



**SOFTWARE EMBARCADO DE CONTROLE PARA
TRICICLO ASSISTIDO POR ESTIMULAÇÃO ELÉTRICA**

GEORGE ANDREW BRINDEIRO

**DISSERTAÇÃO DE MESTRADO EM ENGENHARIA DE SISTEMAS
ELETRÔNICOS E DE AUTOMAÇÃO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**FACULDADE DE TECNOLOGIA
UNIVERSIDADE DE BRASÍLIA**

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

**EMBEDDED CONTROL SOFTWARE FOR A
TRICYCLE ASSISTED BY ELECTRICAL STIMULATION**

**SOFTWARE EMBARCADO DE CONTROLE PARA
TRICICLO ASSISTIDO POR ESTIMULAÇÃO ELÉTRICA**

GEORGE ANDREW BRINDEIRO

ORIENTADOR: PROF. ANTÔNIO PADILHA LANARI BÓ

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA DE
SISTEMAS ELETRÔNICOS E DE AUTOMAÇÃO

PUBLICAÇÃO: PPGEA 661/2017 DM

BRASÍLIA/DF: MARÇO - 2017

**UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**SOFTWARE EMBARCADO DE CONTROLE PARA
TRICICLO ASSISTIDO POR ESTIMULAÇÃO ELÉTRICA**

GEORGE ANDREW BRINDEIRO

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

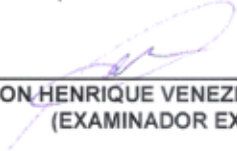
APROVADA POR:



ANTONIO PADILHA LANARI BÓ, Dr., ENE/UNB
(ORIENTADOR)



ADSON FERREIRA DA ROCHA, Dr., ENE/UNB/DF
(EXAMINADOR INTERNO)



WILSON HENRIQUE VENEZIANO, Dr., CIC/UNB
(EXAMINADOR EXTERNO)

BRASÍLIA, 20 DE MARÇO DE 2017.

FICHA CATALOGRÁFICA

BRINDEIRO, GEORGE ANDREW

Software Embarcado de Controle para Triciclo Assistido por Estimulação Elétrica [Distrito Federal] 2017. 95 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2017).

DISSERTAÇÃO DE MESTRADO – Universidade de Brasília, Faculdade de Tecnologia.

Departamento de Engenharia Elétrica

- | | |
|----------------------------|-----------------------------------|
| 1. Arquitetura de Software | 2. Equipamentos Médicos |
| 3. Software Embarcado | 4. Estimulação Elétrica Funcional |
| I. ENE/FT/UnB | II. Título (série) |

REFERÊNCIA BIBLIOGRÁFICA

BRINDEIRO, G. A. (2017). Software Embarcado de Controle para Triciclo Assistido por Estimulação Elétrica, DISSERTAÇÃO DE MESTRADO, Publicação PPGEA 661/2017 DM, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 95.

CESSÃO DE DIREITOS

AUTOR: George Andrew Brindeiro

TÍTULO: Software Embarcado de Controle para Triciclo Assistido por Estimulação Elétrica.

GRAU: Mestre ANO: 2017

É concedida à Universidade de Brasília permissão para reproduzir cópias desta dissertação de mestrado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte dessa dissertação de mestrado pode ser reproduzida sem autorização por escrito do autor.



George Andrew Brindeiro

Departamento de Eng. Elétrica (ENE) - FT

Universidade de Brasília (UnB)

Campus Darcy Ribeiro

CEP 70919-970 - Brasília - DF - Brasil

DEDICATÓRIA

A todos que me apoiaram e acreditaram em mim, mesmo quando eu já duvidava e queria desistir. Se não fossem vocês, eu não teria chegado tão longe.

AGRADECIMENTOS

Depois de um mestrado com a duração de um doutorado, que aconteceu em um período com diversos aprendizados e experiências acadêmicas e pessoais, é um prazer reconhecer as contribuições de todos que fizeram parte desse contexto. Não foi fácil superar todos desafios e certamente nem este trabalho, nem este autor, seriam os mesmos sem pessoas tão especiais.

Em primeiro lugar, agradeço aos membros do projeto EMA pelo acolhimento, amizade e trabalho em equipe: Estevão Lopes, Antônio Padilha, Lucas Fonseca, Ana de Sousa, Miguel Gutierrez, Emerson Martins e Juliana Guimarães. Além de recuperarem minha motivação para concluir o mestrado, conseguiram inculcar em mim a dimensão humana do nosso trabalho como engenheiros. Nossa campanha no Catarse, nossa viagem para Zurique e nossa participação no Cybathlon 2016 ficam marcadas como memórias especiais de nosso tempo juntos.

Além daqueles que contribuíram mais diretamente para este trabalho, agradeço aos amigos que vivenciaram juntos a pós-graduação no LARA neste período: André Geraldês, Claudia Ochoa, David Fiorillo, Hugo Tadashi, Luis Figueredo, Murilo Marinho, Roberto Baptista, Thiago Rocha. Obrigado pela inspiração através dos seus trabalhos, pelo companheirismo, pelas conversas, pelos desabafos, pelos jogos de Age of Empires e outros momentos de descontração que tornaram as dificuldades inerentes à vida acadêmica mais fáceis de lidar.

Ainda no LARA agradeço àqueles que confiaram na minha orientação e me permitiram desenvolver habilidades impossíveis de serem transmitidas em sala de aula: Artur Pio, Gabriel Araújo, Jacqueline Cristina e Mateus Mendelson. Tenho orgulho do seu crescimento, das suas conquistas e de ver que compartilham o conhecimento adquirido com as próximas gerações do LARA. É assim que nossa universidade e nosso grupo de pesquisa se fortalecem e se tornam cada vez mais relevantes para a ciência brasileira.

Um dos aspectos mais marcantes dos últimos cinco anos foram as experiências com o mundo do empreendedorismo e startups, junto aos amigos da Overdrive e do Brasília Fab Lab: André Geraldês, André Gama, André Leal, Apolinário Passos, Bruno Amui, Guilherme Garcia, Juliana Holanda, Luan Freitas, Marina Suassuna, Paula Macedo, Pedro Costa. Obrigado por embarcarem nessa aventura comigo, apesar de todos os perrengues e complicações. Apesar do impacto que isso teve no tempo de conclusão do meu mestrado, não tenho dúvida de que valeu a pena. Vivenciar tudo isso com vocês mudou a minha vida para sempre.

Na reta final, escrevendo a dissertação, contei com a compreensão e apoio de várias pessoas. Obrigado aos meus amigos, à minha família, aos meus colegas e alunos do Colégio Seriös. Minha ausência não foi em vão: finalmente acabou! Contem com minha presença em dobro daqui para frente, agora com algumas toneladas a menos nas costas.

Finalmente, agradeço àquela que faz tudo valer a pena, que me ensina a ser uma pessoa melhor todos os dias, que me aquece o coração. Aquela que me deu o maior presente que eu poderia pedir, ao voltar para Brasília para começarmos nossa vida juntos. Mariana, obrigado pelos 130 meses de felicidade e companheirismo, em todos aspectos das nossas vidas.

RESUMO

Título: Software Embarcado de Controle para Triciclo Assistido por Estimulação Elétrica

Autor: George Andrew Brindeiro

Orientador: Prof. Antônio Padilha Lanari Bó

Este trabalho visa estudar e aplicar diferentes técnicas e práticas no desenvolvimento de software embarcado de controle de um triciclo adaptado para a prática do ciclismo auxiliado por estimulação elétrica funcional. Entre normas técnicas relacionadas a dispositivos médicos e boas práticas de engenharia de software, buscou-se entender que estratégias poderiam ser adotadas para mitigar anomalias e avaliar a qualidade de software nesse contexto. Levando em consideração os riscos presentes no dispositivo, foi proposta uma nova arquitetura de software para mitigá-los, utilizando o framework *Robot Operating System* (ROS). A arquitetura proposta e sua implementação foram avaliadas com base em quatro atributos internos de qualidade de software: modificabilidade, reusabilidade, verificabilidade e proteção.

Palavras-chave:

Arquitetura de Software, Equipamentos Médicos, Software Embarcado, Estimulação Elétrica Funcional

ABSTRACT

Title: Embedded Control Software for a Tricycle Assisted by Electrical Stimulation

Author: George Andrew Brindeiro

Supervisor: Prof. Antônio Padilha Lanari Bó

This work aims to study and apply different development techniques and practices to the embedded control software to a tricycle adapted for functional electrical stimulation cycling. Between technical standards related to medical devices and software engineering best practices, it attempts to understand which strategies could be adopted to mitigate anomalies and evaluate the quality of software in this context. Considering the risks presented by the device, a new software architecture was proposed to mitigate them, using the *Robot Operating System* (ROS) framework. The proposed architecture and its implementation were evaluated based on four internal software quality attributes: modifiability, reusability, verifiability and safety.

Keywords:

Software Architecture, Medical Devices, Embedded Software, Functional Electrical Stimulation

SUMÁRIO

1	INTRODUÇÃO	1
1.1	CONTEXTUALIZAÇÃO	1
1.2	DEFINIÇÃO DO PROBLEMA	1
1.3	OBJETIVOS	2
1.3.1	OBJETIVO PRINCIPAL	2
1.3.2	OBJETIVOS SECUNDÁRIOS	2
1.4	ORGANIZAÇÃO DO MANUSCRITO	2
2	REVISÃO BIBLIOGRÁFICA	4
2.1	PROCESSOS REGULATÓRIOS DE DISPOSITIVOS MÉDICOS	4
2.1.1	HISTÓRICO GLOBAL	4
2.1.2	REGULAMENTAÇÃO NOS ESTADOS UNIDOS	4
2.1.3	REGULAMENTAÇÃO NA UNIÃO EUROPÉIA	7
2.1.4	REGULAMENTAÇÃO NO BRASIL	8
2.2	NORMAS TÉCNICAS SOBRE DISPOSITIVOS MÉDICOS	14
2.2.1	NORMALIZAÇÃO DE SOFTWARE DE DISPOSITIVOS MÉDICOS	15
2.2.2	IEC 62304: PROCESSOS DO CICLO DE VIDA DE SOFTWARE DE DISPOSITIVOS MÉDICOS	16
2.2.3	ISO 14971: APLICAÇÃO DE GERENCIAMENTO DE RISCO A DISPOSITIVOS MÉDICOS	21
2.2.4	ISO 13485: SISTEMAS DE GESTÃO DA QUALIDADE DE DISPOSITIVOS MÉDICOS	24
2.3	ENQUADRAMENTO DE SISTEMAS MECATRÔNICOS	25
3	EMA TRIKE	28
3.1	DESCRIÇÃO	28
3.2	MECÂNICA	29
3.3	YOST LABS 3-SPACE SENSOR	30
3.3.1	CARACTERÍSTICAS TÉCNICAS	32
3.3.2	COMPONENTES SENSORES	32
3.3.2.1	GIRÔMETRO	35
3.3.2.2	ACELERÔMETRO	35
3.3.2.3	MAGNETÔMETRO	35
3.3.2.4	CALIBRAÇÃO E CONFIGURAÇÃO	36
3.3.3	MODOS DE COMUNICAÇÃO	36
3.3.4	PROTOCOLO 3-SPACE SENSOR	37
3.4	ESTIMULADOR HASOMED REHAŠTIM	37
3.4.1	CARACTERÍSTICAS TÉCNICAS	40
3.4.2	FORMATOS DE ONDA	40
3.4.2.1	MODO DE PULSO ÚNICO	42
3.4.2.2	MODO CONTÍNUO DE LISTA DE CANAIS	42
3.4.2.3	MODO DE CICLO ÚNICO DE LISTA DE CANAIS	43
3.4.3	OPERAÇÃO DO REHAŠTIM	44
3.4.4	PROTOCOLO SCIENCEMODE	44
3.5	SISTEMA EMBARCADO	46

4	CRITÉRIOS PARA ESTRUTURAÇÃO DO SOFTWARE	51
4.1	CONCEITOS DE PROJETO DE SOFTWARE	51
4.2	DIAGNÓSTICO DA VERSÃO INICIAL DO SOFTWARE	52
4.2.1	MODIFIABILIDADE.....	52
4.2.2	REUSABILIDADE	55
4.2.3	VERIFIABILIDADE	57
4.2.4	PROTEÇÃO	57
4.3	ARQUITETURA PROPOSTA	58
5	IMPLEMENTAÇÃO DA ARQUITETURA PROPOSTA	60
5.1	ROBOT OPERATING SYSTEM.....	60
5.1.1	CONCEITOS BÁSICOS	61
5.1.2	FERRAMENTAS NATIVAS	63
5.1.3	ADEQUAÇÃO PARA A ARQUITETURA PROPOSTA	66
5.2	VISÃO GERAL DA IMPLEMENTAÇÃO	67
5.3	IMU_NODE.....	68
5.3.1	PROTOCOLO	68
5.3.2	CONFIGURAÇÕES.....	68
5.3.3	MÓDULO PYTHON	68
5.3.4	SCRIPT EXECUTÁVEL	68
5.4	STIMULADOR_NODE	70
5.4.1	PROTOCOLO	70
5.4.2	CONFIGURAÇÕES.....	70
5.4.3	MÓDULO PYTHON	71
5.4.4	SCRIPT EXECUTÁVEL	71
5.5	CONTROL_NODE.....	72
5.5.1	ALGORITMO	72
5.5.2	CONFIGURAÇÕES.....	73
5.5.3	MÓDULO PYTHON	73
5.5.4	SCRIPT EXECUTÁVEL	73
5.6	ASPECTOS ADICIONAIS DE INTEGRAÇÃO	74
6	AVALIAÇÃO DO RESULTADO OBTIDO	76
6.1	DIAGNÓSTICO DA VERSÃO ATUAL DO SOFTWARE	76
6.1.1	MODIFIABILIDADE.....	77
6.1.2	REUSABILIDADE	78
6.1.3	VERIFIABILIDADE.....	80
6.1.4	PROTEÇÃO	81
6.2	AVALIAÇÃO DE FUNCIONALIDADE	81
7	CONCLUSÃO	88
7.1	TRABALHOS FUTUROS	89
7.2	PUBLICAÇÕES.....	89
	BIBLIOGRAFIA	91

Lista de Figuras

2.1	Tratores metálicos de Perkins, um dos primeiros dispositivos médicos comprovadamente fraudulentos de que se tem registro, eram populares no fim do século XIX.....	5
2.2	Dispositivos médicos comercializados no início do século XX: endireitador de nariz e capacete térmico para combate à calvície.....	5
2.3	Caminhos de aprovação de dispositivos médicos nos EUA.	7
2.4	Caminhos de aprovação de dispositivos médicos da Classe I na UE.....	9
2.5	Caminhos de aprovação de dispositivos médicos da Classe IIa na UE.	9
2.6	Caminhos de aprovação de dispositivos médicos da Classe IIb na UE.	10
2.7	Caminhos de aprovação de dispositivos médicos da Classe III na UE.....	10
2.8	Caminhos de aprovação de dispositivos médicos no Brasil.	13
2.9	Grau de reconhecimento ⁶ de normas técnicas internacionais por membros do IMDRF.	15
2.10	Normas técnicas afetando a implementação de software para dispositivos médicos.....	17
2.11	Fluxo de processos e atividades de desenvolvimento de software na IEC 62304.	19
2.12	Fluxo de processos e atividades de manutenção de software na IEC 62304.	20
2.13	Processo de gerenciamento de risco segundo a norma ISO 14971.	22
2.14	Atividades de gerenciamento de risco segundo a norma ISO 14971.....	23
2.15	Robô de cirurgia Da Vinci, utilizado pela primeira vez no Brasil em 2008.	26
3.1	<i>EMA Trike</i> com o piloto Estevão Lopes no <i>Cyathlon 2016</i>	28
3.2	<i>EMA Trike</i> com principais elementos rotulados.....	29
3.3	Diagrama de blocos da <i>EMA Trike</i>	29
3.4	<i>Trike Tadpole HP3 20 X 26 CS (HP3 Trikes)</i>	30
3.5	Sistemas de referência de atitude e rumo da família <i>3-Space Sensor (Yost Labs): Embedded, USB/RS232, Bluetooth, Wireless 2.4 GHz DSSS, Data Logger e Wireless Dongle</i>	31
3.6	Orientação padrão do <i>3-Space Wireless Sensor</i> (reconfigurável em software).	32
3.7	Diagrama de blocos do <i>3-Space Wireless Sensor</i>	33
3.8	Conexões e interface do <i>3-Space Wireless Sensor e 3-Space Wireless Dongle</i>	33
3.9	Formato do pacote de dados binário no modo <i>wireless</i> no protocolo <i>3-Space Sensor</i>	38
3.10	Formato do pacote de resposta binária no modo <i>wireless</i> no protocolo <i>3-Space Sensor</i>	39
3.11	Estimulador <i>RehaStim</i> (Hasomed).....	39
3.12	Conexões e interfaces do Estimulador <i>RehaStim</i>	40
3.13	Formato do pulso bifásico gerado pelo estimulador.....	43
3.14	Exemplo do Modo Contínuo de Lista de Canais (<i>ccl</i>).	44
3.15	Formato do pacote de dados no protocolo <i>ScienceMode</i>	46
3.16	Inicialização de modo de lista de canais no protocolo <i>ScienceMode</i> (<i>cl_init, Ident = 00</i>).	46
3.17	Atualização de modo de lista de canais no protocolo <i>ScienceMode</i> (<i>cl_update, Ident = 01</i>).	47
3.18	Interrupção de modo de lista de canais no protocolo <i>ScienceMode</i> (<i>cl_stop, Ident = 10</i>).....	47
3.19	Geração de pulso único no protocolo <i>ScienceMode</i> (<i>single_pulse, Ident = 11</i>).	47
3.20	Pacote de reconhecimento no protocolo <i>ScienceMode</i> (<i>ack</i>).	47
3.21	Computador embarcado Raspberry Pi 3.	49

4.1	Mistura entre implementação de protocolo e variáveis de configuração, sem comentários para guiar o leitor que necessite realizar alterações em parâmetros do sistema.....	56
4.2	Arquitetura <i>dataflow</i> para a <i>EMA Trike</i>	59
5.1	Descrição dos elementos que compõem o Robot Operating System (ROS).....	61
5.2	Etapas de conexão a um tópico no ROS.....	62
5.3	Etapas de conexão a um serviço no ROS.	63
5.4	Visualização 3D de sistema robótico usando a ferramenta <i>rviz</i>	64
5.5	Gráficos de variáveis ao longo do tempo usando a ferramenta <i>rqt_plot</i>	65
5.6	Grafo de execução de sistema visualizado na ferramenta <i>rqt_graph</i>	65
5.7	Arquivo <i>bag</i> com escalares e imagens aberto na ferramenta <i>rqt_bag</i>	66
5.8	Estrutura do sistema da <i>EMA Trike</i> implementado no ROS.	67
5.9	Perfis de estimulação do algoritmo de controle da <i>EMA Trike</i>	72
6.1	Grafo de contribuições do repositório <i>ema_trike</i> , disponível no GitHub	77
6.2	Arquivos de configuração permitem que sejam feitas modificações de parâmetros do sistema sem a introdução acidental de erros no código-fonte	78
6.3	Implementação separada dos protocolos de comunicação em formato de biblioteca, utilizando parâmetros disponibilizados em arquivo de configuração externo ao código-fonte do <i>script</i> principal	79
6.4	Trecho do código do nó <i>stimulator_node</i> , ilustrando a estrutura linear e específica de cada <i>script</i> executável da versão atual do software.....	79
6.5	Visualização do estrutura do sistema da <i>EMA Trike</i> com a ferramenta <i>rqt_graph</i>	80
6.6	Visualização das variáveis controle da <i>EMA Trike</i> com a ferramenta <i>rqt_plot</i>	80
6.7	Visualização do registro de dados da <i>EMA Trike</i> com a ferramenta <i>rqt_bag</i>	81
6.8	Gráficos de controle/ângulo/velocidade para o sujeito 1 com versão inicial do software (Teste 1). 83	
6.9	Gráficos de controle/ângulo/velocidade para o sujeito 1 com versão atual do software (Teste 2)... 84	
6.10	Gráficos de controle/ângulo/velocidade para o sujeito 2 com versão inicial do software (Teste 3). 85	
6.11	Gráficos de controle/ângulo/velocidade para o sujeito 2 com versão atual do software (Teste 4)... 86	
6.12	Comparação da taxa de execução do controlador e aquisição de sensores entre versões do software.	87

Lista de Tabelas

2.1	Diferenças entre os processos de aprovação dispositivos médicos nos EUA e na UE.	11
2.2	Exigências da IEC 62304 para cada classe de software.	18
3.1	Especificações de elementos mecânicos da <i>EMA Trike</i>	31
3.2	Especificações gerais do <i>3-Space Wireless Sensor</i>	34
3.3	Especificações dos sensores do <i>3-Space Wireless Sensor</i>	34
3.4	Especificações gerais do <i>3-Space Wireless Dongle</i>	34
3.5	Situações sinalizadas pelo LED indicador do <i>3-Space Wireless Sensor</i>	35
3.6	Situações sinalizadas pelo LED indicador do <i>3-Space Wireless Dongle</i>	35
3.7	Exemplos de comandos binários no modo <i>wireless</i> no protocolo <i>3-Space Sensor</i>	38
3.8	Especificações do Estimulador <i>RehaStim</i>	41
3.9	Situações de falha sinalizadas pelo LED indicador do <i>RehaStim</i>	42
3.10	Situações de falha sinalizadas pela tela LCD do <i>RehaStim</i>	42
3.11	Como ativar o <i>ScienceMode</i> do estimulador <i>RehaStim</i>	45
3.12	Configurações da interface de comunicação no <i>ScienceMode</i>	45
3.13	Identificadores dos comandos do protocolo <i>ScienceMode</i>	46
3.14	Inicialização de modo de lista de canais no protocolo <i>ScienceMode</i> (<i>cl_init</i> , <i>Ident</i> = 00).	48
3.15	Atualização de modo de lista de canais no protocolo <i>ScienceMode</i> (<i>cl_update</i> , <i>Ident</i> = 01).	48
3.16	Interrupção de modo de lista de canais no protocolo <i>ScienceMode</i> (<i>cl_stop</i> , <i>Ident</i> = 10).	48
3.17	Geração de pulso único no protocolo <i>ScienceMode</i> (<i>single_pulse</i> , <i>Ident</i> = 11).	49
3.18	Pacote de reconhecimento no protocolo <i>ScienceMode</i> (<i>ack</i>).	49
4.1	Atributos comuns de qualidade de software.	53
4.2	<i>Scripts</i> presentes na base de código da <i>EMA Trike</i> em maio de 2015.	54
5.1	Configurações do <i>imu_node</i> disponíveis no arquivo <i>imu.yaml</i>	69
5.2	Tópicos publicados pelo <i>imu_node</i> no ROS.	70
5.3	Configurações do <i>stimulator_node</i> disponíveis no arquivo <i>stimulator.yaml</i>	71
5.4	Tópicos assinados pelo <i>stimulator_node</i> no ROS.	71
5.5	Configurações do <i>control_node</i> disponíveis no arquivo <i>control.yaml</i>	73
5.6	Tópicos assinados pelo <i>control_node</i> no ROS.	74
5.7	Tópicos publicados pelo <i>control_node</i> no ROS.	74
6.1	<i>Scripts</i> presentes na base de código-fonte da <i>EMA Trike</i> em março de 2017.	76

Lista de Símbolos

Siglas e acrônimos

ABNT	Associação Brasileira de Normas Técnicas
Anvisa	Agência Nacional de Vigilância Sanitária
BPFC	Boas Práticas de Fabricação e Controle
CE	<i>Conformité Européene</i>
Conmetro	Conselho Nacional de Metrologia, Normalização e Qualidade Industrial
EMA	<i>Empoderando Mobilidade e Autonomia</i>
EUA	Estados Unidos da América
FDA	<i>Food and Drug Administration</i>
FMEA	<i>Failure Mode and Effects Analysis</i>
FMECA	<i>Failure Mode, Effects and Criticality Analysis</i>
FTA	<i>Fault Tree Analysis</i>
GHTF	<i>Global Harmonization Task Force</i>
GQUIP	Gerência de Tecnologia em Equipamentos Médicos
HACCP	<i>Hazard Analysis and Critical Control Point</i>
HAZOP	<i>Hazard and Operability Study</i>
IEC	<i>International Electrotechnical Commission</i>
IMDRF	<i>International Medical Device Regulators Forum</i>
IN	Instrução Normativa
Inmetro	Instituto Nacional de Metrologia, Normalização e Qualidade Industrial
ISO	<i>International Standards Organization</i>
LE	Laboratório de Ensaio
MDD	<i>Medical Devices Directive</i>
OCP	Organismo Certificador de Produtos
PHA	<i>Preliminary Hazard Analysis</i>
RDC	Resolução da Diretoria Colegiada
ROS	<i>Robot Operating System</i>
SBAC	Sistema Brasileiro de Avaliação da Conformidade
Sinmetro	Sistema Nacional de Metrologia, Normalização e Qualidade Industrial
SOUP	<i>Software of Unknown Provenance</i>
SVS	Secretaria de Vigilância Sanitária
UE	União Européia
UML	<i>Unified Modeling Language</i>

1

INTRODUÇÃO

"The absence of solid architecture and principled engineering practices in software development affects a wide range of medical devices, with potentially life-threatening consequences"

FY2011 OSEL Annual Report (FDA)

1.1 CONTEXTUALIZAÇÃO

Um sistema crítico de segurança é aquele em que uma falha pode resultar em perda de vida, prejuízo material significativo ou danos sérios ao ambiente [1]. Esse é o caso de vários dispositivos médicos, que lidam diretamente com processos vitais do corpo humano. Com a evolução da tecnologia biomédica, torna-se cada vez mais predominante a presença de software neste campo de aplicação. Isso inclui sistemas mecatrônicos, como robôs cirúrgicos, exoesqueletos, próteses robóticas e cadeiras de rodas motorizadas.

Essa dependência crescente de dispositivos médicos em software, aliada ao fato que alterações aparentemente pequenas em um software podem ter implicações importantes no funcionamento de um dispositivo e sua performance clínica, faz com que se torne cada vez mais relevante avaliar a adequação, qualidade e segurança de software de sistemas mecatrônicos aplicados à saúde.

Existem diversas normas técnicas regulamentando aspectos biomédicos e eletromecânicos do funcionamento de dispositivos médicos, porém ainda há pouca supervisão e avaliação do software empregado nestes dispositivos. Isso faz com que falhas de software sejam a maior causa de *recall* de dispositivos médicos entre 2008 e 2012 nos Estados Unidos [2, 3]. A tendência é que esse cenário se agrave, à medida que o ambiente de uso destes dispositivos se torna mais complexo em termos de conectividade e interoperabilidade.

1.2 DEFINIÇÃO DO PROBLEMA

Considerando a importância do software no funcionamento adequado de sistemas mecatrônicos aplicados à saúde, surgem as questões centrais deste trabalho:

- Como estruturar o desenvolvimento de software para minimizar o surgimento de falhas e anomalias?
- Que arquiteturas de software podem auxiliar na manutenção de software seguro?
- Como podemos avaliar a qualidade de software utilizado em dispositivos médicos?

Estas questões são abordadas no contexto do projeto *Empoderando Mobilidade e Autonomia* (EMA), que visa desenvolver tecnologias assistivas voltadas para pessoas com dificuldade de locomoção. Em particular utiliza-

se a *EMA Trike*, um triciclo assistido por estimulação elétrica, para aplicação deste conhecimento na etapa de pesquisa e desenvolvimento em um sistema mecatrônico aplicado à saúde.

1.3 OBJETIVOS

1.3.1 Objetivo principal

Estudar diferentes técnicas e práticas de desenvolvimento de software que possibilitem o desenvolvimento seguro, eficaz e eficiente com exemplo de aplicação no software embarcado de controle para um triciclo assistido por estimulação elétrica.

1.3.2 Objetivos secundários

- Apresentar aspectos do processo de desenvolvimento de sistemas mecatrônicos aplicados à saúde que possibilitem a reutilização de software;
- Catalogar iniciativas existentes para avaliação e certificação de software, especificamente relacionadas a dispositivos médicos e sistemas mecatrônicos aplicados à saúde;
- Desenvolver o código-fonte existente da *EMA Trike*, visando melhorar sua modifiabilidade, reusabilidade, verificabilidade e proteção;
- Documentar aspectos do desenvolvimento do software embarcado de controle da *EMA Trike* que possam ser reproduzidos em outros projetos relacionados.

1.4 ORGANIZAÇÃO DO MANUSCRITO

O Capítulo 2 faz uma revisão bibliográfica de padrões de desenvolvimento de software e sistemas, processos regulatórios e normas técnicas relacionadas a dispositivos médicos, estratégias de mitigação de anomalias e metodologias de avaliação de qualidade de software aplicado à saúde.

O Capítulo 3 apresenta detalhes do projeto da *EMA Trike*, um triciclo que foi adaptado para a prática do ciclismo auxiliado por estimulação elétrica funcional. São descritas características técnicas e aspectos do funcionamento dos componentes mecânicos e eletrônicos que foram integrados ao triciclo para este fim.

O Capítulo 4 inicia um estudo de caso envolvendo a versão inicial do software deste dispositivo, com o objetivo de melhorar sua modifiabilidade, reusabilidade, verificabilidade e proteção. Os riscos presentes no projeto são analisados com o objetivos de definir critérios para a reestruturação do seu software, com uma nova arquitetura proposta para o sistema.

O Capítulo 5 descreve a implementação da arquitetura apresentada no capítulo 4. Apresenta-se o framework *Robot Operating System* (ROS) como alternativa viável para a estruturação do software embarcado de controle da *EMA Trike*, detalhando suas funcionalidades e os benefícios da sua utilização.

O Capítulo 6 apresenta uma análise comparativa do software da *EMA Trike* antes e depois do uso do ROS. Primeiro discute-se o estado dos atributos de qualidade desejados na versão atual do software: modificabilidade, reusabilidade, verificabilidade e proteção. Em seguida, compara-se experimentalmente a funcionalidade da versão inicial e atual do software.

O Capítulo 7 apresenta as conclusões do trabalho e propõe melhorias para trabalhos futuros.

2

REVISÃO BIBLIOGRÁFICA

2.1 PROCESSOS REGULATÓRIOS DE DISPOSITIVOS MÉDICOS

2.1.1 Histórico Global

Durante muito tempo, não havia qualquer controle governamental sobre a venda e uso de dispositivos médicos. Até meados do século XX, grande parte dos dispositivos utilizados pelos médicos da época, como estetoscópios e bisturis, eram relativamente simples, fazendo com que quaisquer perigos ou defeitos fossem imediatamente verificáveis [4]. Porém, ao longo do tempo surgiram algumas exceções notáveis - equipamentos tão milagrosos quanto fraudulentos.

A descoberta, em 1961, que o sedativo talidomida teria causado defeitos congênitos e a morte de milhares de bebês, após suas mães utilizarem essa droga para combater os efeitos do enjôo matinal, comoveu e chocou o mundo. Com isso, formou-se um consenso internacional: medicamentos, tecnologias médicas e outros produtos para saúde deveriam estar sujeitos a uma regulamentação mais rígida que outros bens de consumo [5].

Os Estados Unidos da América (EUA) assumiram a dianteira desse movimento, mas logo foram acompanhados pelo resto do mundo. Diversos países estabeleceram regimes de regulamentação próprios a partir dos anos 1970. O crescimento no número de regras, testes e requisitos necessários para comercialização de dispositivos médicos em diferentes países levou a União Européia (UE) a empregar esforços para uma maior harmonização dos processos regulatórios [5], reduzindo as barreiras para o comércio entre seus países-membros.

A importância de ambos nesse cenário se observa nos números do mercado global de dispositivos médicos. Em 2007, os EUA concentravam cerca de 50% da produção e consumo destes produtos, enquanto a UE era responsável por 30%. O Japão vinha em terceiro lugar, com 10% do mercado [6]. Juntos, representantes da indústria e agências reguladoras desses gigantes do mercado lideraram os esforços mundiais de harmonização através da criação da *Global Harmonization Task Force* (GHTF) em 1992, que contava ainda com a participação do Canadá e Austrália.

Em 2012, a GHTF deu lugar ao *International Medical Device Regulators Forum* (IMDRF), uma organização composta apenas por representantes de agências reguladoras. Com a participação dos membros fundadores da GHTF e inclusão de Brasil, China, Rússia e Cingapura ao grupo, o IMDRF emprega esforços para acelerar a harmonização e convergência da regulamentação internacional de dispositivos médicos [7]. As principais iniciativas se concentram em esquemas de classificação e registro internacional, padronização de normas técnicas reconhecidas para fins regulatórios, programas de auditoria única para dispositivos médicos e intercâmbio de relatórios entre as autoridades nacionais competentes.

2.1.2 Regulamentação nos Estados Unidos

Um dos dispositivos médicos fraudulentos mais antigos que se tem registro nos EUA foi o trator metálico [4], patenteado em 1796 [8] e retratado na Figura 2.1. Tratava-se de um conjunto com um bastão de ferro e outro de latão, que aliviaria dores e inflamações. Apesar da fundamentação do seu funcionamento no “princípio

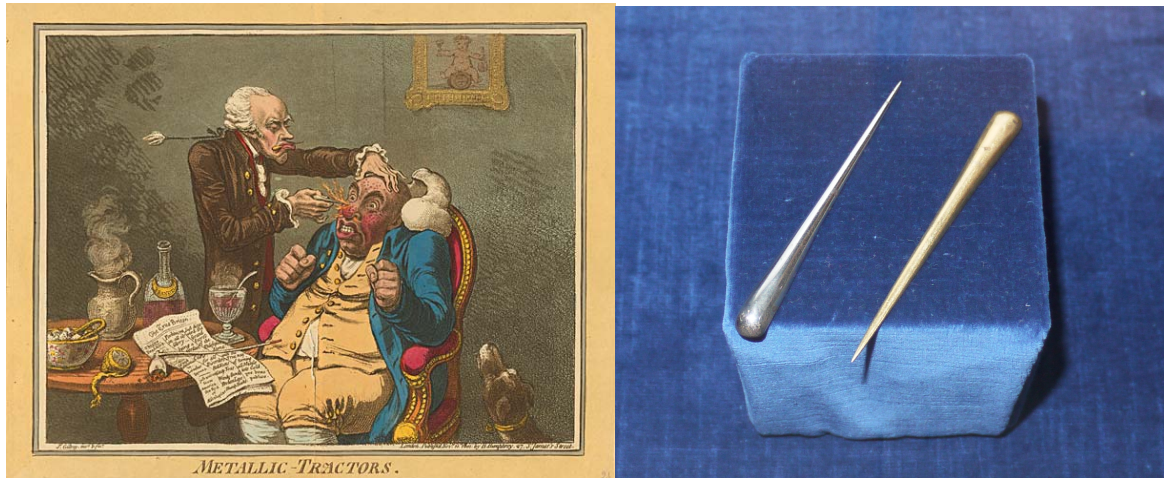


Figura 2.1: Tratores metálicos de Perkins, um dos primeiros dispositivos médicos comprovadamente fraudulentos de que se tem registro, eram populares no fim do século XIX.

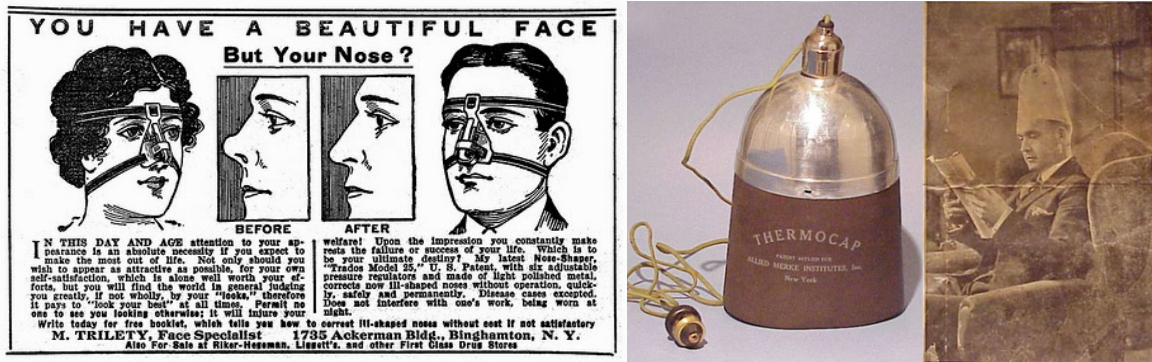


Figura 2.2: Dispositivos médicos comercializados no início do século XX: endireitador de nariz e capacete térmico para combate à calvície.

Galvânico” parecer bastante duvidosa para o olhar da ciência nos dias de hoje, vale ressaltar que pouco se sabia sobre os mistérios do eletromagnetismo à época.

Os tratores metálico de Perkins se tornaram tão populares que até mesmo o presidente dos EUA, George Washington, teria adquirido um conjunto para sua família. John Haygarth provou em 1800 que tais efeitos terapêuticos eram inexistentes, substituindo os bastões metálicos por similares feitos de madeira, em um dos primeiros estudos a utilizarem um placebo em um teste clínico simples cego [9].

Em 1906, por meio do *Pure Food and Drugs Act*, foi criada a *Food and Drug Administration* (FDA), agência do governo dos Estados Unidos hoje responsável pelo controle dos alimentos, suplementos alimentares, medicamentos, cosméticos, equipamentos médicos, materiais biológicos e produtos derivados do sangue humano. Nessa época, porém, não havia nenhuma regulamentação específica para dispositivos médicos [10].

Ao longo do tempo, outros equipamentos médicos de eficácia não comprovada surgiram no mercado, tais como endireitadores de nariz e capacetes térmicos para combate à calvície, ilustrados na Figura 2.2. Em 1917, a FDA se manifestou sobre o crescimento descontrolado do número de fraudes em seu relatório anual ao Congresso dos EUA, afirmando que a legislação vigente apresentava “limitações severas ... que dificultam o controle ... de dispositivos mecânicos fraudulentos utilizados para fins terapêuticos” [11, p. 19].

O aumento de dispositivos e medicamentos que diziam utilizar radiação para tratar diversas doenças levou a FDA a se manifestar novamente em 1926 no seu relatório anual ao Congresso dos EUA. A agência chamou atenção para os riscos à saúde apresentados pela radioatividade, documentados na pesquisa dos químicos franceses Pierre e Marie Curie desde 1898, e a necessidade de monitoramento destes produtos [4].

Grande parte da atuação da FDA no início do século XX se concentrava em retirar do mercado produtos que apresentavam riscos de segurança aos pacientes ou cuja eficácia não era devidamente comprovada. Embora a agência monitorasse esses produtos, ela não tinha autoridade para agir por contra própria no seu bloqueio.

Apenas em 1938, no *Food, Drug and Cosmetic Act*, incluiu-se uma definição clara para dispositivos médicos na legislação estadunidense, tornando esses produtos sujeitos à fiscalização da FDA. A agência avaliaria aspectos de segurança e eficácia, não precisando mais provar que os benefícios terapêuticos de um produto eram intencionalmente deturpados ou falsos para interromper sua comercialização. Contudo, ainda não era obrigatório que dispositivos fossem testados, inspecionados e aprovados antes de serem comercializados [10].

Em 1962, foram propostas mudanças para que dispositivos médicos fossem controlados de forma comparável, porém separada, de novos medicamentos. Essas mudanças não foram implementadas até o início dos anos 1970, quando falhas em milhares de marca-passos e dispositivos intra-uterinos foram reportados. Nessa época, a FDA havia começado a listar todos dispositivos médicos disponíveis no mercado e classificá-los de acordo com seus potenciais riscos, independentemente de qualquer legislação [4].

