Universidade de Brasília - UnB

Departamento de Engenharia Elétrica

Trabalho de Conclusão de Curso

Plataforma para captura e estimação de movimentos em membro superior utilizando Sistemas Dinâmicos Lineares Chaveados

Autor: Leonardo Viana Valle Lins de Albuquerque

Orientador: Prof. Dr. Antônio Padilha Lanari Bó, ENE/UnB

Brasília, DF

2017

Leonardo Viana Valle Lins de Albuquerque

Plataforma para captura e estimação de movimentos em membro superior utilizando Sistemas Dinâmicos Lineares Chaveados

Monografia submetida ao curso de graduação em (Engenharia Elétrica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Elétrica).

Universidade de Brasília - UnB

Departamento de Engenharia Elétrica

Orientador: Prof. Dr. Antônio Padilha Lanari Bó, ENE/UnB

Brasília, DF

2017

Leonardo Viana Valle Lins de Albuquerque

Plataforma para captura e estimação de movimentos em membro superior utilizando Sistemas Dinâmicos Lineares Chaveados/ Leonardo Viana Valle Lins de Albuquerque. – Brasília, DF, 2017-

148 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Antônio Padilha Lanari Bó, ENE/UnB

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Departamento de Engenharia Elétrica , 2017.

1. SLDS. 2. Interface gráfica. I. Prof. Dr. Antônio Padilha Lanari Bó, ENE/UnB. II. Universidade de Brasília. III. Departamento de Engenharia elétrica. IV. Plataforma para captura e estimação de movimentos em membro superior utilizando Sistemas Dinâmicos Lineares Chaveados

CDU 02:141:005.6

Leonardo Viana Valle Lins de Albuquerque

Plataforma para captura e estimação de movimentos em membro superior utilizando Sistemas Dinâmicos Lineares Chaveados

Monografia submetida ao curso de graduação em (Engenharia Elétrica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Elétrica).

Trabalho aprovado. Brasília, DF, 06 de Dezembro de 2017:

Prof. Dr. Antônio Padilha Lanari Bó, ENE/UnB
Orientador

Prof. Dra. Carla Silva Rocha Aguiar
Convidado 1

Prof. Dr. Geovany Araújo Borges
Convidado 2

Brasília, DF
2017

Este trabalho é dedicado a todos aqueles que um dia decidiram sonhar além. Aos pequenos começos e à incansabilidade típica dos homens que mesmo na ciência são movidos por amor.

Agradecimentos

"Dêem-me um ponto de apoio e moverei a Terra." - Arquimedes

Em primeiro lugar, eu gostaria de agradecer a Deus. Sem Ele, este trabalho - e também este curso - talvez tivesse existido, mas certamente não teria sido feito com tanta paixão e entusiasmo. Desde o primeiro momento, Ele foi o motor que me levou em frente; no fim, foi Ele quem deu significado a tudo.

Em segundo lugar, não posso deixar de agradecer à minha família, que me apoiou e me incentivou a continuar lutando quando o cansaço era grande ou quando as noites já não pareciam ter fim. À minha mãe, Jossela, ao meu pai, Hermano, ao meu irmão, Rafael, e à minha segunda mãe, Antônia, um sincero obrigado por todo o carinho e motivação. Aqui eu gostaria de fazer um agradecimento especial ao meu pai, que foi fonte de inspiração constante, sempre de coração aberto para acolher minhas dúvidas e me ensinar o caminho da Engenharia. Sem ele eu nunca teria enxergado a profunda beleza que existe nos números ou a maravilha da criação que passa a ser o dom do Engenheiro.

Gostaria também de agradecer ao meu orientador, Professor Dr. Antônio Padilha, por confiar na minha capacidade para este projeto e por me dar todos os meios para completá-lo da melhor forma possível. Eu não poderia também deixar de agradecer ao Dr. Roberto Baptista, autor da tese que orientou este trabalho e que, por tantas vezes, teve a paciência de me explicar os detalhes mais profundos dos seus modelos, sempre me motivando a seguir em frente com o seu entusiasmo.

Por fim, se existe um grupo de pessoas a quem eu preciso agradecer pelos últimos seis anos da minha vida é o grupo a seguir: **Gabriel Bayomi, Luiz Felipe Campos, João Antônio Rondina, Arthur Carvalho, Stefano Dantas, Gabriel Castellano, Henrique Orefice, André Costa, Letícia Brito, Renata Lopes, Gustavo Cid e Pedro Campos**. Sem eles esse curso nunca teria sido o que foi. Aos meus eternos amigos, obrigado por terem se tornado essa segunda família que certamente vou levar para a vida inteira.

"It's hard! — Yes, I know.
But, forward! No one will be rewarded
— and what a reward! —
except those who fight bravely."
- The Way 720 (St. Josemaria Escriva)

Resumo

O movimento é um fator imprescindível para a vida plena do ser humano. Indivíduos que possuem capacidade limitada de movimentação são em grande parte dos casos submetidos a uma série de sessões de fisioterapia na busca por melhor qualidade de vida. No entanto, como a análise dos fisioterapeutas ainda é feita majoritariamente de forma qualitativa, a pesquisa por métodos mais eficientes de tratamento se torna complicada e o desempenho dos pacientes fica aquém do que poderia ser. Além disso, a subjetividade no acompanhamento oferecido a esses pacientes dificulta também o processo de auto-terapia ou telereabilitação que seriam algumas de suas alternativas naturais. Das tecnologias disponíveis no momento para o campo da fisioterapia, várias estão focadas em prover métodos de aquisição de dados dos pacientes. Entretanto, ainda cabe ao fisioterapeuta a tarefa de analisar esses dados, o que muitas vezes não é um processo fácil e nem rápido. Este trabalho propõe o desenvolvimento de uma plataforma de captura e estimação de movimentos capaz não apenas de monitorar os exercícios realizados durante uma sessão de fisioterapia como também de avaliá-los automaticamente, entregando resultados quantitativos e numéricos ao fisioterapeuta ou a outras aplicações que utilizem a unidade de processamento deste sistema. A título de auxiliar na otimização do processo fisioterapêutico, essa plataforma oferece métodos de rotulação, segmentação e extração de parâmetros do movimento a partir de noções-chave do modelo SLDS (*Switching Linear Dynamic Systems*) para estimação de movimentos humanos em membro unilateral superior com até dois graus de liberdade. A aplicação é dividida em dois módulos: uma interface gráfica desenvolvida em JavaScript, responsável por rastrear o braço do paciente em tempo real através da comunicação com sensores inerciais de medida, e uma unidade de processamento desenvolvida primordialmente em Python, cuja tarefa é processar os dados dos sensores através do algoritmo de estimação e reportar resultados à interface. Quando em integração, esses dois módulos são capazes de gerar resultados pertinentes ao processo de reabilitação e assim aprimorar o desenvolvimento fisioterapêutico. A título de validação, uma bateira de 10 testes foi realizada em diferentes modos de operação do sistema, envolvendo quatro sujeitos distintos e variações de parâmetros intrínsecos ao movimento.

Palavras-chaves: Captura de movimentos. Avaliação automática de movimentos. SLDS. Reabilitação.

Abstract

Motion is an indispensable element of a full human life. Individuals who have hindered motion capability usually undergo several physiotherapy sessions in search of better quality of life. However, because the analyses performed by physiotherapists are still mostly done by qualitative assessment, the research for more efficient treatment protocols becomes arduous and patients' progress falls short of what it could be. Furthermore, the subjective monitoring offered to these patients usually hampers the process of auto-therapy or telerehabilitation, which would be some of their natural alternatives. Amongst the available technology in the physiotherapy filed, there are many which focus on providing methods for data acquisition from patients. However, the task of assessing the data, which is not always easy or fast, still belongs to the physiotherapists. This paper proposes the development of a motion capture & estimation platform capable not only of monitoring exercises executed during physiotherapy sessions but also of automatically assessing them, delivering quantitative and numeric results to the physiotherapist or to other applications which might use this system's processing unit. In an attempt to optimize the physiotherapeutical process, this platform offers labeling, segmentation and motor parameter exctraction methods based on key notions from a SLDS model (Switching Linear Dynamic Systems) for estimating human unilateral superior limb movements with at most two degrees of freedom. The application is divided in two different modules: a graphical interface written in JavaScript, which is responsible for tracking the patient's arm in real time through comunication with inertial sensors, and a processing unit deployed primarily in Python, whose task is to process data from the sensors through the estimation algorithm and provide results to the interface. When operating together, these two modules are capable of generating pertinent results for the rehabilitation process, thus improving the physiotherapeutical development. In order to validate the system, a 10 tests batch was executed with four different subjects in different operation modes, varying several parameters which are intrinsic to human movement.

Palavras-chaves: Motion Capture. Automatic Human Movement Assessment. Switching Linear Dynamic Systems. Rehabilitation.

Lista de ilustrações

Figura 1 – Braço com espasticidade [1]. 29
Figura 2 – Tecnologias portáveis levando a fisioterapia para o ambiente pessoal [2], [3]. 30
Figura 3 – Gráfico de posição angular relativa do antebraço no tempo. 36
Figura 4 – Segmentação do movimento angular do antebraço. 37
Figura 5 – Figura de alongamento de tríceps [4] e diagrama de ciclo do movimento. 37
Figura 6 – Segmentação do diagrama de ciclo do alongamento de tríceps. 38
Figura 7 – Segunda Segmentação do diagrama de ciclo do alongamento de tríceps. 39
Figura 8 – Diagrama de ciclo do alongamento de tríceps com delimitação de fases. 40
Figura 9 – Ator em estúdio de captura de movimentos com veste equipada com marcadores [5]. 42
Figura 10 – Utilização de sistemas MOCAP para animação gráfica na indústria cinematográfica [6]. 43
Figura 11 – Microsoft Kinect [7] e Leap Motion [8], respectivamente. 43
Figura 12 – *Panoptic studio* e imagem do *Open Pose* [9], respectivamente. 43
Figura 13 – Unidade Inercial de Medida (IMU) [10]. 45
Figura 14 – *Myo armband* [11]. 45
Figura 15 – *YEI 3-Space Sensor* e *Wireless Dongle*, respectivamente [10]. 46
Figura 16 – Yost Labs 3-Space Mocap Studio [12]. 50
Figura 17 – Cena em Three.js [13] e em Babylon.js [14], respectivamente. 52
Figura 18 – Diagrama de Trellis de uma cadeia oculta de Markov [15]. 58
Figura 19 – Autômato de um sistema dinâmico linear [15]. 62
Figura 20 – Diagrama de ciclos de flexão de braço segmentado. 64
Figura 21 – Diagrama de ciclos de rotação de pulso segmentado. 65
Figura 22 – Composição dos movimentos de flexão de braço e rotação de punho com nova segmentação. 66
Figura 23 – Autômato de um sistema dinâmico linear chaveado [15]. 70
Figura 24 – Primeiro passo da estimação de uma variável (SLDS). 71
Figura 25 – Predições do Filtro de Kalman (SLDS). 72
Figura 26 – Segundas predições do Filtro de Kalman (SLDS). 72
Figura 27 – Recebimento de uma amostra (SLDS). 73
Figura 28 – Atualizações do Filtro de Kalman (SLDS). 73
Figura 29 – Poda dos caminhos com base na função custo (SLDS). 75
Figura 30 – Iteração completa da etapa de estimação (SLDS). 76
Figura 31 – Árvore de estimações para 4 amostras recebidas. 77
Figura 32 – Melhor caminho para suavização com janela de 4 iterações. 78

Figura 33 – Prospecto da plataforma "Sword Phoenix"da empresa Sword Health [3]. 84
Figura 34 – Plataforma "Sword Arya"da empresa Sword Health [16]. 84
Figura 35 – Imagem de jogo apresentada pelo site da Jintronix [2]. 85
Figura 36 – Diagrama de blocos de mais alto nível do sistema. 88
Figura 37 – Eixos de orientação padrão das IMUs (eixo-X apontando para a direita, eixo-Y para cima e eixo-Z para frente [10]. 90
Figura 38 – Diagrama de blocos do sistema com *hardware* e *driver* definidos. 91
Figura 39 – Etapas do processo de criação do modelo de braço utilizado na interface gráfica. A progressão dos passos está definida da esquerda para a direita, o terceiro modelo sendo aquele utilizado na aplicação. 94
Figura 40 – Sistema de *gimbals* [17]. 95
Figura 41 – Dois *frames* referentes à representação espacial do modelo do braço, o primeiro parado na posição inicial e o segundo durante um movimento de acenar. 98
Figura 42 – Diagrama de blocos do sistema. 98
Figura 43 – Imagens do gráfico construído para a gravação de diagramas de ciclo. Acima, logo antes da gravação; abaixo, após o fim da gravação de um movimento de uma variável apenas, no início da etapa de segmentação. 101
Figura 44 – Imagem obtida durante o processo de segmentação de um caso Multivariável. No topo e no meio, os diagramas parciais do movimento e, por último, o diagrama principal. 103
Figura 45 – *Feedback* em tempo real acerca do estado do movimento. 104
Figura 46 – Diagrama de blocos final do sistema implementado. 112
Figura 47 – Teste de velocidade univariável para movimento de quatro estados. . . 119
Figura 48 – Teste de amplitude univariável para movimento de quatro estados. . . . 120
Figura 49 – Teste de robustez univariável para movimento de quatro estados. . . . 122
Figura 50 – Teste de robustez univariável para movimento de quatro estados sob taxa de atualização de 8,5 Hz. 123
Figura 51 – Teste de robustez univariável para movimento de seis estados sob taxa de atualização de 8,5 Hz. 124
Figura 52 – Teste de velocidade univariável para movimento de seis estados sob taxa de atualização mais elevada (12 Hz). 125
Figura 53 – Teste de amplitude univariável para movimento de seis estados. 127
Figura 54 – Teste de robustez univariável para movimento de seis estados. 128
Figura 55 – Teste de invariância ao usuário realizado com três indivíduos distintos. 130
Figura 56 – Teste de velocidade multivariável para movimento de quatro estados. A curva inferior representa o ângulo entre braço e tronco; a curva superior representa o ângulo entre braço e antebraço. 132

Figura 57 – Teste de amplitude multivariável para movimento de quatro estados. A curva inferior representa o ângulo entre braço e tronco; a curva superior representa o ângulo entre braço e antebraço. 134
Figura 58 – Teste de robustez multivariável para movimento de quatro estados. A curva inferior representa o ângulo entre braço e tronco; a curva superior representa o ângulo entre braço e antebraço. 135

Lista de tabelas

Tabela 1 – Comparação entre métodos de tratamento dos problemas inerentes da avaliação de movimentos [15]. 83

Lista de abreviaturas e siglas

API	Application Programming Interface
DSSS	Direct Sequence Spread Spectrum
DTW	Dynamic Time Warping
EMG	Electromiography
GUI	Graphical User Interface
HMM	Hidden Markov Model
IMU	Inertial Measurement Unit
LDS	Linear Dynamic System
MOCAP	Motion Capture
SLDS	Switching Linear Dynamic System
TUG	Timed Up and Go
ZVC	Zero Velocity Crossing

