransmission();
delay(20);

Wire.beginTransmission(CMPS11_ADDRESS);
Wire.write(0); //command register
Wire.write(0x2A);
Wire.endTransmission();
delay(20);

Wire.beginTransmission(CMPS11_ADDRESS);
Wire.write(0); //command register
Wire.write(0x60);
Wire.endTransmission();

Serial.println("done");
```

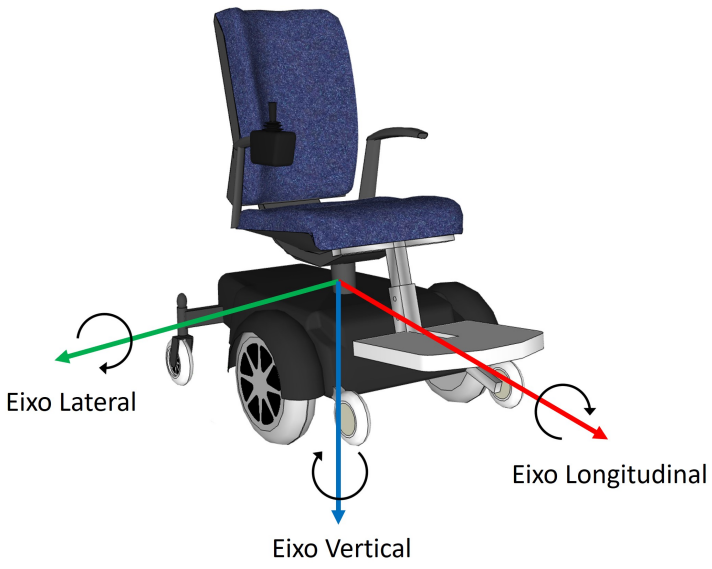


Figura 3.23: Eixos da cadeira de rodas

Tabela 3.4: Registos do módulo CMPS11 utilizados e respetivas funções

Registo	Função
0	Registo de comando (escrita) / Versão do software (leitura)
1	Ângulo de rotação sobre o eixo vertical dado em 8 bits (0-255)
2,3	Ângulo de rotação sobre o eixo vertical dado em 16 bits (0-3599), sendo o registo 2 o byte mais significativo
4	Ângulo de rotação sobre o eixo lateral
5	Ângulo de rotação sobre o eixo longitudinal

A ligação do NodeMCU ao Wi-Fi e ao MQTT *broker* foi suportada pelas bibliotecas ESP8266WiFi [54] e PubSubClient, respetivamente, e é feita de forma idêntica à do programa desenvolvido para o EV3. Já dentro do da fase de *loop*, a leitura dos valores dos ângulos de rotação de cada eixo é feita recorrendo aos registos do CMPS11 que, segundo a tabela 3.4, correspondem aos registos 1 para o ângulo de rotação sobre o eixo vertical, 4 para o ângulo de rotação sobre o eixo lateral e 5 para o ângulo de rotação sobre o eixo longitudinal. Estes valores são constantemente atualizados e enviados, por MQTT, para o tópico "/gyro" que irá ser utilizado por outras componentes do sistema.

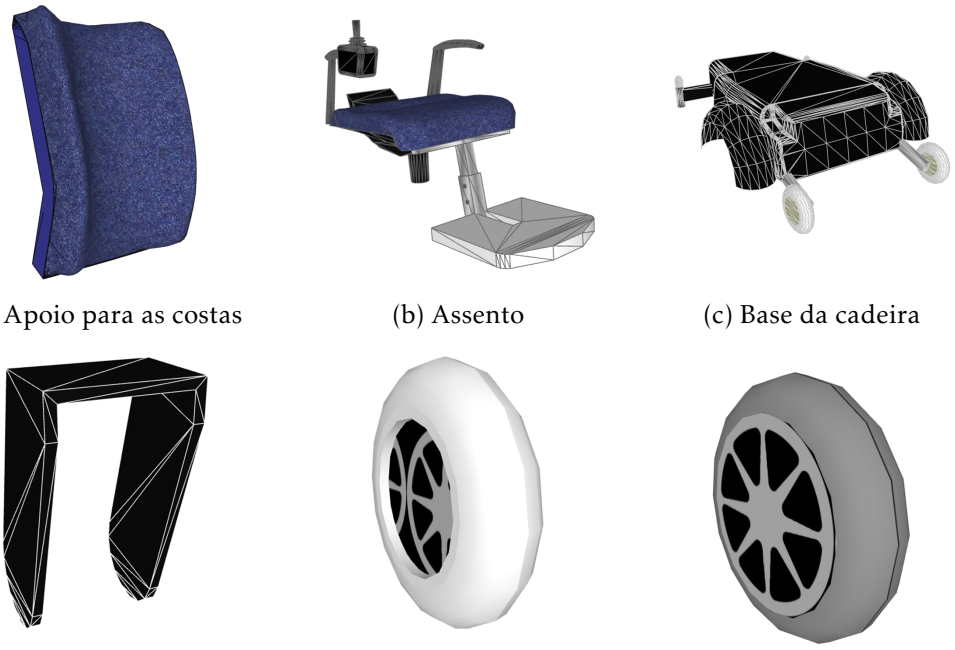
3.3 Modelação Digital

É nesta secção que começa efetivamente o desenvolvimento do DT, na medida em que é modelada, digitalmente, a cadeira de rodas elétrica. Esta modelação abrange três componentes primárias relativamente à cadeira, sendo que a modelação de regras digital não foi incluída, constando nas conclusões como trabalho futuro. As componentes são a sua geometria, caracterizada por um modelo representado a três dimensões, a física, baseada nas suas propriedades físicas, e a comportamental que representa o seu comportamento e mecanismos, face às alterações do ambiente em que se encontra. Começando pela modelação geométrica, o objetivo inicialmente delineado foi adquirir um ficheiro de um modelo 3D, de uma cadeira de rodas elétrica, que pudesse ser importado para o ambiente utilizado para a modelação física e comportamental que, como já foi anteriormente mencionado, foi o ROS. Existem outras plataformas no mercado que permitem a criação, de raiz, de um DT, porém, nem todas apresentam condições favoráveis para a sua utilização, no contexto desta dissertação, nomeadamente o suporte da comunidade, o preço e a sua fiabilidade.

3.3.1 Modelo 3D da Cadeira

O modelo 3D da cadeira de rodas escolhido para o projeto, que já foi anteriormente demonstrado na figura 3.2, foi retirado de [36] de forma gratuita, na fase de desenvolvimento do sistema. O ficheiro encontra-se no formato Collada (DAE), que consiste na descrição do objeto em XML, o que facilita a sua integração em diversas aplicações. Todavia, o modelo completo da cadeira não iria servir para a modelação comportamental, uma vez que não seria possível diferenciar e controlar os diversos *links* que a constituem,

tais como as rodas, o assento e o apoio para costas. Posto isto, foi necessário recorrer a uma ferramenta de edição de modelos 3D para fazer essa distinção e separar o modelo completo em modelos secundários, correspondem aos vários *links* da cadeira. O software *open source* utilizado para esta operação foi o Blender [55], no qual se separou o modelo da figura 3.2 nos modelos dos *links* da figura 3.24, incluindo as respetivas texturas e materiais. Mais adiante é abordada a integração destes *links* na modelação comportamental e física, na *framework* ROS.



(a) Apoio para as costas

(b) Assento

(c) Base da cadeira

(d) Suporte das rodas de apoio traseiras

(e) Roda de apoio traseira

(f) Roda dianteira

Figura 3.24: Modelos 3D dos diferentes *links* da cadeira

3.3.2 Robot Operating System (ROS)

Apesar do nome, o ROS não é propriamente um sistema operativo, mas sim um conjunto de módulos e ferramentas software, desde drivers a algoritmos, que ajudam a construir aplicações, não apenas para robôs, mas também para outros componentes de hardware. Existem mais de 2000 pacotes de ferramentas disponíveis, desenvolvidos pela sua comunidade, cada um especializado numa determinada função [56]. A característica chave do ROS é a forma como está organizado e a maneira como comunica, o que permite o desenvolvimento de projetos complexos e a integração de vários dispositivos num só projeto. A comunicação entre eles, ou neste caso entre os nós, é feita de um modo semelhante ao protocolo de comunicação MQTT, na medida em que podem enviar e receber mensagens para diversos tópicos, podendo estes também ser subscritos. No contexto do sistema desenvolvido, o ROS foi utilizado para criar um gémeo digital da entidade física, replicando o seu comportamento, desde o movimento à posição dos motores, e as suas características

físicas, como o peso e o tamanho.

3.3.2.1 Instalação

Existem várias distribuições do ROS suportadas até à data, havendo até uma versão 2 do ROS, a última a ser lançada, que inclui mais ferramentas do que a primeira. Dado que não havia qualquer experiência prévia no ambiente ROS, optar pela segunda versão seria algo ambicioso no tempo disponível para o desenvolvimento da dissertação, uma vez que esta é bastante mais complexa e difícil de aprender que a primeira. Posto isto, optou-se pela instalação da última distribuição do ROS 1, a Noetic Ninjemys, que é suportada até Maio de 2025 [57].

O ROS está disponível em diversos sistemas operativos, porém, apenas as versões que correm no Ubuntu e no Debian são oficialmente suportadas, enquanto as restantes, ou são versões experimentais, tal como o Windows, ou são suportadas apenas pela comunidade. Para instalar a *framework* ROS no sistema operativo Ubuntu foi necessário recorrer ao software VMware [58] para criar uma máquina virtual, dado que o sistema foi desenvolvido num computador com o Windows. Para esta máquina, foram alocados 7.8 GB de memória RAM, 4 processadores core, 20 GB de disco rígido e instalado o sistema operativo Ubuntu 20.04.3. Dentro do terminal do Ubuntu, procedeu-se à instalação completa do ROS noetic, com todas as ferramentas disponíveis, incluindo simuladores e bibliotecas, através do comando *sudo apt install ros-noetic-desktop-full*. Foi também inicializado o *rosdep*, um pacote que permite a instalação de dependências do sistema.

3.3.2.2 ROS Workspace

Para desenvolver o projeto no ambiente ROS, foi necessário primeiro criar uma pasta *workspace*, através do sistema de *build* "catkin" [59], que permite modificar, fazer *build* e instalar pacotes "catkin", os quais contém todo o código fonte do projeto. A sua criação é feita através da execução dos seguintes comandos, no terminal do Ubuntu:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

O "catkin" *workspace* tem a organização de pastas e ficheiros correspondente à da figura 3.25, já com o sistema da cadeira de rodas incluído. Na pasta "SRC", é onde estão agregados todos os projetos desenvolvidos, em formato de pacote "catkin", que neste caso tem o nome de "wheelchair", bem como um ficheiro denominado "CMakeLists" que configura os projetos na *workspace*. A pasta "BUILD" é onde o *software* CMake [60] é invocado para fazer *build* aos pacotes "catkin" na pasta "SRC" e onde também são guardadas informações *cache* e ficheiros intermediários. Por último, a pasta "DEVEL" é onde os alvos de *build* são colocados antes de serem instalados. É nesta pasta que se encontra o ficheiro

bash "setup", o qual é necessário para sobrepor a *workspace* no ambiente ROS em que nos encontramos, através do seguinte comando:

```
$ source devel/setup.bash
```

De modo a verificar se que a *workspace* está propriamente sobreposta pelo ficheiro "setup", executou-se o comando seguinte e obteve-se o resultado em baixo, que verifica se a variável ROS_PACKAGE_PATH:

```
$ echo $ROS_PACKAGE_PATH
/home/afonso/catkin_ws/src:/opt/ros/noetic/share
```

A navegação sobre os pacotes ROS consegue ser um pouco tediosa e para isso existem diversas ferramentas, tais como *rospack*, *roscd*, *roslog* e *rosls*, que permitem obter informação sobre pacotes, mudar a diretoria para um pacote ou pasta, localizar a pasta onde o ROS guarda os ficheiros log e navegar diretamente para uma pasta ou ficheiro através do seu nome.

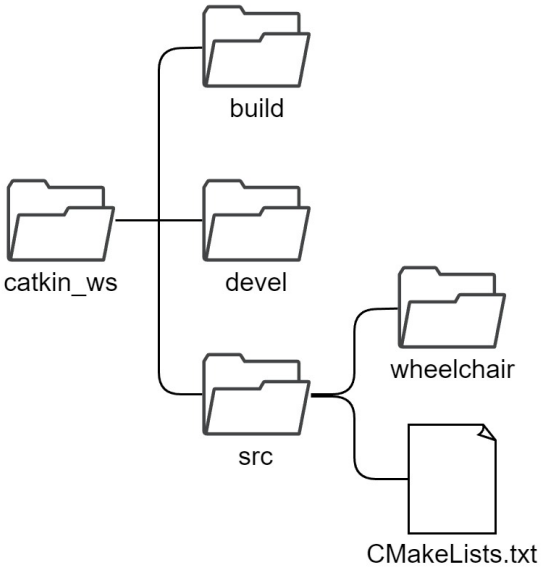


Figura 3.25: Estrutura da "catkin" *workspace* do projeto

3.3.2.3 Pacote ROS do Sistema

Depois da criação da ROS *workspace*, procedeu-se à criação de um pacote "catkin", o qual contem toda a informação do projeto, cuja constituição é a que consta na figura 3.26. Esta pasta inclui dois ficheiros predefinidos que caracterizam um pacote "catkin", o "package" e o "CMakeLists". O primeiro fornece informação relativa ao pacote ROS, tal como a sua versão, descrição, licença e dependências, as quais se encontram no código da listagem 3.5. Estas foram incluídas nos parâmetros do comando *catkin_create_pkg wheelchair [dependencia1] [dependencia2] [dependenciaN]*, aquando da criação do pacote ROS. O segundo

ficheiro, já anteriormente mencionado, inclui algumas configurações do projeto. Antes da criação das restantes pastas e ficheiros, que serão esmiuçados nas secções seguintes, fez-se *build* ao projeto, através do comando *catkin_make*, de modo a conseguir executá-lo. É importante salientar que o pacote ROS e respetivas pastas e ficheiros se encontram no repositório GitHub [61].

Listagem 3.5: Dependências do projeto incluídas no ficheiro XML package

```
<build_depend>message_generation</build_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>opencv2</build_depend>
<build_depend>cv_bridge</build_depend>
<build_depend>sensor_msgs</build_depend>
<build_depend>tf</build_depend>
<build_depend>urdf</build_depend>
<build_depend>xacro</build_depend>
<build_depend>rviz</build_depend>
```

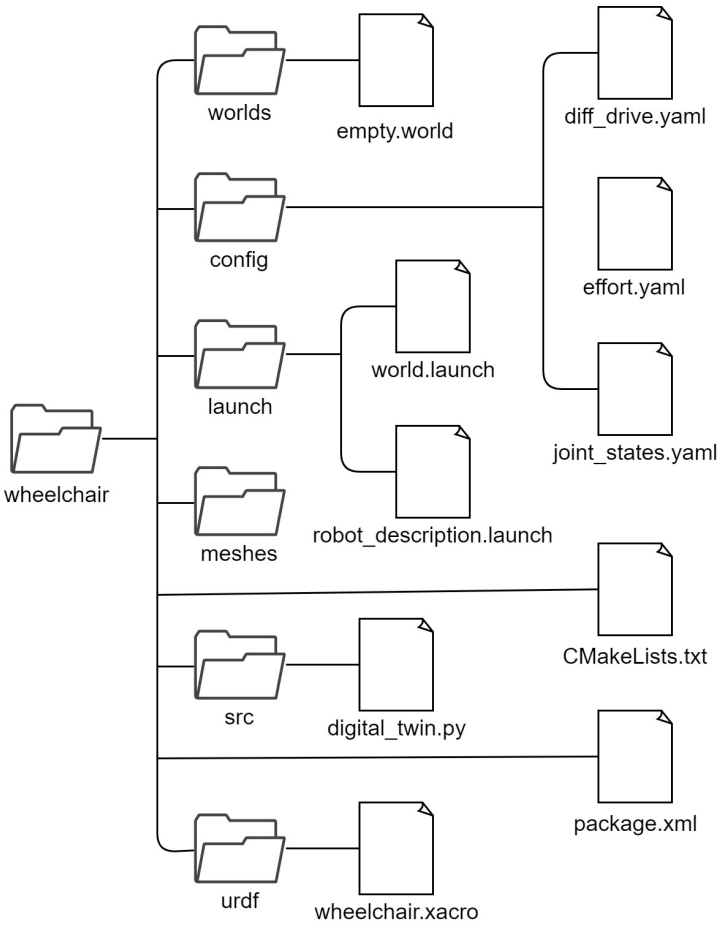


Figura 3.26: Estrutura pacote ROS do projeto

3.3.2.4 Modelo URDF

A pasta "URDF" contém um ficheiro do tipo URDF, no formato XML, que é responsável pela caracterização do modelo da cadeira de rodas elétrica, desde os aspetos visuais aos aspetos físicos, que irão ser visualizados no simulador gráfico Gazebo, o qual irá ser abordado mais adiante. Neste caso particular, utilizou-se a *framework* XACRO, que permite a construção de ficheiros XML mais curtos e legíveis, face ao URDF convencional. A estrutura do ficheiro XACRO está representada na figura 3.27 e consiste em quatro componentes primárias: os *links*, as *joints*, as transmissões e os *plugins*.

1) Links: Os *links* são as partes do esqueleto do modelo, os quais já foram abordados anteriormente e que estão representados na figura 3.24. O modelo da cadeira é constituído essencialmente por 10 *links*: a *footprint*, que projeta o centro da cadeira ao chão, o *chassis* da cadeira, correspondente à parte central com o motor, visível na figura 3.24c, as quatro rodas das figuras 3.24f e 3.24e, os dois suportes para as rodas traseiras da figura 3.24d, o assento da figura 3.24b e as costas da figura 3.24a. A organização estrutural destes *links* pode ser obtida através da execução do comando *check_urdf*, como ilustra a listagem 3.6.

Listagem 3.6: Organização estrutural dos links do sistema