Em 1976 o *Medical Device Regulation Act* distinguiu dispositivos médicos de medicamentos apontando o fato que medicamentos causam reações químicas no corpo humano, enquanto dispositivos médicos agem de formas distintas [10]. Ele instituiu oficialmente o sistema de classificação de dispositivos de acordo com o risco e estabeleceu o grau de controle exigido para cada classe. A nova lei também autorizou a FDA a gerenciar os processos de notificação, reparo, substituição e restituição de dispositivos defeituosos, além de possibilitar o banimento de qualquer dispositivo que apresentasse informações falsas sobre seu funcionamento ou risco excessivo à saúde de seus usuários.

O regime de regulamentação americano continua se aprimorando ao longo dos anos para se adequar aos desafios impostos pelo crescente número de dispositivos que buscam entrar no mercado e pelas novas tecnologias que vêm sendo incorporadas nesses produtos, como testes genéticos caseiros [4], dispositivos conectados [12], equipamentos médicos para uso domiciliar [13] e softwares que atuam como dispositivos médicos [14].

Nos EUA, são definidas três classes regulatórias pela legislação. Todos dispositivos estão sujeitos a controles gerais incluindo rotulagem adequada e adesão a boas práticas de fabricação definidas pela FDA. A classe I inclui dispositivos de baixo risco, cuja segurança e eficácia são razoavelmente garantidos pelos controles gerais, como estetoscópios e abaixadores de língua. A classe II inclui dispositivos de risco moderado, que precisam atingir padrões de qualidade especificados em controles especiais, como tomógrafos computadorizados e endoscópios gastrointestinais. Por fim, a classe III contém dispositivos considerados de alto risco, como marca-passos e próteses de silicone implantadas, cuja segurança só pode ser assegurada com um processo detalhado de avaliação e aprovação pré-comercialização.

Antes de conseguir a aprovação da FDA para comercialização nos EUA, fabricantes precisam demonstrar que o dispositivo é seguro (seus benefícios superam os riscos) e eficaz (faz o que se espera de forma confiável). Para tanto, podem incluir estudos de verificação e validação do projeto do dispositivo, estudos observacionais, testes clínicos randomizados, estudos epidemiológicos, estudos animais, pesquisa de bancada, testes de engenharia ou fabricação e análise estatística de risco.

A maior parte dos dispositivos entra no mercado de duas formas: a partir da demonstração de “equivalên-

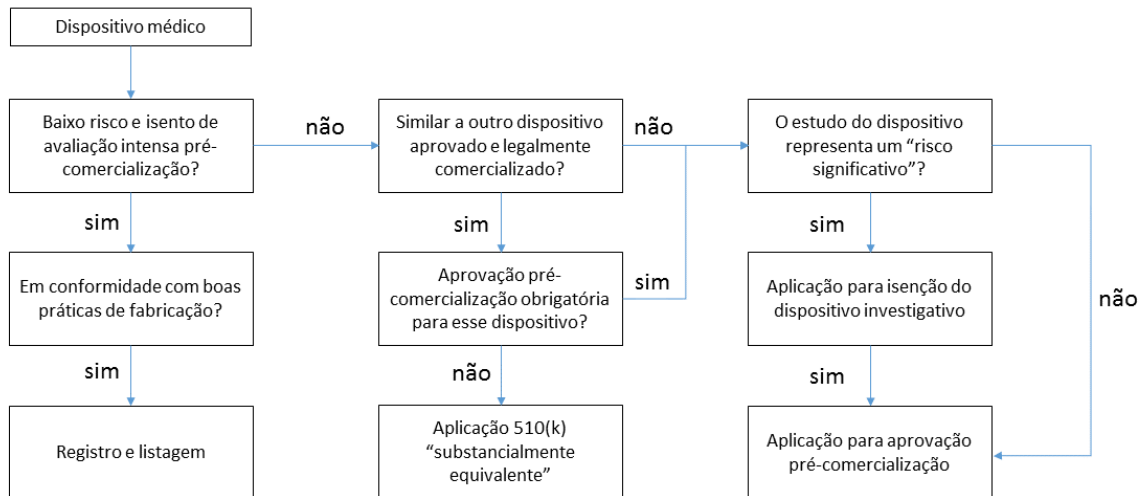


Figura 2.3: Caminhos de aprovação de dispositivos médicos nos EUA.

cia substancial”¹ a um dispositivo previamente aprovado e vendido legalmente, em um processo chamado de notificação pré-comercialização 510(k), ou através da demonstração de segurança e eficácia em um processo de aprovação pré-comercialização. Os dispositivos restantes, de menor risco, são isentos desse processo e somente precisam ser registrados e listados junto à FDA, demonstrando aderência às boas práticas de fabricação. A Figura 2.3 resume como funciona a aprovação de dispositivos na FDA [15].

2.1.3 Regulamentação na União Européia

O surgimento de legislação específica para regulamentação de dispositivos médicos no continente europeu se iniciou na década de 1970. Esse movimento se concentrava no registro de produtos, sem grande ênfase nos processos de fabricação ou integridade clínica. Isso resultou em um alto grau de divergência entre os ambientes regulatórios desses países, o que fez com que o processo de aprovação demorasse até cinco anos em alguns casos. Na prática, foram criadas fortes barreiras técnicas para o comércio, causando imbróglis diplomáticos entre países e impedindo a disseminação de produtos inovadores.

Isso fez com que o Conselho da Comunidade Econômica Européia (hoje Conselho da União Européia) adotasse, em 1985, as medidas propostas na sua resolução intitulada *New Approach to Technical Harmonization and Standards* [16]. Entre seus princípios fundamentais, se incluíam (1) o estabelecimento de regras comuns aos estados membros, limitadas aos requisitos essenciais de segurança ou de interesse geral, que produtos comercializados no mercado europeu precisariam seguir para gozar de trânsito livre na UE, (2) a centralização do desenvolvimento de normas técnicas em organizações especializadas em nível continental, (3) o caráter voluntário das normas técnicas, (4) o reconhecimento obrigatório pelas autoridades regulatórias de que produtos fabricados conforme as normas técnicas harmonizadas se adequam aos requisitos essenciais de segurança e (5) a exigência que o fabricante prove que se adequa aos requisitos essenciais de segurança, caso não siga as normas técnicas harmonizadas.

¹ A determinação de “equivalência substancial” requer que o fabricante especifique um produto aprovado e mostre que este tem as mesmas indicações de uso e usa tecnologias similares ao seu produto. Caso haja alguma diferença nas características tecnológicas dos dispositivos, ainda é possível que ambos sejam substancialmente equivalentes caso as informações enviadas para a FDA não levantem novas questões de segurança e eficácia, comprovando que seu dispositivo é ao menos tão seguro e eficaz quanto o aprovado.

Ao longo da década de 90, essas medidas foram se consolidando para regulamentar um mercado complexo e heterogêneo. Um dos símbolos mais conhecidos da harmonização dos processos regulatórios europeus é a marcação CE (sigla para *Conformité Européene*, ou conformidade europeia), indicativa que um produto atenderia a legislação da UE em quesitos como segurança, higiene e proteção ambiental - ao menos segundo seu fabricante². Diretivas específicas da UE determinam os critérios para que essa marcação seja utilizada [18].

O governo de cada membro da UE deve designar uma autoridade competente, responsável por dispositivos médicos. Trata-se de um organismo com autoridade para agir em nome do estado-membro para garantir que seu governo internalize as diretivas europeias para dispositivos médicos (i.e. faça a transposição dos seus requisitos para legislação nacional) e regule sua aplicação na sua jurisdição.

No sistema europeu, dispositivos médicos são divididos em quatro classes, em ordem crescente de risco: I, IIa, IIb e III. Na classe I, existem ainda duas subdivisões específicas: IIs para dispositivos estéreis e IIm para dispositivos com função de medição. Para determinar a classe de um dispositivo, são aplicadas regras referentes à duração do contato do dispositivo com o corpo do paciente, seu caráter invasivo, uso de fontes de energia, efeitos na circulação sanguínea ou sistema nervoso, impacto diagnóstico ou incorporação de produto medicinal. Por se tratar de um conjunto complexo de regras, a Direção-Geral da Saúde e da Segurança dos Alimentos da Comissão Europeia disponibiliza um guia com diretrizes para facilitar sua aplicação [19].

Dispositivos da classe I podem ser comercializados com uma simples declaração de conformidade, emitida pelo próprio fabricante, desde que não requeiram esterilização ou tenham função de medição. Os demais produtos devem validar sua declaração com um certificado de conformidade emitido por um organismo notificado, uma organização pública ou privada credenciada para atestar a adesão de dispositivos às diretivas europeias.

Dependendo da classificação do dispositivo, seu caminho para aprovação deve seguir diferentes trechos das diretivas europeias. As Figuras 2.4-2.7 ilustram o processo para dispositivos de cada uma das classes. Os anexos referidos podem ser encontrados na Diretiva 93/42/EEC, também conhecida como *Medical Devices Directive* (MDD) [20].

A evolução do ambiente de regulação europeu fez com que surgissem características distintas daquelas observadas no mercado estadunidense. De modo geral, trata-se de um sistema mais flexível e propício à inovação e comércio, porém também mais sujeito a risco. O Quadro 1, traduzido de [18], resume as principais diferenças entre os processos de aprovação de dispositivos médicos nos EUA e na UE.

2.1.4 Regulamentação no Brasil

Seguindo as tendências mundiais, o Brasil começou a estruturar um controle de qualidade maior sobre produtos para a saúde³ nos anos 1970, com a criação da Secretaria de Vigilância Sanitária (SVS) pela Lei nº 6.360/76, conhecida como Lei de Vigilância Sanitária. Na mesma década, também foi criado o Sistema Nacional de Metrologia, Normalização e Qualidade Industrial (Sinmetro) pela Lei nº 5.966/73, com a finalidade de formular e executar a política nacional de metrologia, normalização industrial e certificação de qualidade de produtos industriais, do qual fazem parte o Conselho Nacional de Metrologia, Normalização e Qualidade Industrial (Conmetro) e o Instituto Nacional de Metrologia, Normalização e Qualidade Industrial (Inmetro).

²A marcação CE é colocada pelo próprio fabricante em seus produtos. Ao fazer isto, ele assume integralmente toda a responsabilidade pela conformidade do produto em cumprir as diretivas legais vigentes na Europa [17]. Em muitos casos, não há garantia que o produto é, de fato, seguro ou certificado por uma Autoridade Competente. Apenas dispositivos considerados de alto risco são avaliados, ainda assim apenas por Organismos Notificados - instituições de certificação independentes acreditadas.

³Produtos para Saúde são uma categoria mais abrangente que Equipamentos Médicos, compreendendo ainda Materiais de Uso em Saúde e Produtos de Diagnóstico in vitro.

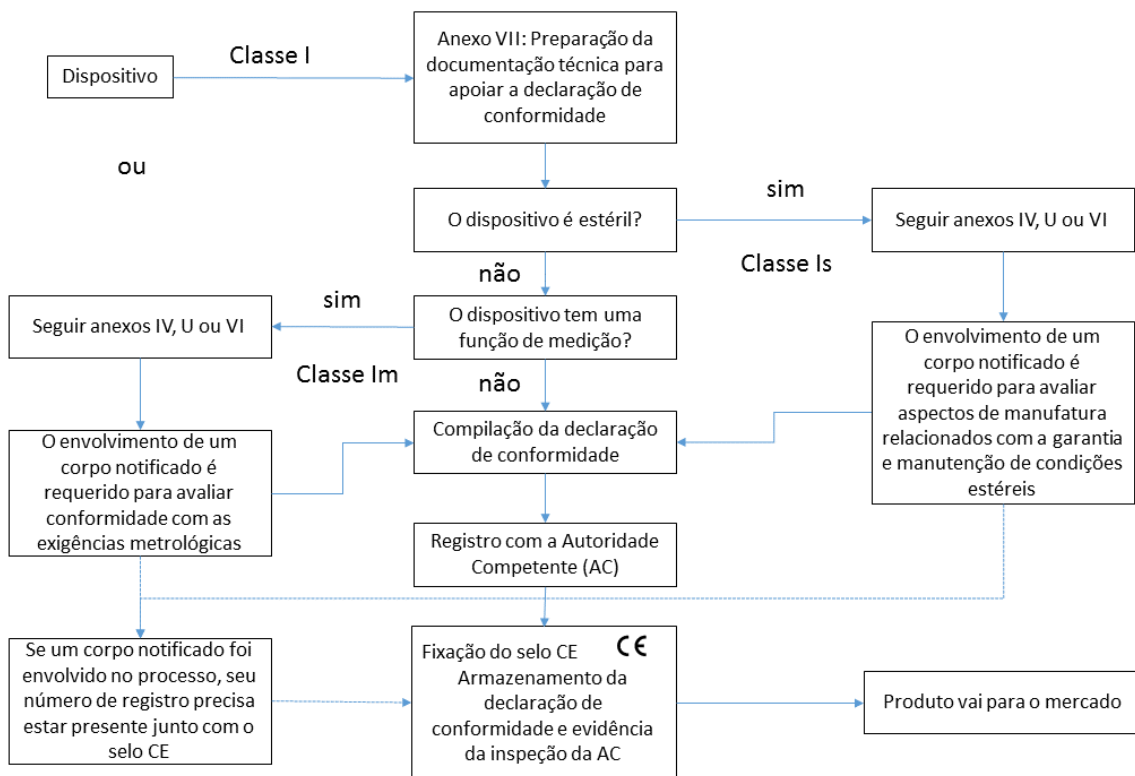


Figura 2.4: Caminhos de aprovação de dispositivos médicos da Classe I na UE.

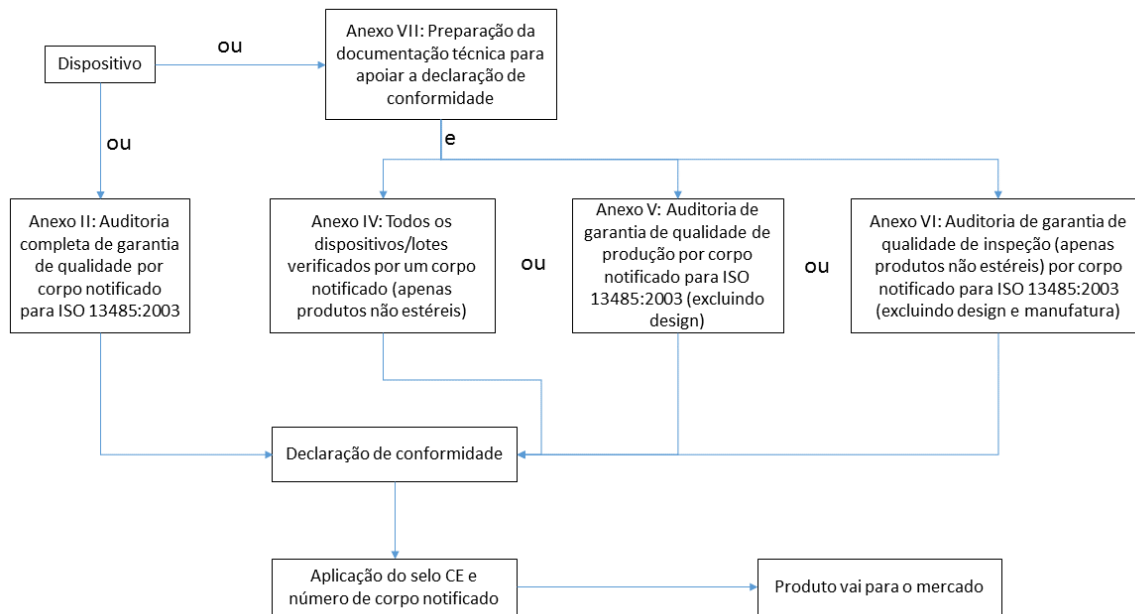


Figura 2.5: Caminhos de aprovação de dispositivos médicos da Classe IIa na UE.

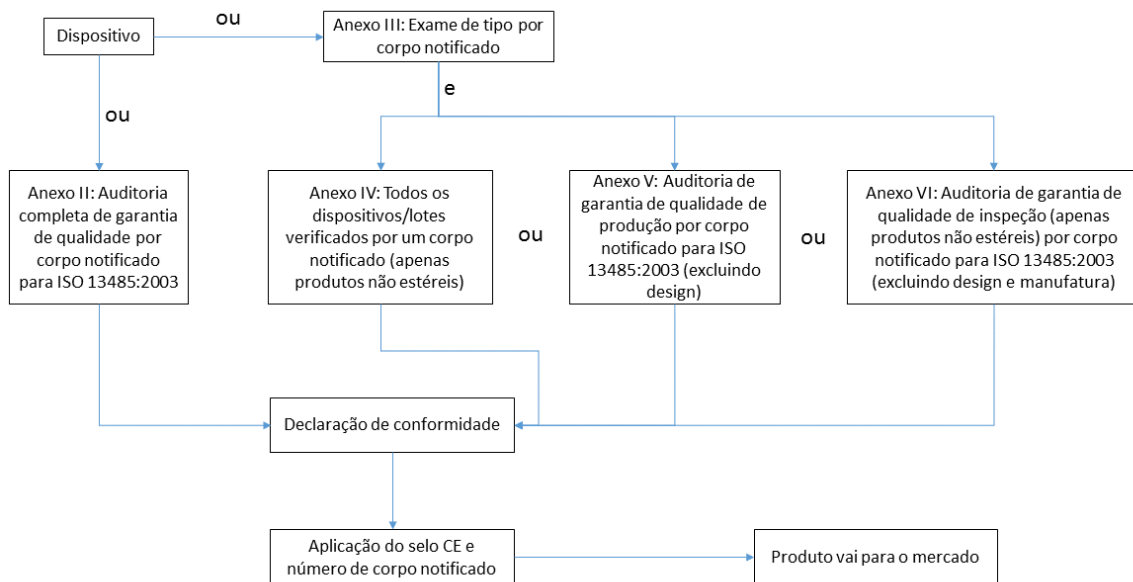


Figura 2.6: Caminhos de aprovação de dispositivos médicos da Classe IIb na UE.

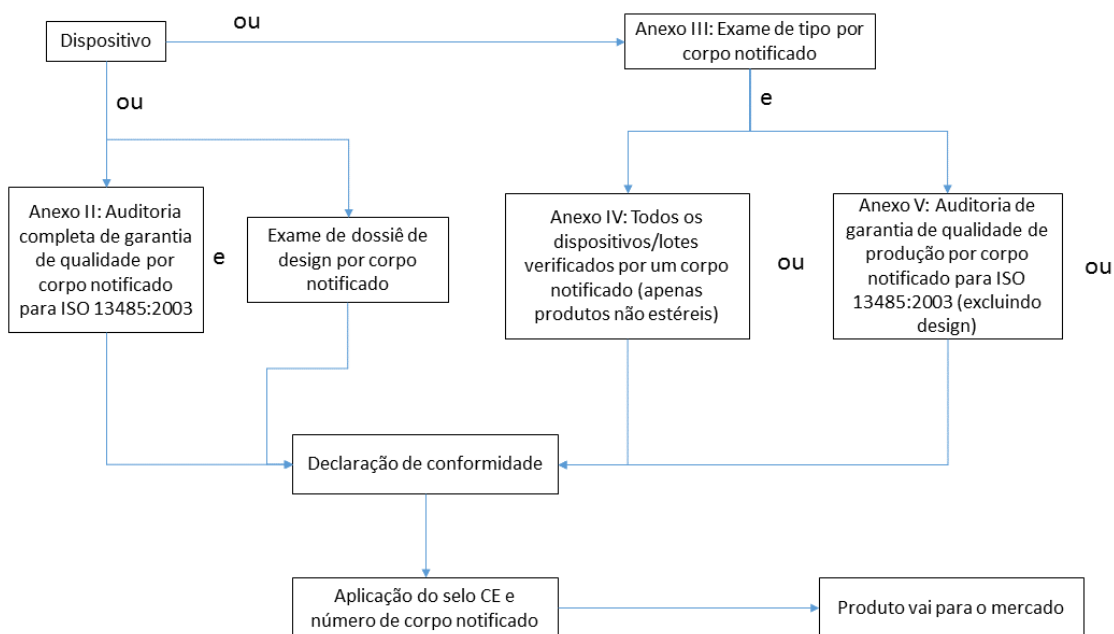


Figura 2.7: Caminhos de aprovação de dispositivos médicos da Classe III na UE.

Tabela 2.1: Diferenças entre os processos de aprovação dispositivos médicos nos EUA e na UE.

Característica	Estados Unidos	União Européia	Possíveis Implicações
Mandato	Supervisão da saúde pública	Segurança de dispositivos (supervisionada por Autoridades Competentes), aprovação de dispositivos (por Organismos Notificados) e facilitação de comércio	Podem influenciar as negociações com clientes da indústria e atenção prestada ao balanço entre efetividade e segurança
Centralização	Supervisão de toda regulamentação de dispositivos pela FDA	Diretivas especificam processos executados por Autoridades Competentes e Organismos Notificados	Padronização e coordenação de avaliações pré- e pós-comercialização são teoricamente mais simples e fáceis de aplicar nos Estados Unidos
Dados requeridos	Garantia razoável de segurança e efetividade para aprovação de dispositivos de alto risco, “equivalência substancial” para liberação 510 (k)	Geralmente análise baseada em desempenho, exigindo prova que o dispositivo funciona como pretendido	Avaliação na UE é feita por fabricantes e Organismos Notificados; fornece menor compreensão dos limites clínicos de dispositivos de alto risco
Transparência	Limites proprietários com relatórios públicos da análise pré-comercialização de dispositivos aprovados, <i>recalls</i> e eventos adversos	Análise de Organismos Notificados não é divulgada; dados pós-comercialização compartilhados entre Autoridade Competentes mas não com o público.	Maior acesso público a evidências nos Estados Unidos
Financiamento	Combinação de recursos federais (> 80%) e taxas de usuários (< 20%)	Financiamento de Autoridades Competentes variável entre países; Organismos Notificados pagos diretamente pelos clientes da indústria	Organismos Notificados podem ser vulneráveis a conflitos de interesse com clientes da indústria; a FDA pode ser influenciada por mudanças no financiamento público e clima político
Acesso	Testes clínicos pré-comercialização de dispositivos de alto risco atrasam o acesso de pacientes a esses dispositivos (não há diferença nos prazos entre EUA e UE para dispositivos de risco baixo e moderado)	Pacientes na UE podem ter acesso a determinados dispositivos de alto risco mais rapidamente que nos EUA, sujeito a limitações financeiras dos pacientes	Pacientes na UE têm acesso mais rápido a determinados dispositivos, mas esses produtos são comercializados com provas menos rigorosas de efetividade e podem ter maior chance de eventos adversos identificados posteriormente

A Lei de Vigilância Sanitária estabelece que “nenhum produto de interesse à saúde, seja nacional ou importado, poderá ser industrializado, exposto à venda ou entregue ao consumo no mercado brasileiro antes de registrado no Ministério da Saúde”, com exceções indicadas na Lei que não precisam passar pelo processo de registro, mas devem ser cadastrados⁴ e continuam sujeitos ao regime de Vigilância Sanitária.

Em 1980, foi criado pelo Conmetro o Sistema Brasileiro de Avaliação da Conformidade (SBAC), estabelecendo um “processo sistematizado, acompanhado e avaliado, de forma a propiciar adequado grau de confiança de que um produto, processo ou serviço, ou ainda um profissional, atende a requisitos pré-estabelecidos em normas e regulamentos técnicos com o menor custo para a sociedade”. Esse sistema não se restringe a equipamentos médicos, abrangendo toda produção industrial brasileira.

Por meio da Lei nº 9.782/99 foi criada a Agência Nacional de Vigilância Sanitária (Anvisa), substituindo a SVS e dando maior autonomia para o novo órgão exercer sua função de regulamentar, controlar e fiscalizar os produtos e serviços que envolvam risco à saúde pública, incluindo a concessão de registro de produtos. Grande parte da legislação vigente, composta principalmente por instruções normativas (IN) e resoluções da diretoria colegiada (RDC), surgiu a partir desse marco histórico.

No Brasil, foram estabelecidas quatro classes de risco para equipamentos médicos (I, II, III e IV) e dezoito regras de enquadramento. A estrutura de classificação corresponde à utilizada na União Européia segundo a MDD [21]. De forma resumida, a classificação por regra obedece aos seguintes critérios:

- Produtos não invasivos: regras 1, 2, 3 e 4;
- Produtos invasivos: regras 5, 6, 7 e 8;
- Produtos ativos: regras 9, 10, 11 e 12;
- Especiais: regras 13, 14, 15, 16, 17 e 18.

Existem dois tipos de regularizações de equipamentos médicos junto à Anvisa: o registro e o cadastro. O cadastro é um procedimento simplificado de regularização, aplicável apenas a alguns produtos das classes I (baixo risco) ou II (médio risco). Os demais produtos das classes I e II, bem como todos das classes III (alto risco) e IV (máximo risco), estão sujeitos a registro.

O sistema brasileiro é complexo, com várias normas que são revistas com frequência para harmonização regulatória com o Mercosul e atualização com parâmetros internacionais. Algumas das normas mais importantes no fluxo de registro e cadastro de equipamentos médicos no Brasil são a (1) RDC nº 16/14, que estipula os critérios para o peticionamento de Autorização de Funcionamento de Empresas junto à Vigilância Sanitária, a (2) RDC nº 16/13, que determina o conjunto de Boas Práticas de Fabricação e Controle (BPFC) exigidos pela Anvisa de todos fabricantes de equipamentos médicos, a (3) RDC nº 185/01, que regulamenta o registro, a alteração, a revalidação e o cancelamento do registro de produtos para saúde, a (4) RDC nº 27/11, que dispõe sobre os requisitos mínimos para comprovar a segurança e eficácia de produtos para saúde, a (5) RDC nº 40/15, que define os requisitos do regime de cadastro para o controle sanitário dos produtos para saúde, a (6) RDC nº 32/07, que dispõe sobre a certificação compulsória dos equipamentos elétricos sob regime de Vigilância Sanitária e a (7) IN nº 03/11, que lista as normas técnicas cujas prescrições devem ser atendidas para certificação de conformidade, no âmbito do SBAC, de equipamentos sob regime de Vigilância Sanitária.

Com o intuito de auxiliar os fabricantes e importadores de equipamentos médicos no processo de solicitação de registro ou cadastro, a Gerência de Tecnologia em Equipamentos Médicos (GQUIP) da Anvisa elaborou um

⁴É possível traçar paralelos entre o cadastramento brasileiro, o *registration and listing* americano e a declaração de conformidade européia - todos são mecanismos para controle de dispositivos de baixo risco.

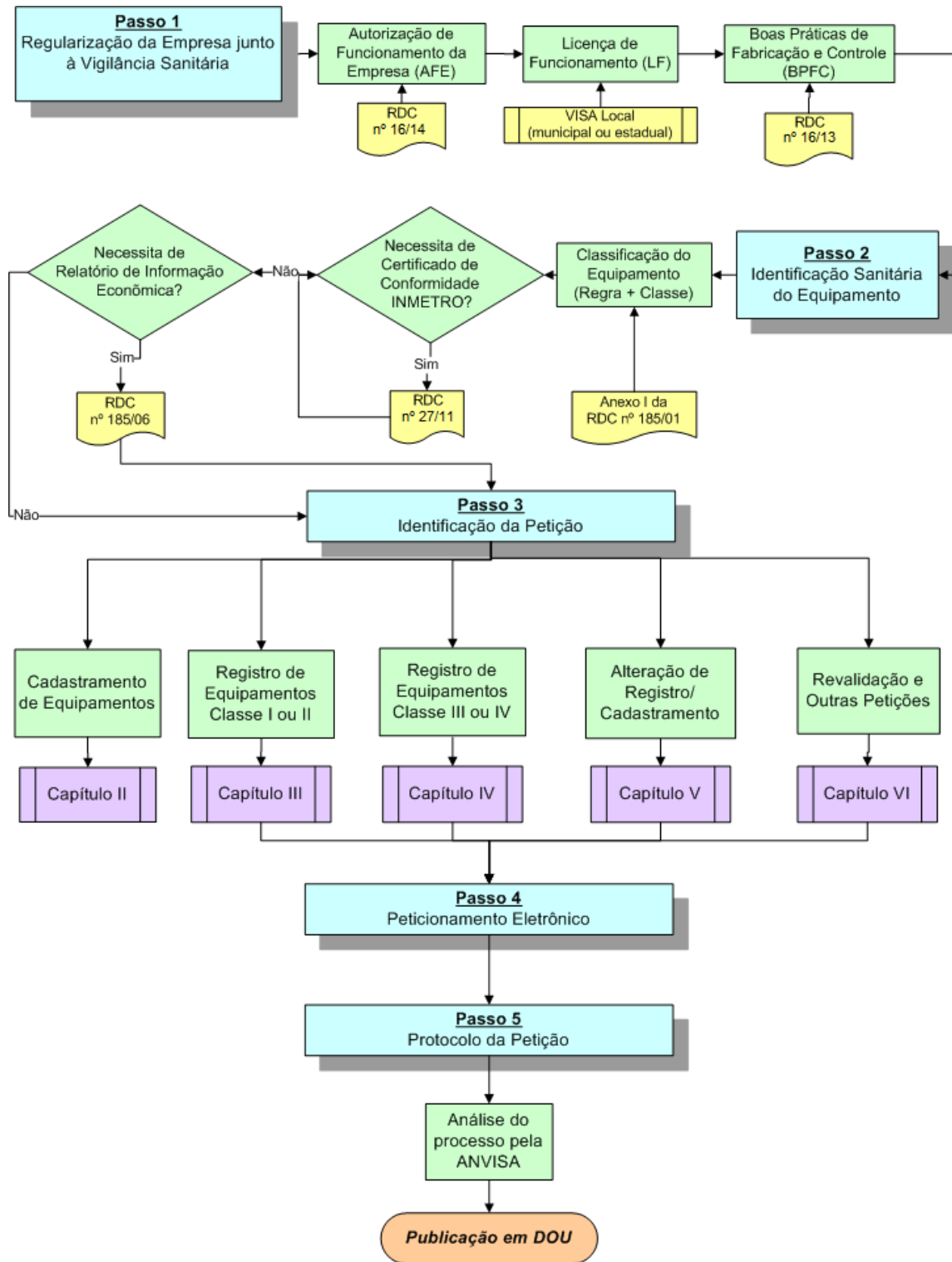


Figura 2.8: Caminhos de aprovação de dispositivos médicos no Brasil.

manual [22] para guiá-los na estrutura normativa brasileira. A Figura 2.8, retirada desse manual e atualizada com a legislação vigente⁵, ilustra o processo de registro ou cadastramento de equipamentos médicos no Brasil.

Dentro do SBAC, está prevista a colaboração da Anvisa com o Sinmetro para a certificação de conformidade dos produtos sob regime de Vigilância Sanitária. Nesse sistema, o Inmetro tem a função de acreditar Organismos Certificadores de Produtos (OCPs) e Laboratórios de Ensaio (LEs). A IN nº 04/15 lista as normas técnicas cujos parâmetros devem ser adotados para a certificação de conformidade no âmbito do SBAC.

OCPs são similares aos organismos notificados do sistema europeu, sendo entidades independentes responsáveis por conduzir todo processo de certificação, identificar laboratórios para realização de ensaios conforme normas técnicas aplicáveis e realizar auditoria no processo produtivo do fabricante. Atualmente existem cerca de 17 OCPs e 15 LEs acreditados pelo Inmetro para equipamentos médicos [23].

2.2 NORMAS TÉCNICAS SOBRE DISPOSITIVOS MÉDICOS

Normas técnicas são documentos, estabelecidos por consenso e aprovados por organismos reconhecidos, que fornecem requisitos, especificações, diretrizes ou características que podem ser usados consistentemente para garantir que materiais, produtos, processos e serviços sejam adequados para seu propósito [24]. Para tanto, tais normas devem ser obtidas a partir de resultados consolidados da ciência, tecnologia e experiência [25].

Fundamentais no cenário regulatório de dispositivos médicos, normas técnicas apresentam critérios harmonizados para assegurar a segurança, qualidade e desempenho destes produtos [26]. Fornecendo uma base técnica representativa de uma ampla gama de especialistas, elas são aplicadas no desenvolvimento de regulamentação nacional e internacional, reduzindo custos e barreiras técnicas ao comércio [27].

Apesar de todas as vantagens apresentadas pela aplicação de normas técnicas em processos regulatórios, desafios imprevistos podem ser introduzidos por produtos inovadores. Para evitar que a aplicação rígida e obrigatória de normas técnicas seja uma empecilho para a inovação tecnológica, muitas autoridades competentes nacionais optam por tornar a conformidade a elas voluntária [26]. A motivação por trás disso é a compreensão que, com o avanço da tecnologia, é mais fácil atualizar normas técnicas que mudar a legislação vigente.

Fabricantes tendem a adotar normas voluntárias de qualquer forma, pois estas apresentam uma forma objetiva de demonstrar conformidade aos requisitos regulatórios, mas contam com flexibilidade para fazê-lo por outros meios se for mais conveniente [28, p. 28]. Para serem aceitos, porém, estes devem ser comprovadamente tão rigorosos quanto as normas técnicas reconhecidas.

Mundialmente, os principais organismos responsáveis pela definição de normas técnicas para dispositivos médicos são a *International Standards Organization* (ISO) e a *International Electrotechnical Commission* (IEC). A Associação Brasileira de Normas Técnicas (ABNT) é a responsável pela normalização técnica no Brasil, fornecendo insumos ao desenvolvimento tecnológico brasileiro. Muitas normas técnicas internacionais definidas pela ISO/IEC foram adotadas no Brasil, após serem traduzidas e publicadas pela ABNT como padrão nacional (ABNT NBR). Essa prática de internalização de normas técnicas converge com o movimento global pela harmonização e redução de barreiras técnicas para o comércio internacional.

Em 2014, o IMDRF publicou um relatório listando as normas técnicas para dispositivos médicos reconhe-

⁵É possível consultar as normas vigentes e alterações em: <http://portal.anvisa.gov.br/legislacao>

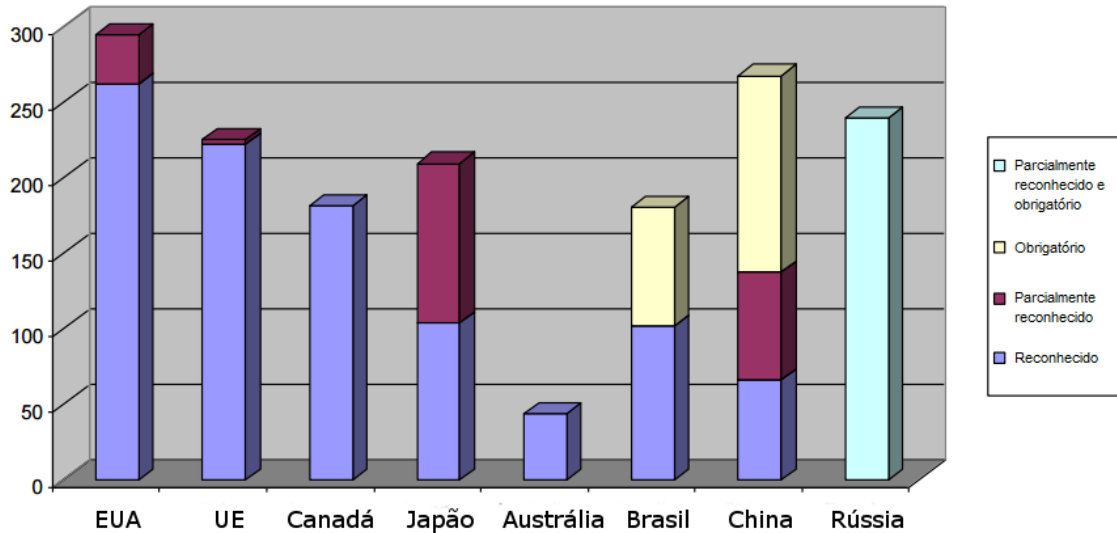


Figura 2.9: Grau de reconhecimento⁶ de normas técnicas internacionais por membros do IMDRF.

cidas⁶ e usadas em processos regulatórios por seus membros [29]. Um painel de especialistas avaliou 1102 normas técnicas internacionais ISO/IEC e indicou o grau de reconhecimento destes: completamente reconhecido, não reconhecido, parcialmente reconhecido ou obrigatório. A Figura 2.9 resume o resultado dessa sondagem, mostrando que no Brasil reconhecia-se 180 dessas normas internacionais, sendo que 78 eram obrigatórias⁷ para aprovação de dispositivos médicos. Outro fator que chama atenção é que apenas três países do grupo consideravam a conformidade com normas técnicas internacionais como requisito obrigatório em processos regulatórios: Brasil, China e Rússia.

Listas apresentadas como anexo ao relatório detalham o status de cada norma técnica analisada nos países membros do IMDRF em 2014. Na lista de padrões reconhecidos pelo Brasil [30], observa-se que nenhuma norma técnica relacionada a software era reconhecida ou obrigatória.

2.2.1 Normalização de software de dispositivos médicos

A nota técnica nº 04/2012/GQUIP/GGTPS/Anvisa foi publicada para servir como um guia orientativo às empresas que desenvolvem software de produtos para saúde, levando em consideração a falta de um regulamento específico no Brasil [31]. Nela é esclarecido o enquadramento sanitário de (1) software produto para a saúde, por si mesmo, (2) software parte (ou acessório) de um produto para a saúde e (3) software não produto para a saúde, bem como os requisitos do dossiê técnico relacionado a eles.

No caso da segunda categoria, que nos interessa neste trabalho, o entendimento é que a classificação de risco de um software em geral acompanha a classificação do hardware, com poucas exceções. Existe uma diferenciação no tratamento de software parte (essencial para a função pretendida) e acessório (não essencial para a função pretendida) de um equipamento médico. Em determinadas circunstâncias, o software pode ser isento de registro, obrigatoriamente registrado junto ao hardware associado ou contar com registro próprio.

⁶Adota-se aqui a definição dada em [26]: uma “norma técnica reconhecida” é aquela que oferece meios suficientes para demonstrar conformidade com ao menos parte dos princípios essenciais de segurança e desempenho adotados por uma autoridade competente.

⁷A IN nº 04/15, que apresenta a lista de normas técnicas cujos parâmetros devem ser adotados para a certificação de conformidade, aumentou esse número para 88 normas obrigatórias.

Não há qualquer indicação nesse documento de que seja feita uma avaliação independente do funcionamento do software de um equipamento médico ou análise do seu código-fonte, sendo suficiente para fins de registro a entrega de arquivos de gerenciamento de risco (segundo a norma ABNT NBR ISO 14971) e relatórios de estudos e testes para verificação e validação da segurança e eficácia do equipamento, realizados segundo critério definidos pelo próprio fabricante.

No *Seminário de Dispositivos Médicos*, promovido pela Anvisa em 2016, houve uma apresentação sobre Regulação de Software no Brasil [32] que sinalizou aspectos importantes sobre o tema. Em primeiro lugar, estabeleceu-se que a verificação de eficácia de software deveria ser fundamentada em métodos consolidados da Engenharia de Software. Em seguida, enfatizou-se a utilização de metodologias de desenvolvimento ágeis ou em cascata, comuns no mercado, bem como a relevância do uso da linguagem UML (*Unified Modeling Language*) no projeto software e a aplicação de testes unitários, testes de integração e testes de sistema.

Dentre as considerações apresentadas, menciona-se que existem lacunas na legislação vigente: não há especificação clara quanto à documentação exigida, nem critérios específicos para o tratamento de software no contexto regulatório. Apesar disso, são citadas sinalizações recentes no Formulário de Cadastro de Software [33] da Anvisa, indicando o que se espera quando se trata de arquitetura de software (diagrama de componentes da UML), hardware (diagrama de implementação da UML) e verificação do sistema (sumário dos testes unitários, de integração e de sistemas, anomalias/bugs não resolvidos).

Mesmo que esta ficha técnica não se aplique a software embarcado, cujas informações devem ser incluídas na solicitação de registro do equipamento médico ao qual se destina, observa-se uma preferência por determinadas formas de apresentação e análise da eficácia de software pelo órgão. Como perspectivas futuras, apresenta-se a possibilidade de ser introduzida legislação própria para o registro de software, especificando seus requisitos de forma explícita.