Lista de símbolos

s	Variável de chaveamento (SLDS univariável)
S	Conjunto de símbolos s
$\mathbb{S}$	Família de conjuntos S
σ	Variável de chaveamento (SLDS multivariável)
D	Conjunto de símbolos σ
Φ	Função de mapeamento $S \rightarrow D$
k	Índice de tempo discreto
t	Índice de tempo contínuo
l	Janela de atraso para suavização
K	Número total de amostras por variável por segmento
M	Número total de estados por variável
N	Numero total de variáveis observadas
C	Função Custo do algoritmo SLDS-Viterbi
Ψ	Argumento mínimo de um conjunto de custos
x	Estado oculto no espaço de estados (posição)
y	Observação no espaço de estados (posição)
q	Ruído no estado oculto
Q	Variância no ruído do estado oculto
r	Ruído na observação
R	Variância no ruído da observação
A	Matriz de transição de estados no espaço de estados
C	Matriz de observação no espaço de estados
v	Parâmetro de velocidade constante

Σ	Parâmetro de variância
e	Erro de inovação do Filtro de Kalman
S	Variância do erro para o Filtro de Kalman
K	Matriz de ganho de Kalman
J	Matriz de ganho suavizado
Π	Matriz de transição de estados no modelo de HMM
B	Matriz de observação no modelo de HMM
π_0	Estado inicial
α	Operador de avanço no algoritmo forward-backward do modelo de HMM
β	Operador de retorno no algoritmo forward-backward do modelo de HMM
γ	Operador composto no algoritmo forward-backward do modelo de HMM
ANGLEpath	Vetor de medidas observadas
COST	Vetor de Custos dos estados
COSTestimates	Matriz de estimativas de custo
COSTpath	Vetor M-dimensional de custos
COVestimates	Matriz de estimativas de variância
COVpath	Vetor M-dimensional de variâncias
PROBestimates	Matriz de probabilidades de transição
Spath	Vetor M-dimensional de estados discretos
Strue	Vetor unidimensional da melhor estimativa dos estados discretos
Xestimates	Matriz de estimativas de posição
Xpath	Vetor M-dimensional de posições
Xtrue	Vetor unidimensional da melhor estimativa de posição

Sumário

Introdução **27**

1 FUNDAMENTAÇÃO TEÓRICA **35**
1.1 Teoria de análise de movimentos **35**
1.2 Sistemas processados para análise de movimentos **40**
1.2.1 Aquisição de dados 41
1.2.1.1 MOCAPs ópticos 42
1.2.1.2 MOCAPs não-ópticos 44
1.2.2 Processamento de dados 46
1.2.3 Interface gráfica 49
1.2.4 Interfaces de comunicação 53
1.3 Fundamentos matemáticos **54**
1.3.1 Modelos 55
1.3.1.1 Cadeias Ocultas de Markov 57
1.3.1.2 Sistemas Dinâmicos Lineares 61
1.3.1.3 Sistemas Dinâmicos Lineares Chaveados (SLDS) 63

2 ESTADO DA ARTE **81**
2.1 Contexto de metodologia **81**
2.2 Contexto de aplicação **83**

3 DESENVOLVIMENTO **87**
3.1 *Hardware* e *driver* de comunicação **88**
3.2 Desenvolvimento de *Software* **91**
3.2.1 Interface Gráfica 92
3.2.2 Camada de Processamento 105
3.3 Fluxo de Operação **113**

4 TESTES E RESULTADOS **117**
4.1 Flexão-Extensão (Univariável) **118**
4.1.1 Teste de Velocidade 118
4.1.2 Teste de Amplitude 120
4.1.3 Teste de Robustez 121
4.2 Flexão seccionada-Extensão (univariável) **123**
4.2.1 Teste de Velocidade 124
4.2.2 Teste de Amplitude 126

4.2.3 Teste de Robustez 126
4.2.4 Teste de Invariância ao Usuário 129
4.3 Alongamento de Tríceps (Multivariável) 131
4.3.1 Teste de Velocidade 132
4.3.2 Teste de Amplitude 134
4.3.3 Teste de Robustez 135
4.4 Outras considerações 136

5 CONCLUSÃO 139

REFERÊNCIAS 145

Introdução

1 Contextualização

"*Sem uma compreensão clara do sentido do movimento não se podem esperar progressos reais.*" - Jigoro Kano. A realidade de que o movimento humano é um fator essencial para o desenvolvimento da sociedade é um fato irrefutável. Desde um piscar de olhos a um aperto de mãos em uma reunião de negócios, de um alpinista escalando as montanhas à respiração de um recém-nascido, é impossível negar que o valor do movimento humano é imensurável. De fato, a capacidade motora e expressiva do homem se tornou ao longo de milhares de anos (e não parece que jamais deixará de ser) uma ferramenta fundamental para a vida.

Por centenas de anos através da história, é possível observar que a importância do controle corporal se restringiu a uma questão de dominância física, ou seja, à capacidade de submeter fisicamente oponentes que ameaçassem o *status quo* do indivíduo. Com o passar do tempo, no entanto, à medida que a sociedade foi se desenvolvendo e ganhando complexidade, a importância do controle corporal foi deixando de ser meramente uma questão de dominância física e foi ganhando território em outras áreas. A habilidade de usar expressões faciais de forma inteligente ou de se portar de maneira confiante em público são exemplos de controle corporal que ganharam valor em uma sociedade desenvolvida, e que hoje são características determinantes no sucesso de um indivíduo inserido nessa sociedade, bem como na sua qualidade de vida.

Em uma escala maior, se o desenvolvimento pessoal depende diretamente da habilidade em controlar o próprio corpo e de, através disso, modificar a realidade ao redor, então também o desenvolvimento da sociedade como um todo depende da capacidade motora de seus integrantes. Isso implica que se existem indivíduos cuja capacidade motora está prejudicada por uma ou por outra razão, é primordial que haja um esforço constante para ajudá-los a recuperar suas faculdades e/ou para criar meios através dos quais eles possam continuar a impactar a sociedade apesar de suas dificuldades.

Um grupo bastante distinto que possui essas características são pessoas que sofreram algum tipo lesão medular ou cerebral. Lesões medulares em grande parte são decorrentes de traumas que geram interrupções no fluxo de informação sensorial e motora em um determinado ponto da medula espinhal. Essas interrupções geralmente levam a uma perda sensorial total ou parcial, dependendo do local do trauma com relação à segmentação da medula [18].

Lesões cerebrais, por sua vez, são definidas de forma geral como danos ao tecido

cerebral e, como atingem o centro do sistema nervoso, podem levar a complicações que muitas vezes ultrapassam o plano motor e chegam ao plano cognitivo. Essas lesões podem ser causadas por diversas fontes: traumatismos externos, genética, intoxicação, tumores, hipóxia (falta de oxigênio), além de uma longa lista de outras possíveis causas. Uma delas, entretanto, se destaca nas estatísticas: o Acidente Vascular Cerebral (AVC).

Além de estar entre as mais comuns causas de lesão cerebral, juntamente com traumatismo cranio-encefálico, o AVC foi considerado pela OMS (Organização Mundial da Saúde) como a terceira maior causa de morte no mundo, atrás apenas de câncer e infarto. Em 2008 o Brasil foi considerado sétimo na escala de países com maior taxa de mortalidade por AVC [19] e, segundo o Ministério da Saúde, entre 2012 e 2014 o AVC foi a maior causa de morte no Brasil, com uma média de 100 mil óbitos por ano [20]. A WSO (Organização Mundial de Derrame, do Inglês *World Stroke Organization*), declara que em média 15 milhões de pessoas sofrem AVCs a cada ano no mundo inteiro e que em torno de 5 milhões não sobrevivem. Isso leva a crer que aproximadamente 10 milhões de pessoas por ano estão sobrevivendo a acidentes vasculares cerebrais e tendo de se adaptar às possíveis sequelas [21].

O termo AVC se refere a um conjunto de sintomas de deficiência neurológica causados por uma alteração significativa na irrigação sanguínea de determinada região cerebral. Essa alteração na irrigação sanguínea pode resultar em comprometimentos irreversíveis das funções cerebrais, como déficits motores, comportamentais, sensoriais e cognitivos [22]. Um AVC pode ser originado por isquemia (suprimento insuficiente de sangue) ou por hemorragia (rompimento de um vaso sanguíneo). Entretanto, 85% dos casos é reportado como sendo relativo a AVCs originados por isquemia, enquanto apenas 15% dos casos são referentes a hemorragia. Ambos possuem consequências específicas e há uma tendência de se estudar cada um de forma muito particular por conta da complexidade das estruturas cerebrais e de suas consequências. Contudo, os déficits neurológicos causados por ambos são muito semelhantes [23].

A consequência física mais comum dos acidentes vasculares cerebrais é a chamada hemiplegia, cuja definição é a paralisia total dos membros de um dos lados do corpo. Dos casos de AVC, 80% apresentam deficiência funcional motora e, associada a ela, quadros patológicos secundários que advém de mudanças na postura e esforços compensatórios sobre o membro incapacitado [23].

Dentre as perdas motoras, talvez aquela que traga mais dificuldades ao indivíduo seja a paralisia do membro superior (braço), que o impossibilita de realizar funções que, apesar de intuitivas para o cotidiano humano, são finas, complexas, e exigem destreza com as mãos. Isso ocorre pois, após o AVC, o membro paralisado sofre de um fenômeno chamado espasticidade, no qual a hipotonia (membro flácido com perda motora geral) inicial é substituída gradualmente por uma forte hipertonia (membro encurtado e flexionado, rí-

gido e tenso por excesso de tônus muscular), como mostrado na Figura 1. Essa contração involuntária dos músculos é causa constante de frustração e incapacidade funcional para o paciente [24].

Figura 1 – Braço com espasticidade [1].

Sob este panorama, é possível compreender a magnitude das complicações. Centenas de milhares de pessoas acabam incapacitadas e com profunda dependência funcional, impedidas de desempenhar diversas funções na sociedade e na vida pessoal.

Ao todo, há três maneiras do indivíduo recuperar suas funções motoras: recuperação espontânea, restituição ou compensação da função perdida. A reabilitação nesses casos só é possível por conta da capacidade monumental do cérebro humano de se adaptar a novas condições. Hoje é sabido que se uma área do cérebro é afetada gravemente e já não pode mais desempenhar suas funções de forma adequada, outras áreas não afetadas podem assumir as funções perdidas, compensando assim pelo déficit. A isso dá-se o nome de neuroplasticidade, e é com esse objetivo que pacientes com sequelas motoras de lesões cerebrais geralmente são submetidos a uma longa série de sessões fisioterápicas e de acompanhamento. [25].

Apesar dessas sessões objetivarem reabilitar ao menos em parte o membro perdido, ainda hoje os tratamentos e terapias de reabilitação para pacientes que sofreram AVC possuem um caráter bastante experimental. Incorporando um leque imenso de técnicas estudadas por pesquisadores, fisioterapeutas e médicos, diferentes abordagens são utilizadas para tentar tratar as deficiências funcionais causadas por AVC. Grande parte desses esforços se concentram em exercícios motores ou respiratórios, que visam prevenir complicações adjacentes (tanto respiratórias como musculares), alongar e fortalecer os músculos potencialmente atrofiados, estimular o reaprendizado motor e treinar pontos de sensibilidade. Esses tratamentos muitas vezes precisam ser constantes e intensivos para obter algum resultado no sentido da real reabilitação do paciente [26].

2 Definição do Problema

Uma das grandes dificuldades dessas sessões de fisioterapia é que, em grande parte das vezes, a análise do desempenho do paciente é feita visualmente pelo próprio fisiotera-

peuta. Julgar quantos graus a mais o braço ou o tronco do paciente está se estendendo ou quão mais forte ele está não é uma tarefa fácil e muito menos precisa quando se confia no olho humano. Além disso, é necessário ao profissional qualificado desempenhar uma série de análises complexas relativas ao movimento do paciente durante as sessões, o que torna o trabalho de avaliar e monitorar sua evolução clínica e física ainda mais intricado.

O mundo atual, no entanto, oferece uma vantagem enorme aos fisioterapeutas: a tecnologia. Nas últimas duas décadas uma nova revolução da tecnologia de sensoreamento e aquisição de dados tem trazido dispositivos muito mais portáveis, amigáveis ao usuário e com alta capacidade de processamento à comunidade médica clínica e hospitalar. Esses dispositivos e sistemas permitem aos próprios fisioterapeutas obter acesso a um conjunto de informações muito mais precisas acerca do desempenho do paciente, e numa taxa muito maior. Isso não só facilita e agiliza o trabalho de avaliação como também o torna mais efetivo na hora de reportar *feedbacks* ao paciente e corrigir sua técnica.

Na realidade, a portabilidade e intuitividade dessas tecnologias têm se tornado tão grandes que os próprios pacientes têm tido oportunidade de praticar em suas próprias casas, recebendo *feedbacks* confiáveis e acurados em tempo real (Figura 2). Poder "estender" as sessões de fisioterapia ao ambiente pessoal é uma vantagem considerável quando se leva em conta que muitos pacientes possuem dificuldades ou não possuem meios de se transportar até clínicas de fisioterapia, e ainda mais quando se considera que hoje em dia há bem mais pacientes para serem tratados do que profissionais qualificados para tratá-los.

Figura 2 – Tecnologias portáveis levando a fisioterapia para o ambiente pessoal [2], [3].

Assim, visando mitigar esses problemas, têm-se investido consideravelmente mais nos últimos anos no sentido de desenvolver aplicações simples e robustas que auxiliem fisioterapeutas e pacientes no processo de reabilitação motora. Isso se deve também ao fato de que o desenvolvimento computacional finalmente têm alcançado patamares de processamento que possibilitam a portabilidade e agilidade dessas soluções.

Portanto, o problema atacado por este trabalho foi o da falta que ainda existe de aplicações de estimação de movimentos que sejam portáveis, simples, eficientes, e que

possam auxiliar fisioterapeutas e pacientes no processo de reabilitação motora.

Para otimizar o protocolo utilizado pelos profissionais, no entanto, é necessário que os sistemas desenvolvidos desempenhem funções que muitas vezes são realizadas instintivamente pelo ser humano, como segmentar movimentos, distingui-los e avaliá-los. Nesse contexto, não é mais suficiente apenas monitorar o paciente com um conjunto de sensores no pretexto de obter uma melhor observação dos seus movimentos. Faz-se necessário criar uma unidade inteligente que processe esses dados e retorne *feedbakcs* úteis e que possam ser analisados mais facilmente ou aplicados diretamente ao/pelo paciente.

Para atender a essa demanda, a engenharia começou então a utilizar algoritmos de estimação e aprendizado para avaliar movimentos humanos. Esses algoritmos, de maneira geral, são "ensinados" o que é um movimento executado de forma correta e, a partir de um grupo de modelos matemáticos específicos, passam a interpretar os dados recebidos sob a ótica desse movimento referencial.

A partir do momento que se tem um algoritmo capaz de "ouvir" dados de um sensor e processá-los de forma a desempenhar aquilo que apenas um fisioterapeuta seria capaz de fazer, "compreendendo" o movimento executado e sendo capaz de avaliá-lo em contraste com um referencial pré-definido, centenas de possibilidades começam a surgir.

De fato, a tarefa de criar um sistema que desempenhe essas atividades de forma satisfatória não é fácil, e inseri-lo em um contexto no qual o paciente ou o fisioterapeuta, ditos leigos em processamento de sinais e programação, possam utilizá-lo sem grandes dificuldades, é um desafio ainda maior. Contudo, como dito anteriormente, esses são esforços importantes que potencialmente auxiliam uma larga parcela da população.

3 Objetivo

É dessa motivação que surge esse trabalho. Inspirado no desejo de auxiliar o processo de reabilitação de pacientes que sofreram AVC resultante em espasticidade unilateral de membro superior, quis-se implementar uma plataforma simples e intuitiva que pudesse ser utilizada por usuários leigos na estimação de movimentos de braço com *feedback* em tempo real.