```
$ check_urdf wheelchair.urdf
robot name is: wheelchair
---------- Successfully Parsed XML ---------------
root Link: robot_footprint has 1 child(ren)
   child(1): chassis
      child(1): left_wheel
      child(2): left_wheel_support
         child(1): left_wheel_back
      child(3): right_wheel
      child(4): right_wheel_support
         child(1): right_wheel_back
      child(5): upper_chassis
         child(1): back
```

Cada *link* é constituído pela sua componente visual, pela inércia e pela área de colisão. A componente visual é definida pelas coordenadas em relação à origem do *link* e pela sua geometria, a qual recebe como parâmetro o respetivo modelo 3D, caracterizado pelo ficheiro Collada (DAE) correspondente. Todos os ficheiros Collada, materiais e texturas dos *links* encontram-se na pasta "MESHES". A colisão é idêntica à componente visual, na medida em que têm as mesmas coordenadas em relação à origem e a mesma geometria, que corresponde aos limites do modelo 3D, tal como se pode possível observar pela figura 3.28, onde as áreas de colisão de cada *link* estão representadas por uma cor alaranjada. Por último, a componente da inércia contém os parâmetros físicos do *link*, tal como a massa, coordenadas do centro de massa e momento de inércia. A massa considerada para os *links* do modelo foi a mesma que a de uma cadeira de rodas real e não a do protótipo

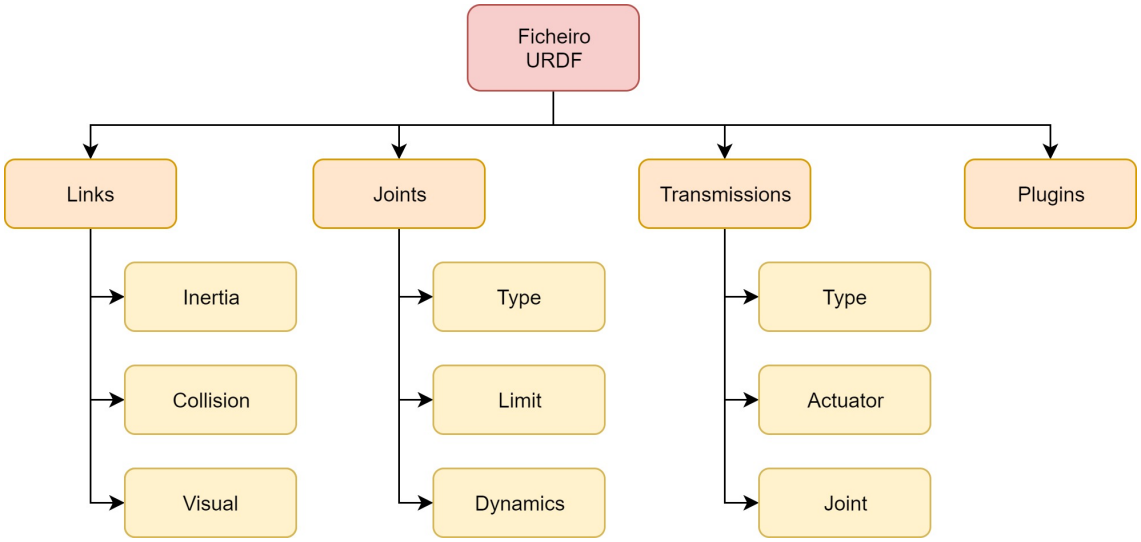


Figura 3.27: Estrutura do ficheiro XACRO

desenvolvido. Foi considerada uma densidade homogénea para todos os *links* pelo que o centro de massa está localizado no respetivo centro geométrico. Para o momento de inércia, utilizou-se o software MeshLab [62] que calcula esse parâmetro de qualquer *mesh* com base no seu centro de massa.

Figura 3.28: Áreas de colisão dos *links* do modelo da cadeira

2) Joints: As *joints* são as ligações entre os diversos *links* do modelo que descrevem as suas cinemáticas e dinâmicas. Cada *joint* é caracterizada pelo seu tipo, coordenadas na origem, nome do *link* "pai" e do *link* "filho", eixo em torno do qual se move, limites e dinâmicas. Existem vários tipos de *joint*, sendo que neste projeto foi utilizada a *prismatic*, para o movimento ao longo do eixo vertical do assento, com a definição de limite inferior e superior, a *revolute*, para a inclinação do apoio para as costas, um movimento de rotação ao

longo de um eixo com a definição de limites superior e inferior, e a *continuous*, utilizada nas quatro rodas, que roda em torno de um eixo sem limites. A relação hierárquica entre os *links* e as *joints* do modelo está representada no diagrama da figura 3.29 e a sua representação gráfica na figura 3.30, com os eixos de rotação de deslize a amarelo.

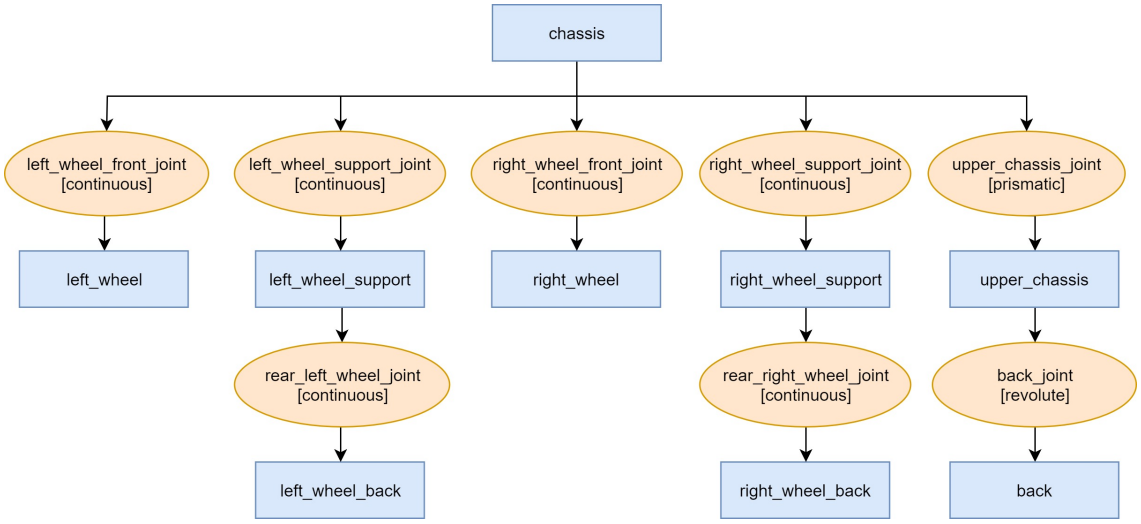


Figura 3.29: Hierarquia dos *links* e *joints* do modelo

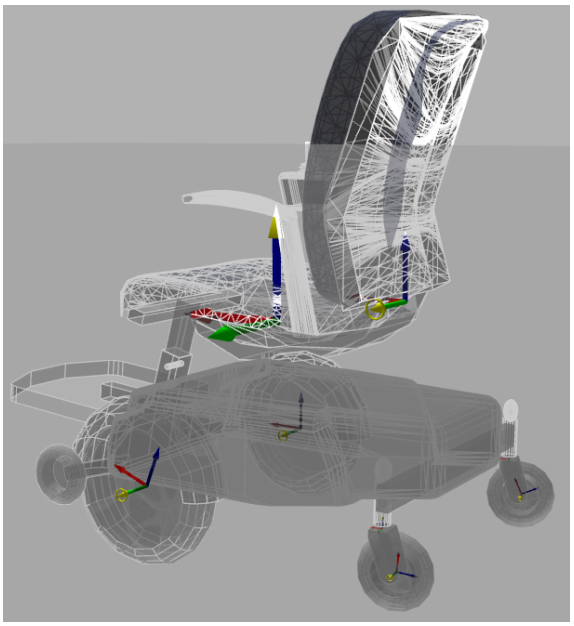


Figura 3.30: Representação gráfica das *joints* do modelo (vista lateral)

3) Transmissões: Este elemento é uma extensão do modelo URDF e é utilizado para descrever a relação entre um atuador e uma *joint*. Para a inclusão das transmissões no modelo, é necessário utilizar o *plugin ros_control*, o qual irá ser abordado a seguir. Uma transmissão é caracterizada pelo seu tipo e pela *joint* e atuador a que está ligada, como consta no exemplo da listagem 3.7, utilizado no ficheiro XACRO.

Listagem 3.7: Implementação das transmissões do modelo virtual

```
<transmission name="upper_chassis_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="upper_chassis_motor">
      <hardwareInterface>EffortJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
    <joint name="upper_chassis_joint">
      <hardwareInterface>EffortJointInterface</hardwareInterface>
    </joint>
</transmission>
```

4) Plugins: O *ros_control*, um conjunto de pacotes de controladores que atuam sobre as *joints* da cadeira, em conjunto com o *plugin gazebo_ros_control* são utilizados para simular os controladores da cadeira de rodas no Gazebo. Para a sua integração é necessário incluir, no ficheiro XACRO, as transmissões já anteriormente abordadas e o *gazebo_ros_control*, tal como demonstra o seguinte código:

```
<gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/</robotNamespace>
    </plugin>
</gazebo>
```

3.3.2.5 Controladores e Plugins

Ao utilizar o *plugin ros_control*, é necessário criar ficheiros de configuração e ficheiros de execução para os controladores que são carregados no simulador Gazebo. Os ganhos dos controladores Proportional Integral Derivative (PID) e as suas definições são gravados num ficheiro do tipo YAML, na pasta "CONFIG", e são posteriormente inicializados no ficheiro "world", dentro da pasta "LAUNCH". O ficheiro "diff_drive" contém as definições do controlador *diff_drive_controller* e é responsável pela rotação das *joints* entre as rodas dianteiras e o *chassis*. O ficheiro "effort" contem as definições dos controladores *JointPositionController* das *joints upper_chassis_joint* e *back_joint*, que correspondem ao movimento vertical do assento e inclinação do apoio para as costas, respetivamente. Por fim, o ficheiro joint_states contém as definições do controlador *joint_state_controller*, o qual monitoriza o estado das *joints*. Os controladores e as respetivas *joints* encontram-se

sistematizados no diagrama da figura 3.31. Os ficheiros do tipo LAUNCH, para além de inicializar os controladores e respetivas configurações, definem ainda as características do mundo virtual que irá correr no simulador Gazebo, a pose com que a cadeira é gerada no mundo e executam os *plugins* em paralelo com a simulação.

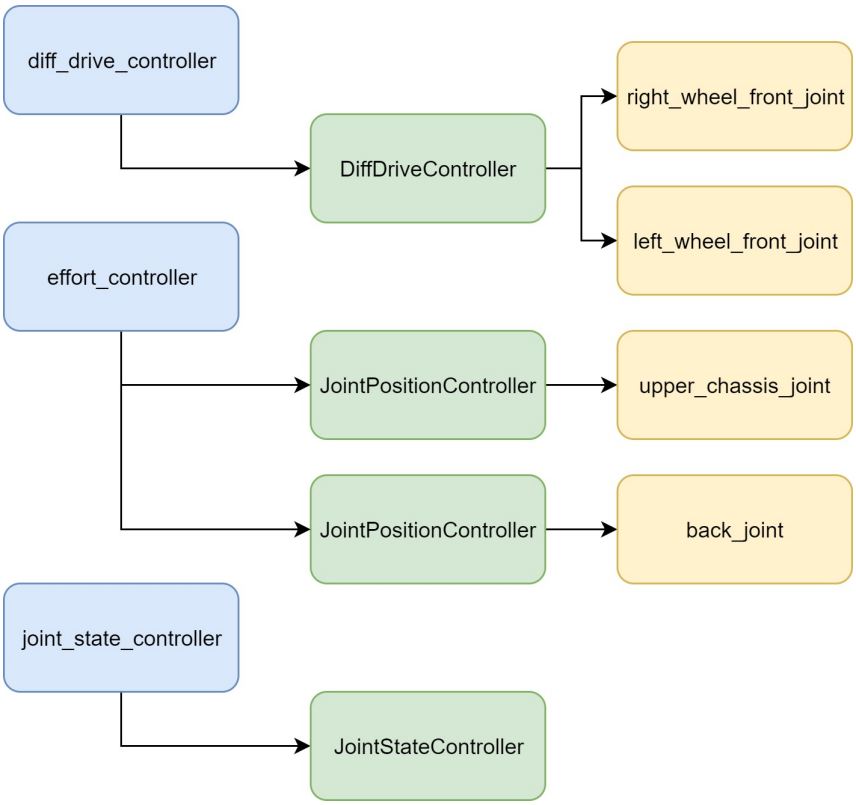


Figura 3.31: Tipos de controladores e respetivas *joints* utilizadas no modelo

Os ganhos dos controladores PID das *joints* de apoio para as costas e do assento foram ajustados com o auxílio de um *plugin* que pertence à ferramenta de desenvolvimento *Graphical User Interface (GUI)* para ROS, *rqt*. O *rqt* apresenta diversas funcionalidades e *plugins* que permitem interagir, em tempo real, com o modelo da cadeira de rodas no simulador Gazebo e até alterar e/ou monitorizar os seus parâmetros. Alguns dos *plugins rqt* utilizados foram os seguintes:

1. **Dynamic Reconfigure** - Este *plugin* permite o ajuste, em tempo real, dos ganhos dos controladores PID, tal como demonstra a figura 3.32, e de parâmetros físicos da simulação no Gazebo.

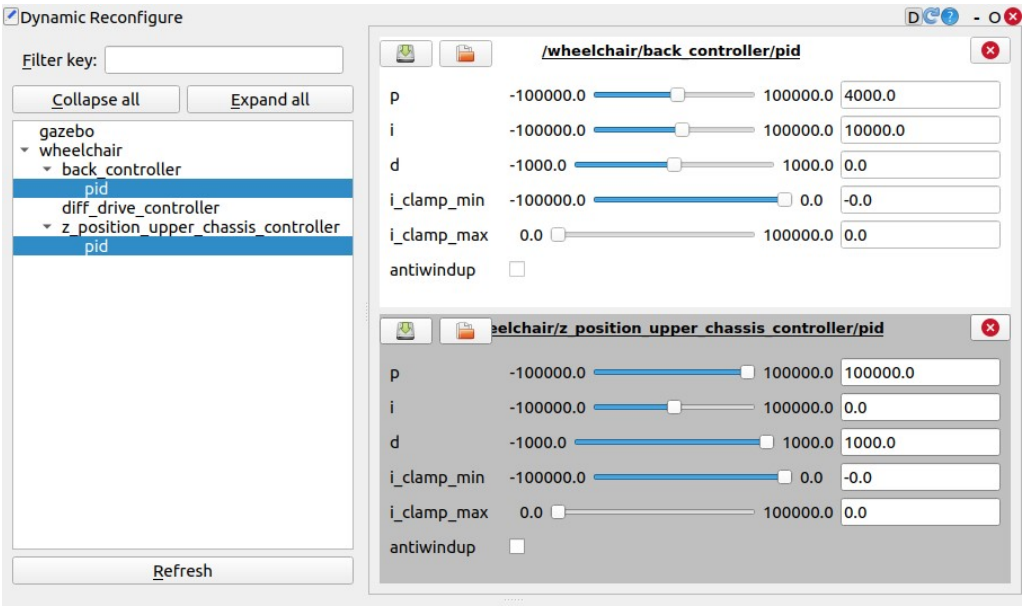


Figura 3.32: Configuração dos ganhos dos controladores PID

2. **MatPlot** - *Plugin* que consiste na apresentação de gráficos com a variação temporal dos valores de um determinado tópico ROS. É particularmente útil para observar a diferença entre os valores enviados dos controladores para as *joints* e os seus valores reais, tal como demonstra o exemplo da figura 3.33. No eixo dos xx tem-se a variável temporal e no eixo dos yy a variável que consta no tópico selecionado.

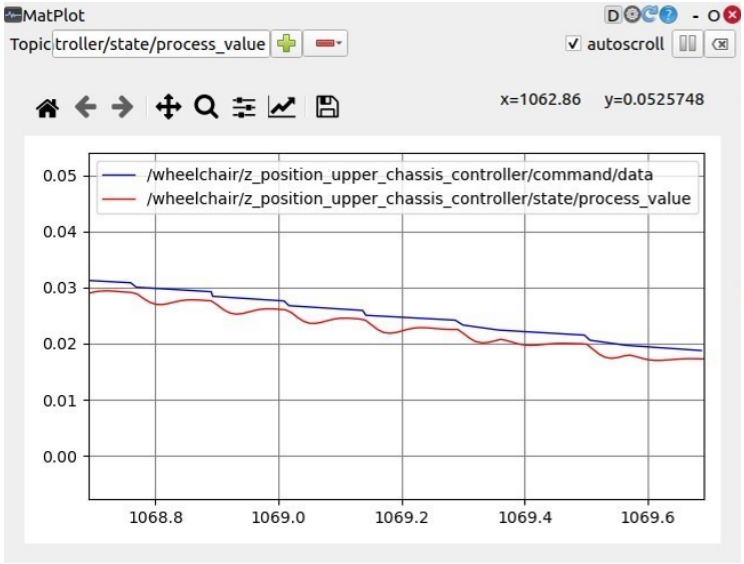


Figura 3.33: Observação dos valores em diversos tópicos ROS presentes no sistema

3. **Message Publisher** - Este *plugin* permite enviar dados para um determinado tópico ROS, a uma determinada frequência, através de expressões numéricas. O tipo de dados tem de ser especificado, a frequência, em Hz, com que se quer enviar e a o respetivo valor. Um exemplo deste tipo de envio está demonstrado na figura 3.34,

onde, neste caso, é enviada uma sinusoide para um determinado tópico a uma frequência de 50 Hz.

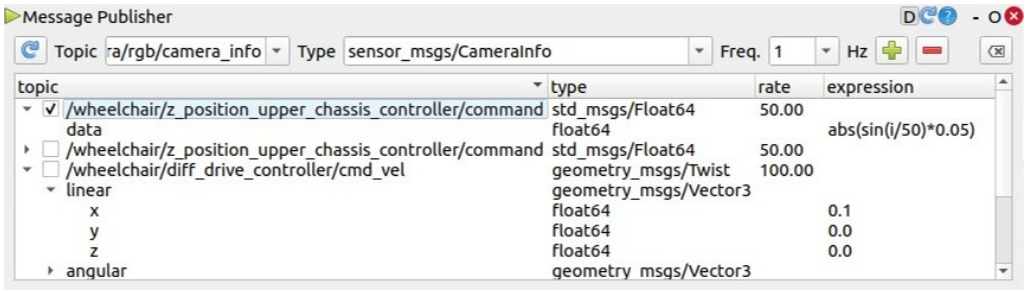


Figura 3.34: Envio de dados para um determinado tópico ROS do sistema

4. **Robot Steering** - Este *plugin* está adaptado para a utilização com o controlador *diff_drive_controller* e possui uma interface que permite o envio da velocidade linear, em milissegundos, e angular, em radianos por segundo, para o tópico *cmd_vel*, tal como demonstra a figura 3.35. Este *plugin* foi utilizado para controlar o deslocamento da cadeira de rodas no simulador Gazebo.

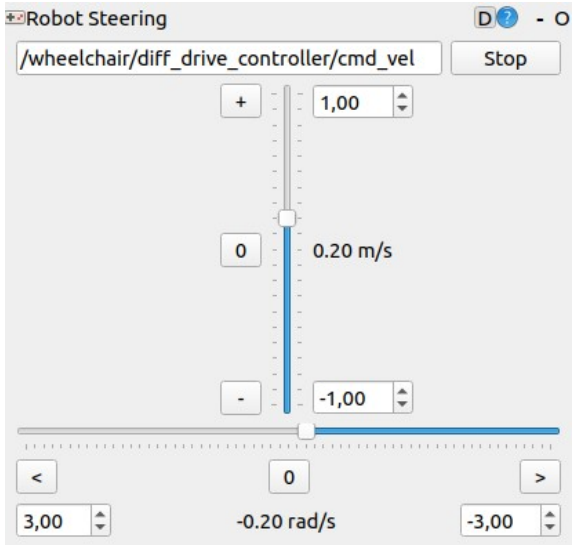


Figura 3.35: Envio da velocidade linear e angular para o tópico *cmd_vel*

5. **Node Graph** - *Plugin* que permite a visualização de todos os nós ROS e quais são os tópicos que estes estão a subscrever ou para quais estão a publicar mensagens. Repare-se, pela figura 3.36, que os nós "/gazebo" e "/Python_ROS" dizem respeito ao modelo da cadeira no simulador e o ficheiro de sincronização com a entidade física, respetivamente. A caixa "/wheelchair" representa o pacote ROS com os diversos tópicos dos controladores, que são subscritos pelo "/gazebo" e "/Python_ROS", como se pode observar pela direção das setas. O tópico "/controller_spawner" inicializa os controladores no Gazebo, o tópico "/gazebo_gui" inicializa o próprio Gazebo e os

seus *plugins* e os restantes tópicos correspondem ao estado e comando enviado para cada um dos controladores das diferentes *joints* do sistema.

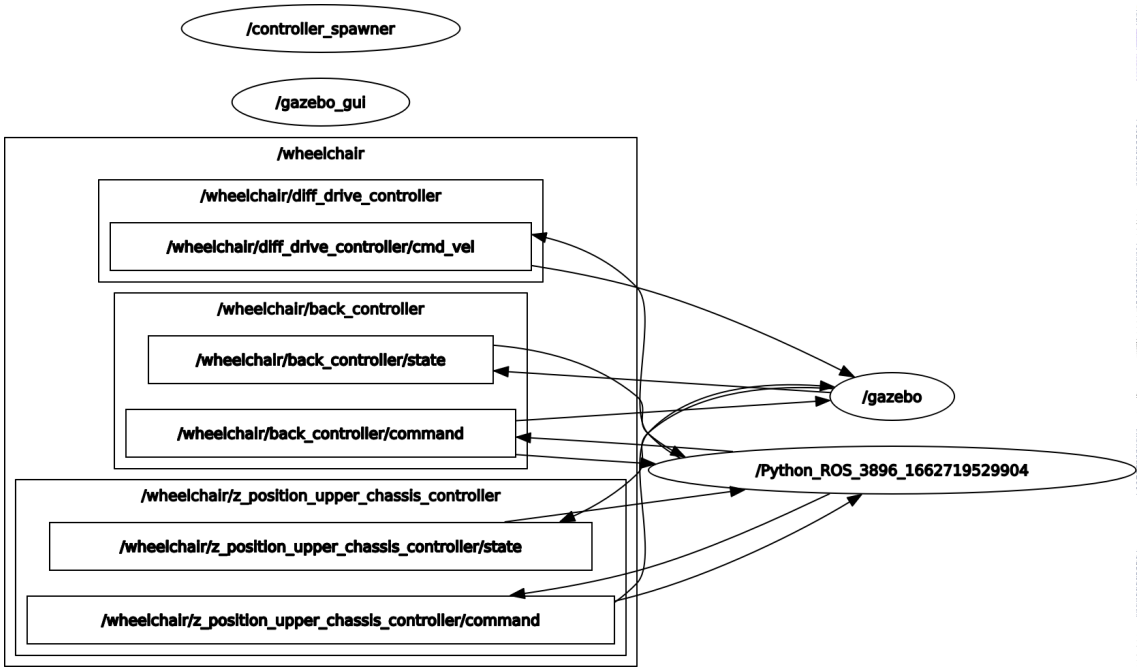


Figura 3.36: Visualização dos nós e tópicos ROS do sistema

3.3.2.6 Simulador Gazebo

O Gazebo é um simulador 3D que permite simular o comportamento não apenas de robôs, como também de outros componentes hardware, com a integração de sensores e atuadores em cenários idênticos à realidade, incluindo os seus parâmetros físicos, tal como a gravidade. No contexto deste sistema, o Gazebo foi utilizado para observar o comportamento do protótipo físico desenvolvido da cadeira de rodas, através do espelhamento do movimento dos seus motores, tendo em conta a sua posição. O mundo gerado para a simulação, descrito no ficheiro "empty"dentro da pasta "WORLDS", é um mundo vazio, apenas contendo o solo. Isto porque o objetivo principal deste projeto é a monitorização da cadeira no mundo real, e não no virtual, ou seja, caso haja obstáculos no mundo real, vai-se perceber na simulação no Gazebo que algo gerou contacto com a entidade física. Já foram incluídas, neste documento, algumas imagens retiradas do simulador Gazebo, tal como 3.30 e 3.28.

3.3.2.7 Ficheiro de Sincronização com a Entidade Física

A última pasta que ainda não foi abordada, a "SRC", contém o ficheiro "digital_twin", o qual possui algoritmos para sincronizar o comportamento da entidade física com o modelo digital, através da posição dos seus motores. O modo de execução do programa consiste, numa fase inicial, na subscrição de tópicos MQTT e ROS, associados a cada motor da cadeira (controladores no caso do ROS). De seguida, dá-se um ciclo onde o programa

vai estar a receber, de forma contínua, mensagens nos tópicos MQTT que subscreveu, relativos ao estado dos motores e, caso este tenha mudado, o sistema vai atuar sobre o modelo digital, enviando mensagens através de *publishers* para as respetivas *joints* de cada motor no ROS. As mensagens enviadas pelo EV3 contêm informação sobre os movimentos dos três grupos de motores da cadeira, nomeadamente se o assento está a subir ou descer, se o apoio para as costas está a inclinar para a frente ou para trás e para que direção a cadeira se está a mover. No caso dos motores das rodas, as mensagens contêm ainda a velocidade a que se encontram, algo que vai ser equacionado para a sincronização com o modelo digital. O espelhamento do comportamento da entidade física no modelo virtual foi possível pelo facto da velocidade das rodas ter sido ajustada para que, quando a cadeira real se deslocasse, a mesma distância seria percorrida no modelo virtual. O mesmo acontece para todos os restantes cenários, nomeadamente para o motor que move o assento e o motor que move o apoio para as costas. No diagrama da figura 3.37, são demonstrados os passos de execução do programa. Tanto a velocidade do motor do assento, como a do apoio para as costas foram previamente ajustadas para igualar ambos os modelos físico e digital.

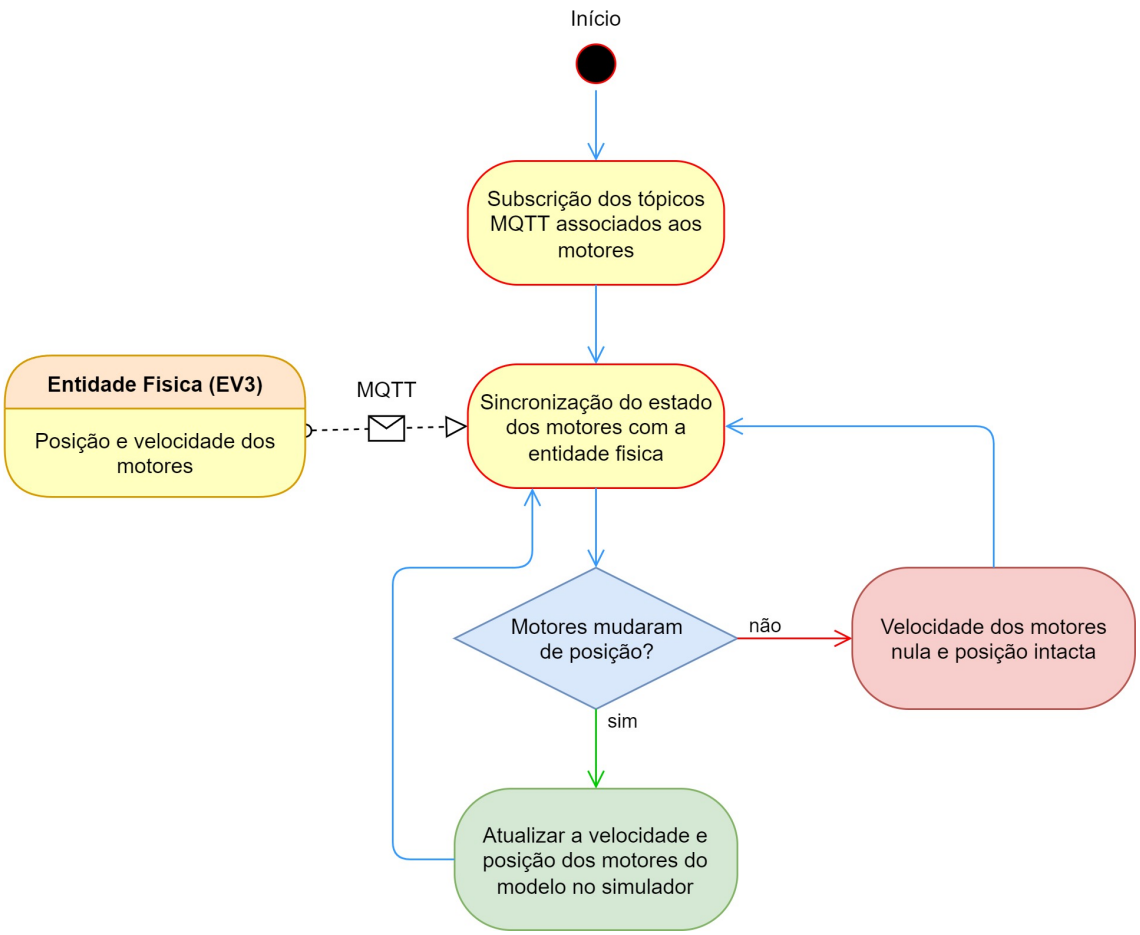


Figura 3.37: Modo de execução do programa de sincronização do modelo digital com a entidade física

A nível de programação do ficheiro, utilizou-se a linguagem Python, conjuntamente com o módulo *rospy*, uma biblioteca dedicada para a utilização com o ROS e que permite a integração de tópicos, serviços e parâmetros ROS. No código da listagem 3.8, foi utilizado o *rospy* para iniciar um nó, que neste caso é a máquina a correr o *script*, subscrever os tópicos ROS das *joints* equivalentes a cada motor da cadeira e definir *publishers* que servem para enviar dados para os respetivos tópicos. Repare-se que cada subscrição de tópico possui um *callback* associado, o que quer dizer que sempre que o valor do tópico muda, a função *callback* é chamada.

Listagem 3.8: Utilização do *rospy* para a ligação com os tópicos ROS

```
# Initiate ROS node on this script
rospy.init_node('Python_ROS', anonymous=True)

# Subscribe ROS topics and define callbacks for each one
rospy.Subscriber("/wheelchair/back_controller/command/", Float64,
    callback_back_data)
rospy.Subscriber("/wheelchair/back_controller/state/",
    JointControllerState, callback_back_state)
rospy.Subscriber("/wheelchair/z_position_upper_chassis_controller/"
"command/", Float64, callback_up_down_data)
rospy.Subscriber("/wheelchair/z_position_upper_chassis_controller/state",
    JointControllerState, callback_up_down_state)