Apesar de não ter sido internalizada pela ABNT, nem constar na lista de normas técnicas reconhecidas pelo Brasil [30], a apresentação cita a norma IEC 62304 como uma referência para auxiliar na comprovação de segurança e eficácia de software de dispositivos médicos. Além desta, considerada a principal norma tratando do assunto, outras normas técnicas influenciam o processo de desenvolvimento de software, como pode ser visto na Figura 2.10. Como observado no diagrama, a conformidade com as normas ISO 13485 e ISO 14971 é um dos requisitos impostos pela IEC 62304. Assim, o entendimento das diretrizes apresentadas pelas três normas é fundamental para o desenvolvimento de software de sistemas mecatrônicos aplicados à saúde.

2.2.2 IEC 62304: Processos do ciclo de vida de software de dispositivos médicos

A norma técnica IEC 62304 define requisitos de ciclo de vida para software de dispositivos médicos, estabelecendo uma estrutura comum para o desenvolvimento e manutenção de software dessa categoria através da descrição de um conjunto específico de processos, atividades e tarefas [34]. Esta norma se aplica tanto a software produto para a saúde, por si mesmo, quanto a software embarcado parte (ou acessório) de um produto para a saúde, estendendo os conceitos apresentados na norma mais geral ISO/IEC 12207.

Em 2014, a norma fazia parte de um seleto grupo de dezessete normas técnicas adotadas, de forma voluntária ou obrigatória, por seis dos oito membros do IMDRF [29]. Em uma declaração conjunta sobre a adoção da IEC 62304 em diferentes jurisdições, os países-membros detalharam o status de reconhecimento da normas por suas autoridades nacionais competentes [35].

Os principais mercados para dispositivos médicos, EUA e UE, consideram a adequação à norma suficiente

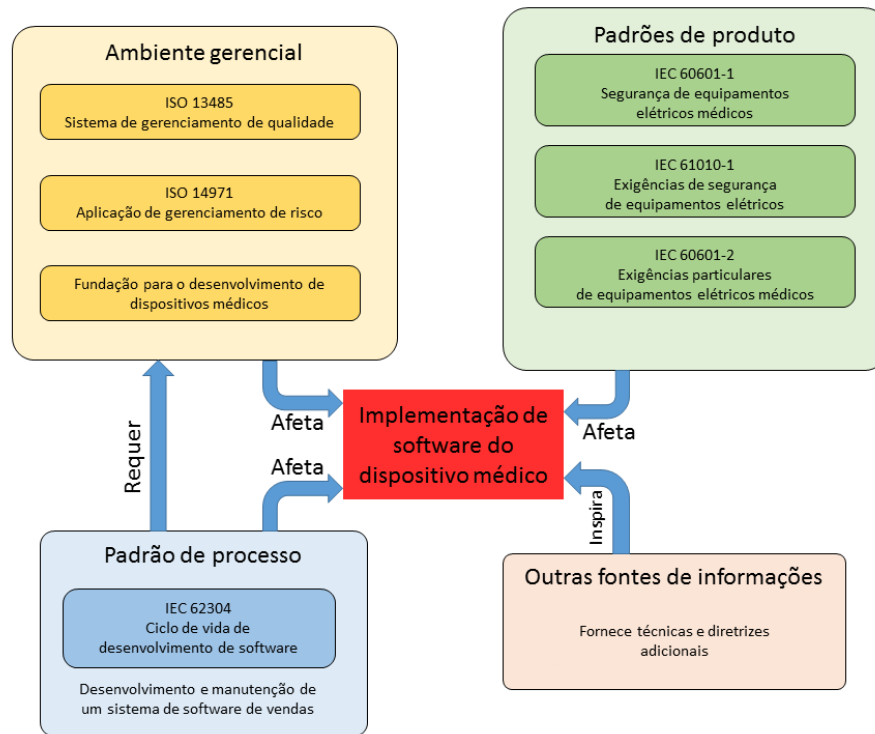


Figura 2.10: Normas técnicas afetando a implementação de software para dispositivos médicos.

para demonstrar conformidade de software de dispositivos médicos com os requisitos legais vigentes — embora enfatizem que outros meios possam ser empregados para o mesmo fim, desde que se mostrem ao menos tão bons quanto a norma. No Brasil não há indicação de que a adoção de suas medidas seja suficiente para atender os requisitos de segurança e eficácia exigidos pela Anvisa. A norma técnica não foi internalizada pela ABNT, porém foi indicado pelo Brasil na declaração conjunta do IMDRF que seus documentos podem fazer parte do dossiê técnico exigido pela Anvisa no registro de equipamentos médicos.

O enfoque da norma é apenas no estabelecimento de processos que possibilitem que o projeto, desenvolvimento e manutenção de software de dispositivos médicos sejam seguros, cumprindo sua função pretendida sem causar riscos inaceitáveis. Ela não aborda, portanto, a validação da versão final do dispositivo médico, mesmo quando este consiste puramente de software.

Um conceito fundamental da IEC 62304 é a presunção de que o software de dispositivos médicos é desenvolvido e mantido dentro de sistemas de gestão de qualidade e gerenciamento de risco. Para tanto, é feita referência normativa aos padrões ISO 13485 e ISO 14971, respectivamente, adicionando apenas aspectos de gerenciamento de risco específicos para software.

O ponto de partida da aplicação da norma é a classificação do software de acordo com seu potencial de criar situações de perigo que possam resultar em uma lesão no usuário, paciente ou terceiros[36]. Essa classificação define as atividades que devem ser desempenhadas pelo fabricante para atingir conformidade com a norma, de acordo com o risco apresentado. São adotadas três classes:

- Classe A: nenhuma lesão ou dano à saúde é possível;
- Classe B: potencial de lesão não-séria;

Tabela 2.2: Exigências da IEC 62304 para cada classe de software.

Documentação de Software Exigida		Classe		
Cláusula	Sub-cláusulas	A	B	C
Software development plan	5.1.1, 5.1.2, 5.1.3, 5.1.6, 5.1.7, 5.1.8, 5.1.9	X	X	X
	5.1.5, 5.1.10, 5.1.11		X	X
	5.1.4			X
Software requirements specification	5.2.1, 5.2.2, 5.2.4, 5.2.5, 5.2.6	X	X	X
	5.2.3		X	X
Software architecture	5.3.1, 5.3.2, 5.3.3, 5.3.4, 5.3.6	X	X	X
	5.3.5			X
Software detailed design	5.4.1		X	X
	5.4.2, 5.4.3, 5.4.4			X
Software unit implementation and verification	5.5.1	X	X	X
	5.5.2, 5.5.3, 5.5.5		X	X
	5.5.4			X
Software integration and integration testing	Todos requisitos (5.6)		X	X
Software system testing	Todos requisitos (5.7)		X	X
Software release	5.8.4	X	X	X
	5.8.1, 5.8.2, 5.8.3, 5.8.5, 5.8.6, 5.8.7, 5.8.8		X	X
Software maintenance process	6.1, 6.2.1, 6.2.2, 6.2.4, 6.2.5	X	X	X
	6.2.3		X	X
Software risk management process	7.4.1	X	X	X
	7.1, 7.2, 7.3, 7.4.2, 7.4.3		X	X

- Classe C: potencial de lesão séria ou morte.

A partir da classificação do software, a norma técnica IEC 62304 propõe que seu ciclo de vida seja dividido em cinco processos: (1) desenvolvimento, (2) manutenção, (3) gerenciamento de risco, (4) gerenciamento de configuração e (5) resolução de problemas. Desenvolvimento e manutenção fazem parte de fluxos distintos porém similares, enquanto os demais processos permeiam todo o ciclo de vida, como ilustrado nas Figuras 2.11 e 2.12. A Tabela 2.2, retirada de [37], lista as exigências impostas pela norma para cada classe de software, citando as cláusulas segundo a numeração adotada no texto da norma.

A norma IEC 62304 não especifica uma estrutura organizacional para o fabricante ou dita que parte da organização deve executar cada processo, atividade ou tarefa, mas exige que todas etapas sejam executadas para estabelecer a conformidade com essa norma técnica. Ela também não prescreve o nome, formato ou conteúdo exato da documentação a ser produzida, mas exige que as tarefas sejam documentadas segundo algum formato determinado pelo usuário. Além disso, a norma não prescreve um modelo específico de ciclo de vida, mas exige que o usuário mapeie os processos, atividades e tarefas da norma ao modelo escolhido.

Algo importante a ser observado é o tratamento dado pela IEC 62304 para software desenvolvido por terceiros, denominado como *Software of Unknown Provenance* (SOUP). Qualquer parte do software do sistema

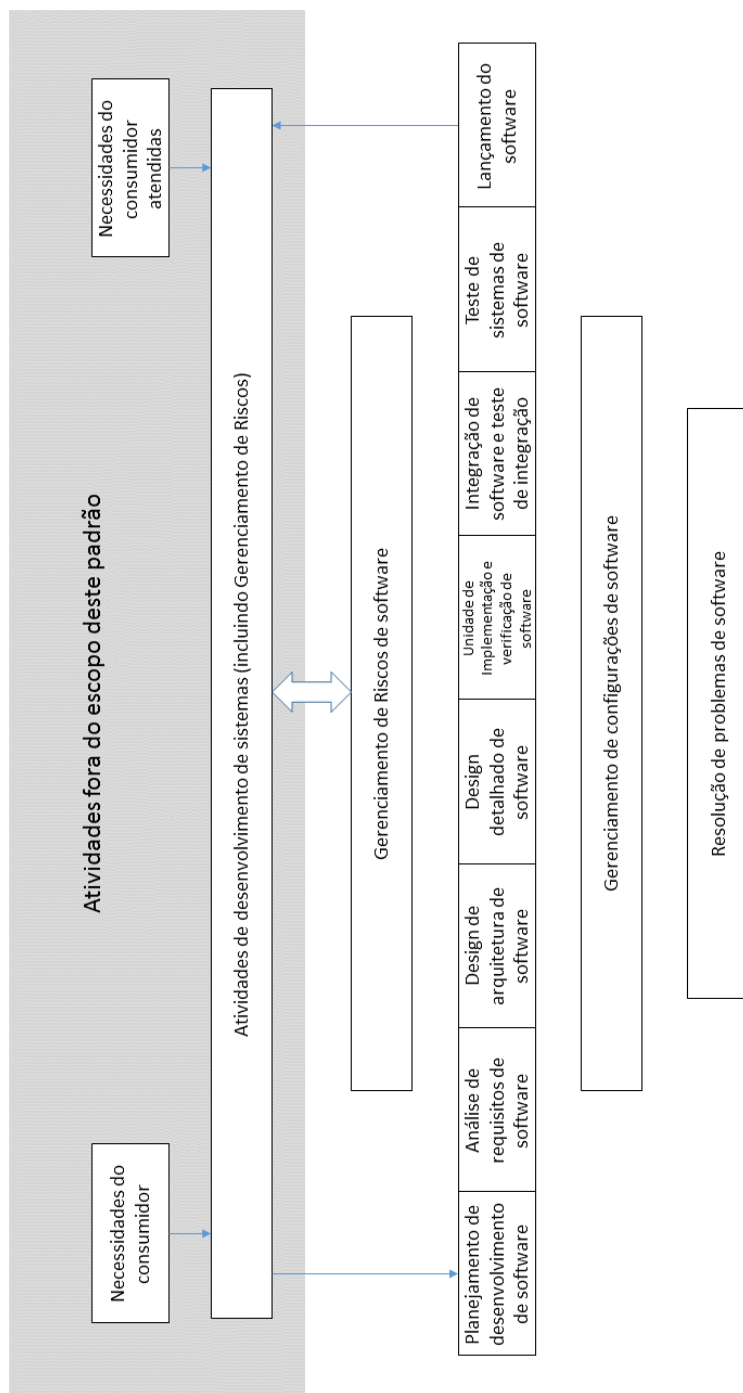


Figura 2.11: Fluxo de processos e atividades de desenvolvimento de software na IEC 62304.

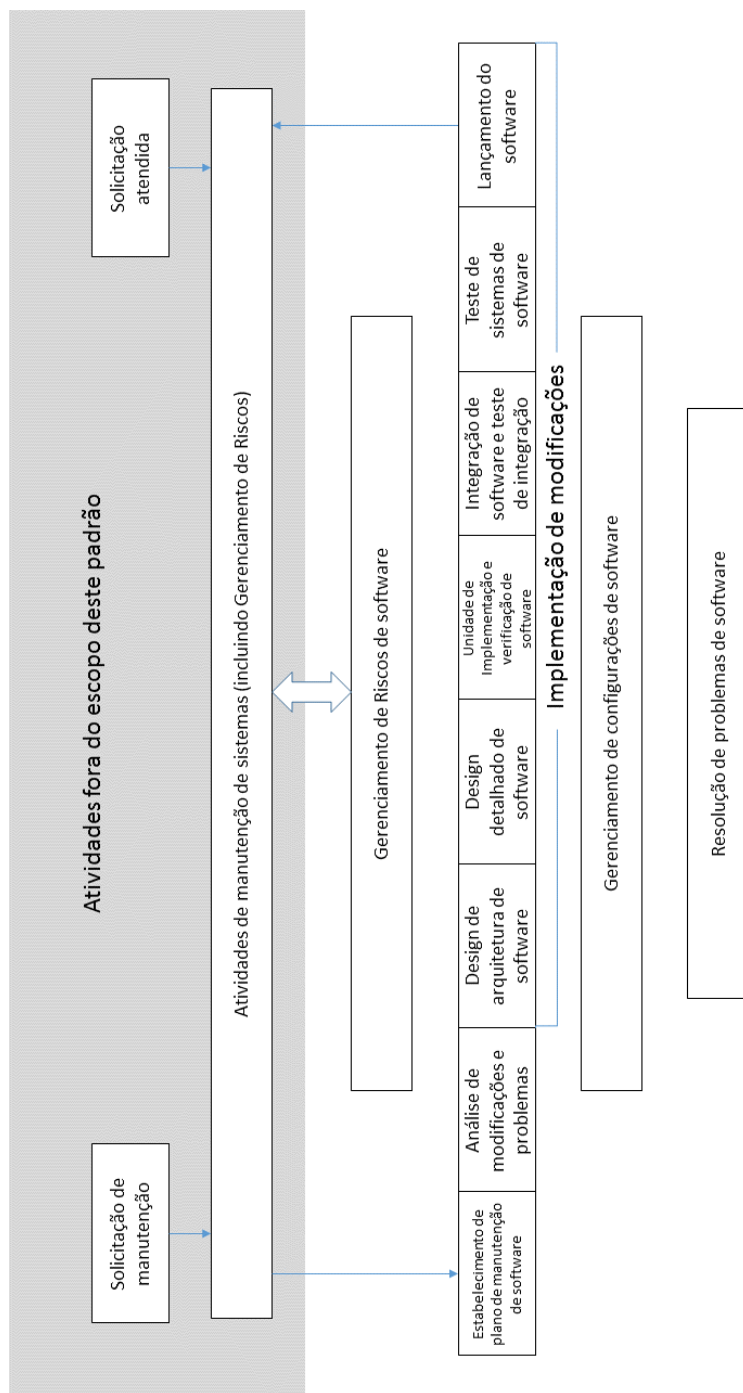


Figura 2.12: Fluxo de processos e atividades de manutenção de software na IEC 62304.

que não tenha sido desenvolvido e documentado segundo as especificações da norma pode ser classificado como SOUP, inclusive elementos pouco enfatizados na concepção de um sistema como sistema operacional, *drivers* e bibliotecas padrão da linguagem utilizada. Embora o uso de software *open source* seja benéfico para muitos projetos, o acesso ao código-fonte por si só não garante conformidade à norma: é necessário incluir todo SOUP no gerenciamento de risco do sistema. Dependendo da classificação de risco do software, pode ser melhor simplesmente não utilizá-lo.

2.2.3 ISO 14971: Aplicação de gerenciamento de risco a dispositivos médicos

A norma técnica ISO 14971 especifica um processo para um fabricante identificar os perigos associados a dispositivos médicos, estimar e avaliar os riscos associados, controlar estes riscos e monitorar a eficácia desses controles [38]. Os requisitos da norma se aplicam a todos os estágios do ciclo de vida de um dispositivo médico, porém esta não se aplica à tomada de decisão clínica, nem define níveis aceitáveis de risco.

Em 2014, a norma fazia parte de um seleto grupo de duas normas técnicas adotadas, de forma voluntária ou obrigatória, por sete dos oito membros do IMDRF [29]. Em uma declaração conjunta sobre a adoção da ISO 14971 em diferentes jurisdições, os países-membros detalharam o status de reconhecimento da norma por suas autoridades nacionais competentes [39].

Os principais mercados para dispositivos médicos, EUA e UE, consideram a adequação à norma suficiente para demonstrar conformidade do processo de gerenciamento de riscos associados a dispositivos médicos com os requisitos legais vigentes, embora enfatizem que outros meios possam ser empregados para o mesmo fim, desde que se mostrem ao menos tão bons quanto a norma. Apesar disso, foi mencionado pelo representante da UE que seria necessário que fabricantes e organismos de avaliação de conformidade fizessem o mapeamento entre os requisitos legais exigidos e as recomendações da norma.

No Brasil é obrigatório que fabricantes e importadores de dispositivos médicos apresentem registros de gerenciamento de risco. A norma técnica foi internalizada pela ABNT e seu uso é reconhecido como suficiente para atender os requisitos da Anvisa. Diversas normas da Anvisa mencionam a ISO 14971, considerando-a aplicável aos estágios de pré e pós-comercialização.

A IEC 62304 faz referência normativa à ISO 14971, estabelecendo que os perigos causados diretamente ou indiretamente por software devem ser considerados e identificados, conforme indicado na norma, para então terem seu risco controlado em um processo integrado com o processo de gerenciamento de risco proposto pela ISO 14971. O gerenciamento de risco de um dispositivo médico não pode ser dissociado do gerenciamento de risco do software embarcado.

O entendimento proposto pela ISO 14971 é que o conceito de risco envolve a combinação da probabilidade de ocorrência de dano e da severidade deste dano. A norma estabelece que o uso de um dispositivo médico implica em algum grau de risco e, portanto, requer que riscos residuais sejam avaliados frente aos benefícios esperados do procedimento. Para tanto, especifica um processo através do qual o fabricante pode gerenciar esse risco, incluindo (1) análise de risco, (2) estimativa de risco, (3) controle de risco e (4) informações de produção e pós-produção. A Figura 2.13 apresenta uma representação esquemática do processo de gerenciamento de risco proposto pela norma. Suas atividades são detalhadas na Figura 2.14.

Ao longo do seu texto, a norma ISO 14971 enfatiza a importância das ações de gerenciamento de risco serem planejadas e documentadas em um arquivo de gerenciamento de risco, usado para aferição de conformidade. Todo perigo identificado deve ser rastreável para as etapas do processo de gerenciamento de risco.

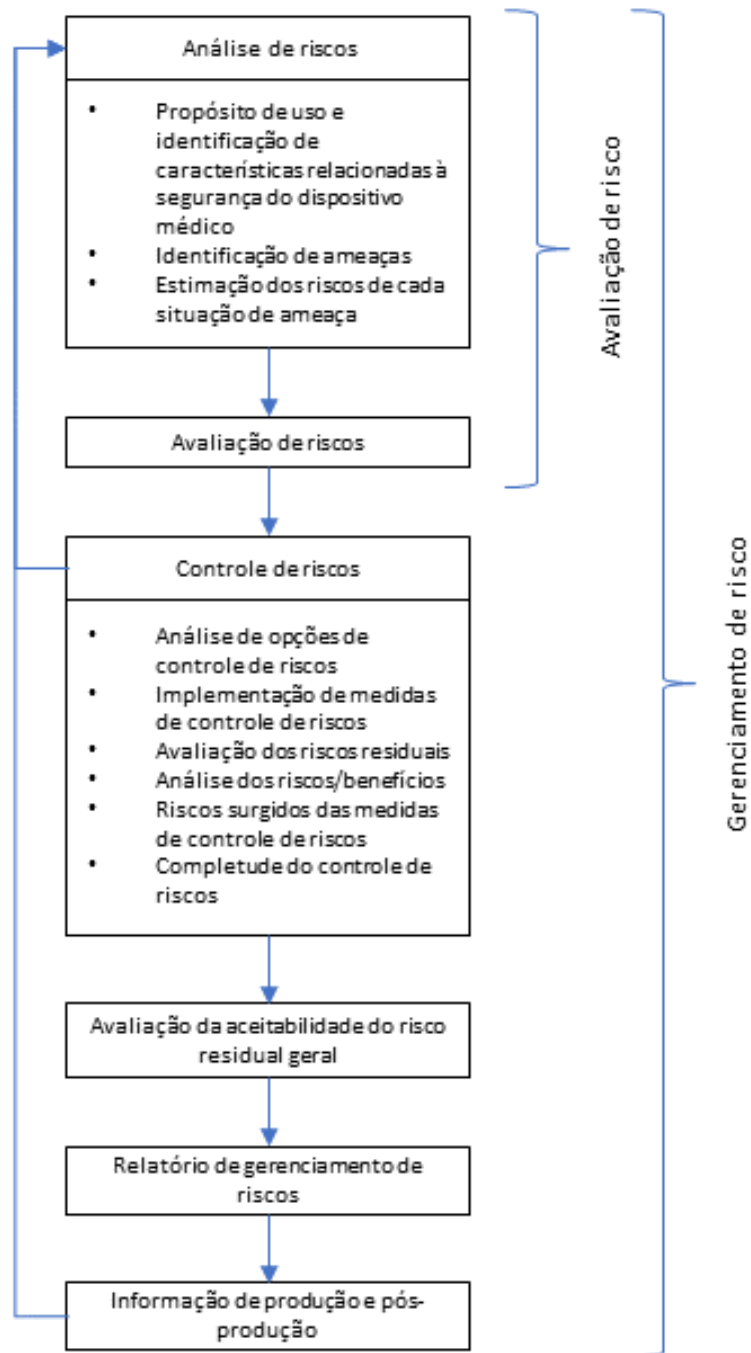


Figura 2.13: Processo de gerenciamento de risco segundo a norma ISO 14971.

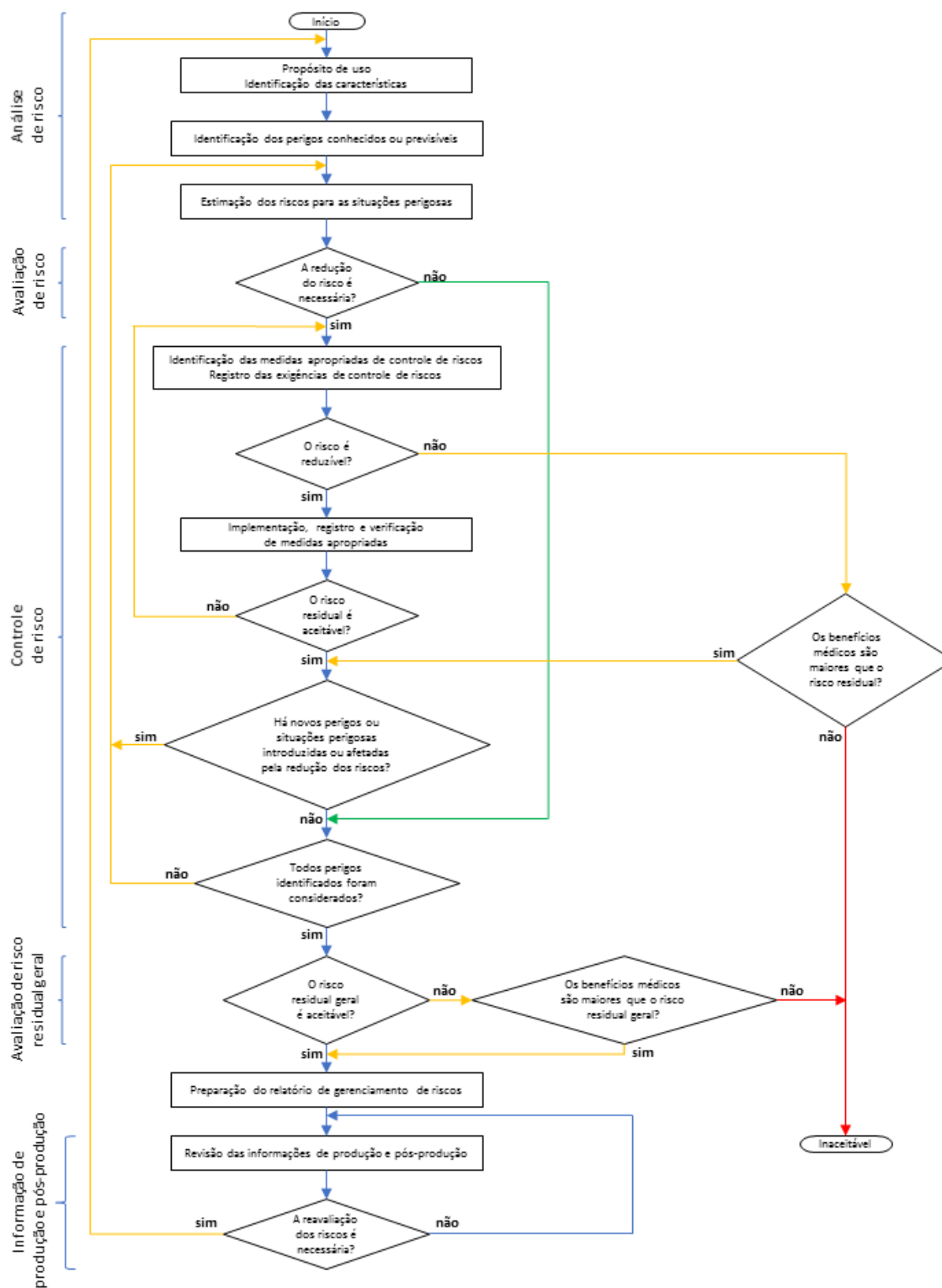


Figura 2.14: Atividades de gerenciamento de risco segundo a norma ISO 14971.

Apesar da norma em si conter recomendações gerais, seus anexos fornecem várias indicações práticas de medidas a serem tomadas em cada etapa do processo de gerenciamento de risco, incluindo: (1) perguntas que podem ser utilizadas para identificar características do dispositivo médico que impactam sua segurança, (2) formas de avaliar risco qualitativamente e quantitativamente e (3) técnicas de gerenciamento de risco.

As técnicas de gerenciamento de risco citadas são complementares, apesar de serem sempre baseadas na análise passo a passo de cadeias de eventos. Segue a descrição dada no Anexo D da norma para cada técnica:

- *Preliminary Hazard Analysis (PHA)*: técnica que pode ser usada desde o início do processo de desenvolvimento de software para identificar perigos, situações perigosas e eventos que podem causar dano quando poucos detalhes do dispositivo médico são conhecidos.
- *Fault Tree Analysis (FTA)*: técnica especialmente útil para a engenharia de segurança, nos estágios iniciais de desenvolvimento, para identificação e priorização de perigos e situações de perigo, bem como para a análise de eventos adversos.
- *Failure Mode and Effects Analysis (FMEA)* e *Failure Mode, Effects and Criticality Analysis (FMECA)*: técnicas em que efeitos ou consequências de componentes individuais são identificados sistematicamente, mais apropriada conforme o projeto se consolida.
- *Hazard and Operability Study (HAZOP)* e *Hazard Analysis and Critical Control Point (HACCP)*: técnicas tipicamente usadas nos estágios finais da etapa de desenvolvimento para verificar e então otimizar conceitos ou mudanças do projeto.

2.2.4 ISO 13485: Sistemas de gestão da qualidade de dispositivos médicos

A norma técnica ISO 13485 especifica requisitos para um sistema de gestão da qualidade em que uma organização precisa demonstrar sua habilidade de fornecer dispositivos médicos e serviços relacionados que consistentemente cumprem os requisitos do consumidor e das regulamentações aplicáveis [40]. A norma surgiu como uma especialização da largamente disseminada ISO 9001 para ser aplicada em processo regulatórios de dispositivos médicos, incluindo requisitos específicos para organizações envolvidas no ciclo de vida destes produtos e excluindo algumas orientações que não se aplicam a processos regulatórios

Segundo as listas de normas técnicas adotadas pelos membros do IMDRF, compiladas em 2014, o grau de reconhecimento da ISO 13485 era variado: no Brasil a norma foi internalizada em 2008 pela ABNT e era reconhecida, porém não obrigatória [30]; na UE a norma era parcialmente reconhecida, sendo necessário ainda a adequação a exigências específicas da MDD [41]; e nos EUA a norma não era reconhecida [42].

Sua versão revisada, publicada em 2016, deve mudar esse panorama. As maiores mudanças observadas na ISO 13485:2016 em relação à ISO 13485:2003 incluem [43]:

- A incorporação de abordagens baseadas em risco além da realização de produtos. Risco é considerado no contexto da segurança e desempenho do dispositivo médico e na adequação a requisitos regulatórios;
- Maior alinhamento com requisitos regulatórios, especialmente quanto à documentação exigida;
- Aplicação a organizações em todo ciclo de vida e cadeia de suprimentos de dispositivos médicos;

- Harmonização dos requisitos de validação de software para diferentes aplicações de software (software de gestão de qualidade, software de controle de processos, software para medição e monitoramento) em cláusulas diferentes da norma;
- Requisitos adicionais no projeto e desenvolvimento com relação a usabilidade, uso de normas técnicas, planejamento da verificação e validação, transição do projeto à etapa de fabricação e registros do projeto;
- Ênfase no tratamento de reclamações e sua comunicação às autoridades reguladoras, de acordo com os requisitos regulatórios e levando em consideração a fiscalização pós-comercialização;
- Planejamento e documentação de ações corretivas e preventivas, além da implementação de ações corretivas sem atrasos indevidos.

A norma ISO 13485 se alinha, em muitos aspectos, com os requisitos de boas práticas de fabricação das diferentes autoridades reguladoras. Por isso, em 2014, foi iniciado um programa piloto de auditoria unificado dentro do IMDRF, chamado de Medical Device Single Audit Program (MDSAP), que inclui a análise de conformidade do fabricante auditado às diretrizes da norma. Atualmente fazem parte dessa iniciativa Austrália, Brasil, Canadá, EUA e Japão [44].

O projeto piloto, que começa a ser implementado em 2017, visa permitir que fabricantes de produtos para saúde contratem um organismo auditor, autorizado no âmbito do piloto, para realizar uma auditoria única que irá contemplar os requisitos relevantes das autoridades reguladoras participantes [45]. Ou seja, além de serem avaliados aspectos relacionados à ISO 13485, também serão incluídos critérios específicos das BPFs da Anvisa (RDC nº 16/13), da Quality System Regulation da FDA (21 CFR Part 820) e diretivas similares de cada autoridade competente participante.

A regulamentação relacionada à gestão de qualidade de dispositivos médicos pode cobrir os métodos, instalações e controles usados pelo fabricante no projeto, fabricação, acondicionamento, marcação, estoque, instalação e manutenção destes produtos. A principal vantagem da implantação de sistemas de qualidade é que eles representam uma abordagem preventiva para assegurar a qualidade de dispositivos médicos, ao invés da abordagem reativa de inspeção e rejeição no fim da linha de produção [28, p.14].

2.3 ENQUADRAMENTO DE SISTEMAS MECATRÔNICOS

A classe de dispositivos estudada neste trabalho (e.g. exoesqueletos, cadeiras de rodas motorizadas, triciclos com eletroestimulação, próteses robóticas) se enquadram, em sua maioria, na classe II do sistema brasileiro, por se tratarem de produtos médicos ativos ou com função de medição, não-invasivos, mas que apresentam riscos consideráveis para seres humanos. Aplicam-se com maior frequência a esses dispositivos a regra 9 (produtos terapêuticos ativos pretendidos para administrar ou trocar energia com o corpo humano) e regra 10 (produtos ativos para diagnóstico).

Robôs de cirurgia são considerados produtos médicos invasivos cirurgicamente e, portanto, são aplicadas regras mais restritivas a eles. Aplicam-se nesse caso a regra 6 (produtos cirurgicamente invasivos para uso transitório) e a regra 7 (produtos cirurgicamente invasivos para uso de curto prazo), fazendo com que esses sejam enquadrados na classe III do sistema brasileiro.

Por apresentarem um risco relativamente menor que equipamentos de radioterapia ou dispositivos implanta-

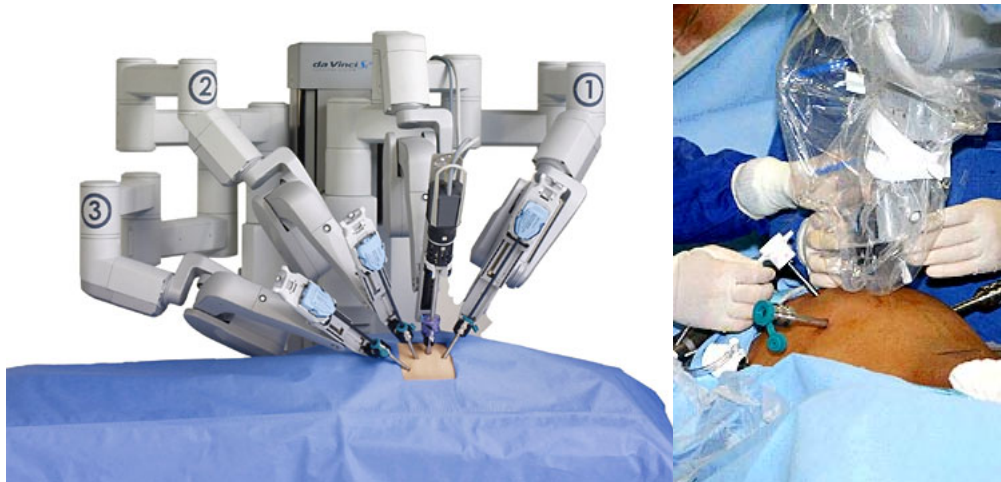


Figura 2.15: Robô de cirurgia Da Vinci, utilizado pela primeira vez no Brasil em 2008.

dos, observa-se um grau de leniência maior de autoridades competentes na liberação de sistemas mecatrônicos. Um exemplo disso é a aprovação pela FDA do robô de cirurgia *Da Vinci*, da empresa *Intuitive Surgical*, retratado na Figura 2.15 em imagem de divulgação e na sua primeira cirurgia no Brasil, realizada em 2008 no Hospital Sírio-Libanês em São Paulo [46].

Nos EUA, aproximadamente 1700 robôs *Da Vinci* estão fazendo cirurgias diariamente [47]. Sua liberação se deu por meio de uma notificação pré-comercialização 510(k) em 2000 [48], em que a empresa apresentou o robô como sendo substancialmente equivalente a equipamentos de laparoscopia existentes, o que fez com que os processos subsequentes de aprovação durassem cerca de três meses, em vez de um ano ou mais — como é comum nos casos em que é exigida uma avaliação completa para aprovação pré-comercialização [49].

Múltiplos processos subsequentes se referiram ao robô originalmente liberado para ampliar seu escopo de atuação, embora relatórios adversos relacionados a suas cirurgias tenham crescido com seu uso. Em 2012, o equipamento foi usado em quase 370 mil cirurgias nos EUA, um número três vezes maior do que em 2008 [50].

Em 2013, a FDA recebeu 3697 notificações da *Intuitive Surgical* e de hospitais que utilizam o robô *Da Vinci*, envolvendo mortes, lesões ou avarias ligadas a procedimentos cirúrgicos com o robô. Esse número mais que dobrou em um ano: em 2012, foram recebidas 1595 notificações desse tipo [51]. Além disso, foram registrados 126 recalls na FDA por diversos motivos, incluindo sete devido a falhas de software [50].

Esse cenário levou a FDA a realizar uma pesquisa com alguns usuários experientes do sistema *Da Vinci* [52], em que foram relatados benefícios e adversidades observados. Nesse relatório, foram reportadas falhas como (1) uma fissura em um dos braços robóticos causou a falha de um instrumento, (2) um dos braços robóticos precisou de manutenção após uma falha de leitura de um instrumento, (3) falha na função de memória, em que configurações de calibração foram perdidas durante trocas de instrumentos e (4) colisões excessivas entre braços durante alguns procedimentos cirúrgicos. Apesar dessas falhas, não houve qualquer sanção da FDA ou mudança de critério para esse tipo de dispositivo.

Na América Latina, estima-se a existência de cerca de trinta robôs *Da Vinci*, sendo doze no Brasil [47]. Em 2007, ele foi classificado pela Anvisa como sistema para cirurgia endoscópica de classe III [53], mantendo sua classificação em 2016/2017 em processos subsequentes de renovação do registro para modelos atualizados [54, 55]. Apesar dos relatos adversos nos EUA, de acordo com o distribuidor do robô no Brasil, não há registro

de complicações graves que tenham causado a morte de pacientes nas 2,5 mil cirurgias robóticas realizadas no país entre 2008 e 2012 [50]— um volume de procedimentos ainda muito inferior ao observado nos EUA.

Na literatura científica em sistemas mecatrônicos existem iniciativas que buscam realizar o modelamento matemático e simulação de equipamentos médicos complexos, incluindo o software no modelo, com o objetivo de realizar verificações formais do sistema. Em [56], propõe-se o uso de um “modelo em tempo-real de coração virtual”, incorporando a operação eletrofisiológica de um coração saudável e com diversas formas de arritmia, para verificação de marca-passos. Ao se extrair as propriedades de temporização do coração e do marca-passos, foi possível construir um modelo de autômato temporizado para a verificação do sistema combinado como um sistema de controle em malha fechada. Em [57], utiliza-se um modelo de rede de Petri colorida de um dispositivo de eletrocardiografia para explorar propriedades relacionadas a segurança e eficácia do software. Em [58], propõe-se uma abordagem para verificação em simulação de propriedades de dispositivos médicos em malha fechada, mesclando um modelo dinâmico farmacocinético do paciente (para a administração intravenosa de medicamentos anestésicos) com verificações em um modelo de autômato temporizado para uma bomba de analgesia controlada pelo paciente.

Embora estes estudos sejam relevantes e ampliem o ferramental disponível para verificação da segurança e eficácia de sistemas mecatrônicos, estes aspectos não serão abordados neste trabalho. Seguindo o espírito da IEC 62304, este trabalho se concentrará em aspectos que contribuem para o desenvolvimento de software seguro e eficaz, e não em formas de testar e validar software que já foi desenvolvido.

3

EMA TRIKE

Este capítulo tem como objetivo descrever o projeto da *EMA Trike*. Trata-se de um triciclo adaptado para ciclistas com lesão medular utilizando estimulação elétrica funcional, visto na Figura 3.1, que servirá como objeto de análise e estudo dentro da temática apresentada nesta dissertação.

A seção 3.1 apresenta detalhes relativos ao projeto conceitual do dispositivo, incluindo uma visão geral do seu funcionamento, a escolha de seus componentes e como estes se inter-relacionam para operar conforme desejado. A seção 3.2 descreve a estrutura mecânica do triciclo que serviu como base para o desenvolvimento. A seção 3.3 lista características técnicas dos sensores utilizados no projeto. A seção 3.4 detalha o funcionamento do eletro-estimulador escolhido. A seção 3.5 apresenta a estrutura do sistema embarcado utilizado para integração de todos componentes.



Figura 3.1: *EMA Trike* com o piloto Estevão Lopes no *Cyathlon 2016*.

3.1 DESCRIÇÃO

A *EMA Trike* é um triciclo sem atuadores eletromecânicos, cujo movimento é induzido pelo movimento dos membros inferiores de um ciclista com lesão medular através de estimulação elétrica. Essencialmente adaptou-se um modelo comercial com eletrônica embarcada que permite medir diversas variáveis de interesse e enviar estímulos elétricos aos nervos da perna do ciclista, provocando o movimento cadenciado característico

dessa modalidade esportiva através de um sistema de controle.