Essa plataforma seria responsável por aprender ciclos de movimentos (executados por indivíduos saudáveis) e então, através da aquisição e processamento de sinais do paciente, estimar e disponibilizar em tempo real informações acerca do movimento praticado (pelo indivíduo lesionado) em uma interface gráfica de fácil acesso. Para o processo de estimação, o algoritmo escolhido (por uma questão de colaboração com o laboratório, suporte e conveniência) foi o algoritmo inovador de sistemas dinâmicos lineares chaveados (SLDS), que será explorado mais profundamente na seção 1.3.

Na realidade, o objetivo global deste trabalho foi que se pudesse utilizar o *feedback* gerado pelo sistema desenvolvido para completar malhas diversas no processo fisioterapêutico de reabilitação, como por exemplo malhas de eletro-estimulação para auxílio do movimento ou jogos computacionais terapêuticos. Para cumprir com esse objetivo, a interface que surgiu ao redor da plataforma, apesar de render ao sistema uma autonomia que não existiria sem ela, foi projetada para não ser essencial e poder ser destacada sem prejuízo à camada de processamento. Essa, por sua vez, pode então ser integrada livremente a outros diversos sistemas.

Em suma, este trabalho teve como objetivo principal a implementação de uma plataforma que fosse capaz de realizar, através do modelo SLDS, uma avaliação automática de movimentos de membro superior de forma a gerar, em tempo real, saídas significativas a nível fisioterapêutico, quer diretamente ou através de outras potenciais aplicações. Para isso, quatro objetivos secundários foram traçadas:

- Selecionar um *hardware* de aquisição de dados apropriado e desenvolver para ele um *driver* de comunicação;
- Selecionar a linguagem de programação apropriada e desenvolver uma unidade de processamento capaz de executar as tarefas de estimação referentes ao modelo SLDS;
- Selecionar a linguagem de programação apropriada e desenvolver uma interface gráfica para interação com o usuário, capaz de reportar em tempo real *feedbacks* acerca do processo de estimação e acerca do rastreamento dos sensores;
- Erguer uma camada de comunicação capaz de encapsular a aplicação, integrando *hardware*, unidade de processamento e interface gráfica.

4 Organização do manuscrito

Este trabalho está organizado em cinco capítulos. O Capítulo 1 dispõe de uma fundamentação teórica que detalha todo o conhecimento técnico necessário para a compreensão do projeto desenvolvido, passando por detalhes de aquisição e processamento de dados, análise e avaliação de movimentos, visualização gráfica, interfaceamento computacional e finalmente pelo modelo matemático de estimação proposto para este sistema. O Capítulo 2 discursa brevemente sobre a condição atual das tecnologias e interfaces para estimação de movimento, e sobre como o mercado e a academia estão lidando com esse desenvolvimento. O Capítulo 3 detalha todo o desenvolvimento do projeto, explicando como cada nuance da plataforma foi implementada e o seu porquê. Aqui a estrutura completa do sistema é mostrada, desde aspectos de hardware e sensoreamento à estrutura interna do software desenvolvido. No Capítulo 4, um conjunto de testes realizado com diversos

indivíduos é apresentado com seus respectivos resultados a fim de avaliar a consistência e a performance do sistema. Finalmente, no Capítulo 5, uma conclusão é feita no sentido de avaliar as diversas expansões e trabalhos futuros que podem vir a ser construídos sobre esta plataforma.

1 Fundamentação Teórica

1.1 Teoria de análise de movimentos

Quando um fisioterapeuta ou um especialista em cinesiologia (ciência que estuda o movimento humano) avaliam o movimento de um paciente, o que eles fazem é seguir um protocolo de análise que passa pela observação de vários detalhes intrínsecos à natureza do movimento. Quanto tempo o paciente levou para estender determinado músculo até sua extensão total? De fato ele alcançou a maior amplitude possível? Quão abrupta foi a transição de um movimento para outro? Houve oscilação na transição? De fato, são muitos detalhes a se observar, e esses profissionais muitas vezes precisam construir uma boa experiência antes de poder intuitivamente avaliar um movimento por completo.

Apesar de confiar no próprio *know-how* para realizarem essas avaliações, e apesar de muitas vezes realizarem-nas de maneira puramente qualitativa, existe uma ciência por trás da mecânica dos movimentos humanos que norteia os fisioterapeutas, à qual dá-se o nome de Biomecânica. Os conhecimentos sobre a forma como os músculos produzem cada tipo de movimento dão aos profissionais nessa área a capacidade de enxergar, através de um comportamento aparente, motivações musculares, ósseas ou posturais para um movimento estar sendo executado de determinada forma.

Assim, se é possível isolar as causas de um determinado movimento bem como suas consequências sobre o corpo do paciente, se é possível extrair características intrínsecas desse movimento para avaliá-lo e se é possível fazer isso através da observação, então imediatamente questiona-se se não seria possível criar um modelo lógico-matemático que descrevesse e parametrizasse esse movimento da mesma forma que o protocolo utilizado pelos fisioterapeutas, e a partir do qual então um computador poderia destacar as nuances desejadas.

Bem, para criar um modelo assim em primeiro lugar é necessário entender o que exatamente é observado pelos fisioterapeutas, ou seja, quais são as variáveis pertinentes ao movimento. A resposta é que na verdade existem três tipos de variáveis associadas ao movimento:

- **Tempo:** diz a duração de cada movimento ou o instante de transição entre movimentos distintos;
- **Variáveis Cinemáticas:** descrevem o movimento em si, e podem ser lineares (posição linear, velocidade linear ou aceleração linear) ou angulares (posição angular, velocidade angular ou aceleração angular);

- **Variáveis Cinéticas:** descrevem as causas dos movimentos (Forças e Torques).

O que é interessante é que indexando as variáveis cinemáticas ou cinéticas no tempo, é possível criar séries temporais com medidas dessas variáveis, e essas séries podem ser então armazenadas em um computador. Já aqui, surge então a necessidade de amostrar o movimento através de algum tipo de sensor ou dispositivo de amostragem. Os detalhes referentes a essa aquisição de dados serão tratados posteriormente na seção 1.2.1.

Através das séries de medidas de qualquer dessas variáveis indexadas no tempo, é possível então gerar aquele que possivelmente é o recurso mais básico na análise de dados: um gráfico no tempo. A Figura 3 mostra um exemplo de um gráfico de posição angular do antebraço (uma variável cinemática) em relação ao braço, indexado no tempo.

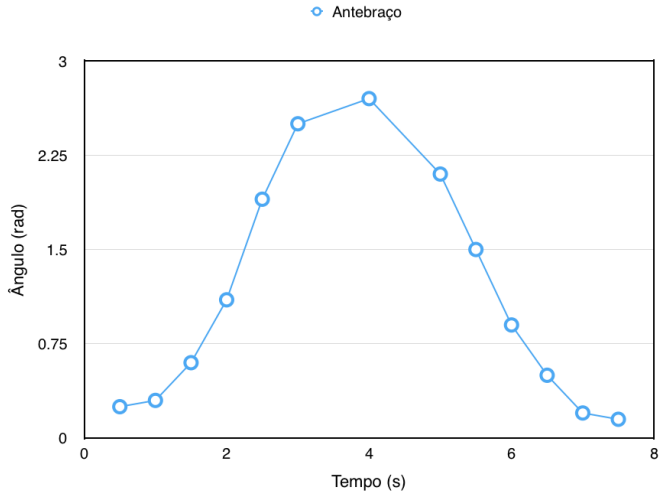

Figura 3 – Gráfico de posição angular relativa do antebraço no tempo.

Não é difícil perceber que, através da análise desses gráficos, já é possível extrair uma vasta gama de informações. Um exemplo de característica que se pode extrair do movimento descrito pela Figura 3 é a identificação de duas "partes" do movimento: uma na qual o ângulo cresce (Extensão) e outra na qual ele diminui (Flexão), bem como o instante da transição entre essas regiões. Essa segmentação do movimento está mostrada na Figura 4.

Caso a amostragem do movimento seja feita a uma taxa razoavelmente alta, também já existe um ganho enorme em resolução na análise do movimento, o que resulta em poder avaliar detalhes e oscilações ainda mais finas na performance do paciente.

Apesar de muitas vezes não ser algo visualmente calculável, esse tipo de gráfico também oferece a oportunidade de avaliar a inclinação das curvas e as nuances nas transições a partir das derivadas e segundas derivadas da série temporal de medidas. Com isso, parâmetros como velocidade e aceleração do movimento podem ser definidos para cada

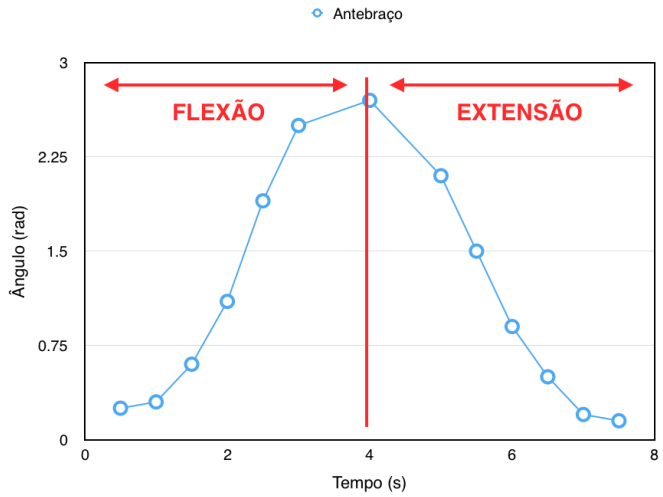

Figura 4 – Segmentação do movimento angular do antebraço.

região identificável dos gráficos. Isso permite, por sua vez, que cada uma dessas regiões seja modelada matematicamente.

Ao criar esses modelos, começa então a surgir a possibilidade não apenas de segmentar o movimento em diferentes "partes" mas também de comparar diferentes execuções de um mesmo movimento a fim de analisar variações nos parâmetros e identificar possíveis padrões. Dessa forma, a parametrização de movimentos passa a ser possível, oferecendo a chance do usuário gravar a curva de um movimento "padrão" e comparar todas as outras gravações subsequentes a esta mesma referência, o que é uma vantagem bastante poderosa por mais que pareça simples.

Grande parte dos movimentos analisados pelos fisioterapeutas, no entanto, não são movimentos que dependem de uma única variável. Na verdade, a maioria deles são movimentos compostos por diversos graus de liberdade e por variações em várias variáveis ao mesmo tempo. Um exemplo é o do alongamento de tríceps mostrado na Figura 5.

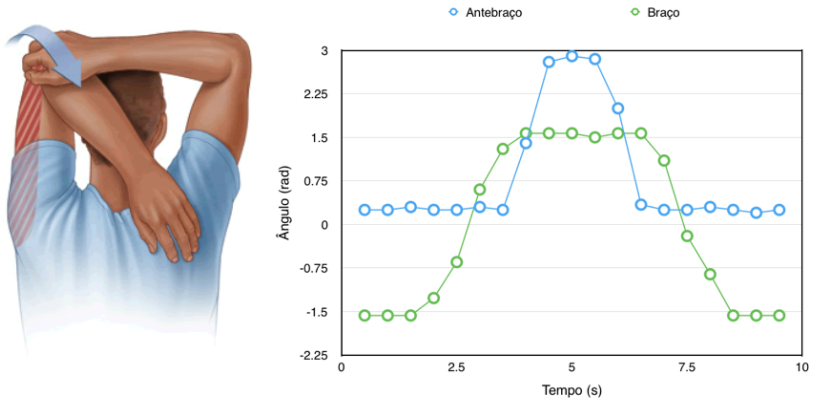

Figura 5 – Figura de alongamento de tríceps [4] e diagrama de ciclo do movimento.

No gráfico da Figura 5, pode-se perceber que há duas variáveis sendo monitoradas: o ângulo do antebraço (girando ao redor do eixo do cotovelo) e o ângulo do braço (girando ao redor do eixo do ombro). Se deseja-se segmentar esse movimento em regiões distintas, o resultado fica como na Figura 6, onde:

- **a:** braço e antebraço estendidos - tríceps relaxado;
- **b:** levantando braço;
- **c:** dobrando antebraço - estendendo tríceps;
- **d:** alongamento - tríceps estendido;
- **e:** estendendo antebraço - relaxando tríceps
- **f:** abaixando braço.

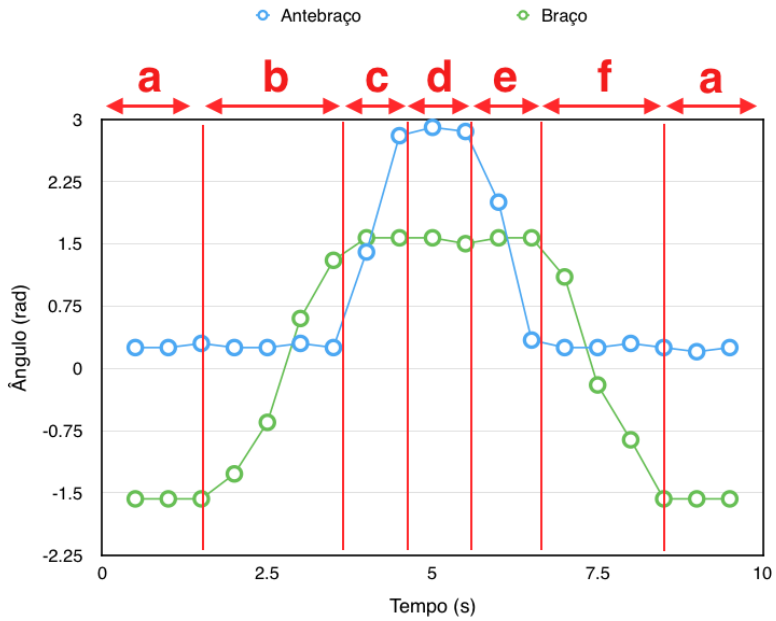

Figura 6 – Segmentação do diagrama de ciclo do alongamento de tríceps.

As informações codificadas por cada uma dessas regiões são de fato úteis, mas ainda não representam o movimento como um todo. Para isso, faz-se uma nova segmentação do movimento, cujo resultado está mostrado na Figura 7 pelas setas azuis. Com base nessa nova segmentação, obtém-se o seguinte:

- **1 ('a'):** tríceps relaxado;
- **2 ('b' e 'c'):** levando braço à posição de alongamento;
- **3 ('d'):** tríceps estendido;

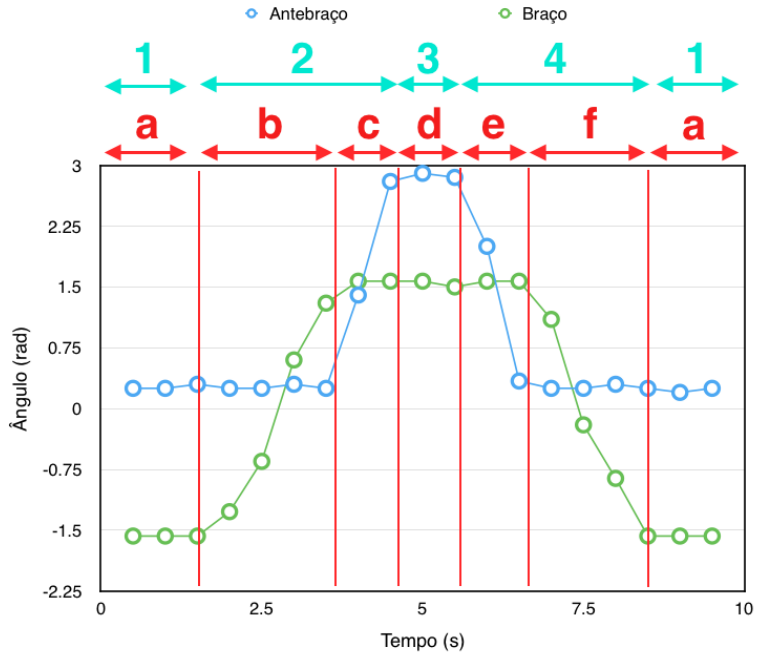

Figura 7 – Segunda Segmentação do diagrama de ciclo do alongamento de tríceps.