# Define publishers for each topic
pub_drive = rospy.Publisher('/wheelchair/diff_drive_controller/cmd_vel/',
    Twist, queue_size=1)
pub_up_down =
    rospy.Publisher('/wheelchair/z_position_upper_chassis_controller/'
'command/', Float64, queue_size=1)
pub_back = rospy.Publisher('/wheelchair/back_controller/command/', Float64,
    queue_size=1)
```

3.4 Interface de Monitorização

Com o intuito de facilitar a monitorização do comportamento da cadeira de rodas, foi desenvolvida uma interface que dispõe de diversas ferramentas de observação de dados, como um gráfico para verificar as variações de rotação sobre cada eixo da cadeira, caixas de texto com a indicação do estado de cada motor, um botão *switch* para guardar os dados num ficheiro CSV e um sistema de notificações que, quando é ultrapassado um determinado limite de rotação sobre um determinado eixo, uma notificação é enviada para o utilizador. A implementação desta interface foi feita com base no Node-RED, uma plataforma de desenvolvimento baseada em fluxo de programação visual (*low-code*) para

conectar dispositivos hardware, API's e serviços online como IoT. O seu ambiente de desenvolvimento encontra-se representado na figura 3.38, incluindo o sistema desenvolvido, também disponível, sob o formato JSON, no repositório GitHub [63], o qual inclui a lógica programada em JavaScript para a receção de dados e envio de notificações. As caixas a roxo representam a subscrição de tópicos MQTT, sendo que as mensagens neles recebidas saem para as funções à direita, a laranja. Nestas funções é processada a informação recebida para ser apresentada nos gráficos e em caixas de texto da interface.

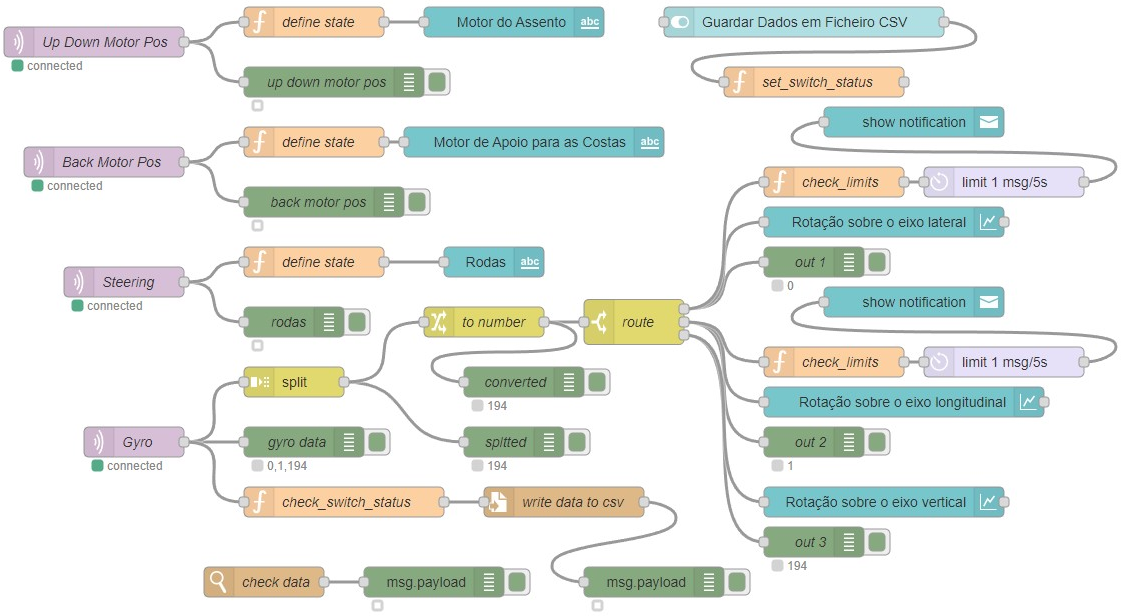


Figura 3.38: Ambiente de programação Node-RED com "*flow*" de desenvolvimento da interface de monitorização

É possível observar, a partir da interface da figura 3.39, os três gráficos correspondentes à rotação da cadeira sobre os três diferentes eixos, sendo que o lateral e longitudinal estão calibrados para começarem sempre em 0. Caso haja uma rotação sobre algum destes eixos, a curva sobe ou desce consoante a direção de inclinação. Para além dos gráficos, é também possível observar três estados a negrito, no canto inferior direito, os quais correspondem aos estados dos três grupos de motores da cadeira de rodas. Finalmente, existe ainda um botão *switch* que, ao ser pressionado, guarda os valores de rotação dos eixos e os estados dos motores num ficheiro CSV até o botão ser novamente pressionado. Uma nota importante é o facto do fuso horário do Node-RED estar uma hora adiantada face aos dados registados que foram guardados no ficheiro CSV.

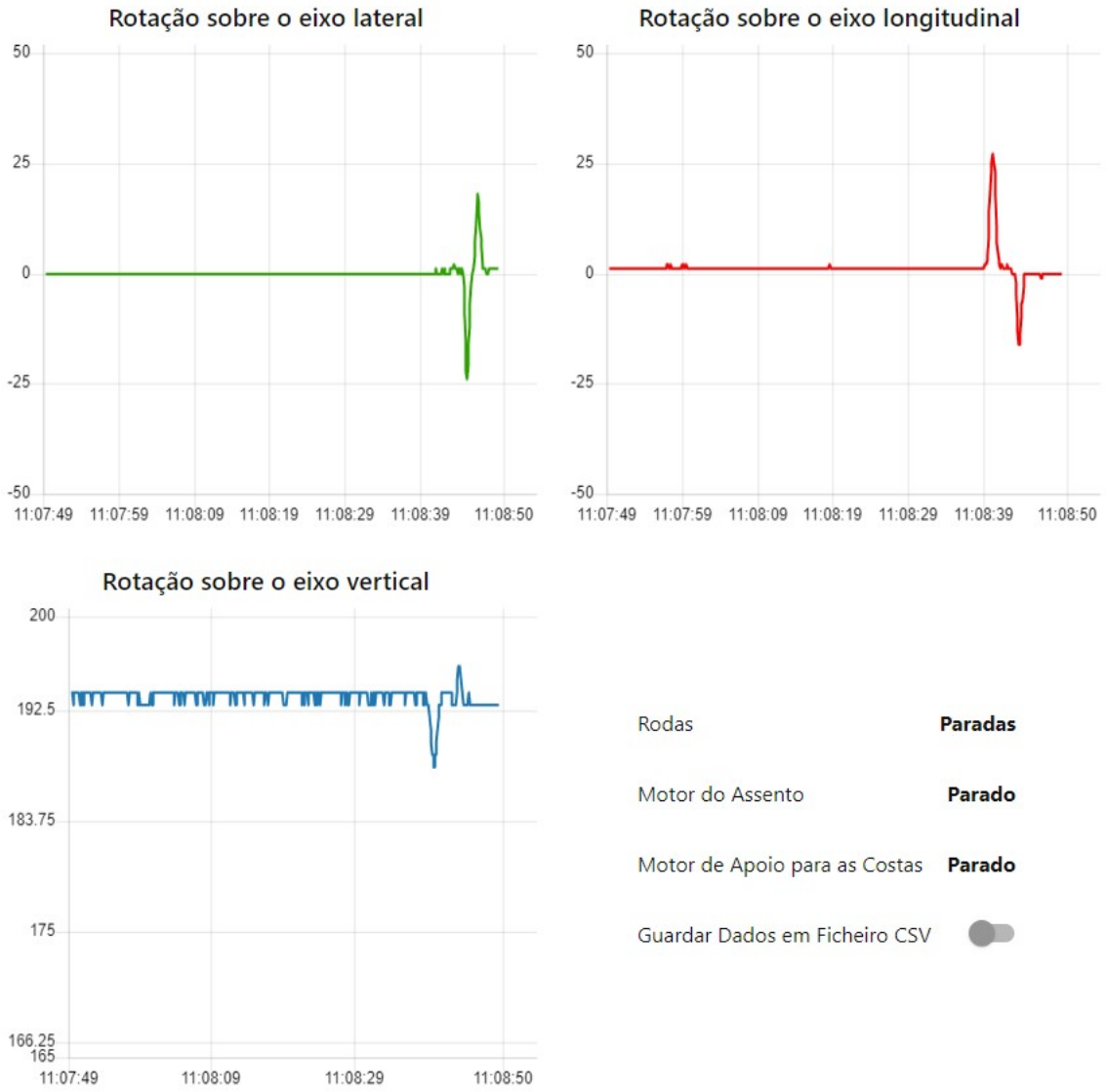


Figura 3.39: Interface desenvolvida para a monitorização do comportamento da cadeira

3.5 Comunicação entre os Elementos do Sistema

A comunicação entre os elementos do sistema desenvolvido é feita com base no protocolo MQTT, como já foi anteriormente mencionado. A lógica por trás deste tipo de protocolo consiste na existência de um *broker* que é responsável pela gestão das comunicações entre os nós que estão ligados a si para subscrever ou publicar mensagens para diferentes tópicos. Neste sistema, uma placa Raspberry Pi assume o papel de *broker*, enquanto os nós são os elementos da arquitetura descritos anteriormente. Existem quatro tópicos principais, através dos quais são enviadas mensagens relativas ao estado dos motores e sensores da cadeira: "/up_down_motor_pos", "/back_motor_pos", "/steering" e "/gyro". No tópico "/up_down_motor_pos" são enviadas mensagens sobre o comportamento vertical do motor do assento, no "/back_motor_pos" mensagens sobre a inclinação do apoio para as

costas, no "/steering" mensagens sobre o comportamento das rodas e no "/gyro" mensagens sobre a rotação da cadeira sobre os três diferentes eixos. No diagrama da figura 3.40, é possível observar a estrutura da comunicação do sistema, através do protocolo MQTT.

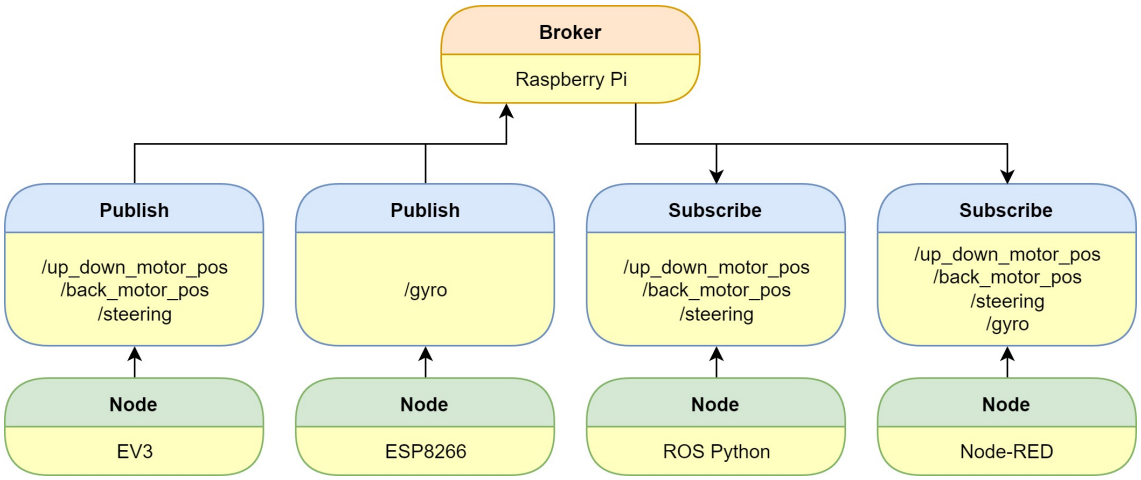


Figura 3.40: Estrutura da comunicação entre os elementos do sistema, através do protocolo MQTT

3.6 Armazenamento de Dados

Os dados gerados pela cadeira de rodas encontram-se, não apenas na interface de monitorização, mas também guardados num ficheiro sob o formato CSV. Isto permite criar tabelas a partir da informação guardada e utilizá-la para qualquer finalidade. A separação de colunas do ficheiro é equivalente à tabela 3.5 e conta com a coluna da "Data", correspondente à data real, incluindo as horas, minutos, segundos e milésimos de segundo, outra para o estado de cada motor e outra para a rotação sobre cada eixo.

Tabela 3.5: Exemplo de uma tabela gerada no formato CSV com os dados da cadeira

Data	Assento	Costas	Rodas	Eixo Lat.	Eixo Lon.	Eixo Ver.
2022-09-12T16:25:47.703Z	Parado	Parado	Paradas	1	0	194
2022-09-12T16:25:47.805Z	Parado	Parado	Paradas	1	0	194
2022-09-12T16:25:47.907Z	Parado	Parado	Paradas	1	0	194
2022-09-12T16:25:48.010Z	Parado	Parado	Paradas	1	0	194
2022-09-12T16:25:48.113Z	Parado	Parado	Paradas	1	0	193
2022-09-12T16:25:48.215Z	Parado	Parado	Paradas	1	0	193
2022-09-12T16:25:48.318Z	Parado	Parado	Paradas	1	0	193
2022-09-12T16:25:48.421Z	Parado	Parado	Paradas	1	0	193

Capítulo 4

Testes de Validação e Análise de Resultados

4.1 Introdução

Neste capítulo são apresentados e analisados os testes de validação dos modelos e algoritmos desenvolvidos. Inicialmente, é feito um teste do funcionamento do sistema, testando diversos aspetos como o controlo da entidade física, a comunicação entre os elementos e a monitorização dos dados. De seguida, são analisados alguns cenários particulares, de modo a compreender o comportamento da cadeira, face a certas adversidades, e a monitorizar os seus parâmetros.

4.2 Operação do Sistema

4.2.1 Descrição

A operação do sistema é feita com base no controlo da entidade física a partir dos botões do controlador remoto, os quais controlam os movimentos dos motores da cadeira de rodas. De modo a testar o funcionamento de todos os elementos do sistema, foi executada uma sequência de comandos, a partir do controlador. Não foram incluídos os botões azuis para os canais 2 e 3, uma vez que têm um comportamento idêntico aos botões vermelhos, na medida em que fazem rodar os motores à mesma velocidade, com a diferença de que a rotação do motor só é feita se o botão estiver a ser constantemente pressionado. Posto isto, a sequência de comandos executada, juntamente com as ações programadas, foi a seguinte:

1. **Botão Vermelho Superior (Canal 1)** - Programado para mover a cadeira para frente a uma velocidade linear constante de 20% (percentagem da velocidade máxima dos motores) e *steering* de 0 (equivalente a deslocar numa linha reta).

2. **Botão Vermelho Inferior (Canal 1)** - Idêntico ao comando anterior, mas com a percentagem da velocidade negativa para se mover para trás.

3. **Botão Azul Superior (Canal 1)** - Programado para virar a cadeira para a sua esquerda, a uma velocidade angular de 9% e *steering* de -100.

4. **Botão Azul Inferior (Canal 1)** - Idêntico ao comando anterior, com *steering* de 100 para virar à direita.

5. **Botão Vermelho Superior (Canal 2)** - Programado para subir o assento da cadeira, a uma velocidade constante de -10% (percentagem da velocidade máxima do motor), até à posição limite definida de -215 (valores negativos devido à posição do motor na cadeira).

6. **Botão Vermelho Inferior (Canal 2)** - Programado para descer o assento da cadeira, a uma velocidade constante de 10% (simétrico ao anterior), até à posição limite definida de 0 (posição inicial).

7. **Botão Vermelho Superior (Canal 3)** - Programado para inclinar o apoio para as costas da cadeira para a frente, a uma velocidade constante de -5% (percentagem da velocidade máxima do motor), até à posição limite definida de -95 (valores negativos devido à posição do motor na cadeira).

8. **Botão Vermelho Inferior (Canal 3)** - Programado para inclinar o apoio para as costas da cadeira para trás, a uma velocidade constante de 5%, até à posição limite definida de 0 (posição inicial, antes da sincronização com o modelo virtual).

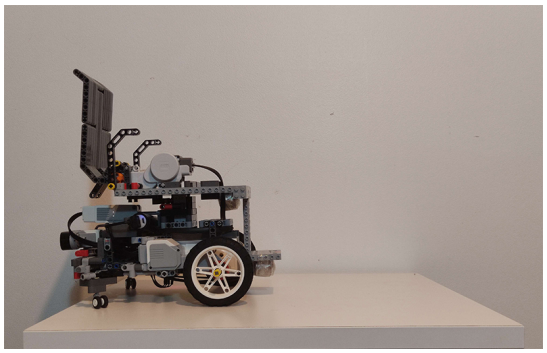
4.2.2 Resultados

Dada a sequência de comandos executada, foram analisados os resultados e guardados num ficheiro CSV, de forma a confirmar as ações que, teoricamente, estavam programadas para acontecer e se a entidade virtual replicou corretamente o comportamento da entidade física. Antes de se pressionar os botões do controlador remoto, colocou-se a cadeira numa superfície plana, de modo que esta conseguisse realizar todos os movimentos necessários.

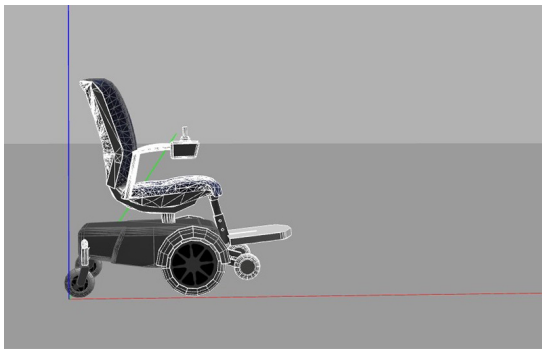
4.2.2.1 Estado Inicial

O estado inicial da cadeira de rodas pode ser observado na figura 4.1, em que a figura 4.1a corresponde à entidade física e a figura 4.1b à entidade virtual, em que ambos têm os motores nas mesmas posições, já depois do acerto da posição do motor de apoio para as costas por parte da entidade física, como demonstra a mensagem seguinte, na consola do ev3dev:

```
Starting: brickrun --directory="/home/robot/ev3dev-wheelchair"
    "/home/robot/ev3dev-wheelchair/main.py"
----------
Posicao Motor Costas: -47
```



(a) Entidade física no seu estado inicial



(b) Entidade virtual no seu estado inicial

Figura 4.1: Entidades física e virtual nos seus estados iniciais

Na interface de monitorização da figura 4.2, pode confirmar-se, pelo estado dos motores, que a cadeira se encontra em repouso, o que corresponde ao estado das suas entidades física e virtual. É importante relembrar que esta informação é sincronizada através do protocolo de comunicação MQTT, com o Node-RED a subscrever os tópicos dos estados dos motores, que por sua vez são publicados pela entidade física, algo que permite diagnosticar o correto funcionamento da comunicação. Nas secções seguintes são apresentados e analisados os resultados da execução dos comandos anteriormente numerados.

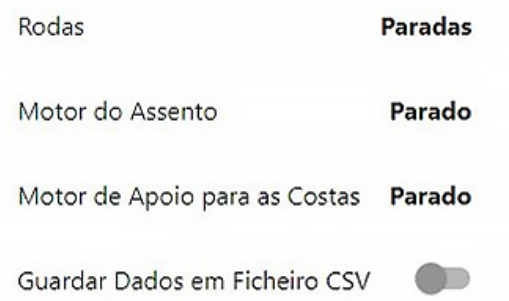


Figura 4.2: Interface de monitorização no estado inicial da cadeira

4.2.2.2 Execução do Comando 1 - Deslocação para a Frente

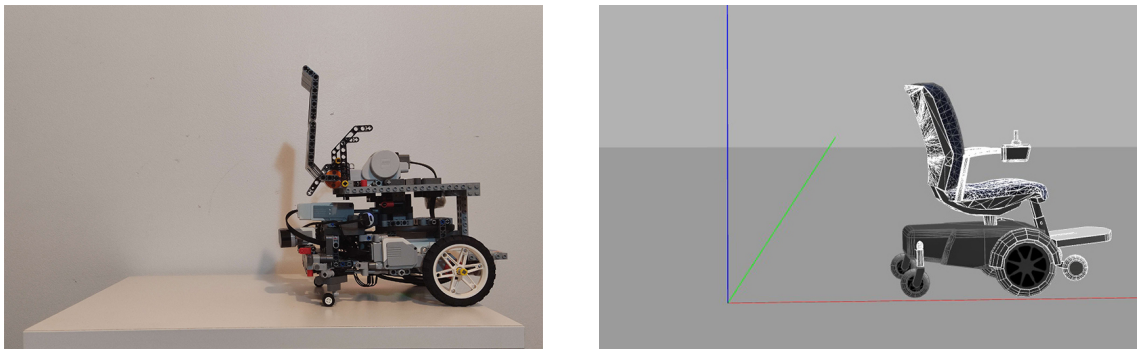
O primeiro comando enviado para o EV3 serviu para rodar os motores das rodas dianteiras, de modo que a cadeira se deslocasse para a frente. Na figura 4.3, é possível observar o estado das entidades física e virtual, após a execução deste comando, onde se conclui que ambas as cadeiras avançaram uma distância idêntica. Durante a deslocação da cadeira para a frente, foi registado o estado das suas rodas na interface de monitorização, tal como

demonstra a figura 4.4. De forma a registar a posição e velocidade dos motores das rodas durante a execução do comando, foram programadas mensagens de *debug* na consola do ev3dev, como demonstra a listagem 4.1.

Listagem 4.1: Posição e velocidade dos motores durante a execução do comando 1