A Figura 3.2 ilustra e categoriza os principais elementos desse dispositivo, enquanto a Figura 3.3 apresenta como esses elementos se relacionam em termos de entradas/saídas com um diagrama de blocos.

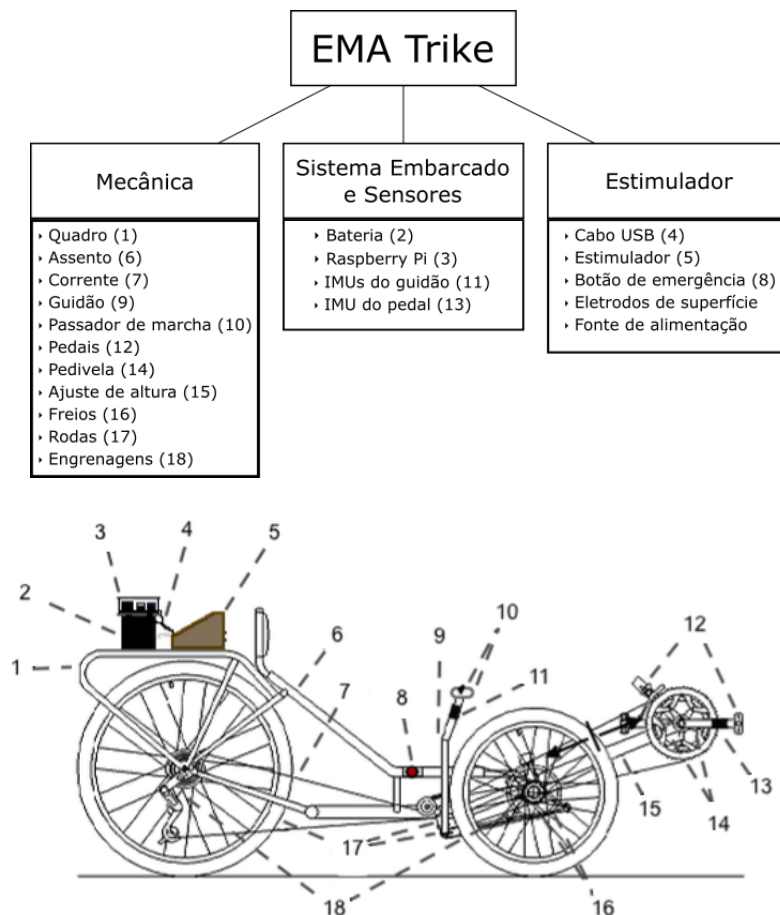


Figura 3.2: *EMA Trike* com principais elementos rotulados.

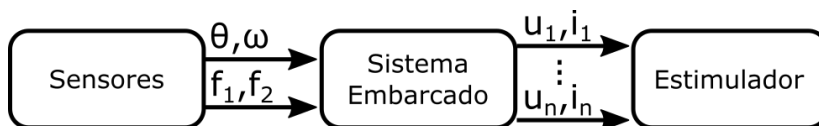


Figura 3.3: Diagrama de blocos da *EMA Trike*.

3.2 MECÂNICA

É utilizada como estrutura mecânica básica uma versão ligeiramente modificada da trike *Tadpole HP3 20 X 26 CS*, comercializada pela empresa *HP3 Trikes*. A Figura 3.4 mostra o triciclo original, antes da incorporação de sensores, estimulador e sistema embarcado.



Figura 3.4: Trike *Tadpole HP3 20 X 26 CS (HP3 Trikes)*.

Trata-se de um modelo recumbente reverso com sistema de direção de barras cruzadas (geometria *Ackermann* modificada), de modo que duas rodas dianteiras são utilizadas para guiagem e a roda traseira é tracionada. Seu quadro é fabricado com tubos de aço carbono através do processo de soldagem TIG, finalizado com pintura automotiva. O guidão é ajustável na largura e na inclinação. A distância do pedal pode ser adaptada para ciclistas de alturas diferentes, usando um mecanismo de ajuste telescópico.

Uma estrutura customizada de aço inoxidável foi fabricada para fixar os pés do piloto ao pedal de forma segura, mantendo o calcanhar perpendicular à superfície de contato. No equipamento estão presentes 8 pinhões e 3 coroas, resultando em 24 possíveis combinações de marcha. A Tabela 3.1 apresenta algumas características técnicas dos elementos mecânicos da *EMA Trike*.

3.3 YOST LABS 3-SPACE SENSOR

O sistema de referência de atitude e rumo *3-Space Sensor* da *Yost Labs* é um dispositivo que integra giroscópio, acelerômetro e magnetômetro triaxiais usando processamento embarcado, fornecendo informações de orientação espacial e velocidade angular em tempo real. Esse sensor é disponibilizado com uma ampla gama de meios de comunicação, incluindo USB, wireless (usando o protocolo ZigBee/IEEE 802.15.4), serial e SPI. A Figura 3.5 mostra algumas variedades do sensor, incluindo uma moeda para referência de tamanho.

Apesar das funcionalidades disponíveis no sensor excederem em muito as necessidades inicialmente levantadas para a *EMA Trike* (medição da posição e velocidade angular do pedivela), optou-se por utilizar o *3-Space Sensor* por sua disponibilidade no laboratório e por não requerer uma instalação no próprio eixo do pedivela. Isso permitiu que sua integração fosse facilitada e que o sensor pudesse ser utilizado em outros projetos durante o processo de desenvolvimento da *EMA Trike*.

Tabela 3.1: Especificações de elementos mecânicos da *EMA Trike*.

Peso aproximado	18 kg
Largura	900 mm
Altura	750 mm
Comprimento	1980 mm
Distância entre eixos (longitudinal)	1040 mm
Distância mínima do solo	110 mm
Ajuste da distância do pedal	Telescópico, amplitude de regulagem 150 mm
Pedivela	MTB alumínio, 170 mm
Roda traseira	Aro 26
Rodas dianteiras	Aro 20
Freios dianteiros	Mecânicos a disco, Shimano, 160 mm
Raio mínimo de curva	2,3 m
Peso máximo do ciclista	100 kgf
Aros dianteiros	Alumínio, tipo aero, 20 - 406; 36 furos
Cubos dianteiros	Alumínio para freio a disco
Pneus dianteiros	20" x 1.75 (Levorin)
Aro traseiro	Alumínio, tipo aero, 26 – 559; 36 furos
Cubo traseiro	Alumínio Shimano Paralax, 36 furos
Pneu traseiro	26" x 1.90 (Levorin)
Passador marcha dianteiro	Shimano Rapid Fire (3 velocidades)
Descarrilador dianteiro	Shimano Alivio
Passador marcha traseiro	Shimano Rapid Fire (8 velocidades)
Descarrilador traseiro	Shimano Alivio (8 velocidades)
Corrente	Shimano HG Index
Conjunto de pinhões	Cassete Shimano (8 velocidades)
Conjunto de coroas	Shimano Tripla: 22; 32; 42
Distância entre pistas de rolamento	800 mm



Figura 3.5: Sistemas de referência de atitude e rumo da família *3-Space Sensor* (Yost Labs): *Embedded*, *USB/RS232*, *Bluetooth*, *Wireless 2.4 GHz DSSS*, *Data Logger* e *Wireless Dongle*.

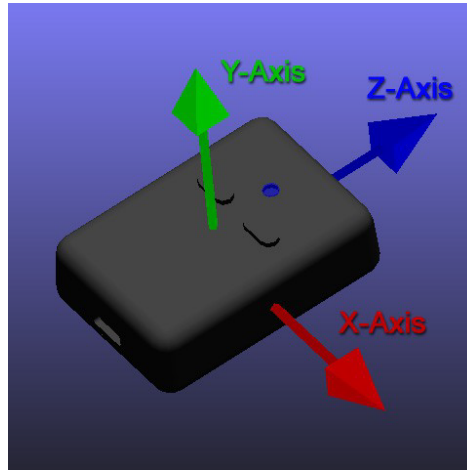


Figura 3.6: Orientação padrão do *3-Space Wireless Sensor* (reconfigurável em software).

3.3.1 Características técnicas

Como utilizou-se apenas módulos *3-Space Sensor Wireless 2.4 GHz DSSS* e *Wireless Dongle*, o escopo desta seção se limita à descrição de apenas algumas características técnicas mais relevantes destes dispositivos. Para características técnicas mais detalhadas desses e de outros sensores da família *3-Space Sensor*, consulte o *3-Space Sensor - Miniature Attitude & Heading Reference System - User Manual* [59].

Todos sensores da família *3-Space* permitem o remapeamento das atribuições e direções dos seus eixos de medição. Essa flexibilidade permite que cada usuário adapte a saída dos sensores à sua aplicação específica. A Figura 3.6 ilustra a orientação padrão do *3-Space Wireless Sensor*.

A Figura 3.7 apresenta um diagrama de blocos da versão wireless do sensor, utilizada neste trabalho. A Figura 3.8 apresenta as conexões e interfaces dos módulos *3-Space Sensor Wireless 2.4 GHz DSSS* e *Wireless Dongle*. A Tabela 3.2 apresenta características gerais do *3-Space Wireless Sensor*. A Tabela 3.3 apresenta especificações dos sensores do *3-Space Wireless Sensor*.

O LED indicador do *3-Space Wireless Sensor* tem como principal funcionalidade a sinalização do estado do dispositivo. Caso o LED esteja configurado para funcionar no modo estático, ele sempre apresentará a cor selecionada pelo usuário em configuração específica. Caso contrário, cada cor e estado do LED tem seu significado listado na Tabela 3.5. As indicações para o *3-Space Wireless Dongle* são listadas na Tabela 3.6.

3.3.2 Componentes sensores

O principal propósito do *3-Space Sensor* é fornecer uma estimativa precisa de orientação espacial. O *3-Space Sensor* disponibiliza três opções de algoritmo para estimação de orientação de atitude: filtro de Kalman, filtro complementar de quaternion (Q-COMP) e filtro de gradiente descendente de quaternion (Q-GRAD).

Estes algoritmos estimam a orientação do dispositivo por meio da combinação de dados que recebe de três tipos de sensores: um girômetro, um acelerômetro e um magnetômetro. Esta seção visa apresentar apenas uma visão geral desse processo, para mais detalhes consulte o manual [59].

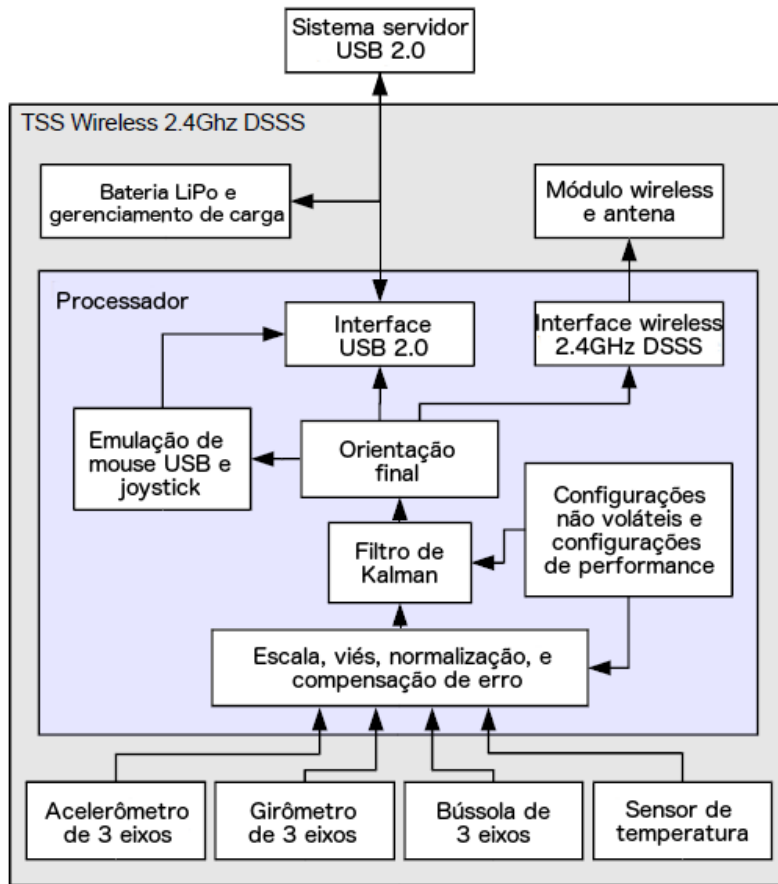


Figura 3.7: Diagrama de blocos do 3-Space Wireless Sensor.

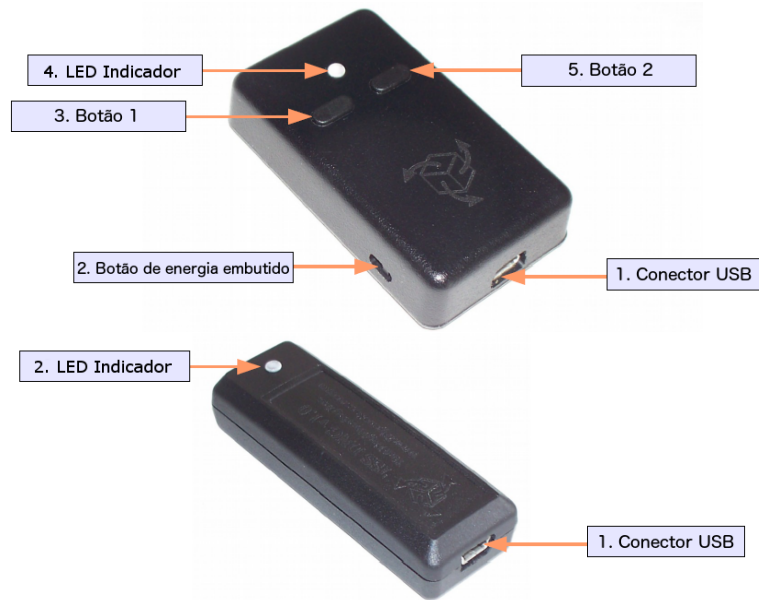


Figura 3.8: Conexões e interface do 3-Space Wireless Sensor e 3-Space Wireless Dongle.

Tabela 3.2: Especificações gerais do *3-Space Wireless Sensor*.

Dimensões	35 mm x 60 mm x 15 mm
Peso	28 gramas
Alimentação	+5V USB
Bateria	Li-Po recarregável
Duração da bateria	5 horas de uso contínuo em performance máxima
Interfaces de comunicação	USB 2.0, 2.4 GHz DSSS Wireless (certificação FCC)
Alcance de comunicação wireless	até 200 metros
Identificadores PAN wireless	65536
Canais wireless selecionáveis	16 (canais 2.4GHz na faixa 11-26)
Taxa de atualização de filtro	até 250 Hz USB / 200 Hz wireless com Kalman AHRS até 850 Hz USB / 230 Hz wireless com QCOMP AHRS até 1350 Hz USB / 260 Hz wireless em modo IMU
Saída de orientação	Quaternion absoluto e relativo, ângulos de Euler, axis angle, rotation matrix, two vector
Outras saídas	Raw sensor data, corrected sensor data, normalized sensor data, temperature
Processador	32-bit RISC funcionando a 60MHz

Tabela 3.3: Especificações dos sensores do *3-Space Wireless Sensor*.

Orientação: Range	360° em todos eixos
Orientação: Precisão	±1° em média para todas orientações em movimento
Orientação: Resolução	< 0,08°
Orientação: Repetibilidade	0,085° para todas orientações
Acelerômetro: Escala	Selecionável entre ±2g e ±8g (padrão ±2g)
Acelerômetro: Resolução	14 bit
Girômetro: Escala	Selecionável entre ±250 e ±2000 °/sec (padrão ±2000 °/sec)
Girômetro: Resolução	16 bit
Girômetro: Estabilidade do bias	2.5°/hr em média para todos eixos (temperatura: 25° C)
Magnetômetro: Escala	Selecionável entre ±0,88 Ga e ±8,1 Ga (padrão ±1.3 Ga)
Magnetômetro: Resolução	12 bit

Tabela 3.4: Especificações gerais do *3-Space Wireless Dongle*.

Dimensões	22.5 mm x 65.6 mm x 15 mm
Peso	12 gramas
Alimentação	+5V USB
Interfaces de comunicação	USB 2.0, 2.4 GHz DSSS Wireless (certificação FCC)
Alcance de comunicação wireless	até 200 metros
Sensores wireless suportados	15 simultâneos
Identificadores PAN wireless	65536
Canais wireless selecionáveis	16 (canais 2.4GHz na faixa 11-26)
Processador	32-bit RISC funcionando a 60MHz

Tabela 3.5: Situações sinalizadas pelo LED indicador do *3-Space Wireless Sensor*.

LED indicador	Carregando?	Frequência	Estado
Verde pulsante	Indiferente	1 pulso por pacote	Pacote recebido
Verde pulsante	Sim	1 Hz	Bateria completamente carregada
Amarelo pulsante	Sim	1 Hz	Bateria carregando
Vermelho pulsante	Não	↓ Carga ↑ Hz	Bateria fraca
Outras cores (padrão Azul)	Indiferente	Contínuo	Funcionamento normal ou modo estático ativado

Tabela 3.6: Situações sinalizadas pelo LED indicador do *3-Space Wireless Dongle*.

LED indicador	Frequência	Estado
Verde pulsante	1 pulso por pacote	Pacote recebido
Vermelho pulsante	1 pulso por pacote	Pacote enviado não chegou ao destino
Outras cores (padrão Azul)	Contínuo	Funcionamento normal ou modo estático ativado

3.3.2.1 Girômetro

Mede velocidade angular, sendo bastante preciso no curto prazo e excelente para a medição de movimento angulares rápidos. Uma estimativa incremental de orientação pode ser obtida através da integração das medições ao longo do tempo a partir de uma orientação de referência, porém esse processo pode levar à divergência da estimativa devido ao ruído de medição. Assim, toda estimativa de orientação obtida usando um girômetro deve ser periodicamente corrigida. Isso pode ser feito por qualquer processo que forneça uma estimativa absoluta de orientação - é comum se utilizar um conjunto com acelerômetro e magnetômetro para essa correção.

3.3.2.2 Acelerômetro

Mede forças de aceleração estáticas (como a força da gravidade) e dinâmicas (causadas pela movimentação ou vibração) agindo sobre o dispositivo. Para estimação de orientação espacial, interessa apenas a obtenção da direção do campo gravitacional, já que se trata de uma referência absoluta que pode ser explorada. Assim, quando o sensor está em movimento, a presença de forças de aceleração dinâmicas degrada sua estimativa.

Mesmo que o sensor esteja estático, apenas um vetor de referência não é suficiente para determinar por completo a orientação espacial de um corpo. Usando apenas o vetor gravidade como referência, consegue-se determinar o ângulo de rolagem (*roll*) e arfagem (*pitch*), mas não o de ângulo de guinada (*yaw*) [60]. Por isso, frequentemente se utiliza um magnetômetro para complementar o acelerômetro.

3.3.2.3 Magnetômetro

Esse sensor mede a direção e magnitude do campo magnético local. Na ausência de materiais ferrosos ou objetos magnéticos, o campo magnético local coincide com campo magnético terrestre, que é aproximadamente constante e perpendicular ao campo gravitacional. A composição desses dois vetores possibilita a obtenção de uma referência absoluta de orientação, usando algoritmos como o TRIAD [61] proposto por Harold Black.

Em suma, quase todo algoritmo de estimação de orientação espacial que utilize esses três sensores trabalha alternando/compondo a estimativa incremental obtida pelo girômetro, que está sujeita a divergência pela integração do ruído de medição, e a estimativa absoluta obtida pela composição do acelerômetro/magnetômetro, que está sujeita a interferência pelo movimento do sensor e alterações no campo magnético local.

3.3.2.4 Calibração e configuração

Os dados brutos de sensores inerciais raramente são imediatamente utilizáveis. Devido a variações no seu processo de fabricação, surgem vieses, fatores de escala e interferência entre eixos. Para corrigir esses efeitos, realiza-se um processo de calibração. O *3-Space Sensor* tem parâmetros de calibração padrão de fábrica, que ficam armazenados na memória não-volátil, mas que podem ser alterados através de comando específico.

Além da estimativa de orientação, o *3-Space Sensor* pode fornecer dados brutos, corrigidos e normalizados dos seus componentes sensores, o estado dos botões de entrada e a temperatura do dispositivo. O processo de normalização fornece os dados do acelerômetro e magnetômetro como vetores unitários, mantendo apenas a direção das grandezas medidas.

Existem várias formas de representar orientação espacial, muitas delas disponíveis no *3-Space Sensor*: quaternions, ângulos de Euler, matriz de rotação, eixo-ângulo, entre outros. Ao usar este sensor, é importante estar ciente das limitações de cada representação: efeitos como o *gimbal lock*, podem comprometer a estimativa fornecida inadvertidamente, mesmo sem qualquer erro de medição.

Por fim, é preciso definir qual orientação corresponde ao ângulo zero em todos eixos para se determinar a orientação do dispositivo. É possível estabelecer esse valor de tara para o sensor no *3-Space Sensor*. Outros parâmetros de configuração avançados estão disponíveis, como *offset orientation*, *oversampling*, *running average* e *trust values*, entre outros.

3.3.3 Modos de comunicação

O *3-Space Wireless Sensor* permite a configuração de alguns aspectos da transmissão de dados para o sistema embarcado:

- **Wireless/Wired:** No modo *wireless*, o *3-Space Wireless Sensor* transmite seus dados para o receptor através do *3-Space Wireless Dongle*, que deve ser conectado via USB no computador. Ele também pode ser conectado diretamente no modo *wired*, caso a conexão com fio não seja um empecilho, para uma comunicação mais ágil e menos suscetível a erros de transmissão. Cada modalidade de comunicação adota um protocolo diferente.
- **Polling/Streaming:** No modo *polling*, o receptor faz uma requisição por dados, que é respondida em seguida. No modo *streaming*, o receptor configura até oito *slots* de comunicação, para que o sensor transmita continuamente esses dados em uma frequência determinada. Isso permite a transmissão mais eficiente dos dados, caso não haja variação significativa nos dados desejados. Mesmo que o modo *streaming* esteja ativado, ainda é possível usar o modo *polling* para fazer requisições.
- **ASCII/Binary:** O *3-Space Sensor* aceita comandos ASCII para que eles possam ser escritos e compreendidos por um humano, em um terminal serial. Porém o modo binário comprime mais os dados e permite uma transmissão mais rápida e com menos bytes.

Em geral, a transmissão *wireless* de dados binários em modo *streaming* é a que traz os maiores benefícios em termos de flexibilidade e desempenho. O uso do modo *polling* é interessante apenas para configuração ou obtenção de dados eventuais de forma assíncrona, enquanto a transmissão *wired* só deve ser usada se as limitações de mobilidade não inviabilizem a aplicação em questão. O uso do protocolo ASCII é recomendado apenas em situações de depuração do sistema, quando o desenvolvedor precisa rapidamente identificar as mensagens enviadas e o retorno dado pelo *3-Space Sensor*

3.3.4 Protocolo 3-Space Sensor

Como mencionado na subseção anterior, o *3-Space Wireless Sensor* permite vários diferentes modalidades de comunicação. Cada opção feita pelo desenvolvedor — *wired* ou *wireless*, *polling* ou *streaming*, ASCII ou binário — tem suas particularidades, de modo que apenas a descrição do protocolo demanda nove páginas do manual, somando-se a mais catorze páginas descrevendo cada comando possível. Considerando esse nível de detalhamento injustificado para esse trabalho, nesta seção são descritas apenas as informações básicas da combinação *wireless*, *polling* e binário.

A Figura 3.9 apresenta o formato do pacote de dados binário no modo *wireless* no protocolo *3-Space Sensor*. Todos pacotes se iniciam com um byte que marca o início da transmissão (0xF8), seguido pelo identificador lógico do sensor conforme registrado no *3-Space Wireless Dongle* associado e o comando escolhido dentre as mais de 90 mensagens disponíveis no protocolo¹. Após o comando, segue um trecho de tamanho variável, que contém seus parâmetros — esse tamanho é fixo para cada comando, sendo especificado como *DataLen* na listagem de comandos. Por fim, adiciona-se um byte de *checksum* para detecção de erros de transmissão, correspondente à soma módulo 8 de todos outros bytes do pacote.

A Figura 3.10 apresenta o formato da resposta binária no modo *wireless* no protocolo *3-Space Sensor*. O primeiro byte indica se o comando foi recebido e processado corretamente, indicando a falha através de um valor não-nulo. Algumas coisas que podem causar uma falha incluem (1) a ausência de um sensor *wireless* correspondente no endereço especificado, (2) erros de comunicação *wireless* ou perda de pacotes, (3) formatação inadequada do comando, quanto ao seu tamanho ou estrutura. O segundo byte indica qual sensor enviou o pacote de resposta e o terceiro indica o tamanho do restante do pacote, que varia de acordo com o comando que foi enviado previamente. Note que não há menção ao comando que está sendo respondido — é responsabilidade do desenvolvedor fazer a sincronização entre comandos e respostas.

A Tabela 3.7 traz alguns exemplos de comandos *3-Space Sensor* e suas possíveis respostas. Para mais informações, consulte o manual do *3-Space Sensor* [59].

3.4 ESTIMULADOR HASOMED REHASTIM

O estimulador *RehaStim* da *Hasomed* é um dispositivo de estimulação elétrica portátil que gera impulsos, simultaneamente em até oito canais, para ativar músculos paralisados através de eletrodos de superfície. Ele foi desenvolvido para aplicações de treinamento e reabilitação, com uma série de parâmetros ajustáveis individualmente para cada canal, e é certificado segundo a norma EN 60601-2-10 para equipamentos eletromédicos. A Figura 3.11 mostra o dispositivo, sem a fonte de alimentação ou eletrodos de superfície conectados.

¹A seção 4.5 do manual [59] contém a listagem de todos comandos disponíveis.

Tabela 3.7: Exemplos de comandos binários no modo *wireless* no protocolo *3-Space Sensor*.

Comando	Descrição	Resposta Possível
F8 01 00 01	Ler orientação como quaternion do sensor 1	00 01 10 00 00 00 00 00 00 00 00 00 00 00 00 3F 80 00 00
F8 05 6A 02 71	Definir taxa de amostragem como 2 para o sensor 5	00 05 00
F8 03 E6 E9	Ler string de versão do sensor 3	00 03 0C 54 53 53 57 49 52 30 36 30 31 31 31
F8 00 EC EC	Ler velocidade do <i>clock</i> do sensor 0 (desligado)	01 00 (Erro)
F8 09 77 00 00 00 00 BF 80 00 00 00 00 BF	Definir vetor de referência do acelerômetro para (0.0, -1.0, 0.0) no sensor 9	00 09 00

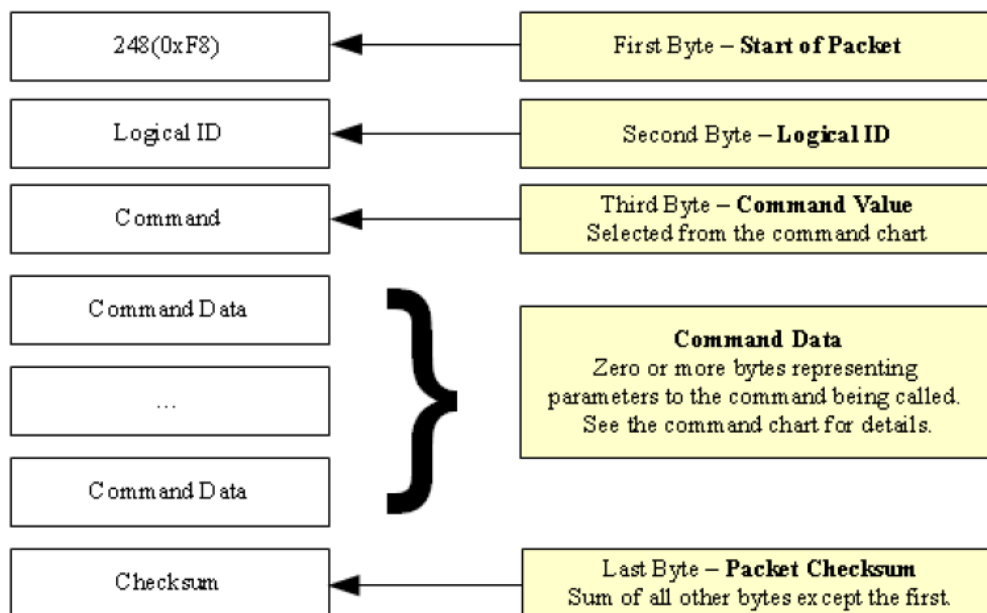


Figura 3.9: Formato do pacote de dados binário no modo *wireless* no protocolo *3-Space Sensor*.

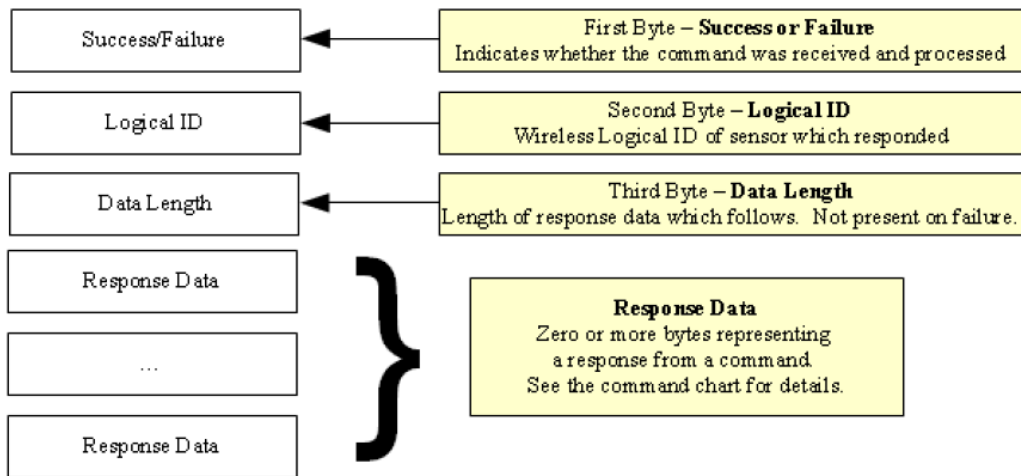


Figura 3.10: Formato do pacote de resposta binária no modo *wireless* no protocolo *3-Space Sensor*.



Figura 3.11: Estimulador *RehaStim* (Hasomed).

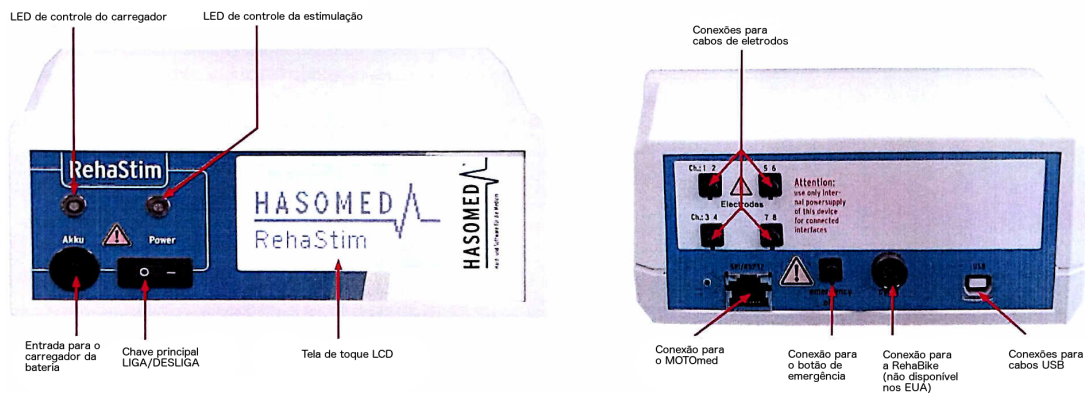


Figura 3.12: Conexões e interfaces do Estimulador *RehaStim*.

Apesar de se tratar de um equipamento comercialmente disponível para uso por pacientes e fisioterapeutas em todo tipo de aplicação de estimulação elétrica funcional, com interface gráfica e programas ajustáveis de terapia de fácil configuração, a grande vantagem deste equipamento é a presença de um modo científico. Nesse modo, é possível controlar os parâmetros de estimulação por meio de um protocolo de comunicação serial. Essa característica facilitou a integração do estimulador na *EMA Trike*.

3.4.1 Características técnicas

A operação do estimulador requer a conexão de eletrodos usando cabos específicos que agrupam até dois pares de eletrodos, na parte traseira do equipamento. Como medida de segurança, essas conexões são testadas antes do início do padrão de estimulação, para evitar que eletrodos mal posicionados causem queimaduras no usuário. Além disso, há a possibilidade de conexão de um botão de emergência para perigo inesperado e repentino. A Figura 3.12 identifica essas e outras conexões e interfaces presentes no dispositivo. A Tabela 3.8 apresenta especificações relevantes do estimulador.

O estimulador está em funcionamento quando o LED indicador de estimulação está aceso e na cor amarela. A Tabela 3.9 indica possíveis causas de falha sinalizadas por outros estados do LED indicador de estimulação. Já a Tabela 3.10 indica possíveis falhas de funcionamento sinalizadas pelo desligamento da tela LCD.

3.4.2 Formatos de onda

O sinal gerado pelo estimulador é um trem de pulsos bifásicos, cuja frequência, amplitude e largura são configuráveis. A Figura 3.13 ilustra o formato de um pulso bifásico típico, aplicado em uma carga resistiva ideal. A amplitude de corrente e largura de pulso são definidas no gráfico. Note que há um intervalo fixo de $100 \mu s$ entre as duas fases do pulso. Ao final do pulso, a carga remanescente nos eletrodos e na pele é removida por um atalho ativo — uma mudança de polaridade dos eletrodos por $1 \mu s$.

O estimulador faz um teste de resistência da pele como medida de segurança. A resistência é determinada através da análise do efeito de um pequeno sinal impulsivo de teste, que é enviado antes de cada pulso de estimulação. Se a resistência não estiver em faixas normais, então o pulso de estimulação não será gerado.

Tabela 3.8: Especificações do Estimulador *RehaStim*.

Largura de pulso de estimulação	20 μ s a 500 μ s, em incrementos de 10 μ s
Corrente	20 mA a 130 mA, em incrementos de 5 mA
Número de canais	2 módulos com 4 canais cada, funcionando constantemente em paralelo
Frequência de estimulação	10 a 50 Hz em passos de 5 Hz
Reserva de tensão elétrica	150 V
Tempo de operação	de 2 a 2,5 horas com padrões estimulação intermediários (carga de 1100 Ω e 100 nF)
Tempo de carga para bateria integrada	cerca de 3,5 horas
Comprimento	13,5 cm
Largura	15 cm
Altura	7 cm
Peso	5 kg
Isolamento de corrente de linha	TR 30RAM090 EN60601-1 SANYO, NiMh, C= 2700 mAh
Alimentação	100-240 VAC 50-60 Hz
Potência de entrada	max. 150W
Temperatura de uso	0 °C a 40 °C
Display/Interface	LCD sensível ao toque
Comunicação	USB / RS232
Sistema Operacional	software customizado
Tensão máxima de saída	154 V
Número máximo de canais	8
Corrente de saída por canal	20 - 130 mA, em incrementos de 5 mA
Forma de onda de saída	Bifásica, com carga balanceada
Grau de proteção	Parte aplicada do tipo BF

Tabela 3.9: Situações de falha sinalizadas pelo LED indicador do *RehaStim*.

LED indicador	Estado	Possível causa	Reparo
Desligado .	Estimulação em stand-by ou inativa		Desligue e reinicie o dispositivo
Vermelho pulsante	Estimulação interrompida	Falha de operação: eletrodo caiu ou não foi conectado corretamente	Verifique a conexão dos eletrodos e seleção de canais.
Vermelho contínuo	Estimulação não iniciada	Falha de sistema: falha de comunicação, erro no módulo de estimulação	Desligue e reinicie o dispositivo.

Tabela 3.10: Situações de falha sinalizadas pela tela LCD do *RehaStim*.

Tela LCD	Estado	Possível causa	Reparo
Luz fraca	A tela não está acesa mas os botões estão ligeiramente visíveis	A iluminação da tela é desligada automaticamente depois de um período determinado.	Toque em uma área da tela em que há um botão.
Escura	Desligamento automático	Erro no autodiagnóstico (falha no módulo de estimulação)	Desligue e reinicie o dispositivo.
Escura	Desligamento automático	Bateria descarregou	Recarregue a bateria, conectando o carregador fornecido
Escura	Desligamento automático	Botão de emergência foi pressionado	Gire o botão de emergência para desarmá-lo. Desligue e reinicie o dispositivo.

Existem três modos de estimulação disponíveis: modo de pulso único (*single_pulse*), modo contínuo de lista de canais (*ccl*) e modo de ciclo único de lista de canais (*oscl*). Os modos *single_pulse* e *oscl* permitem (e requerem) que um dispositivo externo envie comandos continuamente para manter a estimulação. Já o modo *ccl* realiza o controle da geração de pulsos por conta própria, seguindo uma frequência definida, o que facilita a aplicação de padrões complexos de estimulação. A seguir é feita uma descrição breve de cada modo.

3.4.2.1 Modo de Pulso Único

Ao receber um comando externo, o estimulador gera um pulso único em um canal específico, com a amplitude de corrente e largura de pulso desejados. O estimulador gera o pulso imediatamente após o processamento do comando. Padrões complexos de estimulação podem ser gerados através do envio de mais de um comando, fazendo com que um dispositivo externo seja responsável pelo controle do intervalo entre pulsos e demais aspectos de temporização da estimulação.

3.4.2.2 Modo Contínuo de Lista de Canais

Este modo possibilita o controle simultâneo de vários canais, para geração de um pulso em cada um dos canais ativos. Antes de ser executado, é necessário enviar um comando de inicialização, definindo: (1) quais canais devem ser ativados, (2) quais desses canais devem ser executados em baixa frequência, (3) o período de estimulação, (4) o intervalo entre pulsos referente aos canais em modo *doublet* ou *triplet* e (5) o fator

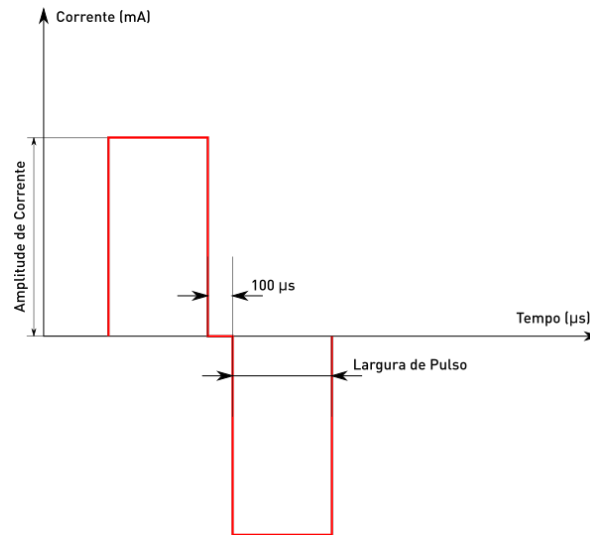


Figura 3.13: Formato do pulso bifásico gerado pelo estimulador.

multiplicativo que regula o período dos canais de baixa frequência. Após a inicialização, devem ser enviados comandos de atualização quando o modo, largura de pulso ou amplitude de corrente de um dos canais ativos precisar ser alterado².

Internamente o estimulador mantém um controlador principal, que gerencia a interface gráfica e decodifica os comandos recebidos, e dois módulos de estimulação, que regulam a temporização de canais associados. O módulo A controla os canais 1-4 e o módulo B controla os canais 5-8, atuando de forma simultânea.

Para que o controlador principal possa enviar essas atualizações para os módulos de estimulação, é reservado um período de $0,6\text{ ms}$ no início de cada ciclo principal de estimulação para comunicação. Os pulsos de estimulação são gerados sequencialmente nos canais ativos de forma compacta, isto é, não há janelas de ativação fixas para cada canal. Assim, não são introduzidos atrasos desnecessários referentes a canais inativos³.