- **4 ('e' e 'f'):** voltando o braço para a posição relaxada.

De modo a facilitar as discussões posteriores, de agora em diante os conceitos citados acima serão referenciados de acordo com o formalismo definido por [27] e mostrado abaixo:

- **Eventos:** mudanças significativas no padrão do movimento, simbolizando a transição para outra região do gráfico;
- **Componentes:** intervalos do movimento que são delimitados pelos eventos e que seguem um mesmo padrão de comportamento;
- **Fases:** combinações de componentes para dar lugar a noções mais gerais do movimento;
- **Movimentos:** combinações de fases para caracterizar o movimentos como um todo.

No exemplo do alongamento de tríceps, observando a Figura 7 fica fácil classificar os divisores vermelhos como um conjunto de eventos, as letras como componentes e os números como fases. O movimento em questão seria fruto de uma última segmentação do gráfico, dessa vez codificando a sequência inteira como "Alongamento de Tríceps". Assim, finalmente haveria uma definição global do movimento.

Gráficos compostos como o da Figura 7, mostrando o desenvolvimento das variáveis no tempo bem como sua segmentação, são chamados diagramas de ciclos do movimento.

Para o intuito deste trabalho, as análises serão restritas aos chamados diagramas de ciclos de movimentos discretos, definidos por posturas significativamente distintas com começo e fim.

A partir deste panorama, pode-se começar a classificar movimentos semelhantes que venham surgir no futuro como na Figura 8, avaliando a fase apropriada para a região observada do gráfico de acordo com a segmentação realizada na primeira análise do movimento.

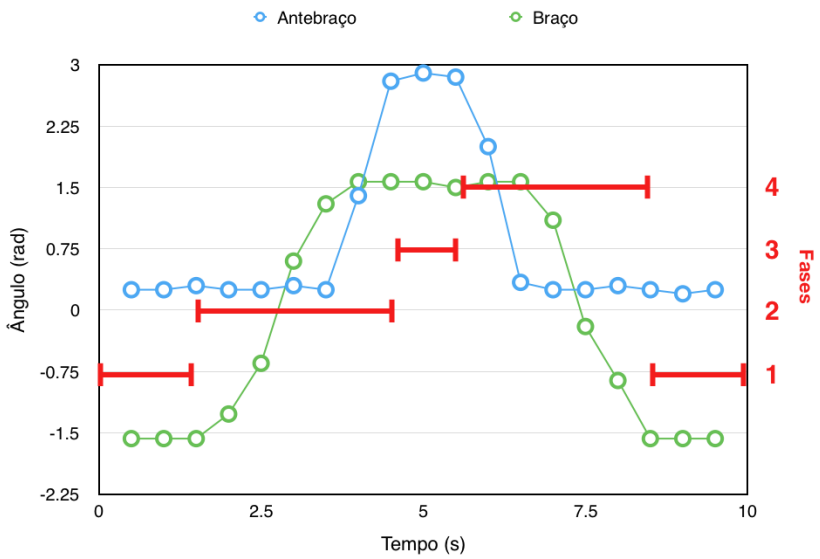

Figura 8 – Diagrama de ciclo do alongamento de tríceps com delimitação de fases.

1.2 Sistemas processados para análise de movimentos

Essa seção vem introduzir conceitos sistemáticos da análise de movimentos, adicionando à teoria vista na seção anterior um contexto prático de aplicação.

É importante nesse ponto dizer que a maioria dos sistemas de processamento são divididos em dois ou três módulos:

1. *Hardware* de aquisição de dados;
2. *Software/ Firmware* de processamento de dados;
3. Interface para interação com o usuário e exibição de resultados (opcional)

As seções a seguir tratarão especificamente de cada um desses módulos, discursando sobre as opções possíveis para cada um e sobre suas características básicas, de forma a criar uma noção geral acerca das etapas de planejamento de um sistema de processamento capaz de realizar análises de movimentos.

Além disso, cada seção trará, já como um exemplo de aplicação, as escolhas feitas no planejamento do sistema criado neste trabalho.

1.2.1 Aquisição de dados

Um ponto importante discutido na seção 1.1 foi que a análise de um movimento pode ser tanto qualitativa quando quantitativa. Na análise qualitativa, o fisioterapeuta confia na própria experiência e, julgando a natureza do movimento pela sua funcionalidade, indica ou não uma mudança na forma como o paciente está realizando aquele movimento. Já a análise quantitativa depende diretamente da capacidade que se tem de modelar e parametrizar o movimento em questão a partir de medidas concretas e numéricas da variável que se quer avaliar [15]. É só a partir dessas medidas indexadas no tempo que se pode gerar os diagramas de ciclo do movimento, para então sugerir ao paciente mudanças baseadas em resultados matemáticos.

Assim, a primeira pergunta que surge é: como obter medidas indexadas no tempo da variável em questão? Se o movimento que se deseja analisar é o alongamento de tríceps, por exemplo, como obter medidas numéricas do ângulo do antebraço ou do braço do paciente enquanto ele pratica o movimento? A resposta é que, para isso, é necessário que se tenha à disposição algum dispositivo eletrônico que seja capaz de amostrar esses ângulos, ou seja, um dispositivo que seja sensível à variação espacial do eixo do braço ou antebraço e que tenha a capacidade de transmitir os resultados dessas variações para uma unidade de processamento.

Como as variáveis de interesse são, para o objetivo deste trabalho, variáveis cinemáticas de posição, velocidade e/ou aceleração, então o foco dessa seção será o estudo de diferentes sensores e métodos que permitam amostrar esses tipos de dados.

Na indústria, o termo utilizado para processos de amostragem e gravação de movimentos de pessoas, animais ou objetos é "Captura de Movimento", do Inglês *Motion Capture*. O mais comum é que esses processos e os sistemas que os executam sejam referenciados como *MOCAPs*, uma abreviação do termo em Inglês.

Os primeiros sistemas de captura de movimento surgiram em torno dos anos 1970 - 1980 com pesquisadores de biomecânica interessados em registrar movimentos de atletas. Rapidamente, o potencial desses sistemas se espalhou por diversas áreas e, submetidos a um desenvolvimento vertiginoso, hoje eles estão presentes não apenas nos esportes como também na indústria cinematográfica, na indústria de jogos, nas ciências médicas (como a fisioterapia), em aplicações de rastreamento de animais e objetos, no desenvolvimento de realidades virtuais, na robótica, em estudos de visão computacional e em diversos outros campos.

MOCAPs estão divididos basicamente em dois grupos: ópticos e não-ópticos.

1.2.1.1 MOCAPs ópticos

MOCAPs ópticos são sistemas de captura de movimento baseados em processamento de imagem. A sua estrutura clássica é a de um conjunto de câmeras que monitora o agente e cuja composição das filmagens gera uma triangulação da posição das juntas sendo rastreadas em seu corpo. A identificação dos pontos estudados no corpo do agente pode ser feita de duas formas: através de marcadores ou sem eles.

Marcadores são pequenos dispositivos acoplados ao corpo do agente em pontos específicos e que, de uma forma ou de outra, indicam para o software de processamento de imagem que o ponto ao qual ele está acoplado está aparecendo naquele *frame*. A Figura 9 mostra um exemplo de agente utilizando uma veste com marcadores.

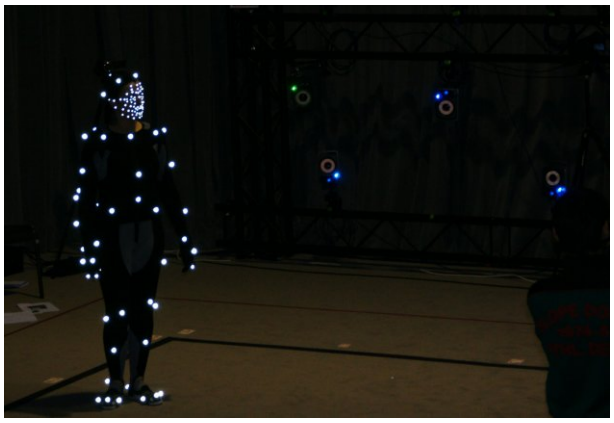

Figura 9 – Ator em estúdio de captura de movimentos com veste equipada com marcadores [5].

Os marcadores podem ser passivos, ativos, semi-passivos, modulados, e ainda de outros tipos. Como não é o foco deste trabalho analisar cada um desses modelos de marcador, basta que se saiba que é a partir da identificação desses marcadores nas cenas filmadas por múltiplas câmeras que é então possível triangular em tempo real a posição de cada um deles e recriar, em computador, um modelo computacional do agente sendo observado, indicando precisamente a posição numérica de cada junta no espaço tridimensional [28]. Este modelo pode inclusive ser alterado para representar outra aparência, que é a técnica utilizada pela indústria cinematográfica para criação de personagens inumanos nos filmes (Figura 10).

Duas empresas que estão à frente dos sistemas de ponta de MOCAPs ópticos com marcadores são a Vicon [29] e a Qualisys [30], que desenvolvem hardware e software de aquisição para diversas aplicações [15].

Os sistemas de MOCAP ópticos que não usam marcadores são fruto do recente desenvolvimento de algoritmos de aprendizado de máquina e visão computacional. Esses sistemas permitem identificar em filmagens corpos e silhuetas pré-definidas sem qualquer

Figura 10 – Utilização de sistemas MOCAP para animação gráfica na indústria cinematográfica [6].

dispositivo de rastreamento ou equipamento especial acoplado ao corpo do agente. Isso representa uma grande vantagem uma vez que o custo e a falta de praticidade dos marcadores é eliminada. Por outro lado, a complexidade do sistema de processamento se torna bem maior.

Exemplos de MOCAPs ópticos sem marcadores são o Microsoft Kinect, o Leap Motion (Figura 11) e o recente projeto da Universidade Carnegie-Mellon intitulado *Open Pose*, realizado com o *Panoptic Studio* (uma câmara isolada para gravação dos agentes) no intuito de gerar uma biblioteca aberta de estimação de movimentos (Figura 12).

Figura 11 – Microsoft Kinect [7] e Leap Motion [8], respectivamente.

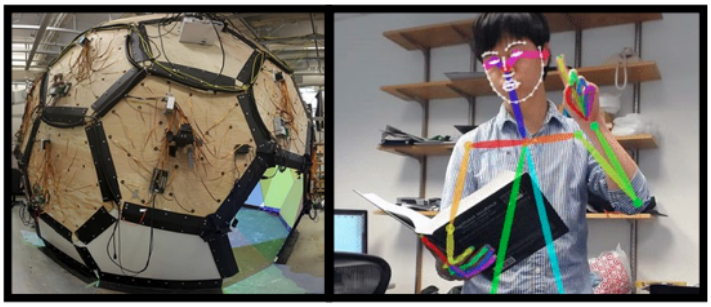

Figura 12 – *Panoptic studio* e imagem do *Open Pose* [9], respectivamente.

A desvantagem de MOCAPs ópticos é que, além de ser necessário possuir uma poderosa ferramenta de filmagem, a área de cobertura do sistema é proporcional ao número de câmeras. Isso implica que o agente não pode abandonar a área delimitada pelo campo de visão das câmeras ou então os dados não serão mais adquiridos [28].

1.2.1.2 MOCAPs não-ópticos

Como o próprio nome já diz, MOCAPs não-ópticos utilizam outros métodos para aquisição de dados que não o processamento visual do agente. Isso traz uma vantagem imediata ao sistema: a independência de uma área de cobertura óptica.

É claro que, independente do *hardware* de aquisição utilizado, a interface com a unidade de processamento vai exigir alguma limitação espacial, seja por conta da cobertura de rede sem fio (WiFi, Bluetooth, etc) ou pela extensão do cabeamento dos sensores. Contudo, a área de cobertura nesses casos, especialmente quando é utilizada alguma rede sem fio, é consideravelmente maior do que aquela a que MOCAPs ópticos submetem os agentes.

Existem três tipos principais de MOCAPs não-ópticos: Magnéticos, Mecânicos e Inerciais [28].

De forma sucinta, MOCAPs magnéticos são dispositivos equipados com três bobinas ortogonais que geram campos magnéticos através dos quais se pode calcular orientação e posição dos sensores, e MOCAPs mecânicos são utilizados em estruturas acopláveis ao corpo do agente, como exoesqueletos ou órteses, nas quais se inserem sensores mecânicos de posição [28].

MOCAPs inerciais, no entanto, são um pouco mais robustos e incluem sensores inerciais em miniatura, geralmente processados e equipados com modelos matemáticos e algoritmos de filtragem. Em geral, MOCAPs inerciais, ou IMUs (Unidades Inerciais de Medida, do Inglês *Inercial Measurement Units*), são o acoplamento de três sensores distintos: um acelerômetro, um magnetômetro e um giroscópio, dos quais se colhem os dados desejados. Em geral esses dados são fundidos por algum filtro (um filtro de Kalman Estendido, por exemplo) para gerar estimativas mais acuradas de posição. Apesar de terem uma resolução tipicamente baixa e de serem sucetíveis a perturbações eletromagnéticas, IMUs entregam informação completa sobre os eixos de movimento do agente além de serem altamente portáveis e de possuírem uma instalação extremamente simples (Figura 13). Grandes fabricantes de IMUs são empresas como Delsys, YEI, Xsens, entre outros [15].

Por fim, é interessante citar um último tipo de dispositivo que se talvez se encaixe nessa categoria por permitir também a aquisição de dados relativa ao movimento do agente: os sensores de Eletromiografia (EMG). Apesar desses sensores não codifica-

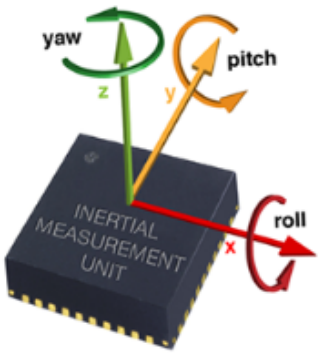

Figura 13 – Unidade Inercial de Medida (IMU) [10].

rem variáveis cinemáticas diretamente, eles fornecem informação acerca da intensidade dos impulsos elétricos percorrendo os músculos do agente. Esses impulsos, por sua vez, codificam movimentos individuais que podem ser analisados.

Atualmente o mercado tem tentado incorporar o uso de sensores de EMG a soluções de entretenimento, negócios e saúde pela sua praticidade e principalmente pelo caráter inovativo que eles trazem ao cotidiano. Entretanto, como há a necessidade de uma camada extra de processamento para filtrar os impulsos elétricos e mapear suas relações com movimentos observáveis, essas soluções ainda estão "engatinhando".

Um exemplo de aplicação envolvendo sensores de EMG que tem tido bastante sucesso no mercado é o *Myo armband*, que utiliza uma rede de sensores ao redor do antebraço do agente para monitorar impulsos, codificar movimentos e assim controlar aplicativos de computador, máquinas industriais, entre outros (Figura 14).

Figura 14 – *Myo armband* [11].