```
Posicao Roda Esquerda: 0
Posicao Roda Direita: 0
Velocidade Roda Esquerda: 0
Velocidade Roda Direita: 210
Posicao Roda Esquerda: 16
Posicao Roda Direita: 17
Velocidade Roda Esquerda: 210
Velocidade Roda Direita: 210
Posicao Roda Esquerda: 41
Posicao Roda Direita: 36
...
Posicao Roda Esquerda: 413
Posicao Roda Direita: 409
```

Como se pode observar, a velocidade das rodas começa a 0, partindo do estado inicial, e de seguida, após o comando, a da direita sobe para 210 (unidade de rotações do codificador), mantendo este valor constante até parar. A roda esquerda assume o mesmo valor da roda direita, derivado da utilização da classe MoveTank que sincroniza ambas as velocidades. Relativamente à posição, esta começa igualmente a 0 e aumenta progressivamente até à paragem da cadeira, nunca sendo exatamente o mesmo valor em cada roda, devido a variações de direção no momento do arranque, causado pelo atrito das rodas traseiras de suporte.



(a) Estado da entidade física após execução do comando 1

(b) Estado da entidade virtual após execução do comando 1

Figura 4.3: Estado das entidades física e virtual após execução do comando 1

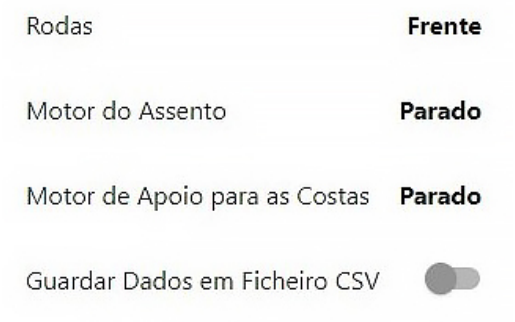


Figura 4.4: Interface de monitorização durante a execução do comando 1

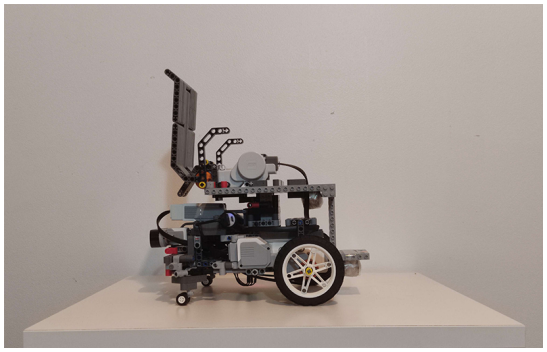
4.2.2.3 Execução do Comando 2 - Deslocação para Trás

O segundo comando deslocou a cadeira ligeiramente para trás, como demonstra a figura 4.5b, o que corresponde à posição da entidade virtual da figura 4.5b e ao estado das rodas da interface da figura 4.6. À semelhança do que foi feito com o comando anterior, nomeadamente a programação de mensagens com informação referente à velocidade e posição dos motores das rodas, para este comando fez-se o mesmo, sendo a saída na consola a da listagem 4.2.

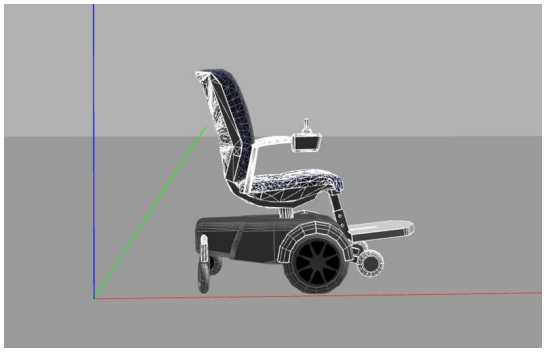
Listagem 4.2: Posição e velocidade dos motores durante a execução do comando 2

```
Velocidade Roda Esquerda: -210
Velocidade Roda Direita: -210
Posicao Roda Esquerda: 401
Posicao Roda Direita: 401
Velocidade Roda Esquerda: -210
Velocidade Roda Direita: -210
Posicao Roda Esquerda: 376
Posicao Roda Direita: 370
...
Posicao Roda Esquerda: 32
Posicao Roda Direita: 23
```

Assumindo que a posição das rodas é equivalente à posição final do comando anterior, ao executar o segundo comando, a velocidade das rodas assume o valor negativo constante -210 (por estar a andar para trás) e a posição das rodas vai diminuindo, como seria de esperar, até às posições 32, para a roda esquerda, e 23, para a roda direita. Estes valores não atingem o valor 0 uma vez que a cadeira não se deslocou totalmente para a posição do estado inicial. A diferença que existe na posição de cada roda provocou até um pequeno desvio na direção da cadeira real, apesar de não ser visível, algo que não aconteceu na entidade virtual, pelo facto do atrito ser desprezável nas rodas de suporte.



(a) Estado da entidade física após execução do comando 2



(b) Estado da entidade virtual após execução do comando 2

Figura 4.5: Estado das entidades física e virtual após execução do comando 2

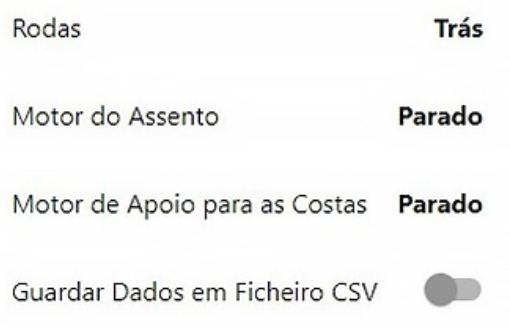


Figura 4.6: Interface de monitorização durante a execução do comando 2

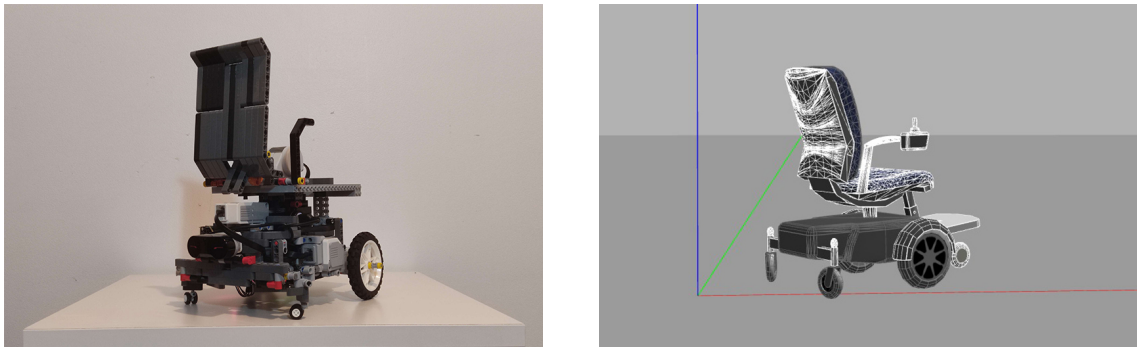
4.2.2.4 Execução do Comando 3 - Virar à Esquerda

O terceiro comando rodou a cadeira sobre si mesma, para a esquerda, com uma velocidade angular constante, deixando-a na posição da figura 4.7a. A entidade virtual da figura 4.7b encontra-se numa posição idêntica à cadeira real, no entanto dá a sensação que não rodou tanto como devia, o que pode ser justificado pela pequena mudança de direção causada na deslocação anterior. Durante o movimento da cadeira, registou-se, na interface de monitorização da figura 4.8, a sua rotação sobre o eixo vertical, com o ângulo de direção a sofrer uma diferença negativa, assim que é iniciado o movimento, o que faz sentido, uma vez que a cadeira rodou. O estado das rodas apresenta o texto "Esquerda", algo que está coerente com o movimento das rodas. Á semelhança dos comandos anteriores, também se registaram mensagens na consola do EV3 para monitorizar a posição e velocidade das rodas, as quais se encontram na listagem 4.3.

Listagem 4.3: Posição e velocidade dos motores durante a execução do comando 3

```
Velocidade Roda Esquerda: -94
Velocidade Roda Direita: 94
Posicao Roda Esquerda: 24
Posicao Roda Direita: 30
Velocidade Roda Esquerda: -94
Velocidade Roda Direita: 94
Posicao Roda Esquerda: 18
Posicao Roda Direita: 33
...
Posicao Roda Esquerda: -165
Posicao Roda Direita: 224
```

A velocidade das rodas, sendo simétrica, provoca a rotação da cadeira sobre o eixo vertical, neste caso para a esquerda, alterando também a sua posição no mesmo sentido da velocidade. Repare-se que a posição da roda esquerda diminui ao longo do tempo, ao passo que a da roda direita aumenta.



(a) Estado da entidade física após execução do comando 3

(b) Estado da entidade virtual após execução do comando 3

Figura 4.7: Estado das entidades física e virtual após execução do comando 3

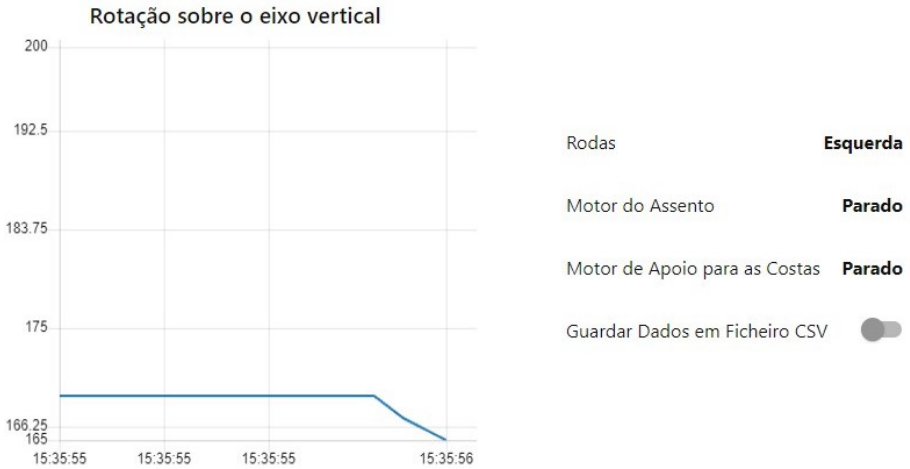


Figura 4.8: Interface de monitorização durante a execução do comando 3

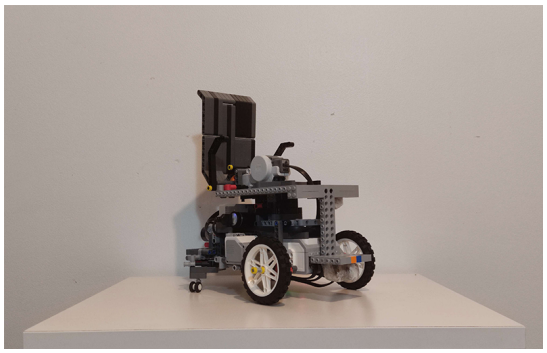
4.2.2.5 Execução do Comando 4 - Virar à Direita

Comando semelhante ao anterior, mudando apenas o sentido de rotação da cadeira sobre o eixo vertical, o qual se pode verificar no gráfico da interface da figura 4.10. A velocidade angular mantém-se constante, mas simétrica para cada roda. A posição final da entidade física encontra-se representada na figura 4.9a e a da entidade virtual na figura 4.9b. Mais uma vez, as entidades encontram-se em posições semelhantes, com uma pequena diferença causada pelo movimento do primeiro comando. A velocidade e posição dos motores registados são as que constam na listagem 4.4.

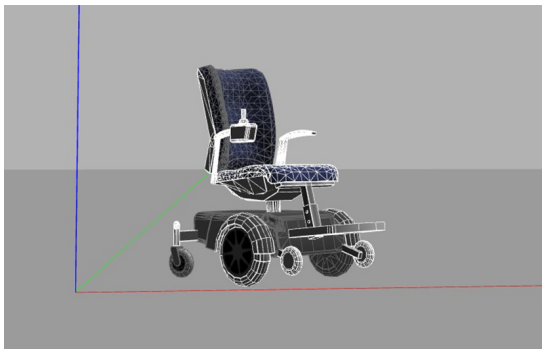
Listagem 4.4: Posição e velocidade dos motores durante a execução do comando 4

```
Velocidade Roda Esquerda: 94
Velocidade Roda Direita: -94
Posicao Roda Esquerda: -156
Posicao Roda Direita: 218
Velocidade Roda Esquerda: 94
Velocidade Roda Direita: -94
Posicao Roda Esquerda: -142
Posicao Roda Direita: 191
...
Posicao Roda Esquerda: 48
Posicao Roda Direita: 3
```

Tal como foi dito, a velocidade é simétrica ao comando anterior, por estar a rodar para a direita, e a posição da roda esquerda aumenta ao longo do tempo, enquanto a da roda da direita diminui.



(a) Estado da entidade física após execução do comando 4



(b) Estado da entidade virtual após execução do comando 4

Figura 4.9: Estado das entidades física e virtual após execução do comando 4

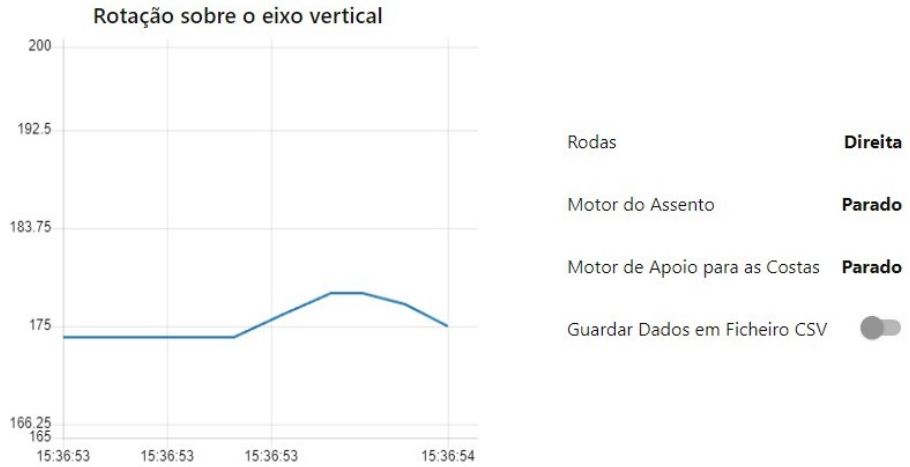


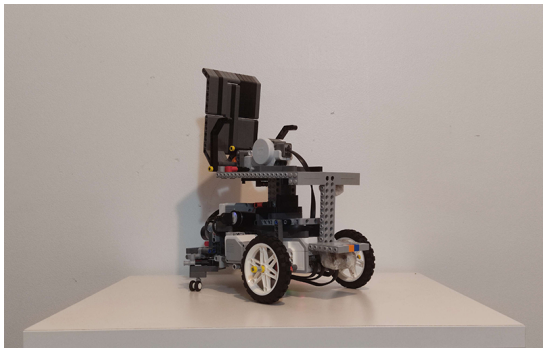
Figura 4.10: Interface de monitorização durante a execução do comando 4

4.2.2.6 Execução do Comando 5 - Subir o Assento

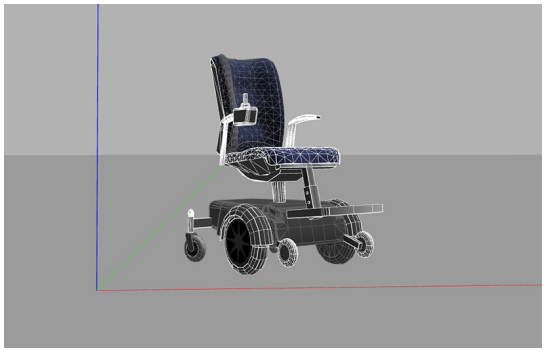
Este comando já controla outra componente da cadeira de rodas, o movimento vertical do assento, o qual foi direcionado para cima, deixando a cadeira nas posições das figuras figuras 4.11a e 4.11b, correspondentes à entidade física e virtual. Comparando com as figuras do comando anterior, nota-se a diferença na subida do assento em ambas as entidades. Recorrendo ao *plugin* MatPlot da figura 4.12, durante a execução do comando, verifica-se que o valor enviado para o tópico do controlador da *joint* do assento, *data*, é muito semelhante ao valor registado no simulador, *process_value*, contendo apenas um pequeno atraso na resposta, derivado da regulação dos ganhos PID. A nível de comunicação, o estado deste motor está a ser subscrito pela interface de monitorização da figura 4.13, a qual apresenta corretamente a palavra "Cima". Foram igualmente registados o estado e a posição deste motor no final do seu movimento, sendo que o resultado foi o seguinte:

```
Estado Motor Assento: ['holding']
Posicao Motor Assento: -211
```

Note-se que o estado do motor no final do movimento é de *holding*, o que significa que o motor está propositadamente a manter a sua posição, não permitindo a sua rotação. Esta situação foi programada para que não houvesse forças externas que conseguissem, com facilidade, alterar a posição dos motores e perturbar a movimentação da cadeira. A posição final do motor de -211 representa o limite superior definido para o assento da cadeira e apresenta-se negativa pela forma como este foi colocado no protótipo.



(a) Estado da entidade física após execução do comando 5



(b) Estado da entidade virtual após execução do comando 5

Figura 4.11: Estado das entidades física e virtual após execução do comando 5

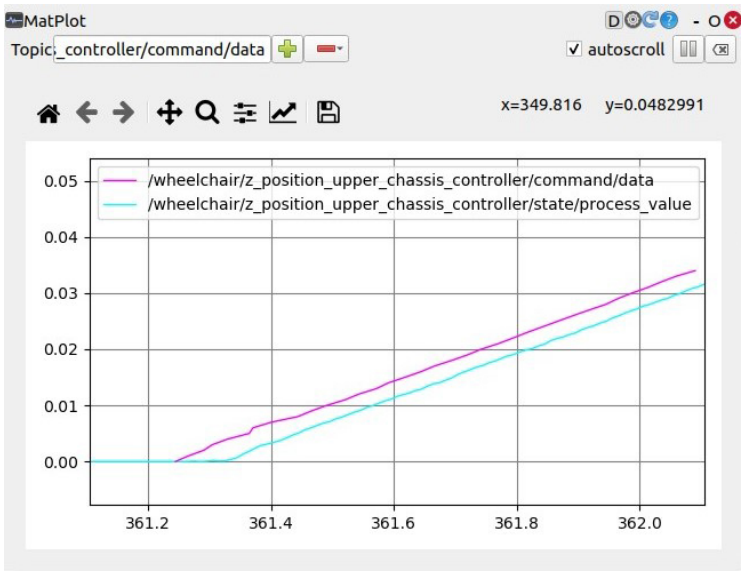


Figura 4.12: Janela do *Plugin* Matplot durante a subida do assento da cadeira

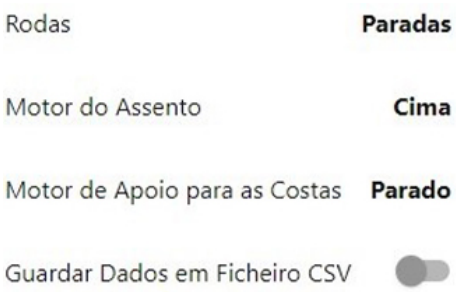


Figura 4.13: Interface de monitorização durante a execução do comando 5

4.2.2.7 Execução do Comando 6 - Descer o Assento

Após a execução deste comando, a cadeira regressou à posição inicial da figura 4.9, uma vez que o assento foi rebaixado, situação registada na interface de monitorização da figura 4.15. Utilizando novamente o MatPlot, é possível observar, pela figura 4.14, que os valores enviados para o tópico do controlador da *joint* do assento em comparação com os valores reais são muito idênticos, permitindo uma sincronização precisa com os movimentos da entidade física. O estado e posição do motor foram também registados no EV3, a partir das seguintes mensagens na consola, no final da execução do comando:

```
Estado Motor Assento: ['holding']
Posicao Motor Assento: 0
```

À semelhança do comando anterior, o estado é de *holding*, uma vez que o motor está a forçar a sua posição, que neste caso é a posição inicial de 0.

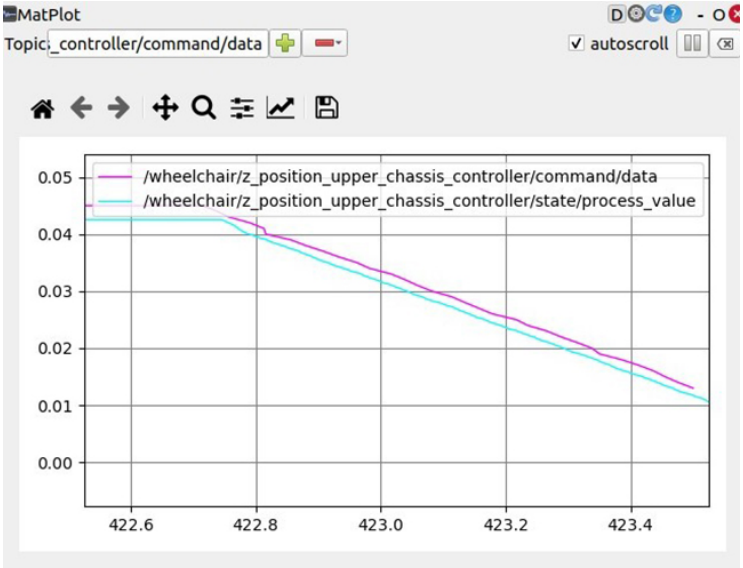


Figura 4.14: Janela do *Plugin* Matplot durante a descida do assento da cadeira

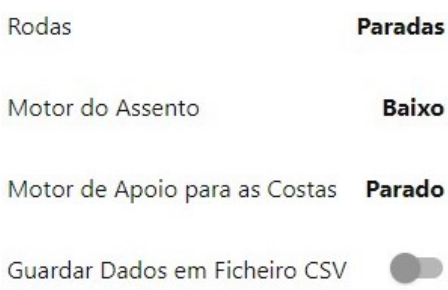


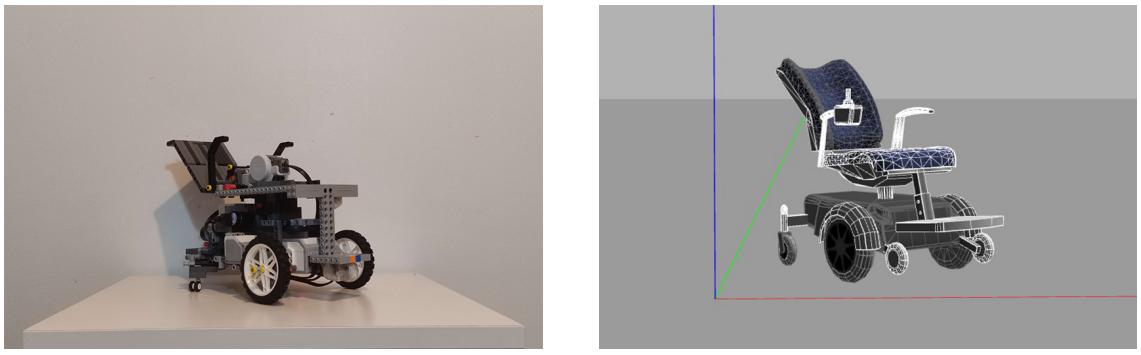
Figura 4.15: Interface de monitorização durante a execução do comando 6

4.2.2.8 Execução do Comando 7 - Inclinar Apoio de Costas para Trás

O último par de comandos executados controla o último motor da cadeira, responsável pela inclinação do apoio para as costas. Após a execução do comando para inclinar as costas para trás, chegou-se ao estado das entidades física e digital das figuras 4.16a e 4.16b, respetivamente, o que, comparando com o estado final do comando anterior, difere visivelmente na posição das costas. O espelhamento do comportamento da entidade física para a entidade virtual, registado na interface da figura 4.17, foi feito através do envio de valores para o tópico do controlador da *joint* das costas, tal como demonstra o gráfico no MatPlot da figura 4.18. Mais uma vez, os valores enviados são idênticos aos valores processados, o que indica que praticamente não existe atraso no movimento do motor da cadeira no simulador, face à entidade física. O estado e posição do motor das costas foi registado no EV3, como demonstra a saída da consola seguinte:

```
Estado Motor Costas: ['holding']
Posicao Motor Costas: -1
```

À semelhança dos motores anteriores, o seu estado encontra-se no modo *holding*, o qual força a a posição em que se encontra, que neste caso é de -1, correspondente à posição inicial (a posição inicial original é de 0).



(a) Estado da entidade física após execução do comando 7

(b) Estado da entidade virtual após execução do comando 7

Figura 4.16: Estado das entidades física e virtual após execução do comando 7

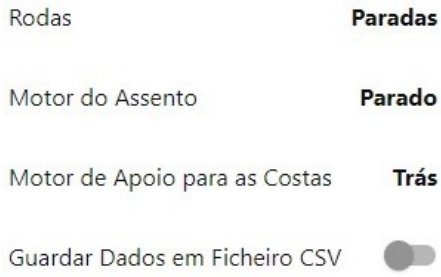


Figura 4.17: Interface de monitorização durante a execução do comando 7

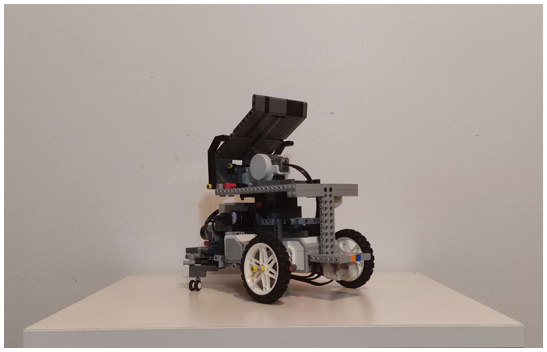
Figura 4.18: Janela do *Plugin* Matplot durante a inclinação para trás do apoio para as costas

4.2.2.9 Execução do Comando 8 - Inclinar Apoio de Costas para a Frente

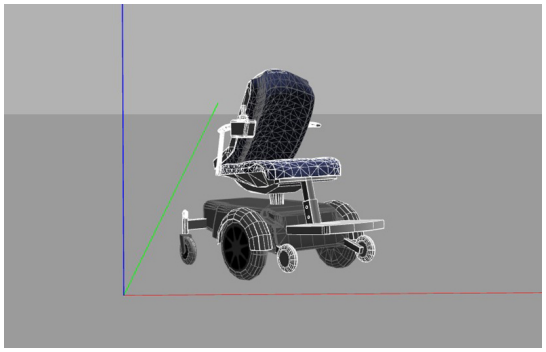
O último comando inclinou o apoio de costas da cadeira para a frente até ao limite definido, chegando ao estado das figuras 4.19a e 4.19b, correspondentes às entidades física e virtual, respetivamente. Durante a execução do comando, registou-se o estado do motor, representado na interface da figura 4.21, e os valores enviados para o tópico do controlador da sua *joint*, tal como demonstra o gráfico da figura 4.20. A partir deste gráfico, é possível observar que, apesar de haver uma certa ondulação, causada pelos valores dos ganhos PID definidos, os valores processados são idênticos aos que foram enviados para o controlador, não havendo, portanto, atraso na resposta ao movimento da entidade física. Na saída da consola do EV3 seguinte, constam o estado e posição do motor das costas no final da execução do comando:

```
Estado Motor Costas: ['holding']
Posicao Motor Costas: -95
```

O estado de *holding* significa o mesmo face ao comando anterior e a posição de -95 representa a posição limite definida de inclinação.



(a) Estado de entidade física após execução do comando 8



(b) Estado da entidade virtual após execução do comando 8

Figura 4.19: Estado das entidades física e virtual após execução do comando 8

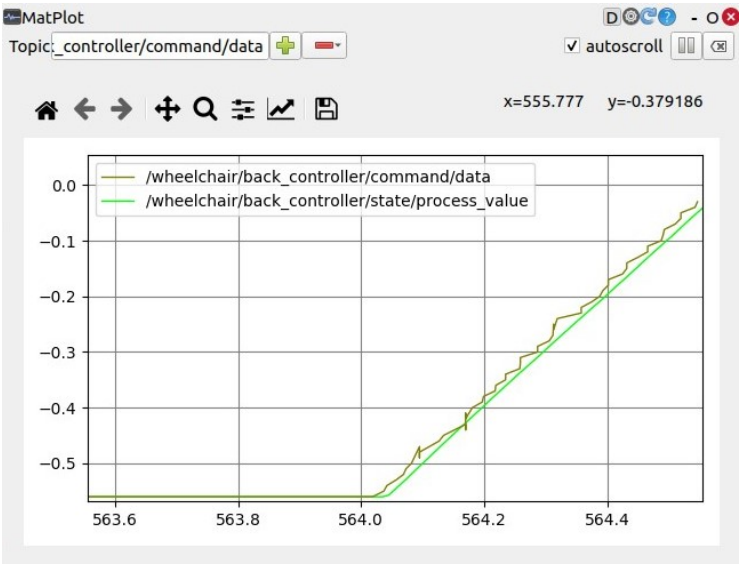


Figura 4.20: Janela do *Plugin* Matplot durante a inclinação para a frente do apoio para as costas

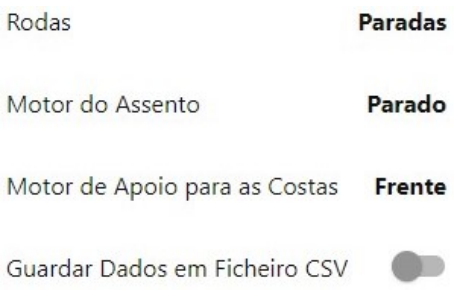


Figura 4.21: Interface de monitorização durante a execução do comando 8

4.2.2.10 Ficheiro CSV com Dados do Teste

Os dados referentes ao estado dos motores e rotação da cadeira, durante a execução dos comandos anteriores, foram registados num ficheiro CSV, através da interface de monitorização. A tabela 4.1 é uma amostra do produto desse ficheiro contendo apenas os momentos nos quais houve uma transição de estados de qualquer motor da cadeira, juntamente com a rotação sobre os eixos lateral, longitudinal e vertical. É possível concluir, através da análise da tabela, que a variação do estado de cada motor corresponde à sequência de comandos executada. Verifica-se também que após a rotação da cadeira sobre o eixo vertical, ou seja, virando para a esquerda e direita, o ângulo altera, enquanto nos restantes instantes temporais assume praticamente o mesmo valor. Após a movimentação para a esquerda, o ângulo muda de 170 para 179 e após a movimentação para a direita muda de 179 para 169, o que é significativo visto que a escala deste ângulo é representada em 8 bits, de 0-255. Todavia, estima-se que, dado a rotação considerável que a cadeira executou, a alteração do ângulo deveria ser maior, o que se pode justificar pela posição do módulo da bússola CMPS11 no protótipo da entidade física, condicionando a precisão do eixo de rotação vertical.

Tabela 4.1: Dados guardados no ficheiro CSV referentes aos comandos executados

Data	Assento	Costas	Rodas	Eixo Lat.	Eixo Long.	Eixo Ver.
2022-09-14T14:47:24.704Z	Parado	Parado	Paradas	0	0	170
2022-09-14T14:47:26.348Z	Parado	Parado	Frente	3	0	172
2022-09-14T14:47:27.891Z	Parado	Parado	Paradas	-1	0	167
2022-09-14T14:47:28.198Z	Parado	Parado	Trás	-3	0	167
2022-09-14T14:47:29.225Z	Parado	Parado	Paradas	2	0	170
2022-09-14T14:47:31.691Z	Parado	Parado	Esquerda	2	0	177
2022-09-14T14:47:31.999Z	Parado	Parado	Paradas	1	0	179
2022-09-14T14:47:33.957Z	Parado	Parado	Direita	0	2	171
2022-09-14T14:47:34.055Z	Parado	Parado	Paradas	0	1	169
2022-09-14T14:47:35.289Z	Cima	Parado	Paradas	1	0	170
2022-09-14T14:47:36.727Z	Parado	Parado	Paradas	0	0	170
2022-09-14T14:47:38.268Z	Baixo	Parado	Paradas	1	0	171
2022-09-14T14:47:39.604Z	Parado	Parado	Paradas	0	0	170
2022-09-14T14:47:39.707Z	Parado	Frente	Paradas	0	0	170
2022-09-14T14:47:41.659Z	Parado	Parado	Paradas	1	0	171
2022-09-14T14:47:42.276Z	Parado	Trás	Paradas	0	0	170
2022-09-14T14:47:43.919Z	Parado	Parado	Paradas	0	0	170

4.3 Cenários Particulares

Para alem da testagem do funcionamento do sistema, foram também realizados testes em casos particulares, nomeadamente a aplicação de uma força externa para movimentar a cadeira e a sua rotação sobre os eixos vertical, longitudinal e lateral.

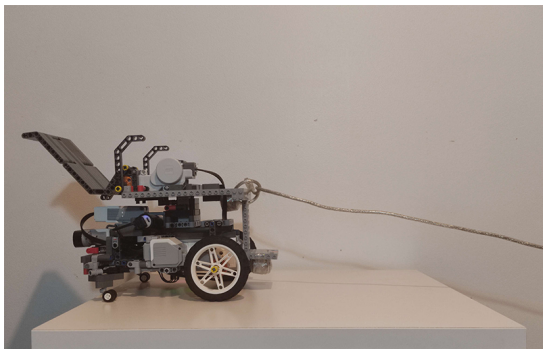
4.3.1 Aplicação de Força Externa para a Rotação das Rodas

Este teste foi realizado com o intuito de confirmar se a entidade virtual estaria efetivamente a replicar o comportamento da entidade física a partir dos valores lidos pelos sensores dos motores e não pelos dos valores que estavam programados. Para tal, criou-se um cenário em que a entidade física da cadeira é puxada com uma corda, a uma determinada força, de modo que esta se desloque para a frente sem a utilização do controlador remoto. O objetivo deste teste é observar no simulador virtual se de facto a cadeira assume o mesmo comportamento face ao mundo real e se os estados dos motores das rodas são corretamente registados na interface de monitorização.

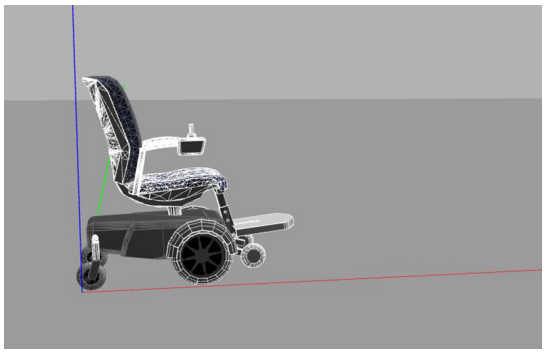
Nas figuras 4.22a e 4.22b, é possível observar as entidades física e virtual no seu estado inicial, sendo que na entidade física é visível a corda que irá puxar a cadeira para a frente. Após a aplicação da força externa, as entidades física e virtual chegam às posições finais das figuras 4.22c e 4.22d, respetivamente, nas quais podemos observar um avanço significativo, o que confirma que o modelo virtual efetivamente sincroniza o estado da sua cadeira consoante os valores lidos pelos sensores dos motores reais. Relativamente à interface de monitorização, é possível observar pela figura 4.23, os dados referentes ao estado dos motores e eixos de rotação durante a aplicação da força. O estado das rodas mostra o texto "Frente", o que condiz com a realidade, e o gráfico da rotação sobre o eixo vertical possui uma ligeira variação derivado de um pequeno desvio aquando da aplicação da força. Estes dados foram ainda registados num ficheiro CSV, cuja tabela se encontra em 4.2, desde o momento inicial ao momento de repouso da cadeira. A partir dos dados da tabela, confirma-se a variação do ângulo no eixo de rotação vertical e o estado das rodas. Por último, registaram-se também a posição e velocidade das rodas no EV3, a partir de uma porção das mensagens impressas na consola, ilustradas na listagem 4.5. O estado dos motores ao ser *ramping* significa que estes estão a rodar, mas não conseguem atingir uma velocidade constante. Conclui-se que, na fase inicial da aplicação da força externa, a posição e velocidade das rodas aumentam, seguido de uma diminuição da velocidade até a cadeira parar.

Listagem 4.5: Posição e velocidade dos motores durante aplicação de força externa

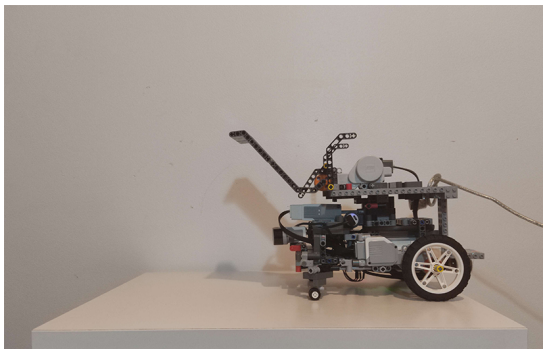
```
Estado dos Motores das Rodas: ['ramping']
Posicao Roda Esquerda: 150
Posicao Roda Direita: 150
Velocidade Roda Esquerda: 96
Velocidade Roda Direita: 96
Posicao Roda Esquerda: 250
Posicao Roda Direita: 350
Velocidade Roda Esquerda: 143
Velocidade Roda Direita: 143
Posicao Roda Esquerda: 287
Posicao Roda Direita: 386
Velocidade Roda Esquerda: 121
Velocidade Roda Direita: 121
```



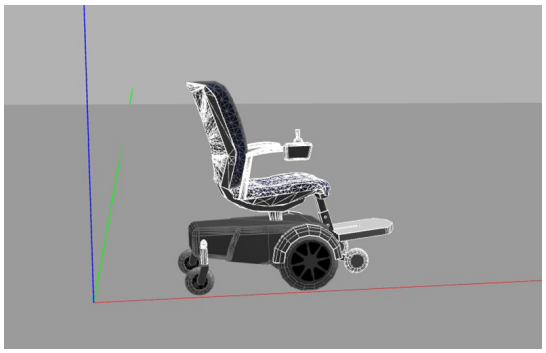
(a) Estado da entidade física antes da aplicação de uma força externa



(b) Estado da entidade virtual antes da aplicação de uma força externa



(c) Estado da entidade física depois da aplicação de uma força externa



(d) Estado da entidade virtual depois da aplicação de uma força externa

Figura 4.22: Estado das entidades virtual e física antes e depois da aplicação de uma força externa

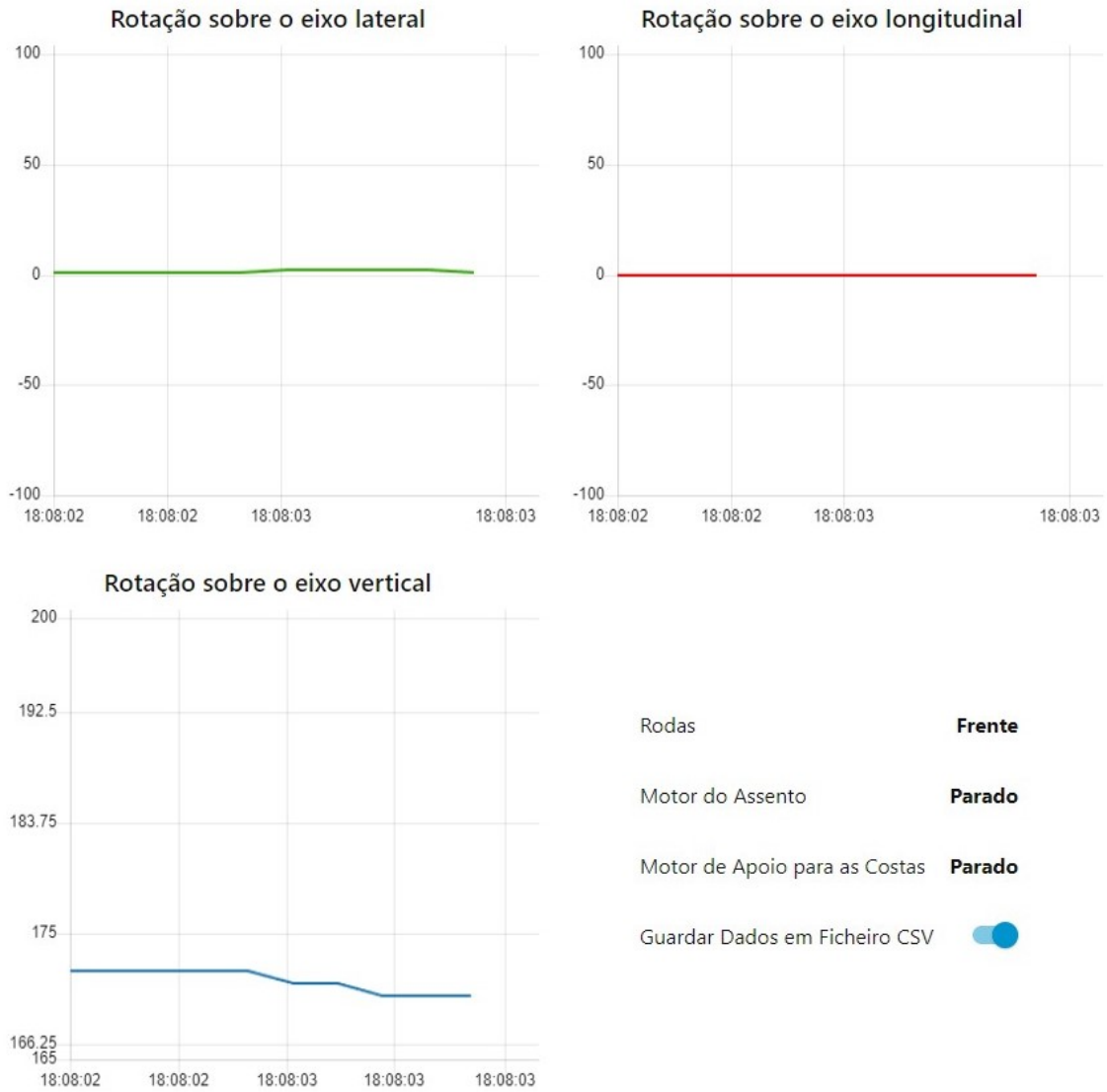


Figura 4.23: Interface de monitorização durante a aplicação de uma força externa

Tabela 4.2: Dados guardados no ficheiro CSV durante a aplicação de uma força externa

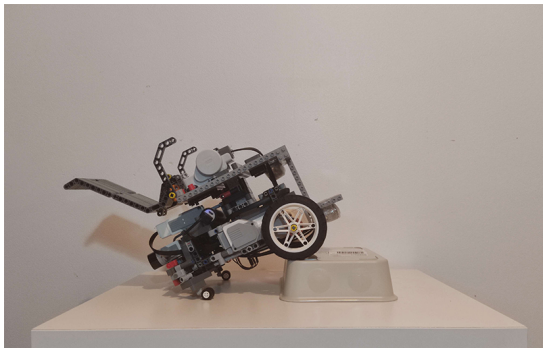
Data	Assento	Costas	Rodas	Eixo Lat.	Eixo Long.	Eixo Ver.
2022-09-15T17:08:03.062Z	Parado	Parado	Paradas	1	0	172
2022-09-15T17:08:03.166Z	Parado	Parado	Frente	2	0	171
2022-09-15T17:08:03.268Z	Parado	Parado	Frente	2	0	171
2022-09-15T17:08:03.885Z	Parado	Parado	Frente	2	0	169
2022-09-15T17:08:04.091Z	Parado	Parado	Frente	1	0	168
2022-09-15T17:08:04.296Z	Parado	Parado	Frente	0	0	167
2022-09-15T17:08:04.706Z	Parado	Parado	Frente	0	0	166
2022-09-15T17:08:04.912Z	Parado	Parado	Frente	0	0	166
2022-09-15T17:08:05.221Z	Parado	Parado	Paradas	0	0	167

4.3.2 Rotação da Cadeira Sobre os Eixos Longitudinal e Lateral

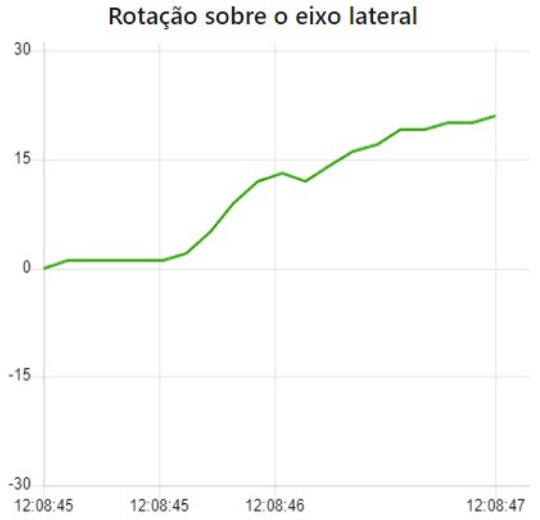
O último teste realizado no sistema consiste na rotação da entidade física sobre os eixos longitudinal e lateral, de modo a monitorizar as variações de inclinação da cadeira e alertar o utilizador para a ultrapassagem de limites predefinidos.

4.3.2.1 Rotação sobre o Eixo Lateral

Para este cenário utilizou-se um objeto, de modo que a cadeira se encontrasse numa posição inclinada para trás, tal como demonstra a figura 4.24a, e registaram-se os valores da variação de rotação sobre o eixo lateral durante a inclinação, como demonstra o gráfico da figura 4.24b. À semelhança dos outros eixos, o módulo CMPS11 gera o ângulo de rotação numa escala de 0 a 255, daí o intervalo das ordenadas do gráfico ser entre -30 e 30, o que facilita a sua observação. Os valores exatos da variação do ângulo de rotação sobre o eixo lateral foram registados num ficheiro CSV, tal como demonstra a tabela 4.3, a partir do qual podemos concluir que, de facto, o ângulo aumenta com a inclinação da cadeira no sentido positivo, não alterando os restantes eixos.



(a) Entidade física inclinada para trás sobre o eixo lateral



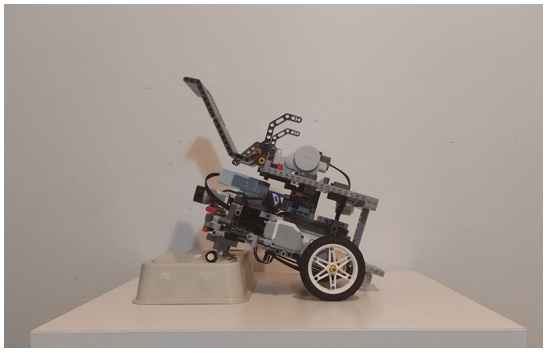
(b) Gráfico de monitorização da rotação da cadeira sobre eixo lateral

Figura 4.24: Entidade fisica e interface de monitorização durante a inclinação da cadeira para trás

Tabela 4.3: Dados guardados no ficheiro CSV durante a inclinação da cadeira para trás

Data	Eixo Lat.	Eixo Long.	Eixo Ver.
2022-09-16T11:08:45.172Z	0	0	193
2022-09-16T11:08:45.789Z	2	0	193
2022-09-16T11:08:45.892Z	5	0	193
2022-09-16T11:08:45.994Z	9	0	192
2022-09-16T11:08:46.097Z	12	0	192
2022-09-16T11:08:46.405Z	14	0	193
2022-09-16T11:08:46.508Z	16	0	193
2022-09-16T11:08:46.714Z	19	0	193
2022-09-16T11:08:47.125Z	21	0	193

Foi adotado o mesmo procedimento para inclinar a cadeira para a frente, como se pode observar pela figura 4.25a, registou-se a variação da inclinação durante o movimento no gráfico da figura 4.25b e foram guardados os valores do sensor num ficheiro CSV, o qual se encontra representado na tabela 4.4. A partir do gráfico e da tabela pode-se concluir que efetivamente a cadeira rodou no sentido negativo sobre o eixo lateral, com o ângulo de rotação a aumentar em módulo ao longo do tempo, até à posição final da figura 4.25a, sem qualquer alteração considerável nos outros eixos.



(a) Entidade física inclinada para a frente sobre o eixo lateral

(b) Gráfico de monitorização da rotação da cadeira sobre eixo lateral

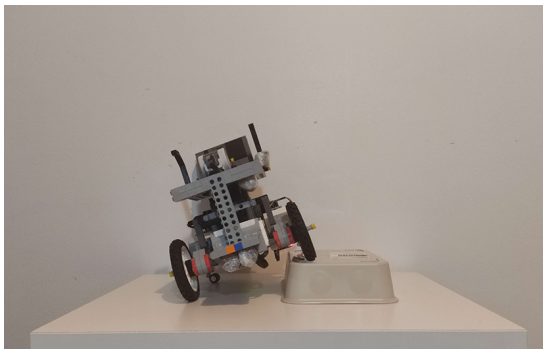
Figura 4.25: Entidade física e interface de monitorização durante a inclinação da cadeira para a frente

Tabela 4.4: Dados guardados no ficheiro CSV durante a inclinação da cadeira para a frente

Data	Eixo Lat.	Eixo Long.	Eixo Ver.
2022-09-16T11:00:57.895Z	0	0	193
2022-09-16T11:00:57.998Z	-1	0	193
2022-09-16T11:00:58.101Z	-3	0	193
2022-09-16T11:00:58.204Z	-7	0	193
2022-09-16T11:00:58.306Z	-10	0	193
2022-09-16T11:00:58.409Z	-13	0	193
2022-09-16T11:00:58.512Z	-16	0	193
2022-09-16T11:00:58.615Z	-20	0	194
2022-09-16T11:00:58.923Z	-23	0	193

4.3.2.2 Rotação sobre o Eixo Longitudinal

Os testes de rotação sobre o eixo longitudinal são idênticos aos anteriores, na medida em que é utilizado o mesmo objeto para inclinar a cadeira, tal como demonstra a figura 4.26a. À semelhança do eixo lateral foi também registada a variação do ângulo de rotação durante a inclinação, presente no gráfico da figura 4.26b e na tabela 4.5. Pode-se concluir a partir do gráfico que, ao inclinar para a sua direita, a cadeira roda no sentido negativo ao longo do eixo longitudinal, algo que é suportado pelos valores registados no ficheiro CSV da tabela, na coluna do respetivo eixo.



(a) Entidade física inclinada para a direita sobre o eixo longitudinal

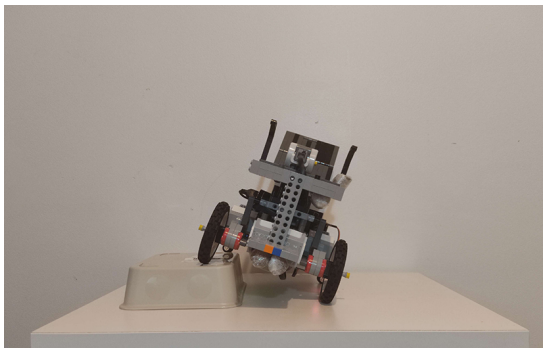
(b) Gráfico de monitorização da rotação da cadeira sobre eixo longitudinal

Figura 4.26: Entidade física e interface de monitorização durante a inclinação da cadeira para a direita

Tabela 4.5: Dados guardados no ficheiro CSV durante a inclinação da cadeira para a direita

Data	Eixo Lat.	Eixo Long.	Eixo Ver.
2022-09-16T10:50:42.859Z	0	0	196
2022-09-16T10:50:42.962Z	0	-3	196
2022-09-16T10:50:43.167Z	0	-6	197
2022-09-16T10:50:43.270Z	0	-10	198
2022-09-16T10:50:43.373Z	0	-13	199
2022-09-16T10:50:43.476Z	0	-16	200
2022-09-16T10:50:43.681Z	0	-17	201
2022-09-16T10:50:43.990Z	0	-20	203
2022-09-16T10:50:44.195Z	-1	-23	203

Por último, o teste de inclinação para a esquerda vai de encontro ao anterior, com a cadeira a assumir a posição final da figura 4.27a, após a rotação sobre o eixo longitudinal. A variação do ângulo é naturalmente simétrica ao cenário anterior, como se pode observar pelo gráfico da figura 4.27b e pela tabela 4.6, correspondente ao ficheiro CSV gerado durante a inclinação.



(a) Entidade física inclinada para a esquerda sobre o eixo longitudinal

(b) Gráfico de monitorização da rotação da cadeira sobre eixo longitudinal

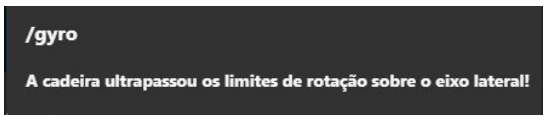
Figura 4.27: Entidade física e interface de monitorização durante a inclinação da cadeira para a esquerda

Tabela 4.6: Dados guardados no ficheiro CSV durante a inclinação da cadeira para a esquerda

Data	Eixo Lat.	Eixo Long.	Eixo Ver.
2022-09-16T10:35:17.761Z	0	2	194
2022-09-16T10:35:17.966Z	1	4	194
2022-09-16T10:35:18.172Z	1	6	193
2022-09-16T10:35:18.275Z	1	8	192
2022-09-16T10:35:18.377Z	1	11	192
2022-09-16T10:35:18.583Z	1	15	191
2022-09-16T10:35:18.789Z	1	18	190
2022-09-16T10:35:18.994Z	1	20	189
2022-09-16T10:35:19.509Z	1	24	188

4.3.2.3 Alertas de Ultrapassagem de Limites de Inclinação

Durante a execução dos testes de inclinação da cadeira foram registados os alertas programados para quando houvesse a ultrapassagem dos limites predefinidos para o ângulo de rotação sobre os eixos lateral e longitudinal. Os limites máximos definidos foram 20 em módulo, valor que é ultrapassado em todos os testes, como se pode observar pelos gráficos e tabelas de cada um. A nível de interface, as notificações assumem o formato da figura 4.28 e foram ambas recebidas nos respetivos testes.



(a) Notificação da passagem de limite de rotação sobre o eixo lateral

(b) Notificação da passagem de limite de rotação sobre o eixo longitudinal

Figura 4.28: Notificações da passagem dos limites de rotação sobre os eixos lateral e longitudinal

Capítulo 5

Conclusões

5.1 Balanço Geral

O objetivo desta dissertação foi desenvolver um sistema digital, idêntico ao de uma cadeira de rodas elétrica, que conseguisse sobretudo monitorizar os seus parâmetros, nomeadamente o estado dos seus componentes, e replicar o seu comportamento através de um simulador virtual. Apesar de não ter sido possível adquirir uma cadeira de rodas elétrica real, foi criado um protótipo físico idêntico a uma cadeira convencional, a partir do módulo programável EV3, o qual abriu portas à integração de sensores e motores na cadeira, o que viria a ser útil para a inclusão das rodas, do motor do assento e do motor para o apoio das costas. Apesar de não terem sido exploradas todas as classes de motores e sensores, a gama de robôs programáveis EV3 possui um vasto leque de produtos e documentação que podem ser utilizados em projetos de grande dimensão. A instalação de um novo sistema operativo baseado em Linux no EV3, o ev3dev, permitiu o desenvolvimento de uma solução mais robusta e a comunicação, através do protocolo MQTT, com os outros elementos do sistema. O controlo dos motores do protótipo foi feito a partir do controlador remoto da gama EV3, sendo que, a qualquer alteração da sua posição, era enviada uma mensagem para o respetivo tópico MQTT, que viria a ser subscrito pelo modelo virtual, o qual replicava esse movimento. Ainda no protótipo, foi integrado o módulo da bússola CMPS11, recorrendo a à placa NodeMCU, com o microcontrolador ESP8266, o qual viria a ser utilizado para monitorizar variações de inclinação da cadeira.