A Figura 3.14 ilustra todas as nuances do funcionamento do estimulador em modo contínuo de lista de canais, descritas nos parágrafos anteriores. A lista de canais ativos inclui os canais 1, 3, 5 e 6. Os canais 3 e 5 estão no modo *doublet*, isto é, geram dois pulsos a cada período de estimulação t_1 , espaçados entre si segundo o intervalo entre pulsos t_2 . Os pulsos de estimulação são representados por barras coloridas para facilitar a visualização, mas seguem o formato de pulso bifásico apresentado na Figura 3.13. As barras cinzas representam períodos em que há comunicação entre o controlador principal e os módulos de estimulação.

3.4.2.3 Modo de Ciclo Único de Lista de Canais

O modo *oscl* é um meio termo entre o *single_pulse* e o *ccl*: ao receber um comando externo, o estimulador gera uma sequência de pulsos em uma lista de canais, com a amplitude de corrente e largura de pulso especificados para cada canal. Essa sequência, equivalente a um ciclo do modo *ccl*, é gerada imediatamente após o processamento do comando. Assim como no modo *single_pulse*, é preciso que um dispositivo externo seja responsável pelo controle do intervalo entre pulsos e demais aspectos de temporização da estimulação para a

²Todos canais são inicializados com amplitude de corrente nula, então é preciso realizar pelo menos uma atualização para que o sistema seja útil nos modos de lista de canais (*ccl* e *oscl*).

³Note que os intervalos de estimulação dos canais 1 e 3 estão adjacentes na Figura 3.14, já que o canal 2 não está ativo.

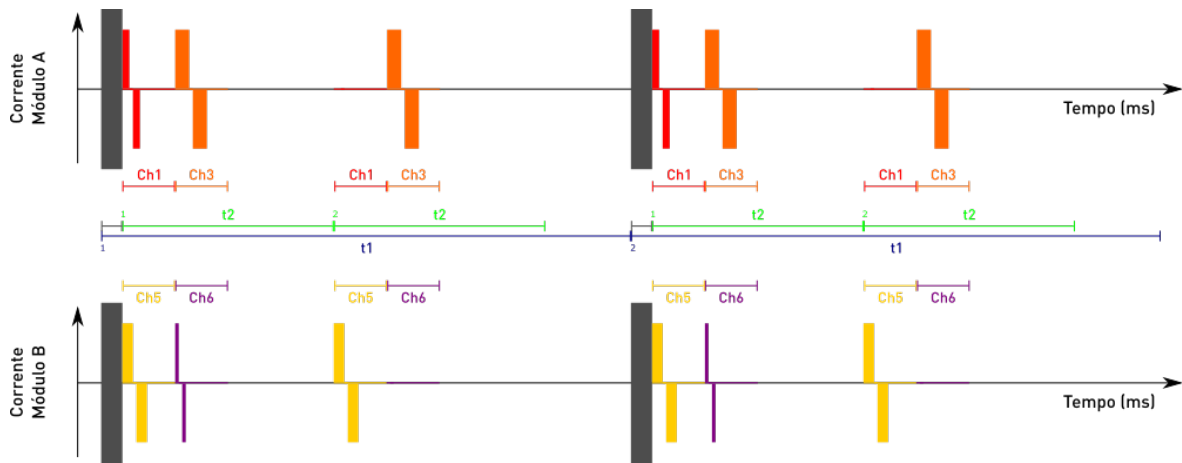


Figura 3.14: Exemplo do Modo Contínuo de Lista de Canais (*ccl*).

geração de padrões de estimulação complexos. A ativação deste modo é feita através da especificação de um período de estimulação t_1 nulo na inicialização do modo de lista de canais.

3.4.3 Operação do RehaStim

A operação usual do estimulador é feita através da seleção de programas de treinamento através da interface apresentada na tela LCD do dispositivo. Esse modo de controle é simples e útil para testes, mas sua descrição foge do escopo dessa dissertação. Para mais informações, veja o manual do estimulador [62].

O modo científico do estimulador *RehaStim* permite que um dispositivo externo conectado à sua interface USB controle diretamente os parâmetros de estimulação dos oito canais, oferecendo grande flexibilidade para aplicações de pesquisa. Para ativar o modo científico, basta seguir os passos listados na Tabela 3.11.

3.4.4 Protocolo ScienceMode

Para se comunicar com o estimulador no modo científico, é necessário que o dispositivo externo implemente o protocolo *ScienceMode*, descrito no documento “*ScienceMode - RehaStim Stimulation Device - Description and Protocol*” [63]. A interface de comunicação utilizada no protocolo *ScienceMode* tem parâmetros fixos, baseados no padrão USB 1.1 implementado com o circuito integrado de interface USB-Serial FTDI. Esses parâmetros são apresentados na Tabela 3.12 para referência.

O formato do pacote de dados no protocolo *ScienceMode*, ilustrado pela Figura 3.15, é composto por um identificador de comando, um valor de *checksum* para detecção de erros de transmissão e os dados específicos de cada comando, que variam conforme a mensagem enviada. Os identificadores dos comandos disponíveis no protocolo estão listados na Tabela 3.13.

Um fator importante a ser observado: *o primeiro bit de cada byte em um pacote de dados não é ocupado por dados do protocolo*. Esse bit é reservado para facilitar a sincronização da leitura de pacotes. Se ele tiver valor 1, trata-se do início de um novo pacote, que pode ser utilizado para identificar o comando que foi pedido. Se tiver valor 0, trata-se da continuação de um comando.

Tabela 3.11: Como ativar o *ScienceMode* do estimulador *RehaStim*.

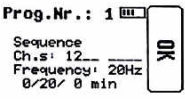
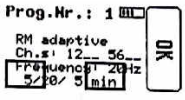



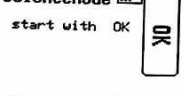

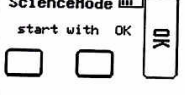
Tela LCD	Instruções
	<p><i>Ligue o estimulador</i></p> <p>Conecte o dispositivo externo pela interface USB e ligue o dispositivo. A tela inicial aparecerá, mostrando as configurações do programa de treinamento selecionado.</p>
	<p><i>Acesse as configurações de terapia</i></p> <p>Existem duas teclas escondidas na linha inferior do display LCD, indicadas com quadrados na imagem ao lado. Pressione o 1º e 3º botões, um após o outro, para acessar as configurações de terapia a partir da tela inicial.</p>
	<p><i>Aviso para usuários comuns</i></p> <p>Antes de entrar nas configurações de terapia, um aviso aparecerá para evitar que usuários comuns mexam nas configurações do estimulador por engano. Pressione OK.</p>
	<p><i>Encontre o ScienceMode na listagem de programas</i></p> <p>Ao entrar nas configurações de terapia, você verá todos programas disponíveis para edição. Pressione as setas para para navegar pela lista até encontrar a opção ScienceMode.</p>
	<p><i>Entre na tela de início do ScienceMode</i></p> <p>Ao encontrar a opção ScienceMode na listagem de programas, entre na tela de início do ScienceMode pressionando OK.</p>
	<p><i>Inicie o ScienceMode</i></p> <p>Essa é a tela de início para o ScienceMode. Clicando OK você ativa o ScienceMode e pode começar a comunicar com o RehaStim através do seu dispositivo externo.</p>
	<p><i>Interrompa o ScienceMode</i></p> <p>Quando estiver pronto para desativar o ScienceMode, pressione STOP. Você retornará para a tela de início do ScienceMode.</p>
	<p><i>Retorne para as configurações de terapia</i></p> <p>Existem duas teclas escondidas na linha inferior do display LCD, indicadas com quadrados na imagem ao lado. Pressione o 1º e 3º botões, um após o outro, para voltar para as configurações de terapia a partir da tela de início do ScienceMode.</p>

Tabela 3.12: Configurações da interface de comunicação no *ScienceMode*.

Parâmetro	Parâmetro <i>pySerial</i>	Valor <i>pySerial</i>
Taxa de Transmissão	baudrate	115200
Paridade	parity	PARITY_NONE
Bits de Dados	bytesize	EIGHTBITS
Bits de Parada	stopbits	STOPBITS_TWO
Controle de Fluxo RTS/CTS	rtscts	True



Figura 3.15: Formato do pacote de dados no protocolo *ScienceMode*.

Tabela 3.13: Identificadores dos comandos do protocolo *ScienceMode*.

Comando	Referência no ROS ⁶	Ident
Inicialização de modo de lista de canais	<i>cl_init</i>	00
Atualização de modo de lista de canais	<i>cl_update</i>	01
Interrupção de modo de lista de canais	<i>cl_stop</i>	10
Geração de pulso único	<i>single_pulse</i>	11

A descrição de cada comando é apresentada nas Tabelas 3.14-3.17, fazendo referência aos parâmetros e formatos de onda discutidos na subseção 3.4.2. As Figuras 3.16-3.19 complementam as tabelas e esclarecem como todos esses parâmetros são aglutinados no pacote, levando em consideração o bit de sincronização em cada byte e também o número de canais ativos⁴, no caso do comando *cl_update*⁵.

A Figura 3.20 e a Tabela 3.18 apresentam o formato do pacote de reconhecimento (*ack*), que diz se a mensagem foi recebida e interpretada com sucesso pelo estimulador. Esse pacote é enviado pelo estimulador como resposta a todos comandos recebidos. Situações que podem levar a um erro incluem erros de configuração da comunicação serial, cálculo incorreto do *checksum* ou o término inesperado de um comando - muitas vezes ocasionado pela montagem incorreta do pacote de dados.

3.5 SISTEMA EMBARCADO

Para processamento embarcado a *EMA Trike* conta com uma placa *Raspberry Pi 3*, comercializada pela *Pi Foundation*, apresentada na Figura 3.21. Trata-se de um computador embarcado ARMv8 com arquitetura 64-bits, contando com quatro núcleos de processamento a 1.2 GHz, 1GB de memória RAM, adaptador *wire-*

⁴O número de canais ativos, que influencia no tamanho do pacote do comando *cl_update*, é indicado pela letra *n* na Figura 3.17.

⁵É importante frisar que no comando *cl_update* os parâmetros *Mode*, *Pulse_Width* e *Pulse_Current* devem ser enviados seguindo a ordem crescente em relação ao número dos canais. Portanto, no exemplo da Figura 3.14 seriam enviados os parâmetros do canal 1, 3, 5 e 6, nesta ordem, sem a necessidade de enviar valores para os canais 2, 4, 7 e 8.

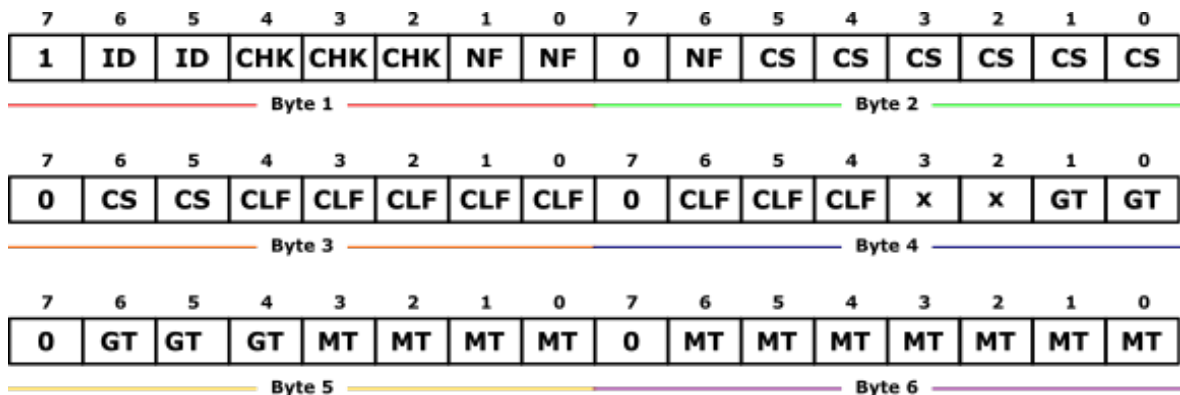


Figura 3.16: Inicialização de modo de lista de canais no protocolo *ScienceMode* (*cl_init*, *Ident* = 00).

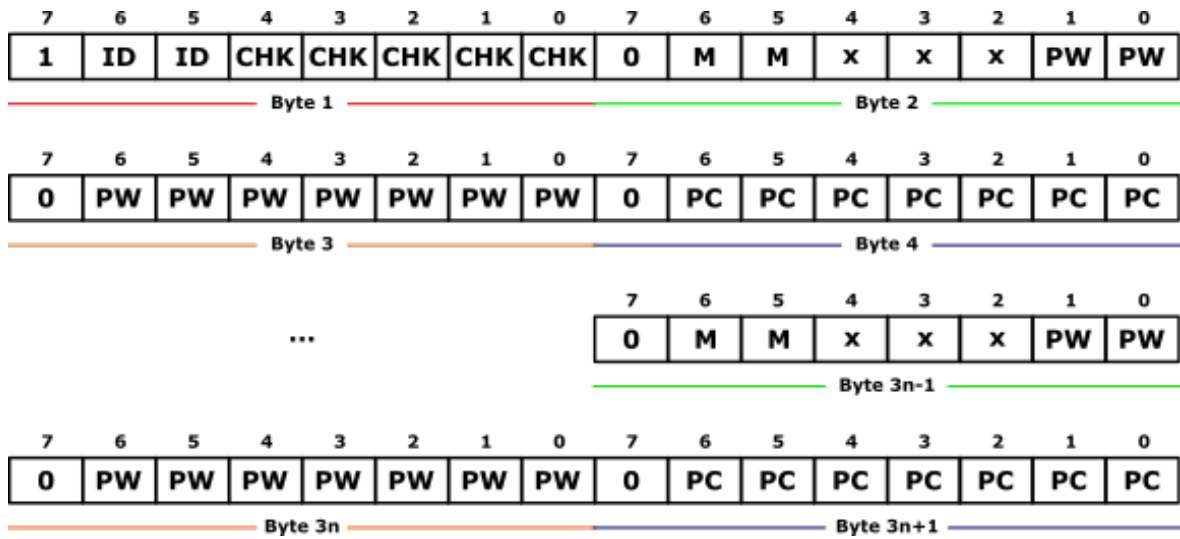


Figura 3.17: Atualização de modo de lista de canais no protocolo *ScienceMode* (*cl_update*, *Ident* = 01).

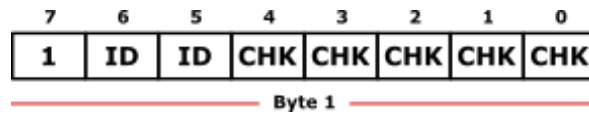


Figura 3.18: Interrupção de modo de lista de canais no protocolo *ScienceMode* (*cl_stop*, *Ident* = 10).



Figura 3.19: Geração de pulso único no protocolo *ScienceMode* (*single_pulse*, *Ident* = 11).

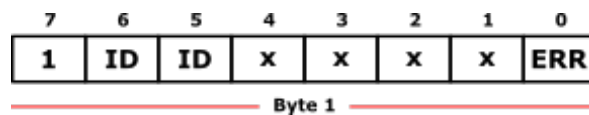


Figura 3.20: Pacote de reconhecimento no protocolo *ScienceMode* (*ack*).

Tabela 3.14: Inicialização de modo de lista de canais no protocolo *ScienceMode* (*cl_init*, *Ident* = 00).

Parâmetro	Bits	Valores	Descrição
<i>Ident</i> (ID)	2	0	Identificador do comando
<i>Check</i> (CHK)	3	0..7	Soma módulo 8 de todas variáveis lógicas: $(NF + CS + CLF + GT + MT) \bmod 8$
<i>N_Factor</i> (NF)	3	0..7	Define quantas vezes o ciclo de estimulação deve ser ignorado em canais de baixa frequência. Assim, o período de estimulação de baixa frequência é dado por $(1 + NF) \times MT$.
<i>Channel_Stim</i> (CS)	8	0..255	Define quais canais devem ser ativados, através de uma máscara de bits em que o valor 1 sinaliza um canal ativo. Cada bit corresponde a um canal, sendo o bit 0 associado ao canal 1 e os demais associados em sequência, até o bit 7 associado ao canal 8.
<i>Channel_Lf</i> (CLF)	8	0..255	Define quais canais devem funcionar em baixa frequência, com período de estimulação definido pela variável <i>N_Factor</i> . OBS: Todos canais de baixa frequência devem ser ativados na variável <i>Channel_Stim</i> .
<i>Group_Time</i> (GT)	5	0..31	Define o intervalo entre pulsos t_2 , conforme ilustrado na Figura 3.14, seguindo a equação $t_2 = \frac{1}{2}(3 + GT) \text{ ms}$.
<i>Main_Time</i> (MT)	11	0..2047	Define o período de estimulação t_1 , conforme ilustrado na Figura 3.14, seguindo a equação $t_1 = \frac{1}{2}(2 + MT) \text{ ms}$. Se $MT = 0$, o modo de ciclo único de lista de canais é ativado.

Tabela 3.15: Atualização de modo de lista de canais no protocolo *ScienceMode* (*cl_update*, *Ident* = 01).

Parâmetro	Bits	Valores	Descrição
<i>Ident</i> (ID)	2	1	Identificador do comando
<i>Check</i> (CHK)	5	0..31	Soma módulo 32 de todas variáveis lógicas: $(M + PW + PC) \bmod 32$
<i>Mode</i> ⁵ (M)	2	0: single pulse 1: doublet 2: triplet	Define a quantidade de pulsos por ciclo
<i>Pulse_Width</i> ⁵ (PW)	9	0, 10..500	Largura do pulso de corrente (μs)
<i>Pulse_Current</i> ⁵ (PC)	7	0..127	Amplitude do pulso de corrente (mA)

Tabela 3.16: Interrupção de modo de lista de canais no protocolo *ScienceMode* (*cl_stop*, *Ident* = 10).

Parâmetro	Bits	Valores	Descrição
<i>Ident</i> (ID)	2	2	Identificador do comando
<i>Check</i> (CHK)	5	0	Soma módulo 32 de todas variáveis lógicas: $0 \bmod 32 = 0$

Tabela 3.17: Geração de pulso único no protocolo *ScienceMode* (*single_pulse*, *Ident* = 11).

Parâmetro	Bits	Valores	Descrição
<i>Ident</i> (ID)	2	3	Identificador do comando
<i>Check</i> (CHK)	5	0..31	Soma módulo 32 de todas variáveis lógicas: (<i>CN</i> + <i>PW</i> + <i>PC</i>) mod 32
<i>Channel_Number</i> (CN)	3	0..7	Define o canal em que o pulso deve ser gerado. 0 valor 0 corresponde ao canal 1 e os demais são associados em sequência, até o valor 7 que corresponde ao canal 8.
<i>Pulse_Width</i> (PW)	9	0,10..500	Largura do pulso de corrente (μs)
<i>Pulse_Current</i> (PC)	7	0..127	Amplitude do pulso de corrente (<i>m.A</i>)

Tabela 3.18: Pacote de reconhecimento no protocolo *ScienceMode* (ack).

Parâmetro	Bits	Valores	Descrição
<i>Ident</i> (ID)	2	0..3	Identificador do comando recebido
<i>Error_Code</i> (ERR)	1	0: error 1: ok	Bit sinalizador de erro no reconhecimento

less 802.11n, Bluetooth 4.1 e 4 portas USB. Uma das suas maiores vantagens é a disponibilidade de diversas interfaces multimídia, GPIO e sua ampla adoção global.

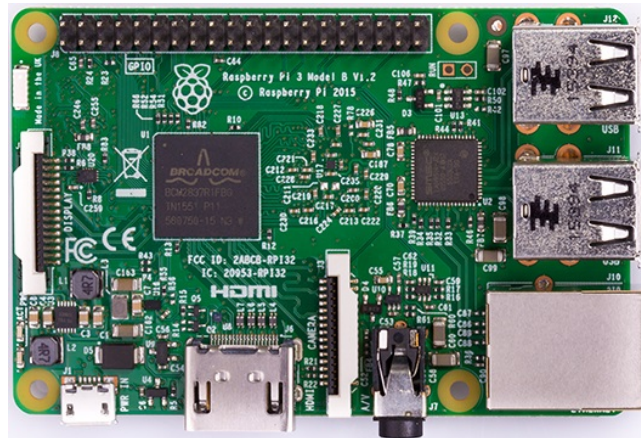


Figura 3.21: Computador embarcado Raspberry Pi 3.

Como sistema operacional adotou-se a distribuição *Raspbian Jessie*, uma variante do *Debian* adaptada para uso com a *Raspberry Pi*. Foi instalado ainda o meta-sistema operacional *ROS Indigo Igloo* a partir do código-fonte, segundo instruções específicas⁷ obtidas na *ROS Wiki*.

Para que o sistema se iniciasse assim que a *Raspberry Pi* fosse ligada, utilizou-se o serviço *cron*, disponível na maioria das distribuições Linux. Trata-se de uma ferramenta que permite programar a execução de comandos e processos, segundo condições listadas em um arquivo chamado de *crontab*. No caso, utilizou-se a condição “reboot” para executar o script principal assim que o sistema terminasse sua inicialização.

Outra particularidade do sistema embarcado utilizado é a opção feita pela execução do código-fonte a partir de um *pen drive*. Isso permitiria que o código fosse atualizado externamente ao sistema, em uma situação

⁷As instruções podem ser consultadas em: <http://wiki.ros.org/ROSBerryPi/Installing%20ROS%20Indigo%20on%20Raspberry%20Pi>

emergencial, sem que fosse necessário adicionar periféricos (monitor, teclado, mouse) ou acessar o *Raspberry Pi* remotamente (via SSH, por exemplo). Para que o *pen drive* fosse sempre montado com o mesmo caminho na estrutura de diretórios do *Raspbian*, foi necessário adicionar uma linha à sua *fstab* - um arquivo de configuração que contém informações sobre todas partições e dispositivos de armazenamento em um sistema Linux. Essa linha associa o identificador único universal do *pen drive* ao caminho desejado, além de incluir algumas informações adicionais sobre a forma como o dispositivo deve ser montado.

4

CRITÉRIOS PARA ESTRUTURAÇÃO DO SOFTWARE

Define-se refatoração de código como uma mudança feita na estrutura interna de software para torná-lo mais compreensível e menos custoso de modificar, sem alterar seu comportamento observável [64]. Além das alterações “*pequenas demais para valerem o esforço de serem feitas*” que fazem parte desse processo¹, decidiu-se realizar mudanças mais profundas no código da *EMA Trike*.

Levando em consideração que no início deste trabalho já tinha-se um sistema funcional, essas alterações foram propostas com o objetivo de também melhorar seu desempenho e segurança². Apesar dessas medidas terem sido adotadas após a finalização de uma primeira versão do código, as questões levantadas neste capítulo também podem ser úteis no planejamento inicial de software para sistemas mecatrônicos aplicados à saúde.

4.1 CONCEITOS DE PROJETO DE SOFTWARE

O conceito de projeto de software é definido como o “*processo de definição da arquitetura, componentes, interfaces e outras características de um sistema ou componente*” quanto o “*resultado deste processo*” [65]. De forma geral, este processo pode ser decomposto em duas etapas:

- **Projeto de arquitetura:** descreve como software é organizado em componentes
- **Projeto detalhado:** descreve o comportamento desejado destes componentes

O resultado deste processo é um conjunto de modelos e artefatos que vão registrar as principais decisões que foram tomadas, bem como a fundamentação de cada decisão não-trivial. Ao registrar a fundamentação, a manutenibilidade de longo prazo do software produzido é aprimorada.

Princípios de projeto de software são premissas fundamentais, noções-chave que fornecem a base para diferentes abordagens e conceitos utilizados neste processo [65]. Alguns princípios amplamente reconhecidos e desejáveis em projetos de software incluem:

1. **Acoplamento** é a medida de interdependência entre módulos de um programa;
2. **Coesão** é a medida da força de associação entre elementos em um módulo;
3. **Decomposição e modularização** são etapas de um processo em que software é dividido em um número de componentes com interfaces bem definidas, que descrevem as interações entre componentes, com o objetivo de atribuir funcionalidades e responsabilidades distintas a componentes diferentes;
4. **Encapsulamento e ocultação de informação** significa que os detalhes internos de um componente estão agrupados e empacotados, sendo inacessíveis a entidades externas;

¹Um catálogo desse tipo de alteração pode ser encontrado em: <https://refactoring.com/catalog/index.html>

²Neste trabalho, quando forem definidos atributos de qualidade, o termo “proteção” passará a ser usado, para diferenciar do sentido de “segurança” que trata da restrição de acesso não-autorizado à aplicação e seus dados.

5. **Separação entre interface e implementação** é uma consequência dos itens 3 e 4: a implementação da funcionalidade de um componente e seus detalhes internos é separada da interface pública, utilizada na interação entre diferentes componentes;
6. **Responsabilidade única** é a noção que cada componente deve ser responsável por uma parte da funcionalidade fornecida pelo software, e que esta responsabilidade deve ser inteiramente contemplada pelo componente associado.

As diretrizes que guiam um projeto de software podem ser divididas em requisitos funcionais, que dizem respeito ao comportamento especificado, e não-funcionais, que dizem respeito ao seu modo de funcionamento. Atributos de qualidade são requisitos não-funcionais que diferenciam projetos que simplesmente fazem o que é necessário, daqueles que o fazem da forma mais adequada. Estes podem ser divididos em externos (características perceptíveis em tempo de execução) e internos (características estáticas do projeto). Na Tabela 4.1 são apresentados alguns atributos de qualidade adotados com frequência.

Atributos de qualidade externos são importantes primariamente para usuários, enquanto atributos internos são mais significativos para equipes de desenvolvimento e manutenção de software. Apesar disso, atributos internos contribuem indiretamente para a satisfação do usuário, uma vez que tornam o produto final mais fácil de melhorar, corrigir, testar e migrar para novas plataformas [66].

A definição de arquiteturas de software é um processo criativo, em que o projetista deve tomar decisões que afetam o produto e o processo. O impacto em atributos de qualidade serve como base para decisões de projeto, sendo muitas vezes necessário fazer escolhas entre atributos de qualidade concorrentes entre si.

Existem várias notações gráficas e textuais para representar artefatos de um projeto de software. Representações estruturais descrevem a estrutura organizacional de um projeto, incluindo os principais componentes e como eles se interconectam. Representações comportamentais descrevem o comportamento dinâmico do software, possivelmente incluindo a fundamentação de algumas decisões de projeto.

4.2 DIAGNÓSTICO DA VERSÃO INICIAL DO SOFTWARE

A base de código da *EMA Trike* em maio de 2015 era formada por cinco *scripts* em linguagem Python, cada um voltado para uma funcionalidade específica (i.e. não eram executados de forma conjunta). A Tabela 4.2 apresenta uma breve descrição do uso de cada *script*, formulada a partir da leitura do código-fonte. Nesta seção, será feita uma análise da qualidade do software apresentado a partir dos atributos de modifiabilidade, reusabilidade, verificabilidade e proteção.

4.2.1 Modifiabilidade

O que pode ser visto na Tabela 4.2 é que a base de código da *EMA Trike* continha diversas versões do mesmo *script*, que se diferenciavam temporalmente ou por plataforma-alvo. Não havia consistência suficiente, porém, entre as versões mais atuais para diferentes plataformas, nem documentação adequada para que fossem aplicadas as mudanças mais recentes de uma plataforma para outra. Não havia nenhum nível de controle versão, nem mesmo o mais básico possível — diversas cópias do mesmo código-fonte, sequencialmente numeradas.

Tabela 4.1: Atributos comuns de qualidade de software.

Atributo	Tipo	Resultado esperado
Disponibilidade	Externo	Os serviços do sistema estão disponíveis para o usuário quando e onde necessários
Instalabilidade	Externo	Instalar, desinstalar e reinstalar a aplicação corretamente é fácil
Integridade	Externo	O sistema está protegido contra perda e imprecisão de dados
Interoperabilidade	Externo	O sistema pode se interconectar e trocar informações com dados com outros sistemas ou componentes facilmente
Desempenho	Externo	O sistema responde a entradas do usuário ou outros eventos rapidamente e previsivelmente
Confiabilidade	Externo	A probabilidade de falha do sistema é baixa, possibilitando seu funcionamento ininterrupto por longos períodos de tempo
Robustez	Externo	O sistema responde bem a condições de operação inesperadas
Proteção	Externo	O sistema está protegido contra lesões e danos que poderiam ser causados ao usuário, ao ambiente e outros
Segurança	Externo	O sistema está protegido contra o acesso não autorizado à aplicação e seus dados
Usabilidade	Externo	Usuários conseguem aprender e lembrar como usar o sistema facilmente, sem orientações adicionais
Eficiência	Interno	O sistema usa os recursos computacionais disponíveis de forma eficiente, sem processamento desnecessário
Modifiabilidade	Interno	Desenvolvedores conseguem manter, mudar, melhorar e reestruturar o sistema facilmente
Portabilidade	Interno	Desenvolvedores conseguem colocar o sistema para funcionar em outros ambientes de operação facilmente
Reusabilidade	Interno	Componentes do sistema podem ser usados por desenvolvedores em outros sistemas
Escalabilidade	Interno	Desenvolvedores conseguem fazer o sistema lidar com mais usuários, transações ou servidores facilmente
Verifiabilidade	Interno	Desenvolvedores podem confirmar que o software foi implementado de forma correta prontamente

Apesar da maior parte dos testes do sistema à época serem estáticos (i.e. usando um rolo de treinamento), o que permitiria o uso de um notebook, a expectativa de uso sempre foi com o triciclo em movimento. Assim, pensar na compatibilidade do software com diferentes plataformas não seria só uma questão de comodidade ou estética, mas de sua efetividade: com menos espaço para o sistema embarcado, sem previsão de interface de vídeo, memória e poder de processamento reduzidos, é importante que seja possível avaliar rapidamente se a mudança de plataforma afetaria o desempenho do sistema. A taxa de amostragem dos sensores no dispositivo em questão, por exemplo, afeta diretamente a sincronização e cadência do movimento.

Grande parte do desenvolvimento foi feito usando um notebook, mas o sistema final utilizaria um computador embarcado, o que demandaria alterações no código-fonte que não foram documentadas ou sequer

Tabela 4.2: *Scripts* presentes na base de código da *EMA Trike* em maio de 2015.

Nome	Linhas	Funcionalidade
<i>readAngles_copa_brasil.py</i>	396	Plataforma: Notebook Exibir ângulo do pedivela no terminal enquanto executa um padrão de estimulação no quadríceps
<i>singleCommand.py</i>	48	Plataforma: Notebook Trecho de código utilizado para testar a IMU
<i>trike_raspberry.py</i>	397	Plataforma: Computador Embarcado <i>readAngle_copa_brasil.py</i> com pequenas mudanças para usar portas seriais no Linux
<i>trike_stim_MAC.py</i>	847	Plataforma: Notebook Versão atualizada do código-fonte, adicionando: <ul style="list-style-type: none"> • Perfis de estimulação • Filtro de média • Controlador PI de velocidade • Gráfico de dados • Registro de dados
<i>trike_stim_safer_control_since_indented.py</i>	919	Plataforma: Notebook <i>trike_stim_MAC.py</i> com mais canais de estimulação e pequenas alterações de escopo de trechos de código (através de indentação)

planejadas. Levando isto em consideração, percebe-se o risco latente de introdução de erros no momento em que alterações fossem transpostas de uma plataforma para outra.

No guia de estilo para código Python [67], Guido Van Rossum apresenta a ideia de que “*código é lido com uma frequência maior do que ele é escrito*”. No mesmo espírito, Abelson & Sussman dizem, no prefácio da primeira edição de seu livro [68], que “*programas são escritos para serem lidos por humanos e apenas incidentalmente executados por computadores*”. Ambas citações apontam para a importância da legibilidade de código, fundamental para que o trabalho de uns possa ser continuado por outros - mesmo que “os outros” sejam os mesmos, apenas meses mais tarde.

Na base de código original da *EMA Trike*, observa-se um uso reduzido de comentários. Isso não seria um problema se a estrutura do código fosse autoexplicativa, porém a falta de consistência no nome de variáveis e funções ofusca a funcionalidade em muitos trechos. Ao contrário do que se poderia imaginar, por exemplo, a função *getEulerAngles* não se resume a retornar ângulos de Euler, mas também atualiza sinais de estimulação e só retorna quando se encerra o programa. Na prática, trata-se do laço principal de execução do *script*.

A maior parte dos comentários é utilizada para alternar configurações do código, em geral valores de variáveis. Em alguns casos trechos inteiros de código foram comentados, quando um *script* foi reaproveitado mas determinada funcionalidade não interessava. Ao invés de clarificar, os comentários muitas vezes contribuíam para tornar o código mais obscuro.

Não se observa estrutura clara no código: variáveis globais e funções são definidas de forma intercalada, ao longo de *scripts* com centenas de linhas. Sendo interrompido por grandes trechos ocupados por definições de funções, o caminho de execução do código apresenta-se bastante confuso. O trecho principal de todos *scripts*, por estar mais próximo ao final do arquivo e logo após a definição uma função curta, pode ser facilmente confundido como parte da função referida.

Um dos pontos mais graves no que se refere à legibilidade é o tratamento dado a protocolos de comunicação no código-fonte. No corpo das funções que devem obter algum dado de sensores ou enviar comando para o atuador, são montadas sequências de bytes como pacotes a serem enviados, sem qualquer explicação do que está sendo feito. A Figura 4.1 ilustra como variáveis de configuração, comentários e implementações de protocolo foram intercalados de forma ilegível para o leitor desavisado que, para decodificar a intenção do programador, deveria consultar o manual de operação de todo hardware utilizado no dispositivo. Isso com o efeito colateral de, sem uma implementação consistente do protocolo para ser utilizada em todo código, surgir o risco de pacotes serem montados incorretamente apenas em alguns trechos do *script*.

Por fim, cabe observar que não foi criada nenhuma documentação externa ao código-fonte que permitiria compreender o fluxo de execução apresentado. A leitura e interpretação de um código monolítico de 800 linhas deve ser feita inteiramente por quem se interessar, sem auxílio externo. As possibilidades de melhoria e reestruturação da base de código da *EMA Trike* eram limitadas pelo acoplamento das diferentes etapas do algoritmo, resultado da baixa modularidade do software.

4.2.2 Reusabilidade

A extensão de um código-fonte em que aplicou-se um nível baixo de abstração indica outro risco no projeto do software: a criação de um “objeto todo-poderoso” [69], um módulo que se propõe a resolver diversos problemas de natureza distinta sozinho e que, por isso, mantém o controle de todo o contexto de dados do sistema. Isso é considerado um “anti-padrão” de desenvolvimento de software, contrário ao princípio da responsabili-

```

180 def initialization(freq, channels):
181     # try:
182
183     #valores para teste#
184     ts1 = round((1/float(freq))*1000)
185     print ts1
186     ts2 = 1.5
187     main_time = int(round((ts1 - 1) / 0.5))
188     # main_time = int(round(ts1))
189     group_time = int(round((ts2 - 1.5) / 0.5))
190     channel_stim = channels
191     channel_lf = 0
192     n_factor = 0
193     check = int((n_factor + channel_stim + channel_lf + group_time + main_time)%8)
194
195     init_1 = (1<<7) | (0<<6) | (check<<2) | (n_factor>>1)
196     init_2 = ((n_factor&1)<<6) | (channel_stim>>2)
197     init_3 = ((channel_stim & 3) << 5) | (channel_lf >> 3)
198     init_4 = ((channel_lf & 7) << 4) | (group_time >> 3)
199     init_5 = ((group_time & 7) << 4) | (main_time >> 7)
200     init_6 = main_time & 127

```

Figura 4.1: Mistura entre implementação de protocolo e variáveis de configuração, sem comentários para guiar o leitor que necessite realizar alterações em parâmetros do sistema

dade única: “*uma classe deve ter apenas um motivo para mudar*” [70].

Na versão mais atualizada do *script* em questão, observa-se a implementação de (1) uma interface gráfica para exibição do histórico de variáveis de interesse, (2) protocolos de comunicação com sensores e atuadores, (3) um sistema de registro persistente de dados para análise posterior, (4) um controlador PI de velocidade e (5) uma interface de usuário para controle do sistema. Existe um fluxo dados natural entre essas funcionalidades, o que apresenta o risco de corrupção acidental de dados à medida em que eles transitam por trechos de leitura e escrita sem nenhum encapsulamento ou restrição de acesso.

Além das funcionalidades listadas coexistirem no mesmo código, a abstração do fluxo de dados é baixa, encadeando sequências longas de comandos dedicadas a resultados parciais do processo. Isso faz com que o leitor “não consiga ver a floresta pelas árvores”, i.e. se perca no detalhes de cada um desses resultados parciais e não consiga perceber o processo como um todo.

Uma mudança em qualquer uma dessas funcionalidades afeta diretamente o sistema como um todo, podendo resultar em falhas e inconsistências não previstas. Mesmo uma mudança na interface gráfica, como a exibição do sinal de controle, apresenta o risco de alterar o nível de estimulação aplicado no usuário, caso essa variável seja sobrescrita por engano. Um eventual travamento na interface de usuário pode levar à falha no registro persistente de dados, que somente é realizado no fim da execução do programa, prejudicando a verificabilidade do sistema pós-falha.

A modularidade do sistema leva à melhoria do atributo de reusabilidade, que tem uma vantagem adicional: caso o processo de desenvolvimento dos módulos tenha levado em consideração aspectos de proteção relevantes para sistemas mecatrônicos aplicados à saúde, projetos que tiram proveito desses módulos já se iniciam com uma qualidade superior. Isso permite que os desenvolvedores possam dedicar mais tempo à segurança dos componentes adicionais do novo sistema.

4.2.3 Verifiabilidade

Em geral escreve-se software crítico com a intenção de que não ocorra falha alguma. Caso isso não seja possível, porém, é extremamente desejável que se possa avaliar o contexto em que aquilo se realizou, para que medidas corretivas possam ser tomadas. Isso se torna especialmente importante quando não se pode constantemente apresentar informações sobre o estado do sistema, como é o caso de muitos sistemas embarcados.

O código-fonte original mantém poucos registros do fluxo de execução, se limitando à exibição de algumas mensagens no terminal, sem deixar claro o instante em que cada uma foi gerada. A não ser que a saída padrão seja redirecionada para um arquivo durante a execução (e.g. usando *pipes*), essas informações serão perdidas em uma situação de falha. Não observa-se, porém, a intenção explícita de se manter um *log* de execução.

Além da disponibilidade dos rastros da execução do software serem úteis para a depuração de causas de falha, o registro dos valores de algumas variáveis ao longo do tempo pode ser valioso. Ele pode, por exemplo, evidenciar uma falha de hardware que tenha ocorrido em tempo de execução. Outro benefício colateral é a possibilidade de se analisar o desempenho do sistema de forma quantitativa, medindo-se taxas de atualização de sensores, frequência de ocorrência de exceções, relações de entrada e saída de funções e similares.

O software analisado mantém um registro persistente de dados na versão mais recente, porém a gravação no disco só é feita no fim do programa, se ele for encerrado de forma adequada. Infelizmente, os casos em que estes dados se mostram mais necessários são aqueles em que ocorre uma exceção não tratada.

4.2.4 Proteção

No sistema da *EMA Trike*, a principal fonte de perigo é a estimulação elétrica. Períodos de estimulação muito longos, com valores de corrente elevados, podem levar a queimaduras na pele. Grandes variações na intensidade da corrente ou uma cadência inadequada da estimulação de diferentes grupos musculares podem gerar movimentos bruscos que levem a lesões nos membros inferiores. Falhas que podem levar a esse estado perigoso incluem o (1) não recebimento de dados de sensores, (2) corrupção de dados de sensores, (3) divergência ou instabilidade do controlador e (4) falhas no envio de comandos de atualização para o estimulador.