Cada um desses tipos de MOCAP possui uma interface própria que permite acessar os dados colhidos, seja por meio de protocolos cabeados, através de redes sem fio ou por meio de aplicativos dedicados. Quando da escolha de um desses dispositivos, no entanto, é importante lembrar que deve-se levar em conta não apenas critérios como resolução ou área de cobertura, mas também sua complexidade de instalação, sua conveniência de uso, custo e, caso o projeto venha a atuar em conjunto com pacientes em condições de saúde alterada, também conveniência, praticidade e capacidade de higienização. Nem sempre o

sistema mais avançado é o melhor para todas as situações.

Para este trabalho, dado o escopo enxuto da solução e a escolha por um *hardware* que fosse portável e de fácil instalação, escolheu-se utilizar como dispositivos de aquisição de dados as IMUs *YEI 3-Space Sensors* no modo de comunicação sem fio com uso de um Dongle do mesmo pacote (Figura 15). MOCAPs ópticos seriam impraticáveis nesse caso e, dada a disponibilidade dessas IMUs em laboratório, a decisão de utilizá-las para o projeto foi imediata.

Figura 15 – *YEI 3-Space Sensor* e *Wireless Dongle*, respectivamente [10].

Os detalhes referentes às especificações dos *YEI 3-Space Sensors* estão documentados em [10]. Parte desses detalhes será explorada na seção 3.1.

1.2.2 Processamento de dados

Uma vez adquiridos os dados referentes às variáveis cinemáticas em observação, esses dados devem ser lidos por uma unidade de processamento. Essa unidade é a responsável por armazenar, tratar e analisar esses dados até que se possa gerar resultados coerentes para o usuário.

Essa unidade pode estar localizada em dois tipos de ambientes de desenvolvimento diferentes: em um computador clássico ou em um *hardware* dedicado como um microcontrolador ou alguma outra plataforma microprocessada. Entretanto, grande parte das ferramentas de processamento exigem um alto poder computacional, característica limitada em muitos microcontroladores do mercado. Isso praticamente elimina essas alternativas dos grandes sistemas de processamento, jogando-a para o final da fila. Além disso, apesar de já haver grande portabilidade de sistemas operacionais em tempo real (RTOS) para microcontroladores, características como *multi-threading* e esquemas diversos de comunicação ainda são razoavelmente trabalhosos de se implementar.

É importante notar que muitos microcontroladores são usados como *gateway* para os dados amostrados pelo *hardware* de aquisição, principalmente quando há vários sensores transmitindo ao mesmo tempo. Entretanto, o processamento é feito, via de regra, em ambientes equipados com um sistema operacional capaz de maior poder computacional.

Outra opção que começa a ser estudada nesses casos é a de *softwares* embarcados que de fato possuem mais poder de processamento e que podem ser portados para plataformas dedicadas. Ainda assim, apesar dessa ser uma possibilidade, atualmente o mercado ainda não incluiu essas alternativas no universo do processamento de dados em larga-escala.

Considerando então a abordagem mais clássica, na qual o ambiente de processamento é um computador com poder computacional à altura da tarefa, quais são as ferramentas de processamento de dados mais utilizadas na indústria?

A primeira delas é uma linguagem de programação chamada "R". Em 2014, em torno de 61% dos desenvolvedores diziam utilizar "R" para aplicações envolvendo processamento de dados, o que rendeu-lhe o primeiro lugar na escolha de profissionais associadas a este campo. Ela é conhecida por ser uma excelente ferramenta para lidar com conjuntos de dados complexos e modelamento, e possui uma vasta gama de bibliotecas. Outra forte característica é sua capacidade de gerar gráficos e tabelas com visual arrojado para apresentação de resultados [31].

Entretanto, apesar de atraente, "R" já não possui um desempenho tão bom em larga-escala. Segundo Michael Driscoll, CEO da Metamarkets: "R é mais uma ferramente para rascunhar, e não para construir. Você não vai encontrar R no coração do algoritmo ranqueador de páginas da Google ou nos algoritmos de sugestão de amizades do Facebook. Engenheiros vão usar R para prototipagem, e então entregarão o modelo a alguém para que seja escrito em Java ou em Python."(tradução livre) [31].

Em segundo lugar nas buscas por ferramentas de processamento vem a linguagem "Python", utilizada em 2014 por aproximadamente 39% dos desenvolvedores de aplicações em processamento de dados. Python surgiu em 1980, sendo implementada pela primeira vez em 1989 por Guido Van Rossum, e tem ganhado espaço vorazmente no mercado nos últimos anos [31].

Além de ser mais fácil de aprender, mais intuitiva, rápida, e escalável para a indústria, Python é uma linguagem aberta, flexível e relativamente poderosa. As ricas bibliotecas já implementadas para ela podem ser facilmente obtidas e lidam impressionantemente bem com diversas áreas quentes em processamento de dados, como computação numérica, aprendizado de máquina, processamento natural de linguagem e *Data Science* [32].

As bibliotecas mais utilizadas nesses campos são:

- **Pandas:** para *data frameworks*;
- **SciKit-Learn, Theano e Tensorflow:** para aprendizado de máquina;
- **ggplot2 e matplotlib:** para visualização;
- **Scipy e Numpy:** para processamento numérico e estatística.

Importando bibliotecas como essas aos projetos, é possível implementar operações e funções matemáticas complexas com poucas linhas de código, o que facilita imensamente a implementação de diversas aplicações, minimizando o tempo de desenvolvimento e acelerando o processo de testes e *debug*. Esse é um atrativo que tem feito empresas de todos os ramos da indústria se aproximarem rapidamente de Python.

Um ponto negativo para entusiastas de processamento de dados, no entanto, é que Python é uma linguagem interpretada. Isso quer dizer que ela não é compilada anteriormente, e que suas instruções são "decodificadas" em tempo de execução por um interpretador. Isso diminui a performance do sistema e atrasa o seu processamento. Para processamento em média-escala essa diferença não é tão crítica, mas para grandes volumes de dados os resultados já começam a ser afetados [33].

Para aplicações não muito grandes, todavia, Python continua sendo uma excelente opção de ferramenta, principalmente quando aliado a uma plataforma de visualização que permita apresentar os dados processados de maneira atraente.

Uma linguagem bastante recente mas que tem potencial para se igualar ou mesmo superar Python no que concerne processamento de dados é uma linguagem chamada "Julia". Essa linguagem tem impressionado muitos na indústria pela sua alta performance e impressionante agilidade se comparada a outras linguagens já consagradas no campo. Seu potencial para escalabilidade é talvez ainda maior do que o de Python e, apesar da pouca idade, ela já mostra sinais de expansão [31].

É claro que, por ser uma linguagem muito nova, ainda há poucos pacotes e bibliotecas implementados para Julia, o que significa que a grande maioria das empresas preferem manter-se nas opções tradicionais quando escolhem ferramentas de processamento. Independente disso, o mundo está de olhos abertos para ver o que Julia irá se tornar nos próximos anos.

Para este trabalho, a ferramenta de processamento escolhida, qualquer que fosse, deveria ser capaz de:

1. Receber os dados adquiridos pelo *hardware* de aquisição;
2. Indexar esses dados no tempo a fim de criar diagramas de ciclos;
3. Permitir segmentação dos diagramas para extração de características do movimento;

4. Criar modelos matemáticos referentes a cada região do movimento para parametrização.;

5. Avaliar outros dados de movimentos subsequentes em comparação com o movimento parametrizado, de forma a estimar esses movimentos em tempo real.

Para que todas essas tarefas pudessem ser desempenhadas satisfatoriamente, era então necessário que se utilizasse uma ferramenta capaz de tratar listas e matrizes com diversas dimensões de forma robusta e rápida em procedimentos numéricos. Somando a isso o caráter flexível e portável desejado para a aplicação deste trabalho, escolheu-se a linguagem Python para codificar a unidade de Processamento.

Essa decisão foi bem aceita pelo laboratório no qual o projeto foi desenvolvido, uma vez que havia certa necessidade da implementação do algoritmo SLDS em Python. Essa necessidade foi também o que motivou o tratamento da unidade de processamento como um módulo destacável, independente da interface gráfica, e que possa ser reutilizado em outros projetos.

1.2.3 Interface gráfica

O ser humano é uma espécie primordialmente visual. Por mais que seja extremamente habilidoso em abstrair conceitos e trabalhar com ideias, 90% de toda apreensão cognitiva do cérebro vem de sinais captados pela retina. Isso, biologicamente, indica algo importante: se é possível comunicar uma informação visualmente, essa deve ser a primeira escolha. A frase "Uma imagem vale mais que mil palavras" não é apenas um ditado [34].

Estudos mostram que consumidores que assistem a vídeos de um produto têm 85% mais chance comprá-lo e que, com imagens, um anúncio online tem 180% mais audiência do que outros baseados apenas em texto. Isso ocorre porque, de fato, o cérebro não só se interessa mais por dados visuais como, através deles, consegue criar uma conexão empática com as informações apresentadas. Em outras palavras, imagens e vídeos possuem um forte elemento emocional que vem acompanhado do potencial de manipular a reação do espectador ao conteúdo [34].

Este potencial é profundamente explorado pela indústria do marketing e propaganda, e recentemente vem sendo explorado também pela indústria da tecnologia e da saúde. No mundo dos negócios, a forma com que se apresenta resultados a um investidor ou com que se projeta o design de um produto são cruciais para o sucesso do empreendimento.

Da mesma forma, quando se trata de aplicações que estarão em contato com usuários diversos, e especialmente pacientes dos quais pede-se um nível de engajamento

razoável com o protocolo de tratamento, é sempre interessante criar uma interface gráfica com a qual eles possam se relacionar.

Além disso, criar uma interface gráfica para o sistema é como criar um véu que permite esconder toda a unidade de processamento "por trás" da aplicação. A essa camada escondida dá-se o nome *Back-end*, e ela tipicamente não pode ser acessada pelo usuário. Isso é vantajoso para a solução pois isola todo o código matemático do usuário e apresenta a ele apenas os conteúdos pertinentes. A camada frontal da aplicação, que permite apresentar essas informações e com a qual o usuário pode interagir (neste caso a própria interface gráfica), é chamada então *Front-end*. Na programação, o termo utilizado para um *Front-end* gráfico de interfaceamento com o usuário é GUI (Interface gráfica de usuário, do Inglês *Graphical User Interface*).

Sob este pretexto, decidiu-se então, para este projeto, implementar também uma interface em *front-end* que fosse gráfica (GUI), que isolasse o código de processamento e que gerasse imediatamente uma conexão com o paciente. Dessa forma, passou-se então a buscar qual seria a melhor ferramente de visualização a se utilizar.

A grande maioria dispositivos de captura de movimento, principalmente aqueles mais avançados, já oferecem uma interface própria de visualização dos dados adquiridos. Muitos deles apresentam *softwares* dedicados que mapeiam os marcadores (ou sensores) utilizados pelo agente a modelos esqueletais do corpo humano em computador, e que acompanham então o corpo do agente em tempo real. De fato, o engajamento que esse tipo de interface gera no usuáro é imediato e quase catártico, o que é ótimo considerando o panorama desta aplicação.

As IMUs escolhidas para este trabalho (YEI 3-Space Sensors) possuem uma dessas interfaces. Ela é chamada *Yost Labs 3-Space Mocap Studio* e oferece exatamente o que é esperado: um modelo esqueletal com rastreamento em tempo real baseado nos dados colhidos pelas IMUs (Figura 16)

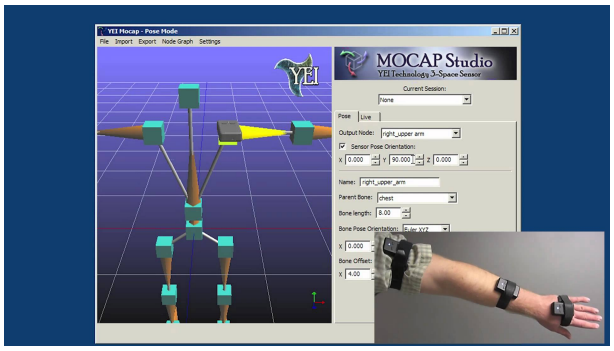

Figura 16 – Yost Labs 3-Space Mocap Studio [12].

Entretanto, no intuito de criar um sistema compacto que integrasse a interface gráfica à unidade de processamento em uma única aplicação, decidiu-se não utilizar o

Mocap Studio e, ao invés disso, criar do zero uma interface de captura de movimento personalizada, capaz de integrar todos os passos do processo de aquisição, segmentação, parametrização e estimação em tempo real.

A partir deste momento foi então necessário definir as especificações da interface de visualização que seria projetada para o escopo deste trabalho. Assim, foi decidido que a interface deveria:

1. Apresentar um modelo esqueletal completo do braço (desde o ombro até a mão), em três dimensões, em conformidade com aquilo que é oferecido pela maioria das interfaces de captura de movimento;
2. Fazer o modelo do braço acompanhar tão fielmente quanto possível o braço do paciente, através do rastreamento das IMUs em tempo real;
3. Apresentar em um menu opções de iniciar, pausar, retomar e cancelar o rastreamento das IMUs;
4. Apresentar, também no menu de opções, botões responsáveis por conduzir o usuário pelo processo de estimação de forma que ele pudesse gravar movimentos, criar modelos e então realizar estimações;
5. Incluir uma região na tela sobre a qual o diagrama de ciclos do movimento inicial gravado pudesse ser registrado para segmentação;
6. Retornar para o usuário, em tempo real, os resultados da estimação dos movimentos.

De forma a atender a essas especificações, era então necessário utilizar uma ferramenta de computação gráfica que permitisse modelamento tridimensional, que possuísse uma API para menu e que pudesse ser atualizada em *frames* contínuos para criar a animação de movimento do modelo do braço.

Considerando as opções disponíveis que atenderiam às especificações, a mais popular é de longe a do design gráfico a partir da linguagem "JavaScript ", uma escolha clássica para programadores de *Front-end*.

JavaScript foi uma linguagem criada em 1995 para desenvolvimento Web. Inicialmente, ela era chamada "Mocha", e apenas depois de passar pelo nome "LiveScript" foi que se consolidou como "JavaScript" por conta de um acordo com a Sun Microsystems (empresa criadora do Java), que queria portar sua linguagem para o navegador mais popular na época, o Netspace Navigator.

Apesar de ter nascido para dinamizar o desenvolvimento Web, o potencial do JavaScript se estendeu largamente nos anos seguintes. Como era uma linguagem destinada aos navegadores web, que começavam a ser usados por milhares de pessoas, com o passar

do tempo reconheceu-se a necessidade de aprimorar o *design* apresentado aos usuários. Aos poucos, o nível das bibliotecas gráficas oferecidas pelo JavaScript foi aumentando e, hoje, ela já é considerada a linguagem mais popular do mundo no desenvolvimento de aplicações [35].

Atualmente, uma das ferramentas mais famosas na renderização de imagens 2D e 3D providas pelo JavaScript é uma API chamada WebGL (Biblioteca Gráfica para Web, do Inglês *Web Graphics Library*). Essa API permite que os navegadores modernos consigam renderizar esses gráficos complexos sem a necessidade de *plug-ins* adicionais.

Um detalhe importante, entretanto, é que, como o próprio nome já diz, WebGL (bem como JavaScript) é uma ferramenta para desenvolvimento *web*. Isso significa que as aplicações desenvolvidas através dela são rodadas em um container HTML através do navegador da máquina. Dessa forma, a escolha de construir a interface gráfica em JavaScript para utilizar a API de WebGL implicou imediatamente na necessidade do usuário ter um navegador disponível para rodar a aplicação. Isso não foi considerado um aspecto crítico do projeto, uma vez que atualmente a disponibilidade de navegadores é uma constante em praticamente todos os computadores pessoais.