O modelo virtual foi desenvolvido no ambiente ROS, no qual foi necessário criar diversos ficheiros com diferentes finalidades, tais como a descrição física da cadeira, onde são especificados os *links* e *joints* que as constituem e as propriedades físicas de cada *link*, a declaração e inicialização dos controladores das diferentes *joints*, incluindo os seus ganhos PID, e a logística de sincronização com o modelo físico, programada em Python,

com recurso ao módulo *rospy*. Para questões de visualização e simulação, recorreu-se ao simulador Gazebo, onde é exibido o modelo 3D da cadeira de rodas a espelhar o comportamento da entidade física. Dada a vasta comunidade e documentação de suporte que o ROS tem, incluindo *plugins* e bibliotecas que permitem visualizar, monitorizar e até mesmo controlar diversos aspetos do modelo, este ambiente mostrou-se uma ótima solução para simular o comportamento da cadeira de rodas, podendo ser adaptado para qualquer objeto físico.

Em termos de monitorização e armazenamento de dados, foi criada uma interface, a partir da ferramenta de desenvolvimento Node-RED, a qual dispõe de gráficos com as alterações do ângulo de rotação sobre os eixos vertical, longitudinal e lateral da cadeira, o estado dos seus motores e ainda uma opção de armazenamento de todos estes dados num ficheiro CSV.

Quanto aos algoritmos desenvolvidos e testes realizados, pode-se concluir que, para o controlo da entidade física, correu tudo como esperado, depois de terem sido ajustados os motores para rodarem a uma velocidade constante e não ultrapassarem certos limites. Para o modelo virtual, foram igualmente ajustadas as velocidades e frequências de resposta de cada motor, face às mensagens que eram enviadas pelo modelo físico, o que resultou numa solução válida, dados os resultados dos testes de sincronização, uma vez que foi possível replicar, com alguma precisão, os movimentos da cadeira. Por último, a interface desenvolvida cumpriu as suas funções de monitorização, tal como foi analisado nos respetivos testes, e gerou diversos ficheiros CSV que podem ser posteriormente processados e utilizados por um profissional ou até mesmo pelo utilizador da cadeira.

5.2 Trabalho Futuro

Existem diversas vertentes que podem ser exploradas neste sistema ou até mesmo criar uma adaptação que contenha alguns dos seus componentes. Começando pela entidade física, o ideal seria obter uma cadeira de rodas elétrica real e integrar os restantes elementos do sistema, nomeadamente o modelo virtual, desenvolvido no ambiente ROS, a comunicação através do protocolo MQTT, e a interface de monitorização. Não abdicando da utilização do módulo EV3, é possível instalar diretamente o ROS no sistema operativo, o que seria benéfico por questões de velocidade de resposta e sincronização. A interface de monitorização pode ser melhorada, adicionando mais componentes, nomeadamente inteligência artificial com os dados adquiridos e o seu armazenamento numa base de dados. Relativamente ao modelo virtual, é possível fazer a modelação física, geométrica, comportamental e de regras com recurso a outros programas, tais como Unity, de modo a expandir algumas das suas características e funcionalidades.

Bibliografia

[1] C. Torkia et al. "Power wheelchair driving challenges in the community: A users' perspective". Em: *Disability and Rehabilitation: Assistive Technology* 10 (3 mai. de 2015), pp. 211–215. ISSN: 17483115. DOI: 10.3109/17483107.2014.898159 (ver p. 1).

[2] LUCI. *The problem Luci solves*. Acedido em 06/02/2022. Jul. de 2021. URL: https://luci.com/the-problem/ (ver p. 1).

[3] S. Fleming. *These 3 tech visionaries are reinventing the wheelchair*. Acedido em 06/02/2022. Mar. de 2021. URL: https://www.weforum.org/agenda/2021/03/technology-wheelchairs-reinvention-accessibility/ (ver p. 2).

[4] Q. Qi et al. "Enabling technologies and tools for digital twin". Em: *Journal of Manufacturing Systems* 58 (jan. de 2021), pp. 3–21. ISSN: 02786125. DOI: 10.1016/j.jmsy.2019.10.001 (ver pp. 2, 4, 6, 8).

[5] T. Ltd. *What is Digital Twin Technology and how does it work?* Acedido em 06/02/2022. URL: https://www.twi-global.com/technical-knowledge/faqs/what-is-digital-twin (ver p. 4).

[6] IBM. *What is a digital twin?* Acedido em 06/02/2022. URL: https://www.ibm.com/topics/what-is-a-digital-twin (ver p. 5).

[7] F. Tao et al. "Digital Twin in Industry: State-of-the-Art". Em: *IEEE Transactions on Industrial Informatics* 15 (4 abr. de 2019), pp. 2405–2415. ISSN: 15513203. DOI: 10.1109/TII.2018.2873186 (ver p. 6).

[8] F. Tao e M. Zhang. "Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing". Em: *IEEE Access* 5 (set. de 2017), pp. 20418–20427. ISSN: 21693536. DOI: 10.1109/ACCESS.2017.2756069 (ver p. 6).

[9] Permobil. *F5 Corpus VS Welcome to the Permobil family*. Acedido em 13/02/2022. 2021. URL: www.permobil.com (ver p. 7).

[10] Elprocus. *Communication Protocols : Basics and Types with Functionality*. Acedido em 13/02/2022. URL: https://www.elprocus.com/communication-protocols/ (ver p. 12).

[11] A. Pandit. *Serial Communication Protocols: Basics, Transmission Modes, Synchronous Asynchronous Serial Protocols*. Acedido em 17/02/2022. Abr. de 2019. URL: https://circuitdigest.com/tutorial/serial-communication-protocols (ver pp. 12, 13).

[12] Marconi. *redes - Qual a diferença entre comunicação assíncrona e síncrona? - Stack Overflow em Português*. Acedido em 17/02/2022. Fev. de 2015. URL: https://pt.stackoverflow.com/questions/51268/ (ver p. 12).

[13] R. T. Vaughan e B. P. Gerkey. *Really Reusable Robot Code and the Player/Stage Project*. 2006. URL: http://www.radish.sourceforge.net (ver p. 13).

[14] G. R. Bradski e A. Kaehler. *Learning OpenCV : computer vision with the OpenCV library*. O'Reilly, 2008, p. 555. ISBN: 9780596516130 (ver p. 13).

[15] S. Srinivasa et al. *The Robotic Busboy: Steps Towards Developing a Mobile Robotic Home Assistant*. URL: https://personalrobotics.cs.washington.edu/publications/srinivasa2008busboy.pdf (ver p. 13).

[16] R. Li et al. *ROS Based Multi-sensor Navigation of Intelligent Wheelchair*. URL: https://core.ac.uk/download/pdf/74369535.pdf (ver p. 13).

[17] *ROS: an open-source Robot Operating System*. Acedido em 13/02/2022. Computer Science Department, Stanford University, Stanford, CA. URL: http://stair.stanford.edu (ver p. 13).

[18] *Peer-to-peer – Wikipédia, a enciclopédia livre*. Acedido em 13/02/2022. Nov. de 2021. URL: https://pt.wikipedia.org/wiki/Peer-to-peer (ver p. 13).

[19] J. s.r.o. *CMakeLists.txt | CLion*. Acedido em 13/02/2022. Mai. de 2021. URL: https://www.jetbrains.com/help/clion/cmakelists-txt-file.html (ver p. 14).

[20] NadimArubai. *Documentation - ROS Wiki*. Acedido em 13/02/2022. URL: http://wiki.ros.org/ (ver p. 14).

[21] O. Robotics. *Understanding services — ROS 2 Documentation: Foxy documentation*. Acedido em 05/09/2022. URL: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html (ver p. 15).

[22] OSUrobotics. *GitHub - OSUrobotics/wheelchair-automation: Code for automating a PERMOBIL wheelchair for the ALS foundation.* Acedido em 14/02/2022. Jan. de 2018. URL: https://github.com/OSUrobotics/wheelchair-automation (ver p. 16).

[23] P. R. Raj Prakash Shinde Shubham Sonawane. *RajPShinde/Autonomous_Wheelchair: ROS Package for an Autonomous Wheelchair capable of navigating in an indoor environments.* Acedido em 14/02/2022. Nov. de 2021. URL: https://github.com/RajPShinde/Autonomous_Wheelchair (ver p. 16).

[24] redragonx. *GitHub - redragonx/can2RNET: This repo has code and documentation to control power-wheelchairs with R-Net electronics.* Acedido em 14/02/2022. Jan. de 2021. URL: https://github.com/redragonx/can2RNET (ver p. 16).

[25] ysshah. *GitHub - ysshah/SmartWheels: A target-following, obstacle avoiding smart wheelchair.* Acedido em 14/02/2022. Mai. de 2017. URL: https://github.com/ysshah/SmartWheels (ver p. 16).

[26] naivoder. *GitHub - naivoder/digital_twin: ROS digital twin simulation, inverse kinematic trajectory planning, collision avoidance and local network communication using UR10 manipulator.* Acedido em 14/02/2022. Mai. de 2020. URL: https://github.com/naivoder/digital_twin (ver p. 16).

[27] programmerall. *Use solidworks to export URDF model, convert to xacro model, and display and control in rviz and gazebo - Programmer All.* Acedido em 14/02/2022. URL: https://www.programmerall.com/article/9241833474/ (ver p. 16).

[28] R. L. Guimarães et al. "ROS navigation: Concepts and tutorial". Em: *Studies in Computational Intelligence* 625 (fev. de 2016), pp. 121–160. ISSN: 1860949X. DOI: 10.1007/978-3-319-26054-9_6 (ver p. 16).

[29] A. Barnard. *Robotics-simulation-digital-twin-Unity | Digital Twin | Siemens Global.* Acedido em 13/02/2022. Fev. de 2021. URL: https://new.siemens.com/global/en/company/stories/research-technologies/digitaltwin (ver p. 17).

[30] hyounesy. *GitHub - Unity-Technologies/Unity-Robotics-Hub: Central repository for tools, tutorials, resources, and documentation for robotics simulation in Unity.* Acedido em 13/02/2022. 2021. URL: https://github.com/Unity-Technologies/Unity-Robotics-Hub (ver p. 17).

[31] mrpropellers et al. *Unity-Robotics-Hub/README.* Acedido em 15/02/2022. Out. de 2021. URL: https://github.com/Unity-Technologies/Unity-Robotics-Hub/blob/main/tutorials/ros_unity_integration/README.md (ver pp. 17, 18).

[32] A. A. Lahcen e S. Belfkih. *NoSQL databases for big data.* 2017, pp. 171–185. URL: https://www.researchgate.net/profile/Ayoub-Ait-Lahcen/publication/318796844_NoSQL_databases_for_big_data/links/6024f1c8299bf1cc26b9df0b/NoSQL-databases-for-big-data.pdf (ver p. 19).

[33] R. Li. *Components, types, and subfield of AI, derived from [25,26]. | Download High-Quality Scientific Diagram.* Acedido em 05/09/2022. URL: researchgate.net/figure/Components-types-and-subfield-of-AI-derived-from-25-26_fig2_358915702 (ver p. 19).

[34] O. Spjuth, J. Frid e A. Hellander. *The machine learning life cycle and the cloud: implications for drug discovery.* 2021. DOI: 10.1080/17460441.2021.1932812 (ver p. 20).

[35] K. Alexopoulos, N. Nikolakis e G. Chryssolouris. "Digital twin-driven supervised machine learning for the development of artificial intelligence applications in manufacturing". Em: *International Journal of Computer Integrated Manufacturing* 33 (5 mai. de 2020), pp. 429–439. ISSN: 13623052. DOI: 10.1080/0951192X.2020.1747642 (ver p. 20).

[36] Epix. *Electric Wheelchair | 3D Warehouse*. Acedido em 18/08/2022. 2014. URL: https://3dwarehouse.sketchup.com/model/602b7217c42d3d365eead856a4605937/Electric-Wheelchair?login=true (ver pp. 24, 46).

[37] L. Group. *EV3 Technology*. Acedido em 19/08/2022. URL: https://www.lego.com/cdn/cs/set/assets/bltbef4d6ce0f40363c/LMSUser_Guide_LEGO_MINDSTORMS_EV3_11_Tablet_ENUS.pdf (ver pp. 24, 26).

[38] D. Etukudo. *Running Torque - Definition, vs Holding, Prevailing, Breakway Torque*. Acedido em 19/08/2022. URL: https://www.punchlistzero.com/running-torque/ (ver p. 26).

[39] robot-advance. *Robot Advance : New Supported WiFi Dongle for EV3*. Acedido em 21/08/2022. URL: https://www.robot-advance.com/EN/actualite-new-supported-wifi-dongle-for-ev3-67.htm (ver p. 30).

[40] T. L. Group. *MINDSTORMS EV3 – LEGO Education*. Acedido em 21/08/2022. URL: https://education.lego.com/en-us/downloads/mindstorms-ev3/software#downloads (ver p. 30).

[41] ev3dev. *ev3dev Home*. Acedido em 21/08/2022. URL: https://www.ev3dev.org/ (ver p. 31).

[42] ev3dev. *Getting Started with ev3dev*. Acedido em 24/08/2022. URL: https://www.ev3dev.org/docs/getting-started/ (ver p. 31).

[43] ykhokhlov. *Realtek RTL8188CUS based Wi-Fi won't work (Edimax EW-7811Un)*. Acedido em 24/08/2022. URL: https://github.com/ev3dev/ev3dev/issues/1442 (ver p. 33).

[44] A. Moriarty e D. Lechner. *moriarty/ros-ev3: How to install ROS (ros_comm) on an ev3 with ev3dev*. Acedido em 26/08/2022. URL: https://github.com/moriarty/ros-ev3 (ver p. 34).

[45] D. Demidov. *ev3dev/ev3dev-lang-python: Pure python bindings for ev3dev*. Acedido em 27/08/2022. URL: https://github.com/ev3dev/ev3dev-lang-python (ver p. 34).

[46] E. Python. *Using Motors - EV3dev Python*. Acedido em 27/08/2022. URL: https://sites.google.com/site/ev3devpython/learn_ev3_python/using-motors (ver pp. 36, 37).

[47] R. H. et al. *Motor classes — python-ev3dev 2.1.0.post6 documentation*. Acedido em 27/08/2022. URL: https://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/ev3dev-stretch/motors.html (ver p. 36).

[48] ev3dev. *Linux Kernel Drivers for ev3dev-stretch — ev3dev-stretch Linux kernel drivers 19 documentation*. Acedido em 27/08/2022. URL: https://docs.ev3dev.org/projects/lego-linux-drivers/en/ev3dev-stretch/index.html (ver p. 37).

[49] A. Alves. *amgalves96/ev3dev-wheelchair: ev3dev package for power wheelchair digital twin*. Acedido em 30/08/2022. URL: https://github.com/amgalves96/ev3dev-wheelchair (ver p. 38).

[50] P. S. Foundation. *paho-mqtt · PyPI*. Acedido em 29/08/2022. URL: https://pypi.org/project/paho-mqtt/ (ver p. 39).

[51] R. Eletronics. *CMPS11-Tilt Compensated Compass Module*. Acedido em 30/08/2022. URL: https://www.pishrobot.com/files/products/datasheets/cmps11.pdf (ver pp. 42, 44).

[52] A. Alves. *amgalves96/esp8266_wheelchair: CMPS11 and ESP8266 integrated system for power wheelchair digital twin*. Acedido em 31/08/2022. URL: https://github.com/amgalves96/cmps11%5C_esp8266-wheelchair (ver p. 44).

[53] Arduino. *Wire - Arduino Reference*. Acedido em 31/08/2022. URL: https://www.arduino.cc/reference/en/language/functions/communication/wire/ (ver p. 44).

[54] I. Grokhotkov. *ESP8266WiFi library — ESP8266 Arduino Core documentation*. Acedido em 31/08/2022. URL: https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html (ver p. 46).

[55] Blender. *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. Acedido em 01/09/2022. URL: https://www.blender.org/ (ver p. 47).

[56] A. Ademovic. *An Introduction to Robot Operating System | Toptal*. Acedido em 01/09/2022. URL: https://www.toptal.com/robotics/introduction-to-robot-operating-system (ver p. 47).

[57] NadimArubai. *ROS/Installation - ROS Wiki*. Acedido em 01/09/2022. URL: http://wiki.ros.org/ROS/Installation (ver p. 48).

[58] VMware. *VMware - Delivering a Digital Foundation For Businesses*. Acedido em 02/09/2022. URL: https://www.vmware.com/ (ver p. 48).

[59] FrancoisVanEeden. *catkin/conceptual_overview - ROS Wiki*. Acedido em 03/09/2022. URL: http://wiki.ros.org/catkin/conceptual_overview (ver p. 48).

[60] kitware. *CMake*. Acedido em 03/09/2022. URL: https://cmake.org/ (ver p. 48).

[61] A. Alves. *amgalves96/ros-wheelchair: ROS package for power wheelchair digital twin*. Acedido em 04/09/2022. URL: github.com/amgalves96/ros-wheelchair (ver p. 50).

[62] P. Cignoni et al. *MeshLab*. Acedido em 05/09/2022. URL: https://www.meshlab.net/ (ver p. 52).

[63] A. Alves. *amgalves96/node_red_wheelchair: Node-RED based system for power wheelchair Digital Twin*. URL: https://github.com/amgalves96/node_red_wheelchair (ver p. 61).