No código-fonte analisado, há uma grande incidência de variáveis globais. Esse tipo de variável pode ser alterada e utilizada em qualquer parte do programa, fazendo com que quaisquer regras implícitas ou explícitas sobre seu uso sejam facilmente quebradas ou esquecidas. Existem ainda problemas de concorrência: as variáveis utilizadas são acessadas por múltiplas *threads* de execução, negligenciando a sincronização necessária com entidades como *mutexes*. Essas questões podem gerar a corrupção de dados em qualquer ponto do fluxo de execução, gerando as situações de perigo.

Outra questão latente relativa à proteção diz respeito ao cuidado com a temporização do sistema. As *threads* existentes não têm suas taxas de execução monitoradas ou controladas, nem são priorizadas entre si. Isso pode gerar atrasos inesperados no recebimento de dados de sensores, atualização do controlador ou envio de comandos para o estimulador.

Não foi implantado um protocolo formal de testes do software, para avaliar seu funcionamento em casos limítrofes ou sua robustez. Testes de bancada foram realizados, porém sem o registro de seus resultados ou condições de operação. Por haver componentes de hardware no sistema, torna-se necessário avaliar seu funcionamento integrado ao software. Porém, como o código não foi modularizado, não é possível fazer testes unitários, isolando cada componente de hardware. Assim, a principal verificação de funcionalidade é feita em

produção (i.e. com um usuário pedalando).

É verdade que testes de software são limitados na sua capacidade de expor todos defeitos latentes no software. A complexidade de sistemas mecatrônicos médicos, que envolvem interações de software, hardware e usuário, previne a possibilidade de testes exaustivos [71]. Assim, a estruturação adequada do processo de desenvolvimento de software é essencial para evitar a introdução de erros. Nesse sentido, a mistura de variáveis de configuração com o código-fonte é uma grande fonte de perigos latentes. A necessidade de se editar o código executável para ajustar o sistema para diferentes usuários e situações faz com que esse processo oportunize a introdução de falhas no sistema.

A principal medida de proteção presente na *EMA Trike* é um grande botão de emergência, ligado diretamente ao hardware do estimulador. Sua função é desconectar imediatamente o circuito interno da fonte de alimentação e descarregar qualquer carga residual através do aterramento. Não há medidas de proteção explícitas no software, como limites impostos para correntes ou *watchdogs* para a atualização do sinal de estimulação.

4.3 ARQUITETURA PROPOSTA

Pensando nas características da *EMA Trike*, tanto em termos de hardware utilizado quanto de etapas do fluxo de execução, foi proposta uma arquitetura *dataflow* para o software, decompondo o processamento em estágios que podem ser executados de forma concorrente. A Figura 4.2 ilustra a estrutura proposta, com três estágios no fluxo de dados principal: IMU, Controle e Estimulador. Cada estágio é implementado como um processo que se encaminha dados para os demais por meio de alguma modalidade de comunicação entre processos, levando em consideração as precauções necessárias quanto à sincronização. Através da separação dos contextos inerente à criação de processos distintos para cada etapa do fluxo de dados, evitou-se o uso de variáveis globais.

Foi proposta ainda a criação de processos adicionais para lidar com a interface com o usuário, para permitir a interação e visualização do estado do sistema em tempo de execução, e o registro de dados, para permitir uma análise detalhada posterior do funcionamento do sistema. Para tanto, seriam criados fluxos de dados secundários, necessitando que sejam especificados apenas os tipos de interações possíveis na interface e as variáveis de interesse para ambos processos.

No Python existe o *Global Interpreter Lock*, um *mutex* que previne que múltiplos *threads* nativos executem bytecodes do Python simultaneamente. Assim, na prática, como a plataforma-alvo contém quatro núcleos de processamento, o sistema original só utiliza um quarto dos recursos disponíveis. Uma consequência da proposta de separação dos módulos do sistema em processos, ao invés de *threads*, é a possibilidade de distribuição da carga computacional entre esses núcleos.

A criação de arquivos de configuração externos ao código-fonte, seguindo uma sintaxe pré-definida, é desejável. Estes arquivos seriam específicos para cada módulo, sendo carregados em tempo de execução sem a necessidade de recompilação ou edição do código-fonte. Ao segmentar arquivos de configuração de acordo com módulos e separá-los, são reduzidas as possibilidades de introdução de erros nesse processo.

Outra medida proposta é a introdução do registro de eventos de execução ao longo do código, para incrementar a verificabilidade do sistema. Deve ser possível redirecionar a saída desses registros para o terminal, arquivos de *log* ou ambos, através de uma configuração do sistema. Também deve ser possível filtrar cada um desses registros segundo a gravidade das mensagens.

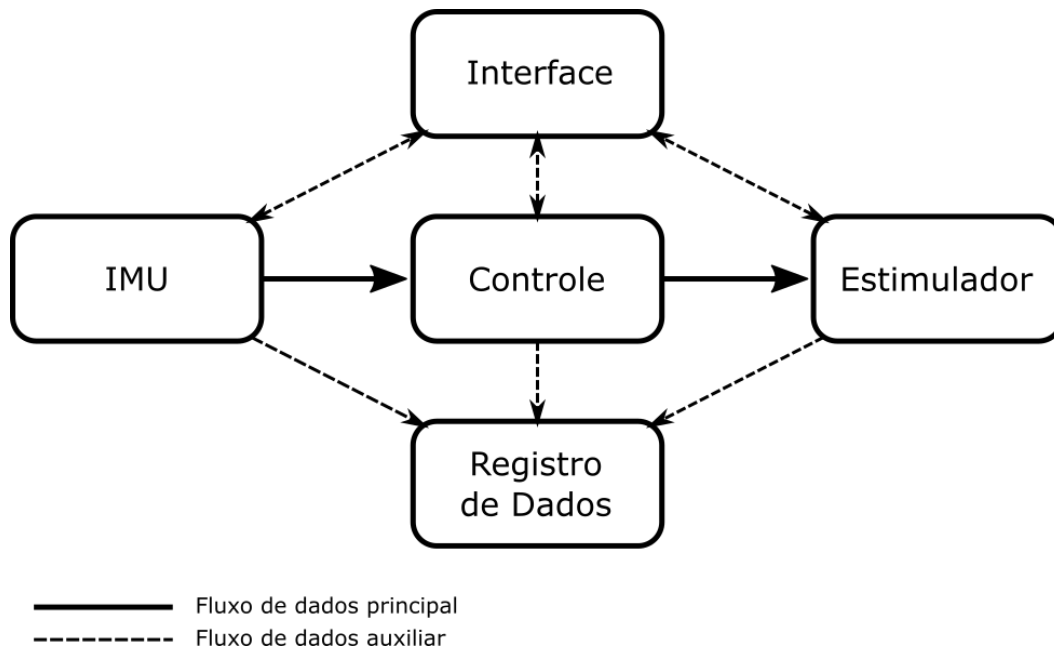


Figura 4.2: Arquitetura *dataflow* para a EMA Trike.

A implantação de um sistema de controle de versão é essencial para possibilitar a verificação de alterações realizadas ao longo do ciclo de vida do software, principalmente se for identificada alguma falha ao longo do tempo. Outros padrões de desenvolvimento de software mais seguros e consistentes devem ser seguidos pelo mesmo motivo, como por exemplo a implementação genérica de protocolos de comunicação com hardware, preferencialmente através do uso de bibliotecas fornecidas pelo fabricante.

Por fim, sugere-se a implantação de mecanismos adicionais de proteção, que possam encerrar a execução do sistema automaticamente em caso de inconsistências temporais ou corrupção de dados. Parte da implementação desses mecanismos pode ser feita em um dos processos do fluxo de dados auxiliar.

5

IMPLEMENTAÇÃO DA ARQUITETURA PROPOSTA

Neste capítulo serão descritos o processo de reestruturação do software da *EMA Trike* e as ferramentas utilizadas nesse processo. Em particular, apresenta-se o *Robot Operating System* (ROS) como alternativa viável para implementação da arquitetura proposta. Além de permitir a organização do software como apresentado no capítulo anterior, ele traz algumas características desejadas de forma nativa, reduzindo o tempo de desenvolvimento. Discute-se, ainda, algumas especificidades relacionadas a cada módulo do sistema e sua integração.

5.1 ROBOT OPERATING SYSTEM

Criado em 2008, o *Robot Operating System* (ROS) [72] consolidou-se como o padrão *de facto* para desenvolvimento de software de sistemas robóticos¹. Trata-se um *framework* flexível que foi construído desde o início para incentivar a reutilização de código e a colaboração. O ROS contém uma coleção de ferramentas, bibliotecas e convenções que têm como objetivo simplificar a criação de comportamento complexo, robusto e que funcione em diferentes tipos de robôs.

Apesar do nome, o ROS não é um sistema operacional no sentido estrito. Sua comunidade se refere ao ROS como um “meta-sistema operacional” porque, apesar de ser executado em plataformas baseadas em UNIX², ele provê serviços que você normalmente encontraria em um sistema operacional, tais como (1) abstração de hardware, (2) controle de dispositivos em baixo nível, (3) implementação de funcionalidades comumente utilizadas, (4) transmissão de mensagens entre processos e (5) gerenciamento de pacotes.

A Figura 5.1 ilustra a descrição usual dada para o ROS por um dos seus idealizadores, Brian Gerkey: “*ROS = Plumbing + Tools + Capabilities + Ecosystem*”. A soma desses elementos é o que faz com que o ROS seja uma alternativa tão poderosa para desenvolvedores: a possibilidade de integração de sistemas complexos, desenvolvidos por especialistas, de forma simples e flexível:

- **Plumbing** (ou encanamento) se refere à infraestrutura que permite que diferentes módulos se comuniquem para serem usados em conjunto;
- **Tools** (ou ferramentas) são aplicações nativas que permitem ao desenvolvedor interagir com sistema em funcionamento de diferentes formas;
- **Capabilities** (ou capacidades) se refere à facilidade com que a estrutura do ROS e sua distribuição online e de código aberto permite a integração de sistemas distintos, muitas vezes desenvolvidos por especialistas;

¹Como normas técnicas foram mencionadas na revisão bibliográfica, é importante ressaltar que não existe atualmente norma específica regendo o desenvolvimento de software para sistemas robóticos. Considera-se um padrão *de facto* devido à alta adoção desse *framework*: somente em julho de 2016, contabilizou-se mais de 8 milhões de downloads de pacotes por mais de 100 mil endereços de IP únicos [73].

²A base de código do ROS é testada principalmente nos sistemas operacionais Ubuntu e Mac OS X, embora a comunidade tenha contribuído para outras plataformas Linux, como Fedora, Gentoo, Arch Linux e Raspbian (que é utilizada neste projeto). Embora existam esforços para portar o código do ROS para o Windows, essa possibilidade ainda não foi explorada por completo.

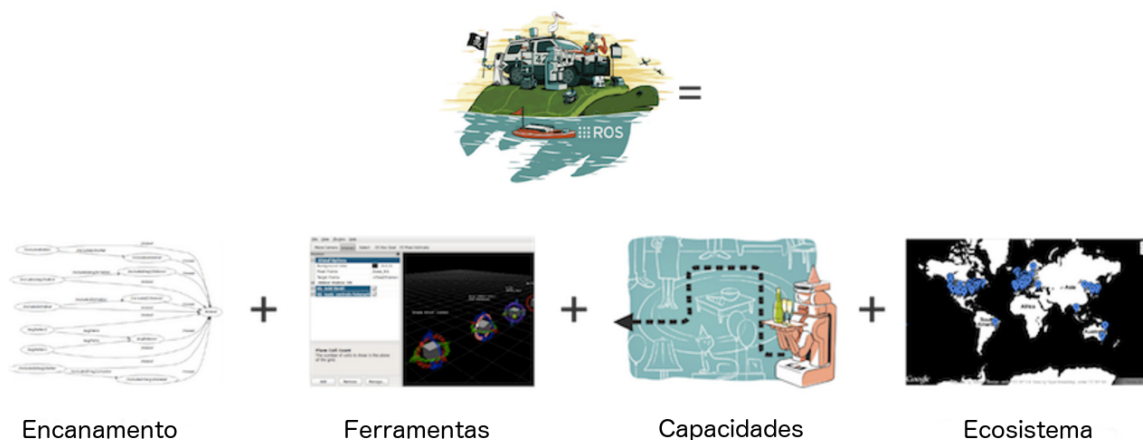


Figura 5.1: Descrição dos elementos que compõem o Robot Operating System (ROS).

- **Ecosystem** (ou ecossistema) se refere à comunidade global que usa, mantém e continua desenvolvendo as capacidades do ROS, através de ambientes online para tirar dúvidas, documentar o trabalho, contribuir e discutir os objetivos da comunidade³.

O objetivo do ROS não é ser o *framework* com mais funcionalidades, mas permitir a reutilização de código em pesquisa e desenvolvimento no campo da robótica. Sua estrutura permite, por exemplo, que programas escritos em linguagens diferentes, como C++, Python ou Lisp, interajam sem grandes dificuldades. Além dessas três, existem doze outras linguagens de programação cujas bibliotecas estão em estágio experimental. Naturalmente, as possibilidades que são abertas com isso fazem com que haja uma grande variedade de funcionalidades disponíveis para a comunidade, mesmo que esse não fosse um objetivo primário.

Diferentes versões do sistema central do ROS são lançadas regularmente como distribuições, de forma similar ao que acontece com sistemas Linux. Além da funcionalidade básica, essas distribuições incluem ferramentas e bibliotecas úteis e compatíveis entre si. A versão utilizada neste trabalho é a versão *Indigo Igloo*, lançada em julho de 2014 e com suporte previsto até abril de 2019.

5.1.1 Conceitos básicos

Todo sistema operando no ROS funciona como uma rede *peer-to-peer* de processos, que podem estar distribuídos em máquinas diferentes. Esses processos conectados usando a infraestrutura de comunicação do ROS são chamados de nós. Para possibilitar essas conexões, um nó mestre é iniciado com o ROS para gerenciar o grafo de execução. Ele é responsável apenas por registrar os nós disponíveis no sistema e mediar a conexão entre eles sempre que requisitado, de forma similar ao papel exercido por um servidor DNS na Internet. Nenhum dado, portanto, passa por ele uma vez que a conexão é estabelecida.

³Esses ambientes podem ser acessados nos seguintes endereços:

- ROS Answers: <http://answers.ros.org>
- ROS Wiki: <http://wiki.ros.org>
- ROS core stacks: <http://github.com/ros>
- ROS Discourse: <http://discourse.ros.org>

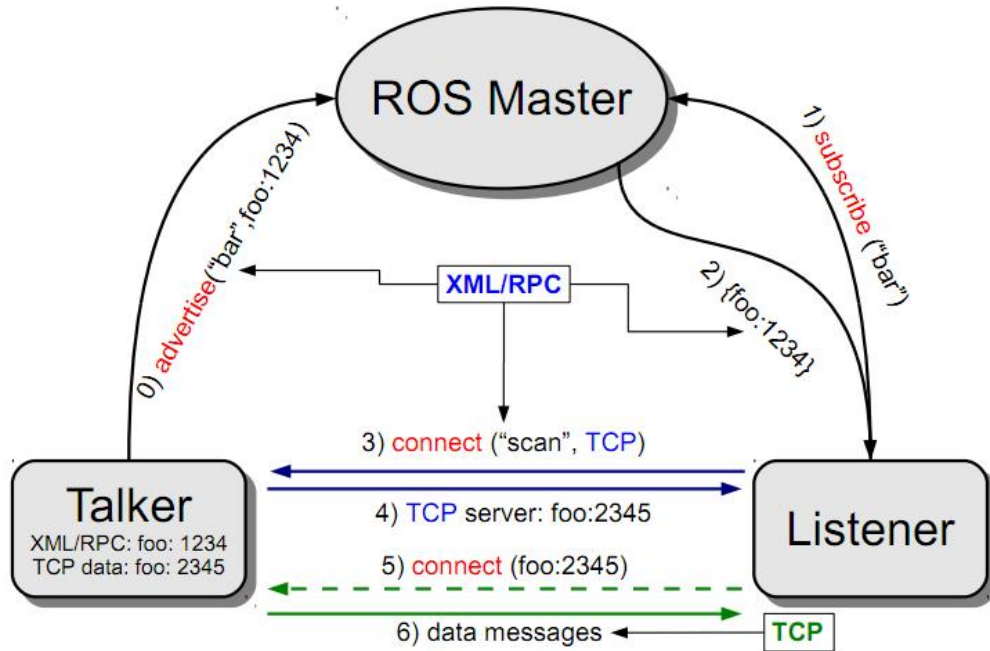


Figura 5.2: Etapas de conexão a um tópico no ROS.

Existem dois tipos principais de comunicação no ROS: tópicos e serviços. Tópicos funcionam como canais de transmissão assíncrona de dados, barramentos em que nós publicadores podem enviar mensagens de um tipo específico para um ou mais nós assinantes. Serviços, por outro lado, seguem o paradigma de transmissão síncrona cliente-servidor, em que o nó interessado deve enviar requisições para o nó que disponibiliza o serviço toda vez que desejar uma resposta. As Figuras 5.2 e 5.3 detalham os respectivos processos de conexão.

Para transmitir dados através de tópicos ou serviços no ROS, são definidas mensagens correspondentes aos tipos de dados transmitidos. Estes dados são serializados em uma representação compacta que corresponde aproximadamente ao formato de uma struct da linguagem C em formato *little endian*. Essa representação implica na necessidade de ambos nós terem acesso à mesma definição do layout da mensagem - para isso, mensagens no ROS são identificadas não só pelo seu nome, mas também por uma *hash* MD5.

Uma das grandes vantagens do ROS é a padronização de unidades de medidas e convenções de sistemas de coordenada [74] adotadas em mensagens e serviços. Inconsistências dessa natureza são uma fonte comum de problemas de integração para desenvolvedores e podem levar a falhas de software. Além disso, elas podem levar a processamento desnecessário para conversão de dados. Essa padronização, amplamente adotada na base de código do ROS, aumenta as possibilidades de interconexão entre módulos desenvolvidos por grupos distintos, permitindo a abstração necessária para que módulos de funcionalidade equivalente sejam intercambiáveis.

A comunicação entre nós no ROS é feita através de sockets, para possibilitar a implementação de sistemas distribuídos. O protocolo de transporte para as mensagens é selecionável, sendo que as principais escolhas no momento são TCP e UDP. Para sistemas de alto desempenho, em que processos numa mesma máquina precisam se comunicar de forma mais eficiente, é possível ainda transmitir mensagens através de ponteiros compartilhados, usando um tipo de nó especializado chamado de *nodelet*.

Outro aspecto importante do ROS é o servidor de parâmetros. Trata-se de um servidor XML-RPC para armazenamento distribuído de parâmetros do sistema, que podem ser requisitados por quaisquer nós. Esses

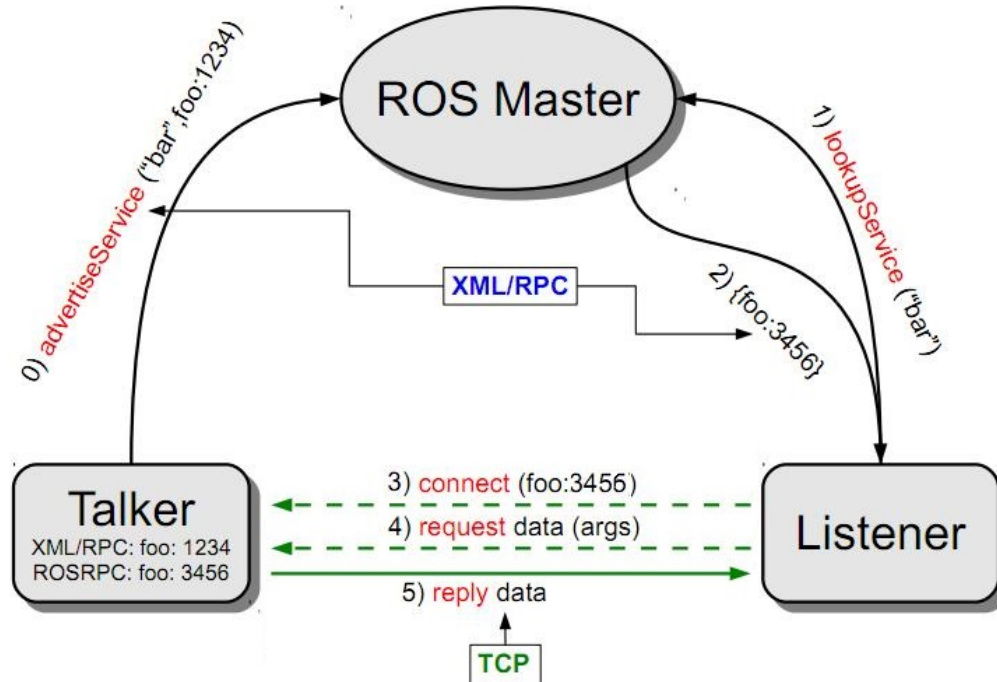


Figura 5.3: Etapas de conexão a um serviço no ROS.

parâmetros podem ser escalares básicos do protocolo, listas e dados binários codificados no formato *Base64*. Esses parâmetros podem ser carregados a partir de arquivos de configuração em formato *YAML* e alterados em tempo de execução, dando grande flexibilidade para reconfiguração simultânea de vários módulos.

Por fim, existe no ROS um mecanismo central baseado em tópicos chamado de *rosout* para o registro de mensagens de status dos nós. Esse registro permite a depuração simultânea do fluxo de execução de todos nós, uma vez que todas são agregadas no mesmo lugar. Existem cinco níveis de verbosidade para esses registros (*DEBUG*, *INFO*, *WARN*, *ERROR* e *FATAL*), dando ao desenvolvedor a possibilidade de registrar os mínimos detalhes da execução sem sobrecarregar o recipiente das mensagens. Isso porque cada recipiente pode filtrar as mensagens recebidas de acordo com esse nível, sendo *DEBUG* o nível mais abrangente e *FATAL* o nível mais restrito. Toda mensagem registrada pode ser direcionada para um terminal, um arquivo de *log* ou ainda ser recebida por um nó através do tópico correspondente.

5.1.2 Ferramentas nativas

Juntamente às capacidades mencionadas na subseção anterior, o ROS contém aplicações que permitem a interação com sistemas em funcionamento. Uma das mais utilizadas se chama *rviz*, um ambiente de visualização 3D que permite a combinação de dados de sensores, modelos de robôs, sistemas de coordenadas, mapas, marcadores e quaisquer outros dados que possam ser representado tridimensionalmente em uma interface única. Esse tipo de visualização permite que o desenvolvedor rapidamente veja o que o robô vê, ajudando a identificar problemas como desalinhamento de sensores ou imprecisões em modelos de robô. A Figura 5.4 mostra o *rviz* em ação, mostrando simultaneamente a imagem da câmera montada na cabeça do robô PR-2, o modelo 3D do robô, obstáculos detectados por seus sensores e o mapa usado para navegação.

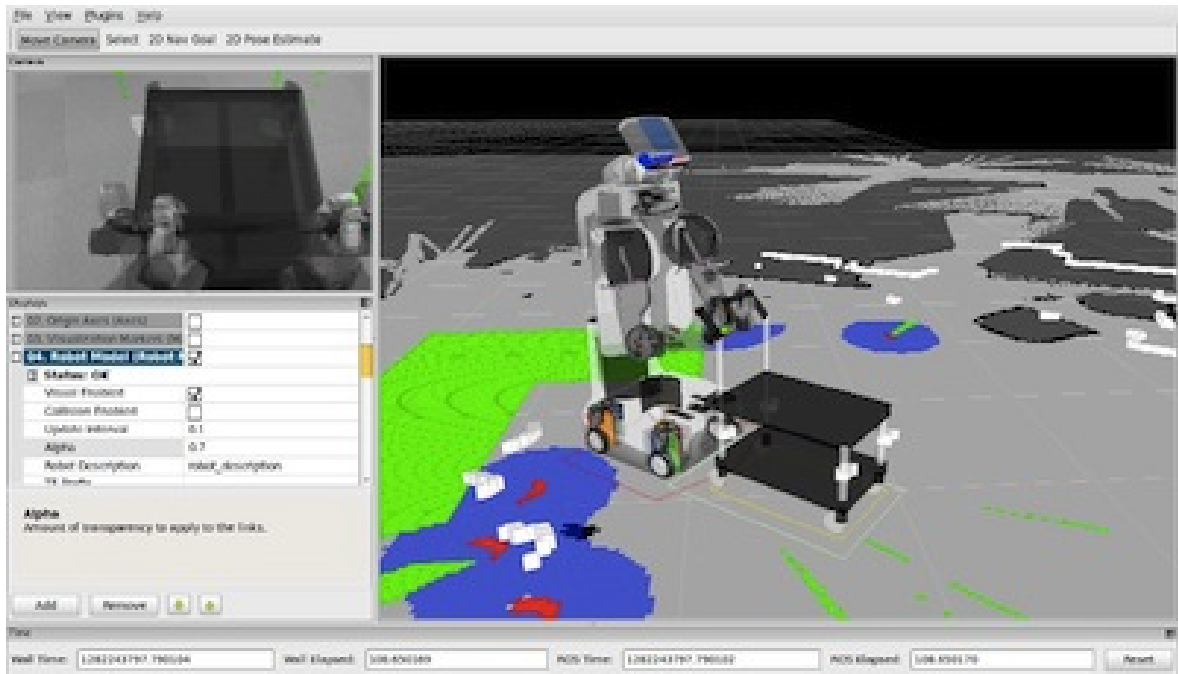


Figura 5.4: Visualização 3D de sistema robótico usando a ferramenta *rviz*.

Dados mais simples podem ser visualizados de outras formas. A ferramenta *rqt_plot*, por exemplo, permite que o valor de variáveis escalares possa ser acompanhado em tempo de execução em um gráfico simples. Múltiplas variáveis podem ser plotadas simultaneamente, sem nenhum esforço do desenvolvedor para disponibilizar essa interface gráfica - basta que os dados sejam publicados em um tópico por algum nó do ROS. A Figura 5.5 mostra um exemplo desse tipo de visualização com o *rqt_plot*, apresentando duas variáveis no mesmo gráfico.

Além de acompanhar o estado de variáveis do sistema, é possível visualizar ainda a estrutura do grafo de execução usando a ferramenta *rqt_graph*. Na interface apresentada, é possível observar os nós sendo executados no ROS, os tópicos que cada um disponibiliza, as conexões existentes entre nós e até mesmo a taxa de transmissão em cada tópico. A Figura 5.6 mostra a estrutura de um sistema complexo visualizado no *rqt_graph*.

Uma das possibilidades apresentadas nativamente pelo ROS é o registro persistente de dados de mensagens em um formato binário chamado de arquivos *bag*. A ferramenta *rosvbag* permite a escolha de um grupo de tópicos para ser registrado no disco, adicionando as mensagens de forma serializada à medida que são recebidas pelos nós assinantes. Esses arquivos também podem ser reproduzidos no ROS nos mesmos tópicos de que foram gravados, com aproximadamente a mesma temporização, como se a situação observada estivesse ocorrendo novamente. Sem nenhum esforço adicional, é possível fazer o registro completo de um experimento, incluindo inclusive imagens, para ser observado e analisado posteriormente de forma simples. Arquivos *bag* podem ser gravados para servirem como *datasets* comuns para validação de algoritmos e compartilhados juntamente a publicações relacionadas, ampliando as possibilidades de escrutínio acadêmico de resultados. A ferramenta *rqt_bag* facilita esse processo, apresentando os dados registrados em uma linha do tempo. A Figura 5.7 mostra um arquivo *bag* com registro de escalares e imagens no *rqt_bag*.

Além das ferramentas gráficas mencionadas, existem mais uma série de ferramentas de linha de comando que facilitam o gerenciamento de pacotes e a instalação, compilação, execução, interação e depuração de sistemas no ROS. Uma das mais utilizadas é a *roslaunch*, que permite a execução simultânea de múltiplos nós

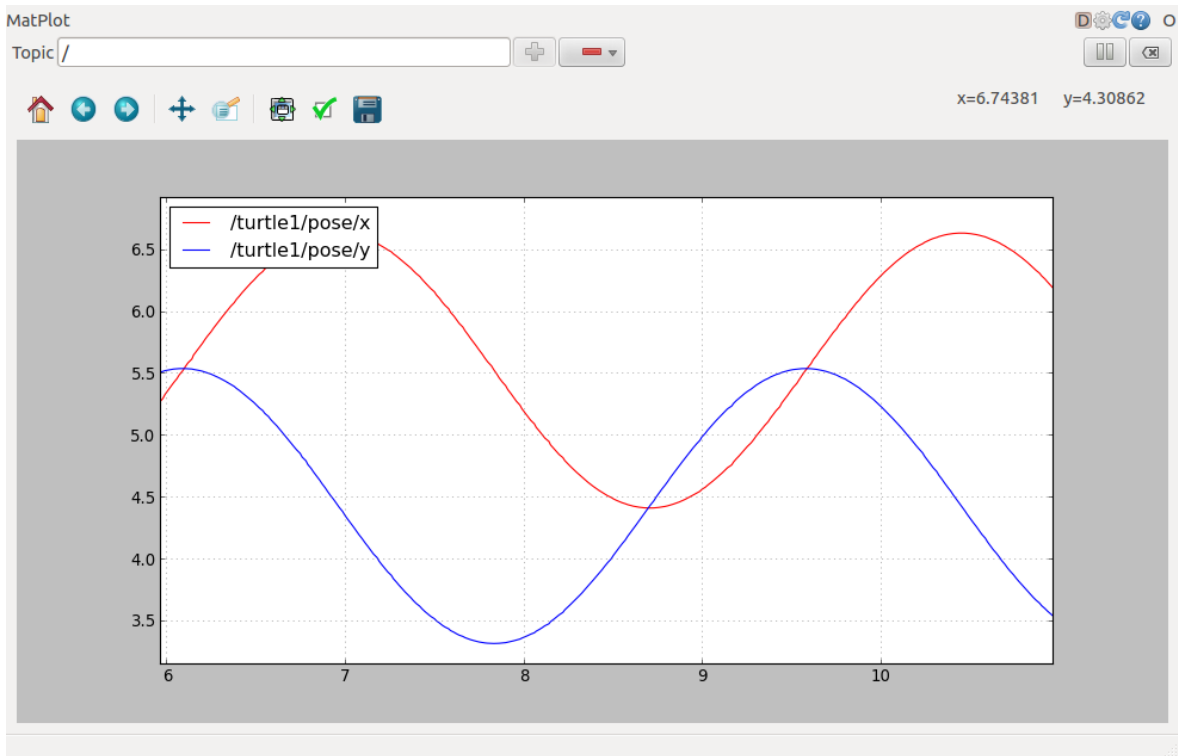


Figura 5.5: Gráficos de variáveis ao longo do tempo usando a ferramenta *rqt_plot*.

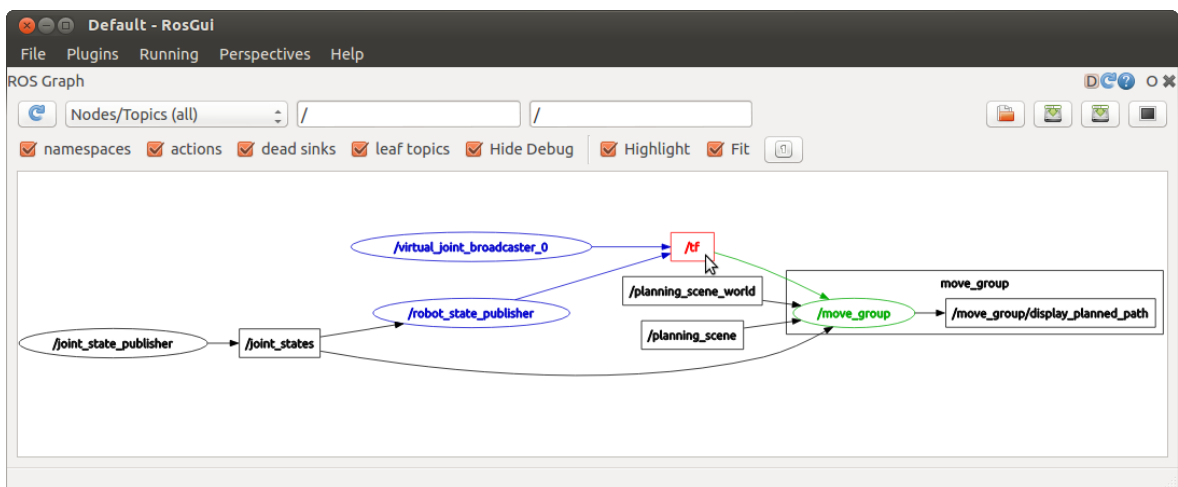


Figura 5.6: Grafo de execução de sistema visualizado na ferramenta *rqt_graph*.

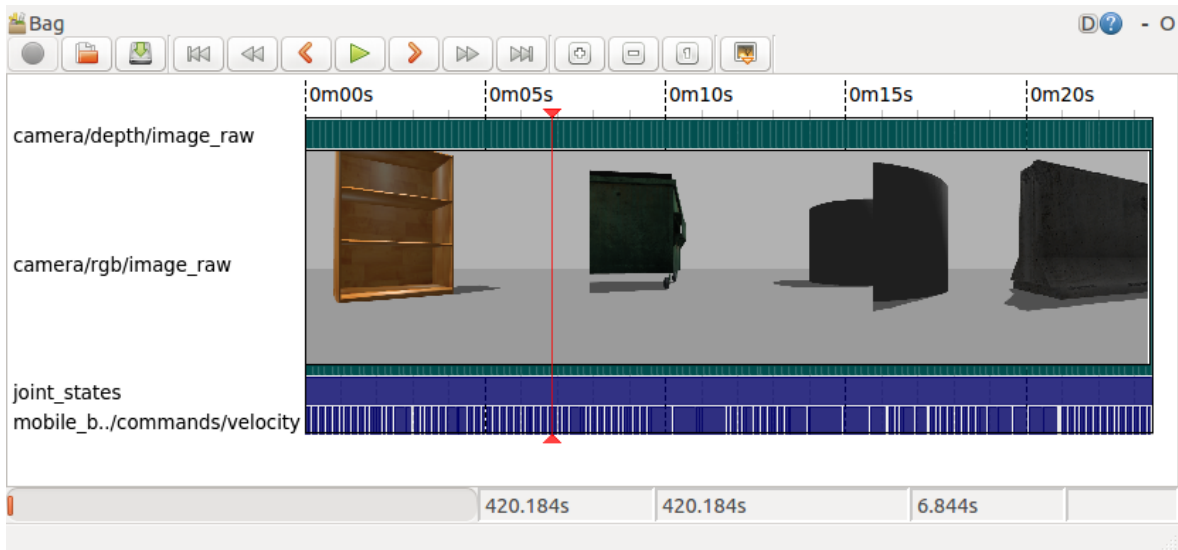


Figura 5.7: Arquivo *bag* com escalares e imagens aberto na ferramenta *rqt_bag*.

do ROS, localmente e remotamente via SSH, especificando parâmetros e opções através de um arquivo XML.

É possível fornecer arquivos de configuração para cada nó carregar valores de interesse no servidor de parâmetros, definir se nós devem ser reiniciados se forem terminados por qualquer motivo, especificar as máquinas em que cada nó deve ser executado e uma série de outras particularidades. Isso dá grande flexibilidade para o desenvolvedor, que pode criar diversos casos de uso complexos para serem lançados com um comando simples.

5.1.3 Adequação para a arquitetura proposta

Dentro do ROS existem várias facilidades para a implementação da proposta da seção 4.3. Cada processo ilustrado no fluxo principal da Figura 4.2 pode ser implementada como um nó do ROS, com a sua interface e comunicação entre processos resolvida através da definição de tópicos do ROS.

Ferramentas nativas podem cumprir o papel dos processos de interface (*rviz*, *rqt*) e registro de dados (*rosbag*). A ferramenta *roslog* possibilita a implementação de registros de eventos. A ferramenta *rqt_graph* permite verificar não só a interconexão entre os sistemas, mas também monitorar a taxa de transmissão de dados em cada tópico. Todas essas ferramentas permitiriam elevar o grau de verificabilidade do sistema, provendo alternativas de depuração em tempo de execução e situações pós-falha, sem que isso leve a um aumento no tempo ou complexidade de desenvolvimento do sistema.

Para o uso de arquivos de configuração separados por módulo e facilmente carregáveis, é possível lançar mão da ferramenta *roslaunch* e o servidor de parâmetros. Aliás, este é outro fator que incrementa a verificabilidade do sistema, uma vez que as configurações no servidor de parâmetros podem ser consultados a qualquer momento. Dentro do *roslaunch*, ainda é possível configurar o sistema para reiniciar processos que tenham falhado por algum motivo — isso permite que o sistema se recupere em caso de falhas.

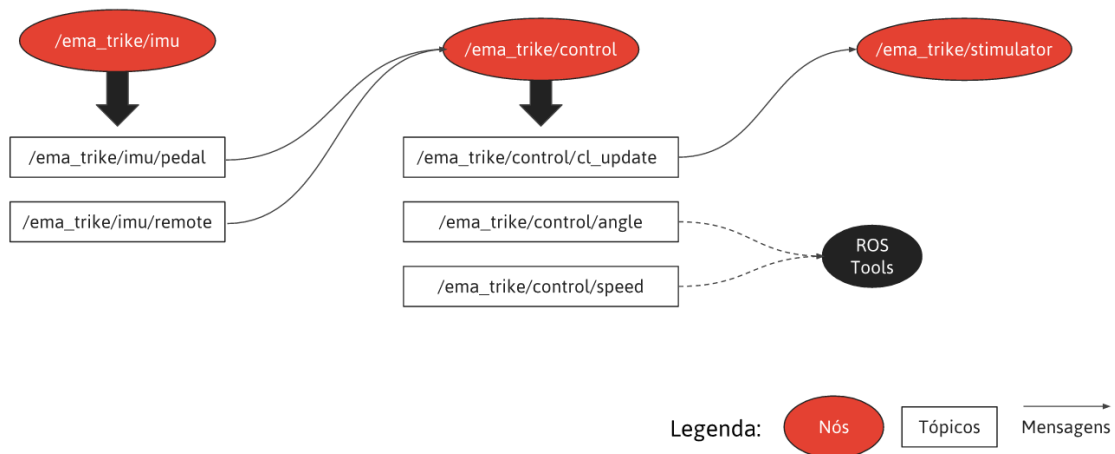


Figura 5.8: Estrutura do sistema da *EMA Trike* implementado no ROS.

5.2 VISÃO GERAL DA IMPLEMENTAÇÃO

A estrutura proposta para o software na Figura 4.2 foi implementada com a criação de um nó do ROS para cada processo no fluxo de dados principal. As funcionalidades presentes no fluxo de dados secundário são providas por ferramentas nativas do ROS. A Figura 5.8 mostra o grafo de execução do sistema desenvolvido, com seus tópicos e nós básicos, em um formato próximo a um diagrama de fluxo de dados.

Cada um dos nós foi programado para que o código específico do ROS fosse separado da implementação da funcionalidade de cada processo. Para isso, foram criados módulos Python que poderiam ser adaptados para outros ambientes caso fosse observado que o desempenho do sistema no ROS não estivesse adequado. Esses módulos contém a implementação completa dos protocolos de cada hardware utilizando, sempre que possível, bibliotecas disponibilizadas pelos fabricantes, para permitir sua reutilização por outros sistemas construídos com os mesmos componentes. O código específico de cada nó, então, apenas precisaria importar o seu módulo correspondente para adicionar a lógica de conexão entre processos do ROS. Para simplificar a configuração do sistema, retirou-se do código-fonte de cada módulo todas variáveis que fossem passíveis de ajuste. Assim, adotou-se o seguinte formato para cada nó:

- Bibliotecas com implementação dos protocolos de comunicação;
- Módulo Python com elementos da funcionalidade básica;
- Configuração flexível de parâmetros ajustáveis do algoritmo em arquivos YAML;
- Tópicos definidos para conexão entres nós.