Apesar de WebGL ser uma API de muitos recursos, seu ponto negativo é a complexidade. Por conta disso, alguns *frameworks* foram desenvolvidos para facilitar a implementação de soluções através de WebGL. Os dois mais conhecidos e mais utilizados são as bibliotecas "Three.js" e "Babylon.js" [36].

Ao comparar as duas bibliotecas, é fácil perceber que ambas desempenham um papel muito parecido e lidam com as animações de maneira muito semelhante. Tipicamente, são criadas: uma variável "scene" (cena), uma variável "renderer" ou "engine" (renderizador), uma variável "camera" (câmera) e os objetos pertinentes à cena, que são o centro da aplicação. Um exemplo de uma cena em Three.js e Babylon.js está mostrada na Figura 17.

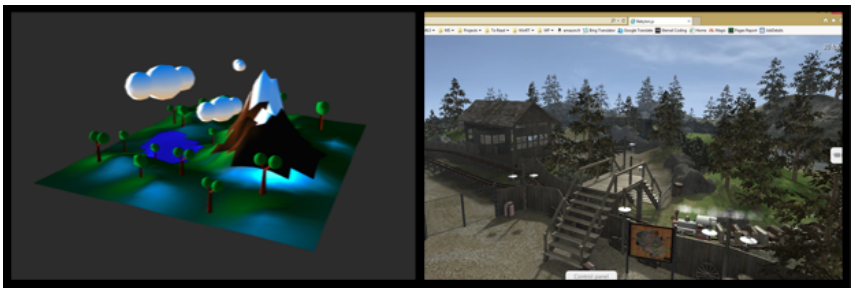

Figura 17 – Cena em Three.js [13] e em Babylon.js [14], respectivamente.

A verdadeira diferença entre as duas bibliotecas reside no intuito de cada uma: Three.js foi criada para tirar vantagem dos recursos de renderização de gráficos 3D da web e com isso gerar animações 3D para outras aplicações diversas a gosto do usuário; Baby-

lon.js foi criara com uma abordagem mais focada na criação e desenvolvimento de jogos para web, apresentando opções mais aplicadas no sentido de cumprir com as especificações para esse tipo de jogo [42].

Considerando este panorama, escolheu-se, para o trabalho em questão, utilizar a lingagem "Javascript" com a biblioteca "Three.js" para a construir a GUI da aplicação.

1.2.4 Interfaces de comunicação

Definir os módulos de um sistema de processamento, apesar de ser o centro do planejamento do projeto, não é tudo. Para que esse sistema venha a ser funcional, é necessário ainda definir como cada módulo irá se comunicar um com o outro. Ter um GUI no *Front-end* da aplicação, uma unidade de processamento de dados no *Back-end* e um *hardware* de aquisição perfeitamente funcional não vale de nada se esses três módulos não trocarem informação entre si. Assim, é ainda importante fazer algumas considerações acerca das interfaces de comunicação do sistema.

As duas interfaces necessárias para o projeto deste tipo de sistema são:

- Interface *Hardware - Back-end*;
- Interface *Back-end - Front-end*.

A Interface *Hardware - Back-end* na grande maioria dos casos depende do protocolo de comunicação utilizado pelo *hardware* escolhido. Caso esse protocolo possa ser atendido pela linguagem da unidade de processamento, então a melhor opção costuma ser escrever, nessa mesma linguagem, um arquivo conhecido como "*driver*".

Um *driver* nada mais é do que um programa que conhece o hardware bem o suficiente para lidar com o seu protocolo de comunicação, "empacotar" os dados recebidos e fornecê-los a quem quer que vá processá-los. É ele quem de fato assume o papel dessa interface entre o dispositivo de aquisição e o software de processamento.

Na realidade, é muito comum que o próprio fabricante dos dispositivos já forneça os drivers necessários para operá-los de forma que o programador não precise implementar bibliotecas inteiras de comunicação antes de poder construir sua aplicação. É também comum que esses drivers sejam fornecidos em algumas linguagens diferentes para um melhor acoplamento com o sistema desenvolvido pelo programador. De qualquer forma, não é sempre que os programadores utilizam os drivers fornecidos pelos fabricantes, e não raro é encontrar drivers personalizados que foram feitos à mão por usuários para aplicações específicas.

Para a comunicação *Back-end - Front-end*, hoje existem basicamente duas arquiteturas utilizadas. Ambas utilizam uma estrutura a que se dá o nome de "Estrutura

cliente-servidor".

Quando se fala de comunicação web, é raro que não haja um servidor envolvido. Um servidor, em termos simples, é uma máquina ou um sistema que serve outras máquinas ou sistemas através da realização de uma tarefa e da reportagem de respostas em relação a essa tarefa. Os clientes são as máquinas ou sistemas servidos pelo servidor. Um cliente pode fazer uma requisição ao servidor, pedindo algum tipo de informação ou serviço, e o servidor então devolve uma resposta. Uma das maneiras mais difundidas de um cliente e um servidor trocarem informação é através de um protocolo chamado HTTP (*Hypertext Transfer Protocol*), que codifica essa troca de informação e que pode responder em diversos formatos padrão (JSON, XML, ...). Isso significa que o servidor deve estar preparado para suportar esses formatos [37], [38].

Sabendo disso, as duas arquiteturas tipicamente utilizadas para estabelecer uma interface entre um *Back-end* e um *Front-end* web são:

1. **Aplicativos renderizados por servidor:** o navegador faz uma requisição HTTP para um servidor e este, por sua vez, responde com uma página web a ser carregada.
2. **Ajax (*Asynchronous JavaScript and XML*):** o código JavaScript envia, de dentro da página web, uma requisição HTTP para um servidor e recebe uma resposta em XML ou JSON que é então tratada pelo código [37].

Tipicamente, o *Back-end* assume o papel de servidor, rodando o processamento de dados isoladamente, e o *Front-end* assume o papel de cliente, requisitando os resultados do processamento quando é conveniente.

Para este trabalho, por uma questão de documentação e conveniência, escolheu-se seguir a estrutura clássica e utilizar a unidade de processamento (*Back-end*) como servidor e o GUI (*Front-end*) como cliente, selecionando Ajax como forma de comunicação entre os dois extremos.

1.3 Fundamentos matemáticos

Na seção 1.1, a base teórica para análise de movimentos foi discutida em relação à conceituação do procedimento de análise, e, na seção 1.2, discutiu-se ferramentas para implementar a teoria proposta anteriormente. Tudo isso, no entanto, ressalta a estrutura do sistema mas não o seu verdadeiro conteúdo.

A pergunta que existe desde o início deste capítulo é: no que consiste, matematicamente, a estimação dos movimentos? Como os dados brutos recebidos do *hardware* de aquisição e se transformam em dados significativos para o fisioterapeuta através do

software de processamento? O que esse software de fato faz com os dados para render-lhes significado?

Essa seção traz a base matemática necessária para entender o processamento de dados em si, desde o seu recebimento até culminar na geração dos resultados da estimação. Como dito anteriormente, o algoritmo implementado foi baseado no modelo SLDS, e o fim maior dessa seção é construir o conhecimento basal para que se possa compreender seu funcionamento.

O embasamento e a formulação aqui derivados seguem a proposta e a linha de raciocínio de [15].

1.3.1 Modelos

Em primeiro lugar, antes mesmo de detalhar os modelos matemáticos envolvidos no processamento de dados, é importante mencionar que o sistema em questão (o braço de um paciente) é um sistema cujo mapeamento no espaço geométrico naturalmente varia com o tempo. Isso o caracteriza como um sistema dinâmico.

Na física, sistemas dinâmicos podem ser representador por um modelo chamado "espaço de estados". Esse modelo é baseado em entradas, saídas e em um conjunto de "variáveis de estado", que são variáveis que, dado seu conhecimento, descrevem completamente o problema em um dado tempo t.

Se todos esses fatores são unidos em uma equação diferencial de primeira ordem modelada como uma função do tempo e se admite-se que o sistema pode ser definido por equações lineares invariáveis de dimensões finitas, então a formulação do modelo fica da seguinte forma:

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{1.1}$$

$$y(t) = Cx(t) + Du(t), \tag{1.2}$$

em que A, B, C e D são matrizes constantes que definem a dinâmica do sistema, x define a variável de estado, u as entradas e y as saídas.

Assumindo que as variáveis estão sendo amostrado no tempo e que este é um sistema causal (só depende de entradas e estados passados), então as equações podem ser reescritas como

$$x_{k+1} = Ax_k + Bu_k, \tag{1.3}$$

$$y_k = Cx_k + Du_k. \tag{1.4}$$

É claro que, no mundo real, todo modelo é perturbado por ruído. Para que elas funcionem de forma satisfatória, um último ajuste pode ser feito nessas equações para

incluir a noção de uma variância das medidas nas variáveis r_k e w_k, que serão tratadas para os fins deste trabalho como sendo ruído branco. Assim, tem-se que

$$x_{k+1} = Ax_k + Bu_k + r_k, \tag{1.5}$$

$$y_k = Cx_k + Du_k + w_k. \tag{1.6}$$

A partir dessas modificações, e uma vez que r_k e w_k são definidas por distribuições aleatórias de gaussianas independentes, de média zero e variância finita, vê-se que, com a adição desses termos, o modelo se torna um modelo estocástico. Além disso, considera-se, desse ponto em diante, que o sistema em questão é um sistema Markoviano, ou seja, um sistema ao qual a propriedade Markoviana se aplica. De forma sucinta, essa propriedade define uma "perda de memória" do sistema, em que um dado estado presente só depende do estado imediatamente anterior e não de todo o histórico de estados anteriores.

Quando se faz estimações a partir desse modelo no espaço de estados, existem quatro tarefas possíveis. Em linhas gerais, cada uma dessas tarefas realiza o mesmo trabalho para janelas diferentes dos dados amostrados. São elas:

1. **Predição:** procura estimar um estado futuro baseando-se nas observações já feitas até um tempo $t > 0$ anterior ao alcance desse estado. Isso é feito através do cálculo de $p(x_k|y_{1:t})$, $0<t<$k, ou seja, da probabilidade do sistema alcançar um estado x no instante futuro k sabendo todas as observações feitas sobre o sistema até um tempo t anterior a k;

2. **Filtragem:** busca estimar o estado presente, ou seja, encontrar a probabilidade de estar no estado em que se encontra considerando todas as observações anteriores. Esse é o cálculo de $p(x_k|y_{1:k})$, que é a probabilidade do sistema estar em um estado x no instante k dado que se sabe todas as observações feitas sobre o sistema até esse instante k.

3. **Suavização:** estima um estado passado a partir das informações obtidas até o tempo presente. Em outras palavras, espera alguns instantes a mais de forma a reunir mais informação e só então estima o estado passado. Essa é a resolução de $p(x_{t-l}|y_{1:t})$, em que l é o tamanho da janela que se deseja esperar antes de estimar o estado passado.

4. **Decodificação com algoritmo de Viterbi:** Para o caso de estados discretos, lida com a estimação da mais provável (melhor) sequência de estados que podem ter gerado o conjunto das saídas observadas. Leva em conta a totalidade das amostras do problema.

Vale aqui dizer que, em geral, a performance de algoritmos de estimação que processam dados *online* é um pouco pior do que aquela de algoritmos com processamento

offline. Isso se dá pois quanto mais informações se tem, mais acurada é a estimação. Dessa forma, quanto mais larga a janela de dados disponíveis no momento da estimação, maiores as chances da resposta estar correta. Dessa maneira, vê-se que o método de Predição é o menos acurado, enquanto a Decodificação com algoritmo de Viterbi dará a melhor estimativa.

De qualquer forma, uma vez que este trabalho possuía como especificação o retorno em tempo real de informação acerca da estimação dos movimentos, decidiu-se então utilizar o método de Filtragem em detrimento dos outros. Assim, daqui para frente a explicação dos algoritmos deste projeto incorporará essa abordagem.

Sabendo que o sistema em questão é um sistema estocástico Markoviano, então existem aqui duas possibilidades: caso a variável oculta X do sistema seja discreta, esse modelo passa a ser chamado "Cadeia Oculta de Markov" (HMM, do *Inglês Hidden Markov Model*); caso o contrário, supondo que a variável oculta seja contínua, então o modelo é um LDS (Sistema dinâmico linear, do Inglês *Linear Dynamic System*).

Ambos serão importantes para este trabalho, e é razoável passar brevemente pelo raciocínio que rege cada um deles.

1.3.1.1 Cadeias Ocultas de Markov

Em uma cadeia oculta de Markov (HMM), a variável oculta do sistema X_t pertence a conjunto discreto de elementos, enquanto a variável observável Y_t pode também pertencer a um conjunto discreto de elementos ou ser uma distribuição Gaussiana contínua.

O autômato clássico das HMMs, conhecido como diagrama de Trellis, está mostrado na Figura 18. Como as variáveis envolvidas no modelo são variáveis estocásticas, existem três parâmetros fundamentais para o modelo de um HMM:

1. $\Pi(i,j) = P(X_k = j | X_{k-1} = i)$**:** matriz estocástica de transição de estados. Diz qual é, para cada estado i, a probabilidade de transitar para qualquer estado j;
2. $B(y,i) = P(Y_k = y | X_k = i)$**:** matriz de observação. Dado um estado presente i, diz qual é a probabilidade de ocorrer uma determinada observação y;
3. $\pi_0(s) = P(X_0 = s)$**:** distribuição de probabilidade para o estado inicial do sistema.

Considerando esses três parâmetros, o modelo pode ser escrito como

$$P(x_k|x_{k-1}) = \mathbf{x_k^T \Pi x_{k-1}} \tag{1.7}$$

$$y_k = B(y_k, x_k) \tag{1.8}$$

$$P(x_0) = \pi_0, \tag{1.9}$$

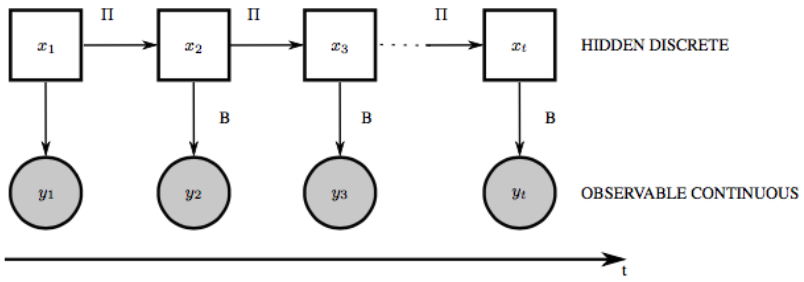

Figura 18 – Diagrama de Trellis de uma cadeia oculta de Markov [15].

em que x é um vetor coluna unitário no índice do estado indicado.

O que é interessante acerca dos HMMs é que eles permitem de maneira simples realizar inferências através de um algoritmo bastante conhecido chamado *Forward-Backward algorithm*. O resultado deste algoritmo é a distribuição $P(x_k|y)$, ou seja, a probabilidade de estar em um estado x qualquer no instante k dado que as saídas observáveis foram a sequência y. Essa informação é extremamente útil, pois permite, a partir da sequência de variáveis observáveis do sistema, estimar as variáveis ocultas.

Outra possibilidade para HMMs é aplicar o algoritmo de Viterbi a fim de inferir a melhor sequência de estados que poderiam ter gerado a sequência observada de medidas, que é encontrar $x^*_{1:k} = argmax_{x_{1:k}} P(x_{1:k}|y_{1:k})$.

Ambas os casos serão tratados abaixo.

1.3.1.1.1 Algoritmo Forward-Backward

Fatorando a expressão desejada, pode-se chegar a

$$P(x_k|y) \propto P(y_{k+1:T}|x_k, y_{1:k})P(x_k|y_{1:k}). \tag{1.10}$$

Pela propriedade Markoviana, $y_{k+1:T}$ independe de $y_{1:k}$. Assim, a equação pode ser reduzida a:

$$P(x_k|y) \propto P(y_{k+1:T}|x_k)P(x_k|y_{1:k}), \tag{1.11}$$

cujo primeiro termo está relacionado à probabilidade de um grupo de observações futuras dado um estado anterior e cujo segundo termo é o próprio processo de Filtragem. Vale ressaltar aqui que o algoritmo *Forward-Backward* assume que são conhecidos os parâmetros do HMM (Π, B e π_0).

Para então definir essas distribuições, divide-se o cálculo em duas partes: o algoritmo *Forward*, referente ao termo da Filtragem e o algoritmo *Backward*, referente ao primeiro termo da equação. A resolução dessas duas etapas resulta na probabilidade $P(x_k|y)$ que se deseja calcular, e fundamenta o algoritmo conhecido como *Forward-Backward*.

A primeira etapa a ser executada é a etapa *Forward*. Utilizando a regra de Bayes para o segundo termo da equação 1.12, tem-se que

$$P(x_k|y_{1:k}) \propto P(y_k|x_k, y_{1:k-1})P(x_k|y_{1:k-1}). \tag{1.12}$$

No entanto, pela propriedade Markoviana, sabe-se que y_k não depende de $y_{1:k-1}$. Além disso, vê-se que o último termo da equação é exatamente o caso explorado pelo método da Predição, comentado anteriormente, e que, também no caso Markoviano, pode ser reescrito em função do estado anterior x_{k-1}. Assim, a expressão fica da seguinte forma:

$$P(x_k|y_{1:k}) \propto P(y_k|x_k) \sum P(x_k|x_{k-1})P(x_{k-1}|y_{1:k-1}). \tag{1.13}$$

Analisando essa expressão acima, é possível ver que o último termo da equação é exatamente o termo que se quer calcular, mas para o instante de tempo imediatamente anterior. Isso significa que essa é uma equação recursiva, e que pode ser resolvida iterativamente. Outro detalhe importante é que o primeiro termo da equação 1.13, $P(y_k|x_k)$, pode ser extraído exatamente do que foi definido como sendo a matriz de observação B, que é conhecida. Além disso, o primeiro termo do somatório, $P(x_k|x_{k-1})$, também é conhecido, proveniente da matriz de transição Π. Assim, com a informação de que essa é uma equação recursiva e sabendo que os outros dois termos, pode-se definir esse resultado como

$$P(x_t|y_{1:t}) = \alpha_t \propto B\Pi^T \alpha_{t-1}, \tag{1.14}$$

no qual o instante k foi generalizado para qualquer tempo t e α foi definido como sendo o resultado do processo de filtragem.

Dessa forma, o cálculo de α, recursivamente, envolve dois passos distintos: primeiro, a partir do resultado da filtragem anterior, prever o novo estado (i.e. calcular $P(x_k|y_{1:k-1})$ através de Π e α_{t-1}); em seguida, atualizar essa predição com base na matriz de Observação B. O conjunto dessas duas etapas, predição e atualização, resultam no novo α e, assim, na probabilidade $P(x_k|y_{1:k})$. De forma a iniciar a recursão, pode-se provar [15] que $\alpha_1 = B\pi_0$. A partir dessa condição inicial, todos os outros αs pode ser calculados.

É interessante notar que, pela propriedade Markoviana, uma vez realizada a predição pode-se eliminar toda e qualquer informação sobre estados passados do sistema, o que reduz imensamente o esforço computacional e torna o modelo independente de outros instantes de tempo.

Para a etapa de *Backward*, a distribuição que se deseja calcular pode ser fatorada em

$$P(y_{k+1:T}|x_k) \propto \sum P(y_{k+1:T}, x_{k+1}|x_k) \tag{1.15}$$

$$P(y_{k+1:T}|x_k) \propto \sum P(y_{k+2:T}|x_{k+1}, x_k, y_{k+1})P(y_{k+1}|x_{k+1}, x_k)P(x_{k+1}|x_k). \tag{1.16}$$

Entretanto, novamente pela propriedade Markoviana, sabe-se que $y_{k+2:T}$ não depende de x_k ou de y_{k+1}. Da mesma forma, y_{k+1} não depende de x_k. Assim, a equação pode ser reduzida a

$$P(y_{k+1:T}|x_k) \propto \sum P(y_{k+2:T}|x_{k+1})P(y_{k+1}|x_{k+1})P(x_{k+1}|x_k). \tag{1.17}$$

A partir dessa equação, fica então fácil perceber que, mais uma vez, o último termo advém da própria matriz de transição Π, o segundo termo advém da matriz de observação B e que o primeiro termo é a própria probabilidade que se quer calcular, apenas um instante de tempo à frente. Com isso, definindo o operador β como sendo o resultado da etapa de *Backward*, tem-se então que

$$P(y_{k+1:T}|x_k) = \beta_k \propto \Pi B \beta_{k+1}, \tag{1.18}$$

Essa também é uma equação recursiva, mas pode-se ver que ela deve ser realizada começando em T e voltando então pelas amostras indexadas no tempo. Eis aqui a explicação do nome *Backward*, que significa "ao contrário", em oposição a *Forward*, que significa "para frente".

Pode-se provar também que $\beta_T = 1$ [15]. Com essa informação e sabendo os parâmetros do HMM, calcula-se então todos os βs.

Finalizado o cálculo dos αs e βs, volta-se para a definição inicial do problema da estimação e, reescrevendo $P(x_k|y)$ como γ_k pode-se reescrever a equação 1.11 como

$$\gamma_k = \alpha_k . * \beta_k, \tag{1.19}$$

na qual o operador ".*"é o produto elemento a elemento dos vetores α e β.

1.3.1.1.2 Algoritmo de Viterbi

Ao aplicar o algoritmo de Viterbi, quer-se resolver a seguinte equação:

$$x^*_{1:k} = argmax_{x_{1:k-1}} P(x_{1:k}|y_{1:k}). \tag{1.20}$$

Essa equação pode ser fatorada como

$$x^*_{1:k} = argmax_{x_{1:k-1}} (P(y_k|x_k)P(x_k|x_{k-1})P(x_{1:k-1}|y_{1:k-1})), \tag{1.21}$$

em que os dois primeiros termos são conhecidos das matrizes Π e B. Rearranjando a equação, pode-se obter

$$x^*_{1:k} = argmax_{x_{k-1}} (P(y_k|x_k)P(x_k|x_{k-1}) max_{x_{1:k-2}} P(x_{1:k-1}|y_{1:k-1})). \tag{1.22}$$

Ao definir o resultado dessa equação como o parâmetro δ_t, pode-se perceber que, novamente, ela assume um perfil recursivo. Além disso, mais uma vez aparecem, respectivamente, as matrizes B e Π que são conhecidas no problema. Logo, acrescentando a isso a condição inicial, termina-se, de forma genérica, com

$$\delta_k = max_{x_{k-1}}(B\Pi^T\delta_{k-1}); \tag{1.23}$$

$$\delta_1 = \pi_0 B_1. \tag{1.24}$$

Esse sistema é muito semelhante àquele encontrado para a etapa *Forward* do algoritmo *Forward-Backward*, mas procura maximizar cada transição para encontrar o melhor caminho dentre as estimativas. O algoritmo *Forward-Backward* apenas encontra a melhor estimativa local para cada iteração.

Ao guardar também o índice da melhor escolha no tempo anterior de cada iteração através de

$$\psi_k(j) = argmax_i P(X_k = j|X_{k-1} = i)\delta_{k-1}(i) \tag{1.25}$$

é possível então, no passo de *Backward* do algoritmo de Viterbi, otimizar o melhor caminho fazendo

$$x_k^* = \psi_{k+1}(x_{k+1}^*). \tag{1.26}$$

Assim, encontra-se então a sequência mais provável de eventos que poderiam ter causado a sequência de observações obtida.

1.3.1.2 Sistemas Dinâmicos Lineares

A outra possibilidade para um modelo estocástico Markoviano reside no caso em que a variável oculta é contínua, e não discreta como no caso de HMMs. Como dito anteriormente, esses sistemas são ditos sistemas dinâmicos lineares.

Nesse tipo de formulação, assume-se que as variáveis aleatórias X_t, Y_t são Gaussianas lineares. Assim,

$$P(X_t = x_t|X_{t-1} = x_{t-1}) = N(x_t; Ax_t + \mu_X, Q) \tag{1.27}$$

$$P(Y_t = y_t|X_t = x_t) = N(y_t; Cx_{t+1} + \mu_Y, R). \tag{1.28}$$

Com isso, escrevendo essas equações na forma vetorial, obtém-se o modelo

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + \mathbf{q}_{t+1} \tag{1.29}$$

$$\mathbf{y}_t = C\mathbf{x}_t + \mathbf{r}_t, \tag{1.30}$$

em que q é o ruído no estado (Gaussiana de média zero e Variância Q) e r é o ruído na observação (Gaussiana de média zero e Variância R).

Essas duas equações governantes do modelo são a representação para sistemas dinâmicos lineares daquilo que seriam as equações 1.7 e 1.8 para HMMs. Analogamente, a matriz A é chamada matriz de transição e a matriz C, matriz de observação. O diagrama para LDS pode ser visto na Figura 19.

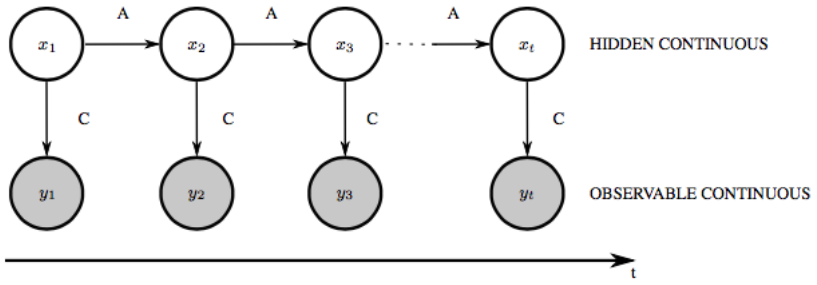

Figura 19 – Autômato de um sistema dinâmico linear [15].

O algoritmo mais conhecido para estimação de sistemas dinâmicos lineares é chamado "Filtro de Kalman". Da mesma forma que no caso de HMMs, em geral o Filtro de Kalman também faz parte de uma estrutura *Forward-Backward*. Abaixo, as duas partes do algoritmo estão detalhadas.

1.3.1.2.1 *Forward*

De maneira semelhante àquela desenvolvida na seção 1.3.1.1, é possível mostrar que o algoritmo do Filtro de Kalman, definido na etapa *Forward* do algoritmo, também é dividido em previsão e atualização, agindo recursivamente.

Na previsão, as estimativas são feitas da seguinte forma:

$$x_{t|t-1} = Ax_{t-1|t-1} \tag{1.31}$$

$$\Sigma_{t|t-1} = A\Sigma_{t-1|t-1}A^T + Q. \tag{1.32}$$

Em seguida, existe um grupo de variáveis auxiliares que devem ser computadas. São elas o custo de inovação (e_t), a variância do erro (S_t) e a matriz de ganho de Kalman (K_t). Suas definições são

$$e_t = y_t - Cx_{t|t-1} \tag{1.33}$$

$$S_t = C\Sigma_{t|t-1}C^T + R \tag{1.34}$$

$$K_t = \Sigma_{t|t-1}C^T S_t^{-1}, \text{ e} \tag{1.35}$$

$$\tag{1.36}$$

A partir dessas variáveis, pode-se então atualizar as estimativas de acordo com

$$x_{t|t} = x_{t|t-1} + K_t e_t \tag{1.37}$$

$$\Sigma_{t|t} = (I - K_t C)\Sigma_{t|t-1}, \tag{1.38}$$

em que I é a matriz identidade.

1.3.1.2.2 *Backward*

Para o retorno do Filtro de Kalman, muitas vezes utiliza-se um algoritmo de suavização chamado Rauch-Tung-Striebel (RTS). Com esse algoritmo, é possível reunir mais informação e tornar a estimativa mais robusta, voltando pelas amostras para otimizar o processo. Para isso, inicialmente toma-se as predições

$$x_{t+1|t} = A_{t+1}x_{t|t} \tag{1.39}$$

$$\Sigma_{t+1|t} = A_{t+1}\Sigma_{t|t}A_{t+1}^T + Q_{t+1}, \tag{1.40}$$

e calcula-se a chamada matriz de ganho suavizado

$$J_t = \Sigma_{t|t}A_{t+1}^T\Sigma_{t+1|t}^{-1}. \tag{1.41}$$

A partir dessas computações, obtém-se então as estimativas de média, variância e variância cruzada desejadas:

$$x_{t|T} = x_{t|t} + J_t(x_{t+1|T} - x_{t+1}|t) \tag{1.42}$$

$$\Sigma_{t,t-1|T} = Cov[X_{t-1}, X_t|y_{1:T}] \tag{1.43}$$

$$\Sigma_{t|T} = \Sigma_{t|t} + J_t(\Sigma_{t+1|T} - \Sigma_{t+1|t})J_t^T \tag{1.44}$$

$$\Sigma_{t-1|t} = J_{t-}\Sigma_{t|T}. \tag{1.45}$$

1.3.1.3 Sistemas Dinâmicos Lineares Chaveados (SLDS)

Como dito no início desta seção, o conhecimento acerca dos modelos matemáticos discutidos acima foi explanado de forma a construir a base necessária para compreender o algoritmo de estimação que é utilizado neste trabalho. Esse algoritmo se baseia no conceito de Sistemas Dinâmicos Lineares Chaveados.

Para explicar o que são esses sistemas, considera-se um exemplo: a tentativa de estimação do movimento de alguém que estende seu braco para pegar um copo d'água. Utilizando HMMs, seria possível estimar sequências discretas de estados, ou seja, seria possível estimar se o braço está estendido ou flexionado em um dado instante, mas não seria possível acompanhar o movimento do braço continuamente entre esses estados. Filtros de Kalman, por outro lado, são o algoritmo ideal para o rastreamento contínuo de variáveis lineares no tempo, o que compensaria pela lacuna existente em HMMs. Contudo, Filtros de Kalman estimam movimento com base em um modelo único que, ainda por cima, é linear. Utilizar esse tipo de abordagem para analisar movimentos de braço, que podem vir a ser complexos, irregulares, e necessitar de acompanhamento multivariável, não seria razoável.

Os Sistemas Dinâmicos Lineares Chaveados vêm para resolver este problema. Aproveitando o melhor dos dois mundos, esse tipo de sistema funciona como uma combinação de cadeias ocultas de Markov e filtros de Kalman, tornando-se assim capaz de descrever movimentos muito mais complexos. Acompanhando movimentos integralmente, tanto em sua vertente contínua quanto discreta, é possível gerar análises muito mais diversas [15].

Para o propósito desde trabalho, a explicação por trás do algoritmo que rege o SLDS será dividida em três partes: Segmentação, Parametrização e Estimação. As duas primeiras são preparações para a última parte, na qual de fato os modelos irão agir. Em geral, as duas primeiras etapas serão realizadas a partir da gravação do movimento de um indivíduo saudável, enquanto a última será utilizada em pacientes de fisioterapia.

1.3.1.3.1 Segmentação

Como discutido na seção 1.1, ao obter um diagrama de ciclos de determinado movimento, a primeira coisa que se faz é segmentá-lo. Matematicamente, o que se faz aqui é associar a cada segmento de comportamento distinto do diagrama de ciclos uma variável s que indexa o comportamento observado naquele segmento. Essa variável é chamada variável de chaveamento.