Algumas vantagens dessa forma de implementação incluem (1) a possibilidade de recombinação de nós para criação de sistemas ampliados, reduzidos ou com funcionalidade distinta, (2) o intercâmbio de nós para avaliação de controladores diferentes ou uso de outros modelos de hardware equivalente e (3) a possibilidade de reuso de boa parte do código com outros *framework* ou *middleware* que suportar a comunicação entre processos e o uso de arquivos de configuração. As seções seguintes detalham esses aspectos para cada nó implementado.

Utilizou-se um sistema de controle de versão ao longo de todo o processo de desenvolvimento. Assim, o registro das alterações realizadas em cada etapa pode ser consultado no repositório do projeto no *GitHub*⁴.

5.3 IMU_NODE

Este nó é responsável pela leitura dos dados de todos *3-Space Sensors* da *Yost Labs* utilizados pela *EMA Trike*. Não há fluxo de dados de entrada dentro do ROS, apenas a leitura de pacotes pela porta USB, que é feita segundo o protocolo descrito no manual. Os dados decodificados destes pacotes são disponibilizados em tópicos individuais para cada sensor conectado.

5.3.1 Protocolo

Para implementar o protocolo de comunicação do *3-Space Sensor* de forma adequada e completa, decidiu-se reaproveitar a biblioteca fornecida pela *Yost Labs*⁵. Apesar do fabricante indicar que a biblioteca seria adequada apenas para ambientes Windows, com algumas alterações foi possível adaptá-la para ambientes Linux. Isso permitiu que estivessem à nossa disposição todas as funcionalidades do sensor, implementadas de forma sistemática e recomendada pelo fabricante, de forma separada do código principal da aplicação.

5.3.2 Configurações

O arquivo de configuração *imu.yaml* foi estruturado mantendo a flexibilidade de uso do *3-Space Sensor* em mente. O usuário pode estipular uma lista de dispositivos, definindo para cada um deles diversas opções de uso. A Tabela 5.1 resume as configurações implementadas para o *imu_node*.

5.3.3 Módulo Python

O módulo *imu.py* é inicializado por meio da passagem de parâmetros presentes em um dicionário Python, correspondentes às configurações apresentadas na Tabela 5.1. São feitas chamadas de função presentes na biblioteca do *3-Space Sensor*, que é importada como módulo externo, para inicialização do sistema segundo as configurações. Além disso, foram implementadas funções auxiliares e *wrappers* para as funções de interesse na biblioteca do *3-Space Sensor*, para que pudessem ser usadas dentro do nó do ROS sem fazer referência direta ao protocolo de comunicação. Isso permitiu que a estrutura do código específico do ROS fosse simplificada.

5.3.4 Script executável

O script *imu_node.py* é inicializado como um nó do ROS, passando as configurações que foram carregadas no servidor de parâmetros para o módulo *imu.py*, que é utilizado como um objeto gerenciador de sensores. São

⁴O repositório do projeto pode ser acessado em: http://github.com/lara-unb/ema_trike

⁵A biblioteca Yost Labs 3-Space Python API Source, versão 2.0.2.3, pode ser obtida em:

https://yostlabs.com/the_downloads/manuals-and-downloads/zip/YEL_3-Space_Python_API_2.0.2.3_Src_19Jun2014.zip

Tabela 5.1: Configurações do *imu_node* disponíveis no arquivo *imu.yaml*.

Configuração	Tipo	Descrição
<code>dev_names</code>	lista de string	Lista de strings identificadores de cada dispositivo ⁶ .
<code>dev_types</code>	dicionário chave: string valor: string	Vetor associativo em que a chave é um string identificador de dispositivo listado em <i>dev_names</i> e o valor é um string indicando se o dispositivo é um <i>dongle</i> (DNG) ou sensor wireless (WL).
<code>imu_mode</code>	dicionário chave: string valor: string	Vetor associativo em que a chave é um string identificador de um sensor listado em <i>dev_names</i> e o valor é um string indicando se ele irá funcionar em modo sem-fio (wireless) ou conectado a uma porta USB (wired). Essa configuração é feita apenas para sensores (<i>dev_types</i> = WL).
<code>wired_port</code>	dicionário chave: string valor: string	Vetor associativo em que a chave é um string identificador de dispositivo listado em <i>dev_names</i> e o valor é um string indicando o endereço da porta USB. Essa configuração é feita apenas para <i>dongles</i> (<i>dev_types</i> = DNG) ou sensores conectados na porta USB (<i>imu_mode</i> = wired).
<code>wireless_dng</code>	dicionário chave: string valor: string	Vetor associativo em que a chave é um string identificador de um sensor listado em <i>dev_names</i> e o valor é o string identificador de um <i>dongle</i> listado em <i>dev_names</i> , ao qual o sensor deve ser conectado para transmissão wireless.
<code>wireless_id</code>	dicionário chave: string valor: int	Vetor associativo em que a chave é um string identificador de um sensor listado em <i>dev_names</i> e o valor é o identificador lógico do sensor que deve ser associado, segundo o mapeamento realizado previamente no <i>dongle</i> definido em <i>wireless_dng</i> .
<code>autocalibrate</code>	bool	Valor booleano indicando se o sistema deve tarar e calibrar todos sensores durante a inicialização
<code>broadcast</code>	bool	Valor booleano que ativa o modo <i>broadcast</i> dos sensores
<code>streaming</code>	bool	Valor booleano que ativa o modo <i>streaming</i> dos sensores
<code>streaming_duration</code>	int/string	Valor inteiro que determina a duração da sessão de <i>streaming</i> , com a opção adicional de se utilizar um string para indicar duração ilimitada (<i>streaming_duration</i> = unlimited)
<code>streaming_interval</code>	int	Valor inteiro que indica a periodicidade em μs com que dados devem ser enviados (um valor nulo indica que dados devem ser enviados a cada iteração do filtro de orientação, todos outros valores devem ser superiores a 1000)
<code>streaming_slots</code>	dicionário chave: string valor: lista de string	Vetor associativo em que a chave é um string identificador de um sensor listado em <i>dev_names</i> e o valor é uma lista de até oito strings indicando as informações que devem ser transmitidas para cada dispositivo em modo <i>streaming</i> .

Tabela 5.2: Tópicos publicados pelo *imu_node* no ROS.

Nome	Tipo	Descrição
<dev_name>	sensor_msgs/Imu	Orientação, velocidade angular e aceleração
<dev_name>_buttons	std_msgs/Int8	Estado dos botões

publicados tópicos para cada um dos sensores listados no arquivo de configuração, segundo a Tabela 5.2.

Define-se, então, a taxa de atualização desejada para o nó e inicia-se o laço principal, em que repete-se sempre a mesma sequência: (1) receber dados dos sensores, (2) preencher as mensagens do ROS e (3) publicar as mensagens. Este processo continua indefinidamente, até que o ROS seja encerrado.

5.4 STIMULADOR_NODE

Este nó é responsável pelo envio de comandos para um estimulador *RehaStim* da *Hasomed*. Não há fluxo de dados de saída dentro do ROS, apenas o envio de pacotes pela porta USB, que é feito segundo o protocolo descrito no manual. Os comandos são codificados segundo as mensagens recebidas em um tópico disponibilizado pelo nó.

5.4.1 Protocolo

Na ausência de bibliotecas disponibilizadas pela *Hasomed* para o *RehaStim*, decidiu-se incluir no módulo *stimulator.py* a implementação do protocolo *ScienceMode*. Por se tratar de um protocolo com apenas quatro comandos e regras simples para montagem de pacotes, conforme descrito na subseção 3.4.4, esse código adicional não aumentou significativamente a complexidade do módulo.

Utilizou-se um dicionário de comandos para agrupar os identificadores dos comandos, seus parâmetros e o tamanho do campo reservado para cada parâmetro. Assim, foi possível documentar o protocolo de forma clara no código-fonte e criar uma única função de montagem de pacotes para todos comandos.

5.4.2 Configurações

O arquivo de configuração *stimulator.yaml* contém poucas opções, tendo em vista que muitos parâmetros do estimulador são estáticos (e.g. as configurações de comunicação são fixas, a frequência de todos canais é a mesma). O usuário pode especificar a porta em que o estimulador será conectado, a lista de canais ativos, a lista de canais de baixa frequência, o fator multiplicativo que deve ser utilizado nos canais de baixa frequência, o período principal de estimulação (ou a frequência equivalente) e o período secundário de estimulação. A Tabela 5.3 resume as configurações implementadas para o *stimulator_node*.

Tabela 5.3: Configurações do *stimulator_node* disponíveis no arquivo *stimulator.yaml*.

Configuração	Tipo	Descrição
port	string	String indicando o endereço da porta USB em que o estimulador está conectado.
channel_stim	lista de int	Lista de inteiros correspondente aos canais ativos do estimulador
channel_lf	lista de int	Lista de inteiros correspondente aos canais de baixa frequência do estimulador
n_factor	int	Inteiro que define quantas vezes o ciclo de estimulação deve ser ignorado em canais de baixa frequência.
freq	int	Inteiro que define a frequência de estimulação (definida como o inverso do período de estimulação, seu uso inutiliza a configuração ts1)
ts1	int	Inteiro que define o período de estimulação t_1 em <i>ms</i>
ts2	int	Inteiro que define o intervalo entre pulsos t_2 em <i>ms</i>

Tabela 5.4: Tópicos assinados pelo *stimulator_node* no ROS.

Nome	Tipo	Descrição
cl_update	ema_common_msgs/Stimulator	Atualizações a serem realizadas no modo de lista contínua de canais, enviadas como uma lista de canais, modos, larguras de pulso e intensidades de corrente.
single_pulse	ema_common_msgs/Stimulator	Comandos de pulso único que devem ser enviados sequencialmente, segundo uma lista de canais, modos, larguras de pulso e intensidades de corrente.

5.4.3 Módulo Python

O módulo *stimulator.py* é inicializado através da passagem de parâmetros presentes em um dicionário Python, correspondentes às configurações apresentadas na Tabela 5.3. Foram implementadas funções auxiliares para montagem e envio de pacotes, além de *wrappers* para os comandos do *RehaStim*, para que pudessem ser usadas dentro do nó do ROS sem fazer referência direta ao protocolo de comunicação.

5.4.4 Script executável

O script *stimulator_node.py* é inicializado como um nó do ROS, passando as configurações que foram carregadas no servidor de parâmetros para o módulo *stimulator.py*, que é utilizado como um objeto gerenciador de um estimulador⁷. O sistema é inicializado a partir das configurações e são assinados tópicos para o recebimento de comandos de pulso único ou lista de canais⁸, conforme a Tabela 5.4.

A partir deste momento, o nó entra em estado de espera, aguardando o recebimento de uma mensagem nos tópicos assinados. A função de *callback* usa as funções definidas no módulo Python do estimulador para enviar o comando adequado, dependendo do tópico de origem da mensagem.

⁷Seria possível gerenciar múltiplos estimuladores com pequenas alterações, mas esse caso de uso não foi previsto durante o projeto.

⁸Note que foi necessário definir uma nova mensagem dentro do ROS, *ema_common_msgs/Stimulator*, uma vez que não se encontravam disponíveis mensagens padronizadas para estimuladores elétricos na versão *Indigo Igloo*.

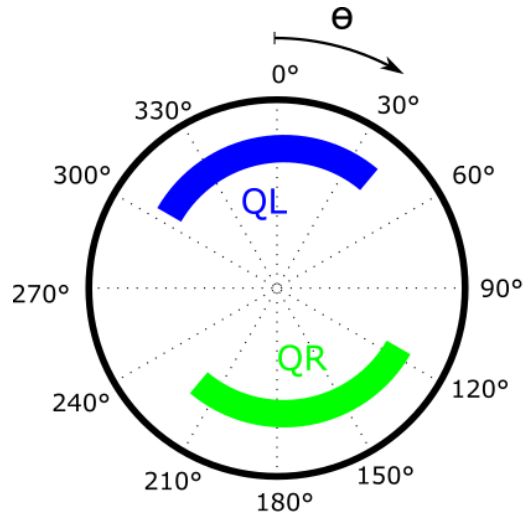


Figura 5.9: Perfis de estimulação do algoritmo de controle da *EMA Trike*.

5.5 CONTROL_NODE

Este nó é responsável pela estratégia de controle da *EMA Trike*. Recebendo os ângulos e velocidades disponibilizados pelo *imu_node*, aplica-se o algoritmo de controle escolhido para determinar que comandos devem ser enviados para o *stimulator_node*.

5.5.1 Algoritmo

Neste trabalho, a estratégia de controle adotada foi a mesma que estava presente na primeira versão do software, descrita em [75]. Vale ressaltar, porém, que a própria estrutura do sistema possibilita que sejam implementados múltiplos outros controladores intercambiáveis, que poderiam ter seu desempenho comparado. A parte importante é que, para isso, ele deve ao menos se adequar à interface apresentada pelo *stimulator_node*, para que seja possível gerar movimento através da estimulação elétrica funcional.

Em primeiro lugar, definiu-se que seria utilizado apenas um par de eletrodos por perna, associados ao quadríceps. Em seguida, foram definidos perfis de estimulação baseados em propriedades biomecânicas estáticas inerentes à pedalada em posição recumbente. A Figura 5.9 ilustra os perfis utilizados, em que a circunferência representa a posição angular do pedivela e as barras coloridas representam os intervalos em que a estimulação é ativada nas pernas esquerda (azul) e direita (verde). Estes perfis de estimulação são defasados de acordo com a cadência da pedalada — quanto maior a velocidade angular medida no pedivela, mais cedo se inicia a estimulação em cada perna. Mais informações sobre os resultados desta estratégia podem ser vistos em [75].

Uma vez definidos os perfis de estimulação, foi projetado um controlador de velocidade proporcional-integral com *anti-windup*. A partir do histórico de erro entre a velocidade angular do pedivela e uma referência pré-estabelecida, calcula-se o sinal de controle como uma porcentagem da largura de pulso máxima do estimulador. O valor de intensidade da corrente aplicada é definido pelo usuário através dos botões de um *3-Space Sensor*, que foi utilizado como controle remoto.

Tabela 5.5: Configurações do *control_node* disponíveis no arquivo *control.yaml*.

Configuração	Tipo	Descrição
speed_ref	int	Inteiro que define a referência de velocidade angular
Kp	int	Inteiro que define o valor do ganho proporcional
Ki	int	Inteiro que define o valor do ganho integral
theta_left	dicionário chave: string valor: int	Vetor associativo em que a chave é um string identificador do limite (min ou max) e o valor é um inteiro que define o ângulo mínimo ou máximo do perfil de estimulação do quadríceps esquerdo.
theta_right	dicionário chave: string valor: int	Vetor associativo em que a chave é um string identificador do limite (min ou max) e o valor é um inteiro que define o ângulo mínimo ou máximo do perfil de estimulação do quadríceps direito.
theta_shift	int	Inteiro que define o deslocamento do perfil de estimulação na velocidade de referência.

5.5.2 Configurações

O arquivo de configuração *control.yaml* contém opções referentes aos parâmetros do algoritmo de controle. O usuário pode especificar a referência de velocidade angular, o ganho proporcional e integral do controlador, os limites dos perfis de estimulação e a defasagem dos perfis de estimulação na velocidade de referência. A Tabela 5.5 resume as configurações implementadas para o *control_node*. Outros parâmetros poderiam ser incorporados ao arquivo de configuração, tais como o identificador dos canais utilizados para cada perna ou a taxa de atualização do controlador, porém decidiu-se mantê-los fixos nesta versão.

5.5.3 Módulo Python

O módulo *control.py* é inicializado através da passagem de parâmetros presentes em um dicionário Python, correspondentes às configurações apresentadas na Tabela 5.5. Foram implementadas as funções $f_x(\theta, \omega)$ e $g(\omega)$, correspondentes ao controle de fase com perfis de estimulação e o controle de velocidade proporcional-integral com *anti-windup* propostos em [75].

Por fim, incluiu-se uma função de interface chamada *calculate* para ser usada pelo nó do ROS, em que são passados os dados de posição angular e velocidade recebidos do *imu_node* para o módulo Python e são retornados os valores de largura de pulso que devem ser enviados para o *stimulador_node*.

5.5.4 Script executável

O script *control_node.py* é inicializado como um nó do ROS, passando as configurações que foram carregadas no servidor de parâmetros para o módulo *control.py*, que encapsula toda a lógica do controlador proporcional-integral com *anti-windup* com perfis de estimulação.

Para receber dados de sensores, o nó assina tópicos correspondentes aos dispositivos *3-Space Sensor* gerenciados pelo *imu_node*: um deles foi acoplado ao pedivela da *EMA Trike* fornece dados de posição e velocidade angular, enquanto o outro funciona como controle remoto e fornece o estado dos seus dois botões. Para enviar comandos de estimulação, o nó publica um tópico para enviar atualizações no modo lista contínua de canais para o *stimulator_node*. Além deste, são publicados dois nós para monitoramento da posição e velocidade

Tabela 5.6: Tópicos assinados pelo *control_node* no ROS.

Nome	Tipo	Descrição
pedal	sensor_msgs/Imu	Orientação, velocidade angular e aceleração do pedivela
remote_buttons	std_msgs/Int8	Estado dos botões do controle remoto

Tabela 5.7: Tópicos publicados pelo *control_node* no ROS.

Nome	Tipo	Descrição
cl_update	ema_common_msgs/Stimulator	Atualizações a serem realizadas no modo de lista contínua de canais, enviadas como uma lista de canais, modos, larguras de pulso e intensidades de corrente.
angle	std_msgs/Float64	Posição angular do pedivela, convertida para graus
speed	std_msgs/Float64	Velocidade angular do pedivela, no sentido da pedalada

angular do sistema, uma vez que os dados enviados pelo *imu_node* são padronizados como quaternions e contém muito mais informação do que o estritamente necessário para avaliação do sistema. A Tabela 5.6 lista os tópicos assinados⁹ e a Tabela 5.4 os tópicos publicados pelo *control_node*.

Define-se, então, a taxa de atualização desejada para o nó e inicia-se o laço principal, em que repete-se sempre a mesma sequência: (1) passar os dados de sensores mais atuais para o controlador, através do módulo *control.py*, (2) receber as larguras de pulso calculadas pelo controlador, (3) preencher as mensagens referentes aos tópicos publicados, (4) publicar as mensagens.

Os dados de sensores são armazenados em variáveis que somente são atualizadas dentro das funções de *callback* correspondentes aos tópicos assinados. Na função *pedal_callback* é feita a conversão de quaternion para ângulos de Euler, a partir da qual se extrai apenas o ângulo no eixo de interesse para o algoritmo de controle, além da conversão da velocidade angular de radianos para graus. Na função *remote_callback*, implementa-se a lógica do controle remoto, avaliando o estado dos botões para saber se o objetivo é aumentar/diminuir a intensidade de corrente do estimulador ou ligar/desligar a estimulação.

5.6 ASPECTOS ADICIONAIS DE INTEGRAÇÃO

Além da implementação dos nós executáveis no ROS, foi necessário criar a lógica de integração que permite a conexão do sistema. Isso foi feito através de *launch files*, arquivos XML utilizados na ferramenta *roslaunch*. O principal arquivo, *trike.launch*, inclui o carregamento dos arquivos de configuração no servidor de parâmetros, a lista de nós que devem ser executados, bem como ferramentas nativas que devem ser inicializadas em paralelo. Para cada nó, ainda é feita a definição da saída dos comandos de registro de eventos (*log* ou *terminal*) e configurações de reinicialização emergencial, caso sejam terminados inesperadamente.

Além do arquivo *trike.launch*, foram criados outros para execução do sistema nas diversas situações de teste e experimentação que se mostraram necessárias. O registro separado dessas configurações de inicialização do sistema permite que o processo seja simplificado, para diferentes casos de uso.

⁹Note que foram especificados os tópicos já levando em consideração a configuração adotada, em que *pedal* se refere a um sensor acoplado ao pedivela da *EMA Trike* e *remote* se refere a um sensor que é utilizado como controle remoto. Caso houvesse outros sensores, seria possível criar tópicos de ambas categorias para eles.

Por fim, apesar de ainda não ter sido utilizado, foi criado um arquivo de configuração com variáveis de interesse para o sistema como um todo, que pode ser compartilhado por diferentes módulos. Isso faz com que não seja necessário replicar valores em múltiplos arquivos de configuração nestes casos.

6

AVALIAÇÃO DO RESULTADO OBTIDO

Neste capítulo é apresentada uma análise das mudanças realizadas no software da *EMA Trike*. Inicialmente, é feito o diagnóstico de software proposto na seção 4.2 após as alterações descritas no Capítulo 5. Em seguida, realiza-se um experimento estático da plataforma com dois sujeitos saudáveis, para verificar se as mudanças realizadas levaram à degradação da funcionalidade do sistema.

6.1 DIAGNÓSTICO DA VERSÃO ATUAL DO SOFTWARE

A base de código da *EMA Trike* em março de 2017 é formada formada por seis *scripts* em linguagem Python, sendo três módulos com definições de classes e três *scripts* executáveis, programados para funcionarem em paralelo. Apesar do objetivo principal ser a execução conjunta dos *scripts*, é possível reutilizá-los em diferentes configurações. A Tabela 6.1 apresenta uma breve descrição do uso de cada *script*.

Tabela 6.1: *Scripts* presentes na base de código-fonte da *EMA Trike* em março de 2017.

Nome	Linhas	Funcionalidade
<i>control.py</i>	69	Fornecer lógica de controle proporcional-integral com <i>anti-windup</i> usada na <i>EMA Trike</i> , segundo as configurações do arquivo <i>control.yaml</i>
<i>control_node.py</i>	151	Disponibilizar a funcionalidade do módulo <i>control.py</i> dentro do ROS, se conectando aos <i>scripts</i> <i>imu_node</i> e <i>stimulator_node</i>
<i>imu.py</i>	347	Fornecer funções para gerenciar um número arbitrário de dispositivos da família <i>3-Space</i> , segundo as configurações do arquivo <i>imu.yaml</i>
<i>imu_node.py</i>	170	Disponibilizar a funcionalidade do módulo <i>imu.py</i> dentro do ROS
<i>stimulator.py</i>	342	Fornecer funções para gerenciar um estimulador <i>RehaStim</i> segundo as configurações do arquivo <i>stimulator.yaml</i>
<i>stimulator_node.py</i>	68	Disponibilizar a funcionalidade do módulo <i>stimulator.py</i> dentro do ROS

Quando são comparadas as Tabelas 4.2 e 6.1, fica claro que a base de código atual está mais adequada no que diz respeito aos princípios de projeto de software. Especificamente:

1. Observa-se grande acoplamento e coesão nos componentes do sistema, que foi decomposto e modularizado de forma satisfatória, respeitando o princípio da responsabilidade única;
2. Os componentes encapsulam e ocultam de informação de forma adequada, tornando os detalhes internos de seu funcionamento inacessíveis aos demais elementos do sistema;
3. A criação de módulos com definições de classe, que compõem a implementação da funcionalidade, utili-

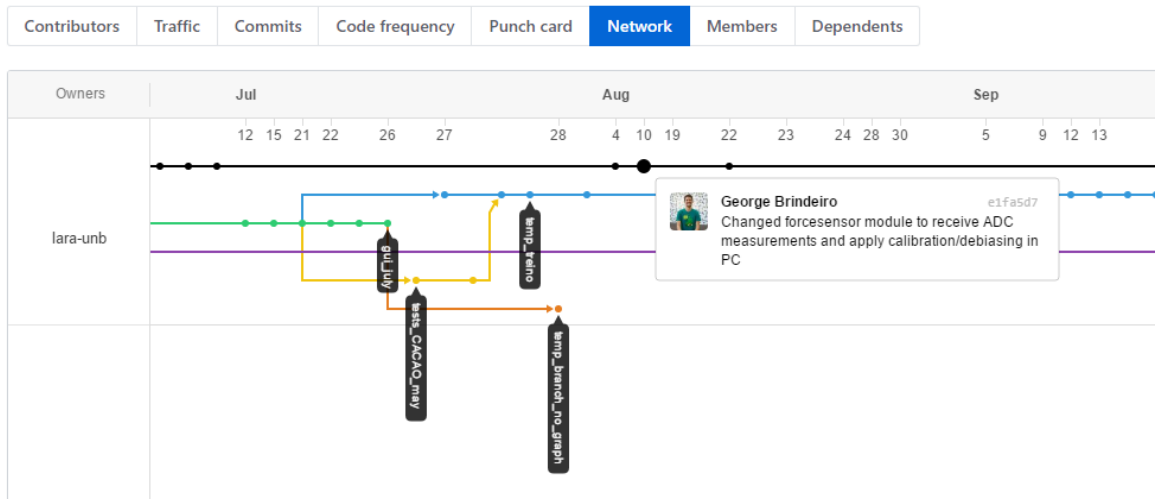


Figura 6.1: Grafo de contribuições do repositório *ema_trike*, disponível no GitHub

zados por *scripts* executáveis que funcionam como nós e definem tópicos no ROS, que funcionam como interface com os demais componentes, seguem o princípio da separação entre interface e implementação.

O projeto contém mais linhas de código no total, porém o fato de cada *script* ter menos linhas e estar mais coeso torna a base de código mais compreensível. É retomada, para fins de comparação, a análise da qualidade do software com os mesmos atributos apresentados na seção 4.2.

6.1.1 Modifiabilidade

Todas alterações realizadas desde o início deste trabalho foram registradas em um sistema de controle de versão e podem ser visualizadas, juntamente com mensagens que resumem as alterações de cada registro no histórico do projeto. Não há dúvidas quanto à obtenção da versão mais atual da base de código.

Outra vantagem do uso de controle de versão é a facilidade para colaboração entre múltiplos desenvolvedores. Todos envolvidos têm acesso a versões sincronizadas do código-fonte e ao conteúdo das modificações realizadas pelos demais. Se necessário, é possível auditar o processo de desenvolvimento e reverter mudanças dentro do sistema de controle de versão. Uma amostra desta funcionalidade é vista na Figura 6.1.

O software desenvolvido executa, sem modificações, em todas plataformas de interesse (notebook e *Raspberry Pi*). Qualquer customização para o ambiente de execução pode ser feita por meio de arquivos de configuração, sem que seja necessário manipular o código-fonte e possivelmente introduzir falhas inadvertidamente. Assim, é facilitada a implementação de melhorias em uma plataforma e sua validação em outra. Um exemplo de arquivo de configuração pode ser visto na Figura 6.2 (a forma antiga foi apresentada na Figura 4.1).

A lógica por trás dos protocolos de comunicação utilizados foi separada em bibliotecas fornecidas pelo fabricante (no caso do *3-Space Sensor*) ou funções específicas (no caso do *RehaStim*), o que torna mais simples a compreensão do que está sendo feito. A cobertura do protocolo do *RehaStim* é completa, contendo todas as possibilidades indicadas pelo fabricante no manual. A cobertura protocolo do *3-Space Sensor*, embora inclua apenas as funções de interesse, é trivialmente extensível para toda sua funcionalidade usando a biblioteca fornecida. A Figura 6.3 ilustra uma das implementações de protocolos codificada para o estimulador (novamente

```

1  ## Stimulator config
2  #
3  # port:          port address for stimulator
4  # channel_stim: list of active channels
5  # channel_lf:   list of low freq channels (NOTE: values must be in channel_stim)
6  # n_factor:     number of skipped periods in low freq channels (0-7)
7  # freq:         main stimulation frequency in Hz (NOTE: this overrides ts1)
8  # ts1:         main stimulation period in ms (1-1024.5 ms in 0.5 steps)
9  # ts2:         inter-pulse time in ms (1.5-17 ms in 0.5 steps)
10
11 port: /dev/ttyUSB0
12 channel_stim: [1, 2]
13 channel_lf: []
14 n_factor: 0
15 freq: 50
16 #ts1: 15
17 ts2: 5
18

```

Figura 6.2: Arquivos de configuração permitem que sejam feitas modificações de parâmetros do sistema sem a introdução acidental de erros no código-fonte

comparável à Figura 4.1).

O código segue uma estrutura razoavelmente padronizada, que pode ser lida e compreendida de forma sequencial, uma vez que o fluxo de execução se mostra bastante linear no código-fonte. Isso ocorre porque cada *script* é responsável por uma parte do funcionamento do sistema. A Figura 6.4 mostra a estrutura de um dos *scripts* criados para funcionar dentro da lógica do ROS.

6.1.2 Reusabilidade

Os módulos criados são utilizáveis de forma independente, em outros contextos, sem a necessidade de qualquer alteração no seu código-fonte para integração. Assim, qualquer outro projeto que se utilize do mesmo hardware é imediatamente beneficiado, já que se trata de reuso *plug and play*.

Mesmo o controlador, que é dependente da existência de sensores e atuadores adequados, pode ser adotado em sistemas com outros sistemas de medição de posição/velocidade angular (e.g. outros sensores inerciais ou *encoders*) ou outros modelos de eletroestimulador, desde que estes estejam integrados ao ROS e adotem um formato de mensagens compatível para seus tópicos.

A separação da implementação (módulo com definição de classes, como visto na Figura 6.3) e interface (*script* executável, como visto na Figura 6.4) de cada nó permite ainda um outro nível de reusabilidade. A implementação está disponível para uso inclusive fora do ROS, sempre que necessário — basta que o sistema seja capaz de executar um interpretador Python. Além disso, a estrutura do projeto em si pode ser reaproveitada, como ponto de partida para novos equipamentos.

```

121     def ccl_initialize(self):
122         cmd = 'channellistModeInitialization'
123         ident = command_dict[cmd]['id']
124
125         nf = self.n_factor # grab from config: n_factor
126         cs = self._bitset(0,[x - 1 for x in self.channel_stim]) # grab from config: channel_stim
127         clf = self._bitset(0,[x - 1 for x in self.channel_lf]) # grab from config: channel_lf
128         gt = int(round((self.ts2 - 1.5) / 0.5)) # grab from config: ts2
129         mt = int(round((self.ts1 - 1.0) / 0.5)) # grab from config: ts1
130
131         # print 'gt',gt
132         # print 'mt',mt
133         # print 'ts1',self.ts1
134         # print 'ts2',self.ts2
135
136         check = (nf + cs + clf + gt + mt) % 8
137
138         fields = {
139             'Ident': ident,
140             'Check': check,
141             'N_Factor': nf,
142             'Channel_Stim': cs,
143             'Channel_Lf': clf,
144             'Group_Time': gt,
145             'Main_Time': mt,
146             'X': 0 #filler
147         }
148
149         pkt = self._buildPacket(cmd,fields)
150
151         return self._writeRead(pkt)

```

Figura 6.3: Implementação separada dos protocolos de comunicação em formato de biblioteca, utilizando parâmetros disponibilizados em arquivo de configuração externo ao código-fonte do *script* principal

```

40 def main():
41     # define stim_manager as global so it can be accessed in callback
42     global stim_manager
43
44     # init stimulator node
45     rospy.init_node('stimulator', anonymous=False)
46
47     # get stimulator config
48     stim_manager = stimulator.Stimulator(rospy.get_param('/ema_trike/stimulator'))
49
50     # list subscribed topics
51     sub_ccl = rospy.Subscriber('stimulator/ccl_update', Stimulator, callback = callback, callback_args = 'ccl_update')
52     sub_single_pulse = rospy.Subscriber('stimulator/single_pulse', Stimulator, callback = callback, callback_args = 'single_pulse')
53
54     # initialize stimulator ccl mode
55     stim_manager.ccl_initialize()
56
57     # spin() simply keeps python from exiting until this node is stopped
58     rospy.spin()
59
60     # stop stimulator ccl mode
61     stim_manager.ccl_stop()

```

Figura 6.4: Trecho do código do nó *stimulator_node*, ilustrando a estrutura linear e específica de cada *script* executável da versão atual do software

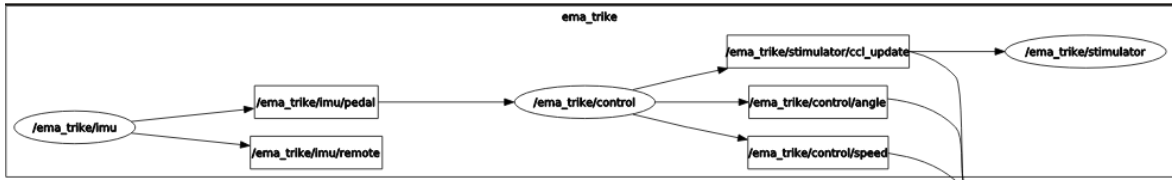


Figura 6.5: Visualização do estrutura do sistema da *EMA Trike* com a ferramenta *rqt_graph*.



Figura 6.6: Visualização das variáveis controle da *EMA Trike* com a ferramenta *rqt_plot*.

6.1.3 Verifiabilidade

O registro de dados publicados no ROS é possível sempre que desejado, sendo necessário apenas executar a ferramenta nativa *rostopic* dentro do ROS. Isso pode ser feito em um terminal à parte, mesmo depois do início da execução, ou incluído no *launch file* utilizado para iniciar o sistema. Se algum dado de interesse não está disponível no ROS para registro, sua inclusão através da criação de um novo tópico ou serviço é simples.

Embora a funcionalidade de registro de eventos ter sido usada abaixo do seu potencial, por falta de tempo para avaliar informações de interesse e incorporar mensagens no código-fonte, sua inclusão é tão simples quanto a impressão de uma mensagem no terminal.

Durante a execução, informações relevantes tais como taxa de transmissão, dados enviados e recebidos em cada tópico, variáveis disponíveis no servidor de parâmetros, estão sempre disponíveis para outros nós conectados dentro do ROS. Em tempo de execução, o usuário pode utilizar todas as ferramentas nativas do ROS para introspecção no funcionamento do software, sem que seja necessário interromper o sistema. As Figuras 6.5-6.7 ilustram algumas das possibilidades de visualização.

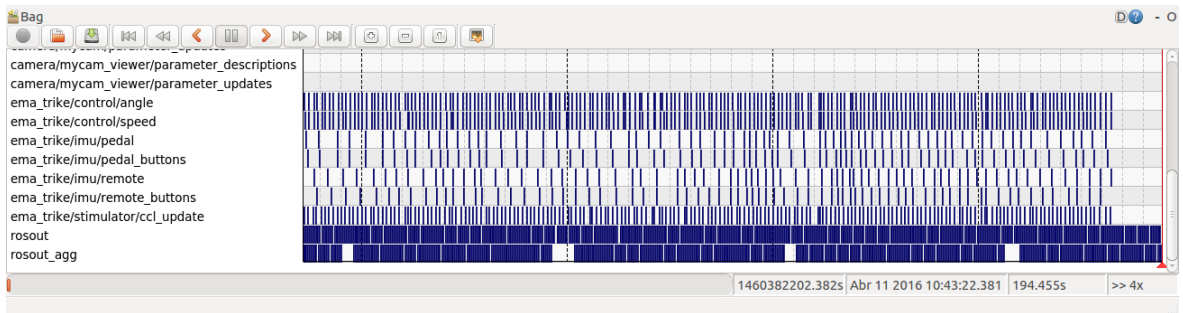


Figura 6.7: Visualização do registro de dados da *EMA Trike* com a ferramenta *rqt_bag*.

6.1.4 Proteção

Nesta versão do software houve uma redução significativa no número variáveis globais e, quando utilizadas, observa-se o seu encapsulamento destas no escopo dos módulos adequados. A transmissão dos valores de variáveis entre módulos não é feito através destas variáveis — ao utilizar-se tópicos no ROS, não é mais necessário se preocupar com questões de segurança ligadas à sincronização e temporização de *threads* concorrentes.

Implementou-se um controle maior sobre a temporização do sistema. Apesar não serem implementados processos em tempo-real, é estabelecida uma taxa de execução para cada módulo. Assim, observa-se uma maior consistência no funcionamento do sistema ao longo do tempo.

Não foi instaurado um protocolo forma de testes unitários e de integração, mas existem *scripts* de teste simples para verificar o funcionamento do hardware de forma isolada. Também não foi implementado nenhum módulo de monitoramento do sistema (*watchdog*) ou restrição no módulo do estimulador.

Apesar disso, o uso de arquivos de configuração protegem o sistema da inclusão de alterações indevidas no código-fonte, que poderiam introduzir falhas imprevistas. Além disso, o ROS ajuda o sistema a se recuperar de falhas simples, reiniciando processos que se encerrarem inesperadamente.

6.2 AVALIAÇÃO DE FUNCIONALIDADE

Para avaliação da funcionalidade da versão atual do software foram executados testes estáticos com dois sujeitos saudáveis. A *EMA Trike* foi fixada em um rolo de treinamento e cada indivíduo pedalou durante um minuto, executando em sequência a versão inicial e a versão atual do software. Para comparação, foram coletados dados dos quatro testes realizados.

Qualitativamente, observou-se desempenho similar de ambos sistemas. Os parâmetros de controle não foram ajustados individualmente para cada usuário, mas eram os mesmos que os utilizados em [75], com velocidade de referência em 300 %. Nenhum dos indivíduos relatou diferenças claras entre as duas versões quanto à resposta do sistema ou sua pedalada.

Entretanto, analisando os gráficos de controle, ângulos e velocidades gerados em cada teste (Figuras 6.8-6.11), observam-se algumas diferenças. Nos gráficos relativos ao sinal de controle, observa-se saturação no valor máximo em ambos testes da versão inicial — provavelmente trata-se de uma divergência no registro das variáveis adequadas. Nos gráficos relativos aos ângulos, observa-se um aspecto serrilhado em ambos testes da

versão atual — provavelmente a taxa de atualização dos sensores no sistema está baixa. Nos gráficos relativos à velocidade, observa-se que nenhuma das versões cumpre o objetivo de controlar adequadamente a cadência da pedalada. O Teste 4 foi o mais bem-sucedido em manter a velocidade de referência.

É possível comparar ainda a temporização das duas versões do software. Na versão inicial do software, todas as etapas são sequenciais e acopladas, executando a uma taxa de aproximadamente 30 Hz. Já na versão atual do software as etapas executam em paralelo, sendo que o controle em si executa à taxa especificada de 50 Hz. Observa-se, porém, que a ação de controle na versão atual é degradada devido ao baixo desempenho da aquisição de dados dos sensores, de cerca de 10 Hz. A Figura 6.12 mostra que, em ambos os casos, a obtenção dos dados é um gargalo para o sistema como um todo. É preciso notar, entretanto, que a variabilidade na temporização foi bastante reduzida na versão atual do software, o que é benéfico no contexto de sistemas de controle e torna o sistema mais previsível.

Neste trabalho, o objetivo estabelecido foi de melhorar atributos internos de qualidade, aqueles que mais contribuem para o trabalho de desenvolvedores, tendo em vista os potenciais benefícios para a segurança e eficácia do produto final. É claro que, do ponto de vista da engenharia, é desejável que isso não venha às custas do essencial: a funcionalidade do produto gerado. Apesar da análise feita apontar alguns pontos de melhoria no sistema, essa é justamente uma das contribuições deste trabalho: tornar visíveis as deficiências do sistema e facilitar o trabalho de abordar as melhorias necessárias para que essas deficiências sejam sanadas.

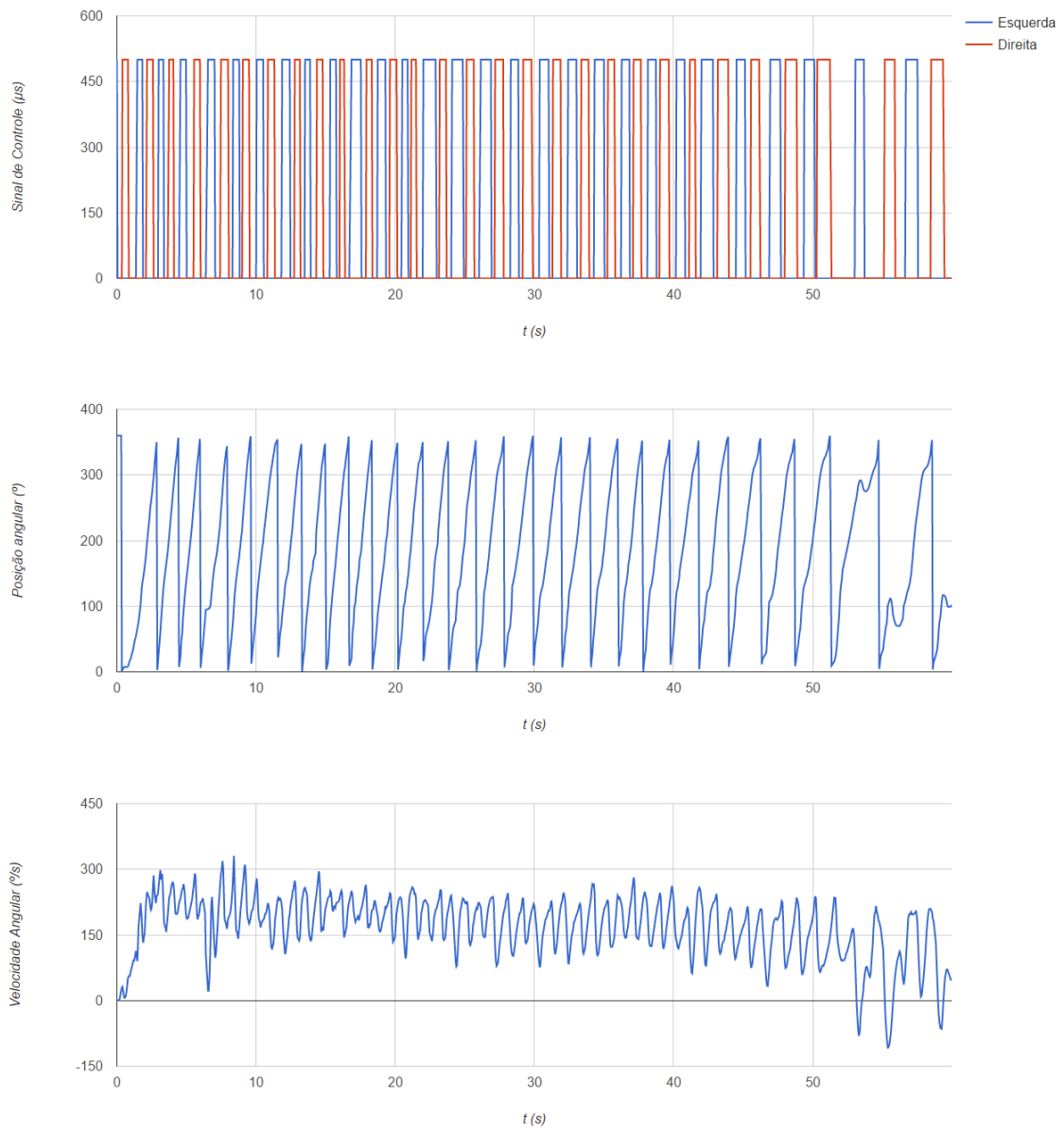


Figura 6.8: Gráficos de controle/ângulo/velocidade para o sujeito 1 com versão inicial do software (Teste 1).

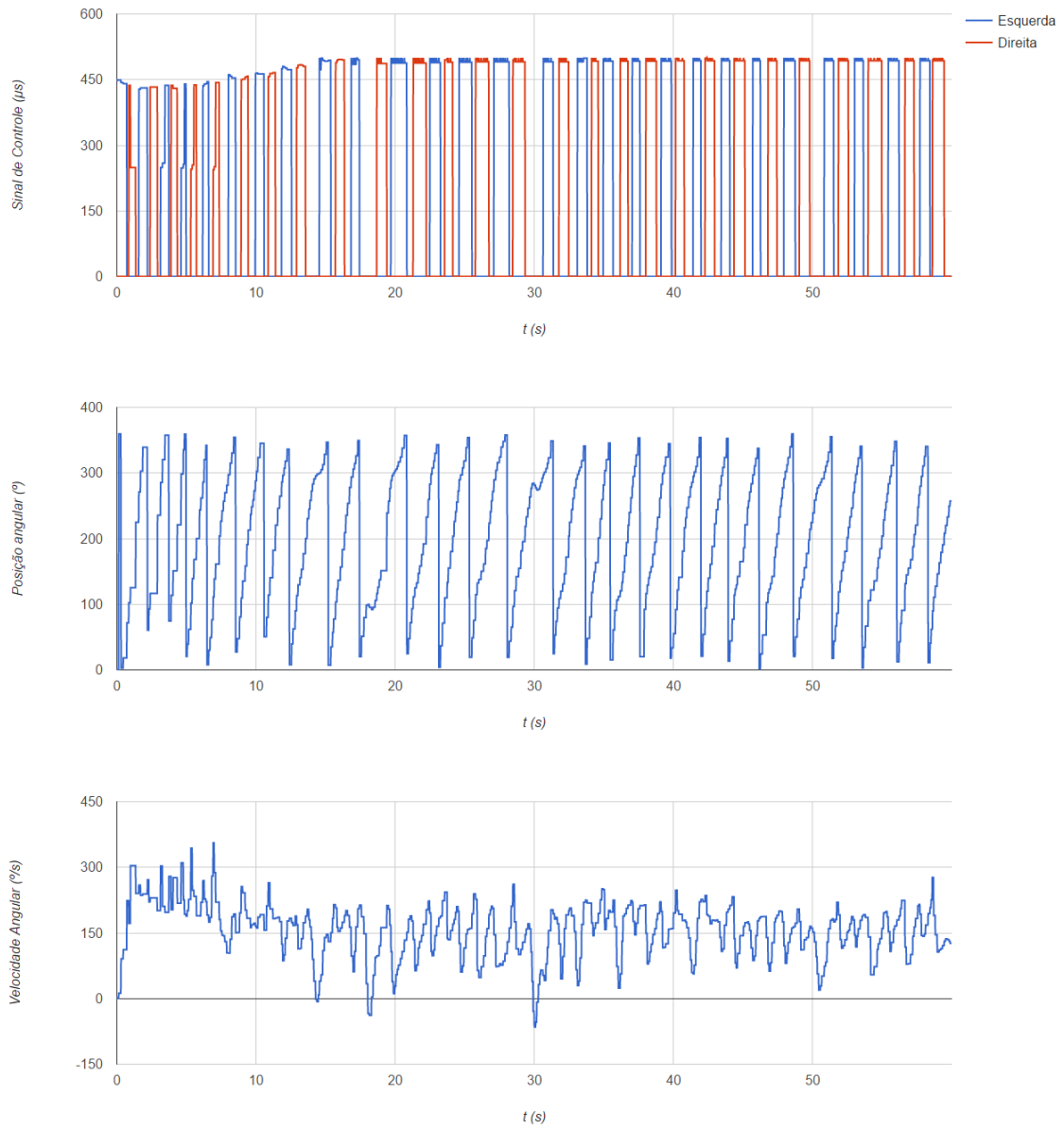


Figura 6.9: Gráficos de controle/ângulo/velocidade para o sujeito 1 com versão atual do software (Teste 2).

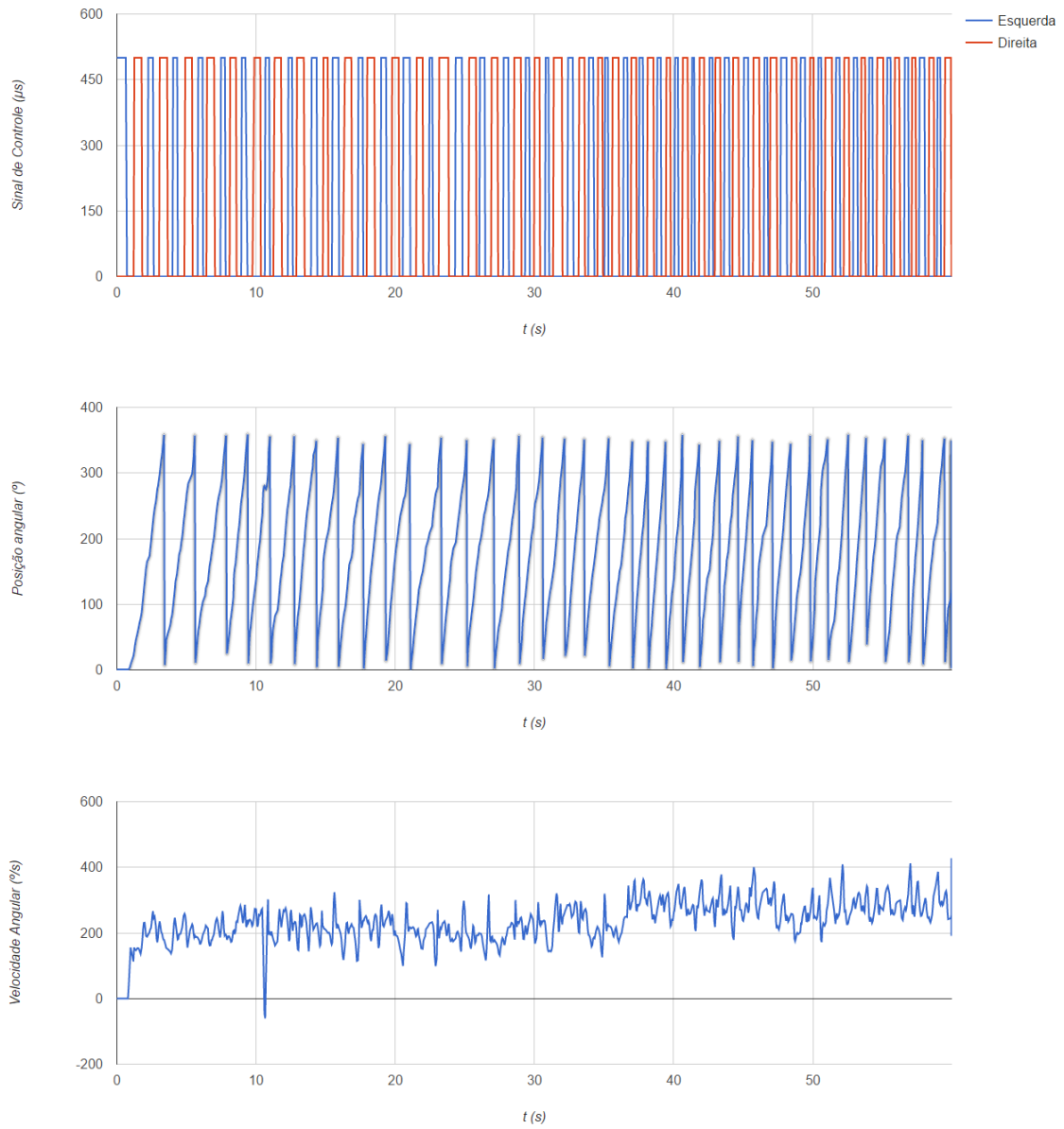


Figura 6.10: Gráficos de controle/ângulo/velocidade para o sujeito 2 com versão inicial do software (Teste 3).

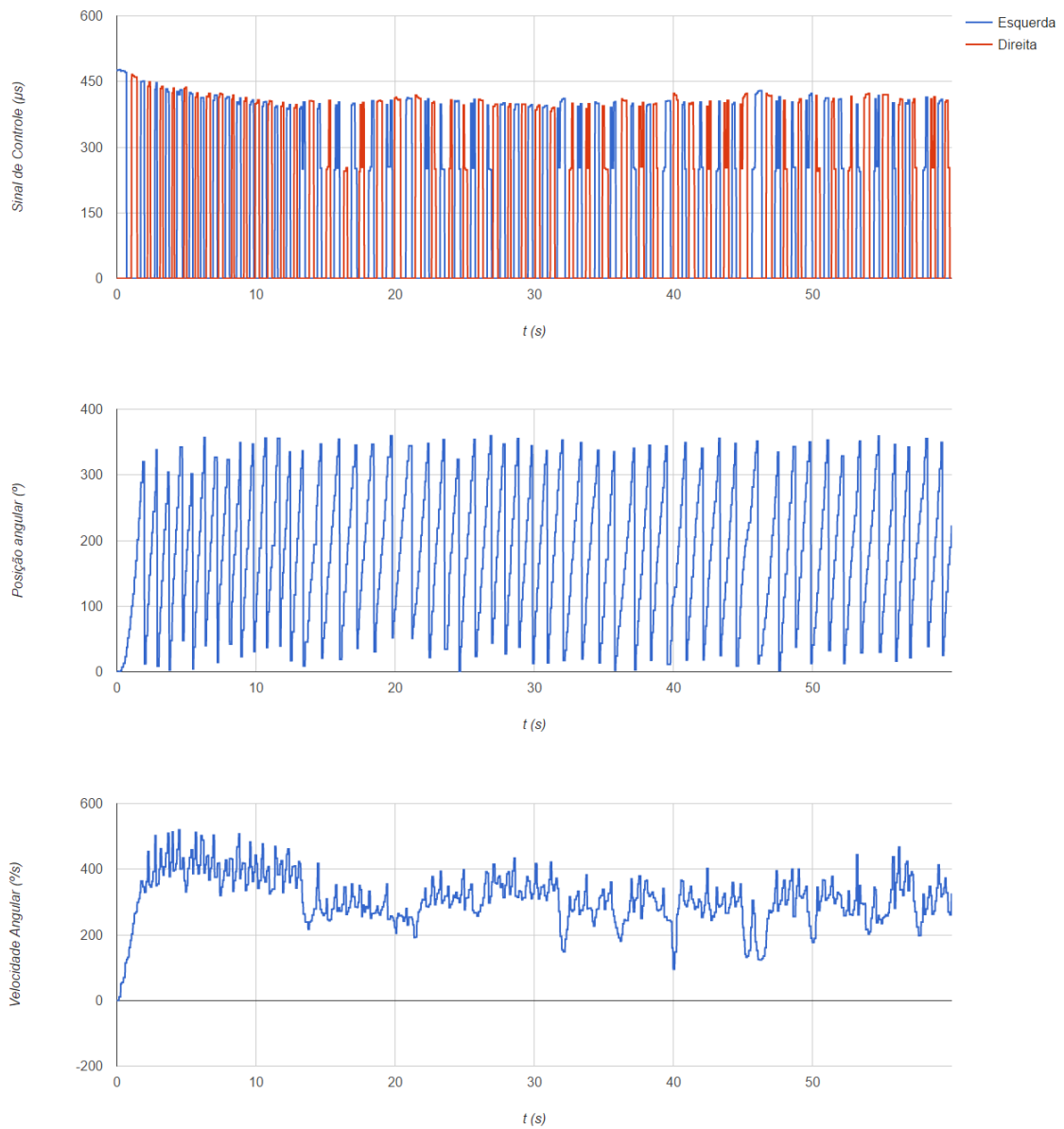


Figura 6.11: Gráficos de controle/ângulo/velocidade para o sujeito 2 com versão atual do software (Teste 4).

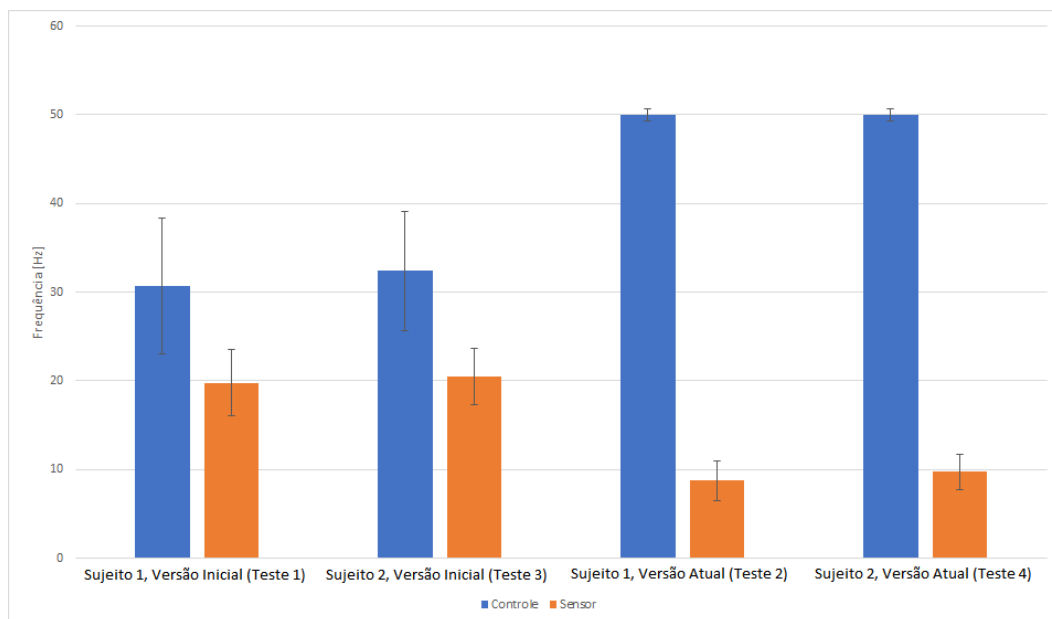


Figura 6.12: Comparação da taxa de execução do controlador e aquisição de sensores entre versões do software.

Equipamentos médicos mecatrônicos, por dependerem de uma combinação de software complexo e hardware sofisticado, incluindo componentes eletrônicos, baterias, sensores e comunicação em rede, apresentam vários desafios em relação a confiabilidade, segurança e proteção. Levando em consideração os riscos possíveis para usuários e a tendência de disseminação desses equipamentos com o avanço da tecnologia, torna-se relevante a aplicação de princípios sólidos de desenvolvimento que possibilitem a mitigação de situações de perigo.

Nesse sentido, este trabalho abordou aspectos de desenvolvimento de software que possam levar a produtos mais seguros e eficazes. Em primeiro lugar, foi feito um levantamento bibliográfico para compreender a regulamentação e as normas técnicas que fundamentam a avaliação e aprovação de equipamentos médicos ao redor do mundo, buscando-se o enquadramento de equipamentos médicos mecatrônicos nesse ambiente regulatório. Observou-se que, apesar do crescimento no número de *recalls* e relatos de incidentes causados por software, ainda não existe um processo formal de verificação ou validação do desempenho e funcionalidade de software para equipamentos médicos. O enfoque maior é dado, em geral, ao processo de desenvolvimento, em vez do produto final. Mesmo a análise do processo é incompleta, pois é fundamentada apenas em documentação gerada pelo fabricante. A principal norma técnica sobre o assunto, a IEC 62304, sequer é amplamente reconhecida por organismos reguladores. De qualquer forma, observa-se uma crescente consciência mundial sobre a importância de serem estabelecidos critérios mais rigorosos para a avaliação de software e seu processo de desenvolvimento, levando em consideração ainda aspectos como cibersegurança e softwares que atuam, por si só, como equipamentos médicos. Assim, é importante compreender esse universo mesmo na etapa de prototipação dessa classe de equipamentos, em ambientes de pesquisa e desenvolvimento.

Como principal contribuição deste trabalho, foram aplicadas diferentes técnicas e práticas de desenvolvimento de software a um equipamento específico, a *EMA Trike*, com o objetivo de melhorar seus atributos internos de qualidade: modifiabilidade, reusabilidade, verificabilidade e proteção. Sabendo que grande parte das falhas de software é introduzida em atualizações de software, considera-se que o processo de manutenção de software é tão importante quanto seu desenvolvimento — isso torna a melhoria dos atributos internos de um equipamento médico mecatrônico extremamente relevante, por reduzir esse tipo de risco e permitir a compreensão de eventuais falhas que vierem a ocorrer.

A arquitetura *dataflow* proposta para o software da *EMA Trike*, o processo de desenvolvimento que foi adotado e, em particular, o *framework* ROS, se mostraram adequados para a prototipação de equipamentos médicos mecatrônicos e se apresentam como uma contribuição secundária deste trabalho, na medida em que podem ser reaproveitados em trabalhos futuros. Apesar dos resultados preliminares indicarem uma piora no desempenho do sistema, sem perda de funcionalidade, o aumento da verificabilidade e modifiabilidade do sistema faz com que possam ser identificadas melhorias possíveis de forma mais rápida e precisa.

Por fim, este trabalho se apresenta também como uma referência básica no próprio desenvolvimento da *EMA Trike*, catalogando diversas informações regulatórias e referências técnicas importantes para que seu projeto possa futuramente ser levado ao mercado como produto comercial.

7.1 TRABALHOS FUTUROS

Como continuação do trabalho realizado, sugere-se o aprofundamento em técnicas de verificação e modelamento de software para equipamentos médicos mecatrônicos, bem como o estudo de supervisórios de segurança que possam ser aplicados para detecção de situações perigosas e redução de risco.

Este mesmo processo de desenvolvimento, incluindo a definição da arquitetura *dataflow*, a criação de módulos funcionais e *scripts* de interface no ROS, dentro de um sistema de controle de versão, pode ser adotado em outros projetos de equipamentos médicos mecatrônicos. Recomenda-se, porém, especificar com mais clareza as etapas de desenvolvimento e a documentação que deve ser gerada em cada etapa. Para que isso resulte em um salto de qualidade nos protótipos produzidos, porém, é importante que seja uma prática adotada por toda a equipe.

7.2 PUBLICAÇÕES

Além deste documento, o trabalho relacionado realizado pelo autor ao longo do período de mestrado gerou as seguintes publicações em congressos internacionais:

Título: EXPERIMENTS ON LOWER LIMB FES CONTROL FOR CYCLING

Autores: Antônio Padilha L. Bó, Lucas O. da Fonseca, Stephany R. Rodrigues, Ana Carolina C. de Sousa, Lucas L. Oliveira, George A. Brindeiro

Publicado em: *XII Simpósio Brasileiro de Automação Inteligente*

Abstract: Cycling aided by Functional Electrical Stimulation (FES) for individuals with neurological disorders, such as Spinal Cord Injury (SCI), has been shown to provide different clinical benefits. The technology is based on the coordinated stimulation of lower limbs muscles to produce cyclic leg movements. Although there are systems available today mostly in clinical settings, studies indicate that the performance of such systems with respect to efficiency, FES-induced muscle fatigue and other aspects are still limited. Based on these premises, in this work we present an innovative FES-aided tricycle for SCI subjects, as well as novel control algorithms that enable cycling cadence tracking and compensation of variable muscle response. Preliminary results on two neurologically intact subjects are presented. Analysis of the data has shown that the system is able to provide satisfactory performance, but additional techniques must be evaluated in order to further reduce the effect of muscle fatigue.

Título: SOFTWARE DESIGN CONSIDERATIONS FOR A FUNCTIONAL ELECTRICAL STIMULATION (FES) TRICYCLE

Autores: George A. Brindeiro, Lucas O. da Fonseca, Antônio Padilha L. Bó

Publicado em: *XXV Congresso Brasileiro de Engenharia Biomédica*

Abstract: This paper presents the software developed for a Functional Electrical Stimulation (FES) tricycle, which aims to enable individuals with spinal cord injury (SCI) to produce cyclic leg movements through coordinated stimulation of lower limbs muscles. Using the Robot Operating System (ROS) framework, ori-

ginally intended for and popular in the robotics research community, software modularity, extensibility and data visibility could be significantly improved. These factors support code reuse and allow for shorter software development lifecycles of computer-controlled medical devices using sensors and/or actuators.

Um artigo ainda foi submetido para uma edição especial da *IEEE Robotics and Automation Magazine* sobre o *Cyathlon*, uma competição em Zurique, na Suíça, que a *EMA Trike* foi colocada à prova em outubro de 2016:

Título: FES CYCLING FOR EMPOWERING SCI INDIVIDUALS

Autores: Antônio P. L. Bó, Lucas O. da Fonseca, Juliana A. Guimarães, Emerson F. Martins, Miguel E. G. Paredes, George A. Brindeiro, Ana Carolina C. de Sousa, Marien C. N. Dorado, Felipe M. Ramos

Submetido para: *IEEE Robotics and Automation Magazine Special Issue on Cyathlon 2016*

Abstract: Physical exercise may produce significant systemic benefits, such as reducing the risk of cardiovascular diseases. For individuals with Spinal Cord Injury (SCI), muscles that are not volitionally contracted may be activated using Functional Electrical Stimulation (FES), a technology based on the application of low energy electric impulses to restore motor function. In this paper we describe our experience in developing a system that enables persons with paraplegia to perform FESassisted cycling. Both technological development and training protocols are presented, including participation at Cyathlon 2016.

Referências Bibliográficas

- [1] J. C. Knight, “Safety Critical Systems: Challenges and Directions,” in *Proceedings of the 24th International Conference on Software Engineering - ICSE '02*, May 2002.
- [2] Center for Devices and Radiological Health (CDRH), “Medical Device Recall Report,” U.S. Food and Drug Administration (FDA), Tech. Rep., Mar. 2014.
- [3] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman, “Analysis of Safety-Critical Computer Failures in Medical Devices,” *IEEE Security & Privacy*, vol. 11, no. 4, pp. 14–26, Jul. 2013.
- [4] C. Rados, “Medical Device and Radiological Health Regulations Come of Age,” *FDA Consumer magazine*, vol. 40, no. 1, pp. 58–65, Jan. 2006. [Online]. Available: https://permanent.access.gpo.gov/lps1609/www.fda.gov/fdac/features/2006/106_cdrh.html
- [5] C. Altenstetter, “Global and Local Dynamics in Regulating Medical Technologies: The EU, Japanese and US Cases,” in *Proceedings of the ECPR Third Biennial Conference: Regulation in the Age of Crisis*, Jun. 2010.
- [6] C. Johnson and H. Sykes, “Medical Devices and Equipment: Competitive Conditions Affecting US Trade in Japan and Other Principal Foreign Markets,” United States International Trade Commission (USITC), USITC Publication 3909, Mar. 2007.
- [7] International Medical Device Regulators Forum (IMDRF). (2017, Feb.) About IMDRF. [Online]. Available: <http://www.imdrf.org/about/about.asp>
- [8] E. Perkins, “Removing pains by metallic points,” U.S. Patent 106X, 19 Feb., 1796. [Online]. Available: <http://www.datamp.org/patents/displayPatent.php?id=39623>
- [9] C. Booth, “The Rod of Aesculapios: John Haygarth (1740-1827) and Perkins’ Metallic Tractors,” *Journal of Medical Biography*, vol. 13, no. 3, pp. 155–161, Aug. 2005.
- [10] L. H. Monsein, “Primer on Medical Device Regulation - Part I. History and Background,” *Radiology*, vol. 205, no. 1, pp. 1–9, Oct. 1997.
- [11] N. F. Estrin, *The Medical Device Industry: Science, Technology, and Regulation in a Competitive Environment*. CRC Press, 31 Aug. 1990.
- [12] S. Hanna, R. Rolles, A. Molina-Markham, P. Poosankam, K. Fu, and D. Song, “Take Two Software Updates and See Me in the Morning: The Case for Software Security Evaluations of Medical Devices,” in *Proceedings of the 2nd USENIX Conference on Health Security and Privacy - HealthSec '11*, Aug. 2011.
- [13] Center for Devices and Radiological Health (CDRH) and Center for Biologics Evaluation and Research (CBER), “Design Considerations for Devices Intended for Home Use,” U.S. Food and Drug Administration (FDA), Guidance for Industry and Food and Drug Administration Staff, Nov. 2014.
- [14] Software as a Medical Device Working Group, “Software as a Medical Device (SaMD): Clinical Evaluation,” International Medical Device Regulators Forum (IMDRF), Proposed Document, Aug. 2016.
- [15] W. H. Maisel, “Medical Device Regulation: An Introduction for the Practicing Physician,” *Annals of Internal Medicine*, vol. 140, no. 4, p. 296, Feb. 2004.

- [16] Council of the European Union, “A new approach to technical harmonisation,” Resolution 85/C 136/01, May 1985. [Online]. Available: <http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:l21001a>
- [17] European Commission (EC). (2017, Feb.) CE marking. [Online]. Available: <http://ec.europa.eu/growth/single-market/ce-marking/>
- [18] D. B. Kramer, S. Xu, and A. S. Kesselheim, “Regulation of Medical Devices in the United States and European Union,” *New England Journal of Medicine*, vol. 366, no. 9, pp. 848–855, Mar. 2012.
- [19] Directorate General for Health & Consumers, “Classification of medical devices,” European Commission (EC), Guidance document MEDDEV 2.4/1 Rev. 9, Jun. 2010. [Online]. Available: <https://ec.europa.eu/docsroom/documents/10337/attachments/1/translations/en/renditions/pdf>
- [20] Council of the European Union, “Medical Devices Directive,” Council Directive 93/42/EEC, Jun. 1993. [Online]. Available: <http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A31993L0042>
- [21] Underwriters Laboratories Inc., “Medical Device Approvals in Brazil: A Review and Update,” White Paper, 2011. [Online]. Available: http://library.ul.com/wp-content/uploads/sites/40/2015/02/UL_WP_Final_Medical-Device-Approvals-in-Brazil_v7_HR.pdf
- [22] Agência Nacional de Vigilância Sanitária (Anvisa), “Manual para Regularização de Equipamentos Médicos na Anvisa,” Jun. 2010.
- [23] J. A. Tesser, “Certificação de Equipamentos Sob Regime de Vigilância Sanitária,” Slides apresentados no I Seminário ANVISA de Dispositivos Médicos, Oct. 2016.
- [24] International Organization for Standardization (ISO). (2017, Feb.) Standards. [Online]. Available: <https://www.iso.org/standards.html>
- [25] European Committee for Electrotechnical Standardization (CENELEC). (2017, Feb.) What is a standard? [Online]. Available: <https://www.cenelec.eu/aboutcenelec/whatwedo/whatisastandard/index.html>
- [26] GHTF Study Group 1, “Role of Standards in the Assessment of Medical Devices,” Global Harmonization Task Force (GHTF), Final Document GHTF/SG1/N044:2008, Mar. 2008.
- [27] International Organization for Standardization (ISO). (2017, Feb.) ISO and policy makers. [Online]. Available: <https://www.iso.org/iso-and-policy-makers.html>
- [28] M. Cheng, *Medical Device Regulations: Global Overview and Guiding Principles*. World Health Organization, 2003.
- [29] IMDRF Standards Working Group, “List of International Standards recognized by IMDRF Members as of March 2014,” International Medical Device Regulators Forum (IMDRF), Final Document IMDRF/Standards WG/N15 FINAL:2014, Sep. 2014.
- [30] —, “Recognized standards - Brazil,” International Medical Device Regulators Forum (IMDRF), Final Document (IMDRF/Standards WG/N15 FINAL:2014), Sep. 2014.
- [31] Gerência-Geral de Tecnologia de Produtos para a Saúde (GGTPS) and Gerência de Tecnologia em Equipamentos (GQUIP), “Guia orientativo às empresas do setor de produtos para saúde para o software para a saúde,” Agência Nacional de Vigilância Sanitária (Anvisa), Nota Técnica n° 04/2012, Mar. 2012.

- [32] T. da Silva Bonfim, “Regulação de Software no Brasil,” Slides apresentados no I Seminário ANVISA de Dispositivos Médicos, Oct. 2016.
- [33] Agência Nacional de Vigilância Sanitária (Anvisa), “Formulário de Petição para Cadastro de Software - Classe I e II.” [Online]. Available: <http://portal.anvisa.gov.br/registros-e-autorizacoes/produtos-para-a-saude/cadastro>
- [34] *Medical device software – Software life cycle processes*, IEC Std. 62 304, 2006.
- [35] IMDRF Management Committee, “Statement regarding Use of IEC 62304:2006 “Medical device software – Software life cycle processes”,” International Medical Device Regulators Forum (IMDRF), Final Document IMDRF/MC/N35 FINAL:2015, Oct. 2015.
- [36] K. Hall, “Developing medical device software to IEC 62304,” *Medical Device + Diagnostic Industry*, Jun. 2010. [Online]. Available: <http://www.mddionline.com/article/developing-medical-device-software-iec-62304>
- [37] A. Kumar, “Simplifying IEC 62304 compliance for developers,” *Medical Device + Diagnostic Industry*, 18 Jan. 2011.
- [38] *Medical devices – Application of risk management to medical devices*, ISO Std. 14 971, 2007.
- [39] IMDRF Management Committee, “Statement regarding Use of ISO 14971:2007 “Medical devices – Application of risk management to medical devices”,” Final Document, International Medical Device Regulators Forum (IMDRF), Final Document IMDRF/MC/N34 FINAL:2015, Oct. 2015.
- [40] *Medical devices – Quality management systems – Requirements for regulatory purposes*, ISO Std. 13 485, 2006.
- [41] IMDRF Standards Working Group, “Recognized standards - Europe,” International Medical Device Regulators Forum (IMDRF), Final Document IMDRF/Standards WG/N15 FINAL:2014, Sep. 2014.
- [42] ———, “Recognized standards - USA,” International Medical Device Regulators Forum (IMDRF), Final Document IMDRF/Standards WG/N15 FINAL:2014, Sep. 2014.
- [43] M. Mezher. (2016, Mar.) New ISO 13485: Device Companies Have Three Years to Transition. Blog post. Regulatory Affairs Professionals Society (RAPS). [Online]. Available: <http://www.raps.org/Regulatory-Focus/News/2016/03/01/24443/New-ISO-13485-Device-Companies-Have-Three-Years-to-Transition/>
- [44] U.S. Food and Drug Administration (FDA). (2017, Feb.) Medical Device Single Audit Program (MDSAP). [Online]. Available: <https://www.fda.gov/MedicalDevices/InternationalPrograms/MDSAPPilot/default.htm>
- [45] Agência Nacional de Vigilância Sanitária (Anvisa). (2017, Feb.) Piloto do Programa de Auditoria Única em Produtos para a Saúde - MDSAP. [Online]. Available: <http://portal.anvisa.gov.br/piloto-do-programa-de-auditoria-unica-mdsap>
- [46] Portal Setor Saúde. (2013, Jun.) Robô-cirurgião Da Vinci vence primeira batalha judicial. [Online]. Available: <https://setorsaude.com.br/robo-cirurgiao-da-vinci-vence-primeira-batalha-judicial/>
- [47] S. S. Arap. (2014, Jul.) Robótica pode tornar cirurgias menos invasivas e mais precisas. Hospital Sírio-Libanê. [Online]. Available: <https://www.hospitalsiriolibanes.org.br/sua-saude/Paginas/robotica-pode-tornar-cirurgias-menos-invasivas-mais-precisas.aspx>

- [48] U.S. Food and Drug Administration (FDA), “Intuitive Surgical Endoscopic Instrument,” 510(K) Number K990144, Feb. 2017. [Online]. Available: <https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfpmn/pmn.cfm?ID=K990144>
- [49] U. Jones. (2000, Jul.) FDA Clears Robotic Surgical System. Med Device Online. [Online]. Available: <https://www.meddeviceonline.com/doc/fda-clears-robotic-surgical-system-0001>
- [50] Portal Setor Saúde. (2013, Apr.) Robôs usados em cirurgias estão sob investigação nos EUA. . [Online]. Available: <https://setorsaude.com.br/robos-usados-em-cirurgias-estao-sob-investigacao-nos-eua/>
- [51] L. Greenemeier, “Robotic Surgery Opens Up,” *Scientific American*, Feb. 2014. [Online]. Available: <https://www.scientificamerican.com/article/robotic-surgery-opens-up/>
- [52] Center for Devices and Radiological Health (CDRH), “Small Sample Survey – Final Report – Topic: da Vinci Surgical System,” U.S. Food and Drug Administration (FDA), Tech. Rep., Nov. 2013.
- [53] Agência Nacional de Vigilância Sanitária (Anvisa), “SISTEMA ENDOSCOPICO ROBOTICO DA VINCI - INTUITIVE,” Registro ANVISA nº 10302860125. [Online]. Available: <https://www.smerp.com.br/anvisa/?ac=prodDetail&anvisaId=10302860125>
- [54] —, “SISTEMA CIRÚRGICO ROBÓTICO DA VINCI - INTUITIVE SURGICAL,” Registro Anvisa nº 10302860146. [Online]. Available: <https://www.smerp.com.br/anvisa/?ac=prodDetail&anvisaId=10302860146>
- [55] —, “Sistema Cirúrgico Robótico da Vinci - Intuitive Surgical,” Registro Anvisa nº 81166920001. [Online]. Available: <https://www.smerp.com.br/anvisa/?ac=prodDetail&anvisaId=81166920001>
- [56] Z. Jiang, M. Pajic, and R. Mangharam, “Cyber-Physical Modeling of Implantable Cardiac Medical Devices,” *Proceedings of the IEEE*, vol. 100, no. 1, pp. 122–137, 2012.
- [57] A. Sobrinho, A. Perkusich, L. D. da Silva, T. Cordeiro, J. Rêgo, and P. Cunha, “Towards Medical Device Certification: A Colored Petri Nets Model of a Surface Electrocardiography Device,” in *40th Annual Conference of the IEEE Industrial Electronics Society - IECON 2014*, Oct. 2014, pp. 2645–2651.
- [58] M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee, “Model-Driven Safety Analysis of Closed-Loop Medical Systems,” *IEEE Transactions on Industrial Informatics*, Oct. 2012.
- [59] Yost Labs, “3-Space Sensor Miniature Attitude & Heading Reference System,” User’s Manual v3.0_r1, Nov. 2014.
- [60] M. Pedley, “Tilt Sensing Using a Three-Axis Accelerometer,” NXP (Freescale Semiconductor), Application Note AN3461 Rev. 6, 03/2013, Mar. 2013.
- [61] H. D. Black, “A Passive System for Determining the Attitude of a Satellite,” *AIAA Journal*, vol. 2, no. 7, pp. 1350–1351, Jul. 1964.
- [62] Hasomed, “RehaStim, RehaMove,” Operating Manual Version 1.3, Dec. 2009.
- [63] T. Schauer, N.-O. Negaard, and C. Behling, “ScienceMode - RehaStim Stimulation Device,” Hasomed, Tech. Rep., Sep. 2009.
- [64] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman, 1999.

- [65] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3.0*. IEEE Computer Society, 2014.
- [66] K. E. Wiegers and J. Beatty, *Software Requirements*. Microsoft Press, 2013.
- [67] G. van Rossum, B. Warsaw, and N. Coghlan. (2001, Jul.) PEP 8 – Style Guide for Python Code. Python Enhancement Proposal. Python Software Foundation (PSF). [Online]. Available: <http://www.python.org/dev/peps/pep-0008>
- [68] J. Sussman, H. Abelson, and G. J. Sussman, *Instructor’s Manual to Accompany Structure and Interpretation of Computer Programs*. MIT Press, 1998.
- [69] A. J. Riel, *Object-oriented Design Heuristics*. Addison-Wesley Reading, 1996, vol. 335.
- [70] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003.
- [71] Center for Devices and Radiological Health (CDRH) and Center for Biologics Evaluation and Research (CBER), “General principles of software validation,” U.S. Food and Drug Administration (FDA), Guidance for Industry and Food and Drug Administration Staff, Jan. 2002.
- [72] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source Robot Operating System,” in *ICRA Workshop on Open Source Software*, vol. 3, 2009, p. 5.
- [73] T. Foote, “ROS Community Metrics Report,” Open Source Robotics Foundation (OSRF), Tech. Rep., Jul. 2016.
- [74] M. P. Tully Foote. (2010, Oct.) REP 103 – Standard Units of Measure and Coordinate Conventions. ROS Enhancement Proposal. Open Source Robotics Foundation (OSRF). [Online]. Available: <http://www.ros.org/reps/rep-0103.html>
- [75] A. P. L. Bó, L. O. D. Fonseca, S. R. Rodrigues, A. C. C. D. Sousa, L. L. Oliveira, and G. A. Brindeiro, “Experiments On Lower Limb FES Control For Cycling,” in *Proceedings of the XII Simpósio Brasileiro de Automação Inteligente (SBAI)*, Oct. 2015.