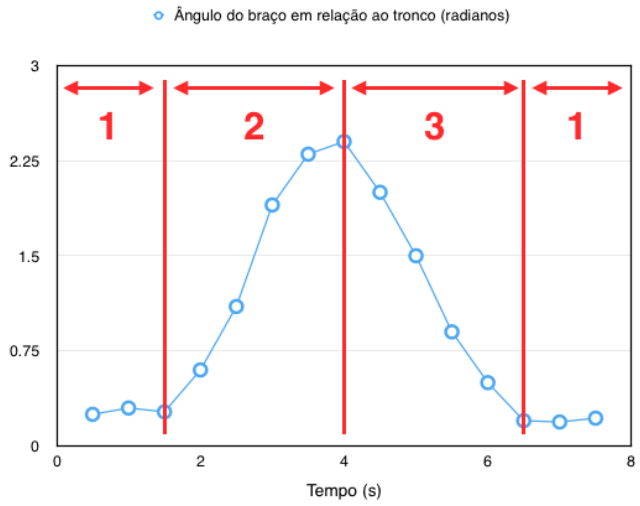

Figura 20 – Diagrama de ciclos de flexão de braço segmentado.

A título de exemplificação, nas Figuras 20 e 21 a variável s pertenceria ao conjunto S ={1, 2, 3}. Alternativamente, esse conjunto poderia também ser alfabético (S ={a, b, c}), lexical (S ={"Relaxado", "Flexionando", "Relaxando"}) ou da formatação que o usuário preferisse. A sua função é apenas indexar um comportamento específico do sistema, indicando o estado em que ele se encontra.

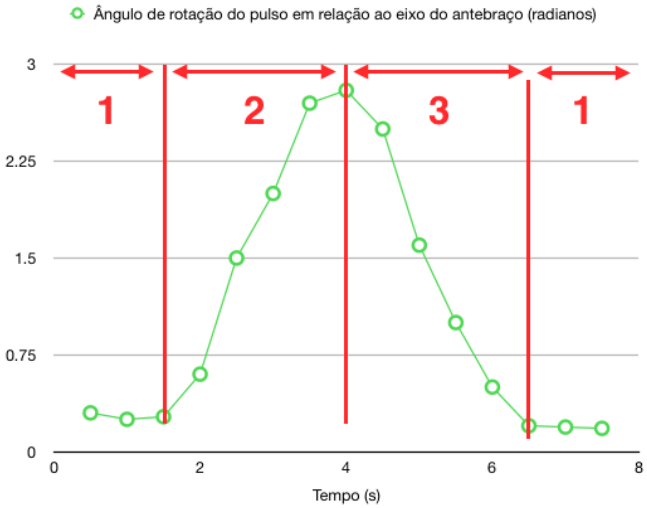

Figura 21 – Diagrama de ciclos de rotação de pulso segmentado.

Caso se tratasse de um movimento composto (mais de uma variável sendo monitorada), seria necessário não apenas segmentar cada um dos diagramas individuais como também o diagrama composto dos dois movimentos que, como pode ser visto na Figura 22, gera um conjunto de símbolos distinto dos casos individuais. Nos casos multivariáveis, esse conjunto não é chamado conjunto S, mas conjunto D, de cujas variáveis de chaveamento são denotados por σ. (No caso da Figura 22, $D = \{1, 2, 3, 4\}$).

A título de consistência, daqui em diante os diagramas que são compostos por todas as variáveis observadas (como aquele mostrado na Figura 22) serão referidos como diagramas principais, enquanto aqueles contendo apenas as variáveis isoladas (como aqueles mostrados nas Figuras 20 e 21) serão referenciados como diagramas parciais. Para o caso univariável, o único diagrama gerado é, obviamente, o diagrama principal. Para o caso multivariável, sempre haverá um diagrama principal e outros N diagramas parciais, onde N denota o número de variáveis sendo observadas.

É importante notar que, dados dois movimentos com conjuntos de variáveis de chaveamento S_1 e S_2, não há prejuízo para o sistema se $S_1 = S_2$. De fato, os movimentos das Figuras 20 e 21 representam este caso. O que é importante é estabelecer que, quando esses movimentos são compostos (como na Figura 22), deve haver uma ordenação dos pares de variáveis de chaveamento s de cada segmento para gerar um conjunto coerente de novas variáveis σ. A função que realiza este mapeamento é a função $\phi(s1, s2, ..., sn)$.

Avaliando a função $\phi(s)$ para o exemplo da Figura 22, obtém-se o seguinte resul-

Figura 22 – Composição dos movimentos de flexão de braço e rotação de punho com nova segmentação.

tado:

$$\phi(s1, s2) = \begin{cases} 1, s1 = 1\text{e } s2 = 1 \\ 2, s2 = 2\text{e } s2 = 1 \\ 3, s1 = 3\text{e } s2 = 2 \\ 4, s1 = 1\text{e } s2 = 3. \end{cases}$$

Dessa forma, são gerados os estados σ e o conjunto D do diagrama composto.

Um vez finalizado o processo de segmentação, deve existir então um conjunto $\mathbb{S} = \{S_1, S_2, ..., S_N\}$. Caso $N > 1$, deve haver ainda os conjuntos D resultantes da combinação dos conjuntos S_n, $n \in 1, 2, ..., N$ a depender do seu comportamento no diagrama principal. Nesse ponto, pode-se então prosseguir para a etapa de Parametrização.

1.3.1.3.2 Parametrização

O primeiro passo da Parametrização é encontrar a matriz de transição de estados Π_{ij} do modelo de Markov para cada movimento. Como os coeficientes dessa matriz dizem quais são as probabilidades de se transitar do estado i para o estado j, então é necessário fazer uma estimativa baseada no diagrama obtido que informe essas probabilidades. Isso é feito apenas para o diagrama principal, da seguinte forma:

$$\Pi_{ij} = \frac{\sum_{t=2}^{T} \xi_t(i, j)}{\sum_{t=1}^{T} \gamma_t(i)}, \tag{1.46}$$

em que

$$\xi_t(i, j) = \begin{cases} 1, \text{ se } s_{t-1} = i \text{ e } s_t = j \\ 0, \text{ caso o contrário.} \end{cases}$$

e

$$\gamma_t(i) = \begin{cases} 1, \text{ se } s_t = i \\ 0, \text{ caso o contrário.} \end{cases}$$

Esse cálculo significa então encontrar quantas vezes o estado i transitou para o estado j e dividir este valor pelo número total de ocorrências do estado i, o que é uma forma bastante simples de calcular a probabilidade aproximada de transição entre os estados i e j.

É fácil ver que, do total de pontos do diagrama, como uma parcela muito pequena são pontos de transição, então a matriz Π terá um comportamento diagonalizado, ou seja, terá valores muito mais altos na diagonal do que nos elementos periféricos. Isso acrescenta inércia aos estados, indicando que, em geral, é muito mais provável que um estado se mantenha como está do que que transite para outro estado.

Essa matriz será sempre uma matriz quadrada $(M \times M)$ em que M simboliza o número de estados definidos para um determinado movimento. Assim, o procedimento da equação 1.45 é repetido até que toda a matriz esteja completa.

Cabe aqui notar que cada movimento gravado possui apenas uma matriz de transição de estados que relaciona todos as suas variáveis de chaveamento, uma vez que ela só é calculada para o digrama principal do problema.

Observando os diagramas de ciclos, não é difícil perceber que cada segmento do diagrama possui um conjunto $(N \times K)$ de dados, sendo N ainda o número de variáveis monitoradas/ observadas simultaneamente e K o número de amostras por variável no segmento.

O segundo passo da parametrização é extrair, das K amostras pertencentes a cada segmento de cada variável amostrada, duas informações:

1. Um parâmetro de velocidade constante;
2. Um parâmetro de variância;

Ambos os parâmetros são obtidos de forma bastante intuitiva: a obtenção do parâmetro de velocidade é feita tirando-se a média das velocidades instantâneas de cada amostra recebida no segmento; a variância é calculada com a definição padrão. Os procedimentos estão detalhados como

$$v_s^n = \frac{\frac{\sum_{t=0}^{T-1}(x_{t+1}-x_t)}{t_{amostra}}}{K}; \tag{1.47}$$

$$\Sigma_s^n = \frac{\sum_{t=0}^{T}(x_t - \overline{x})^2}{K}. \tag{1.48}$$

Ao contrário do cálculo da matriz de transição de estados Π, o cálculo desses parâmetros só é feito para os segmentos dos diagramas parciais.

Vale apontar que, como no caso das Figuras 20 e 21, se algum estado ocorrer mais de uma vez no diagrama (estado 1 em ambas as Figuras), ambos devem ser levados em conta juntamente no cálculo da velocidade e variância. A uma mesma variável de chaveamento dos diagramas parciais, só deve estar associado um par $\{velocidade, variância\} = \{v_s^n, \Sigma_s^n\}$.

Considerando que o exemplo mostrado nas Figuras 20, 21 e 22 é um caso multivariável, os parâmetros definidos até o momento seriam os seguintes:

1. **Diagrama parcial 1 (Figura 20):** Três pares $\{v_s^1, \Sigma_s^1\}$, um para cada estado s do diagrama;
2. **Diagrama parcial 2 (Figura 21):** Três pares $\{v_s^2, \Sigma_s^2\}$, um para cada estado s do diagrama;
3. **Diagrama principal (Figura 22):** Uma matriz Π com as probabilidades de se transitar entre seus quatro estados σ compostos.

Como cada estado dos diagramas parciais é codificado por uma variável de chaveamento s, então na realidade cada par $\{v_s^n, \Sigma_s^n\}$ está associado a um s distinto. Sabendo disso, pode-se criar, para cada estado dos diagramas parciais, um trio $\{v_s^n, \Sigma_s^n, s\}$. O próximo passo é associar os trios $\{v_s^n, \Sigma_s^n, s\}$ adequados para montar esses mesmos vetores no diagrama principal do movimento.

Para entender este processo, o melhor a fazer é voltar a considerar o exemplo da Figura 22. No exemplo mostrado nela, tem-se o seguinte:

$$\sigma = \begin{cases} 1: & s^1 = 1, s^2 = 1, v_1^* = \{v_1^1, v_1^2\}, \Sigma_1^* = \{\Sigma_1^1, \Sigma_1^2\} \\ 2: & s^1 = 1, s^2 = 2, v_2^* = \{v_1^1, v_2^2\}, \Sigma_2^* = \{\Sigma_1^1, \Sigma_2^2\} \\ 3: & s^1 = 2, s^2 = 3, v_3^* = \{v_2^1, v_3^2\}, \Sigma_3^* = \{\Sigma_2^1, \Sigma_3^2\} \\ 4: & s^1 = 3, s^2 = 1, v_4^* = \{v_3^1, v_1^2\}, \Sigma_4^* = \{\Sigma_3^1, \Sigma_1^2\}. \end{cases}$$

em que variáveis indexadas com * pertencem ao diagrama principal.

Seguindo esta lógica, é possível então criar, para cada estado do diagrama principal, vetores semelhantes aos dos diagramas parciais da seguinte forma: $\{v_\sigma^*, \Sigma_\sigma^*, \sigma\}$. É claro que, para o caso univariável, em que apenas um grau de liberdade do movimento está sendo monitorado, esses vetores serão os próprios vetores $\{v_s, \Sigma_s, s\}$ do diagrama univariável (que no caso será o próprio diagrama principal).

Finalmente, com base nesses vetores, pode-se construir, para cada estado do diagrama principal, um modelo linear no espaço de estados. Esses modelos são da seguinte forma:

$$\mathbf{x}_{t+1} = \mathbf{A}_s \mathbf{x}_t + \mathbf{q}_s \tag{1.49}$$

$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \mathbf{r}_t \tag{1.50}$$

$$\mathbf{x}_0 = \mathbf{q}_{s_0}, \tag{1.51}$$

nos quais as matrizes são definidas como

$$x_t = \begin{bmatrix} x_t^1 \\ \vdots \\ x_t^N \\ 1 \end{bmatrix} \qquad y_t = \begin{bmatrix} y_t^1 \\ \vdots \\ y_t^N \end{bmatrix} \qquad q_s^n = \begin{bmatrix} q_s^1 \\ \vdots \\ q_s^N \end{bmatrix} \qquad r_s^n = \begin{bmatrix} r_s^1 \\ \vdots \\ r_s^N \end{bmatrix}$$

$$A = \left[\begin{array}{c|c} I^N & \begin{matrix} v_s^1 \\ \vdots \\ v_s^N \end{matrix} \\ \hline & \end{array}\right] \qquad C = \left[\begin{array}{c|c} I^N & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline & \end{array}\right]$$

e I^N é a matriz identidade de dimensão N.

Também para o diagrama principal, a variável de chaveamento s (ou σ nos casos multivariáveis) evolui segundo a equação 1.7 do modelo de HMM:

$$P(s_{t+1}s_t) = \mathbf{s}_{t+1}^T \Pi \mathbf{s_t}, \tag{1.52}$$

$$P(s_0) = \pi_0. \tag{1.53}$$

Com os modelos lineares construídos para cada estado σ do diagrama principal do movimento e com a equação de evolução da variável, enfim há uma descrição matemática concreta de como cada um deles se comporta, o que finaliza a etapa de parametrização. Com essa informação, finalmente é possível dizer qual é a essência do algoritmo de SLDS.

1.3.1.3.3 Estimação

Ao associar cada modelo do diagrama principal a um estado σ definido no processo de segmentação, criou-se uma maneira de saber como as observações do sistema se comportam para cada estado discreto do movimento.

Assim, se os estados são então alimentados em um HMM, podendo transicionar de acordo com a matriz Π, e supondo que existe algum movimento sendo monitorado pelos

sensores, é possível "chutar" um dos estados do conjunto D e se perguntar: a forma com que o modelo linear associado a esse estado se comporta é coerente com a forma com que as medidas dos sensores estão se comportando? Se sim, o sistema deve manter o "chute". Se não, o sistema deve alternar o "chute" para um estado que tenha um modelo linear de comportamento mais semelhante àquele mostrado pelas amostras.

A maneira com que se faz essa comparação entre os comportamentos das medidas e dos modelos dos estados a fim de decidir se o "chute" do sistema está certo ou não foi explicada na seção 1.3.1.2.1: implementando-se um Filtro de Kalman. Dessa forma, ao detectar uma diferença muito grande entre o comportamento das medidas e o comportamento do modelo, o filtro de Kalman sinaliza para o sistema que é necessário chavear de estado e, consequentemente, de modelo. Por outro lado, a maquina de estados do HMM só permite que o sistema transicione para estados alcançáveis a partir do estado atual.

Uma vez que a dinâmica do sistema consiste justamente em chavear continuamente modelos lineares através de um consenso entre as respostas do Filtro de Kalman e da Cadeia Oculta de Markov, o nome do processo, "Sistemas Dinâmicos Lineares Chaveados", ganha sentido.

Com isso, fica fácil discernir o autômato do modelo, que consiste na estimativa não de uma mas de duas variáveis ocultas ao mesmo tempo: o movimento contínuo do braço do paciente (x_t) e os estados discretos que ele percorre (s_t, ou σ_t no caso multivariável). Ambas devem ser estimadas através da única variável contínua observável: as medidas dos sensores inerciais.

O autômato para Sistemas Lineares Dinâmicos chaveados é apresentado na Figura 23, revelando a relação entre as variáveis ocultas e observáveis em cada nível de interação.

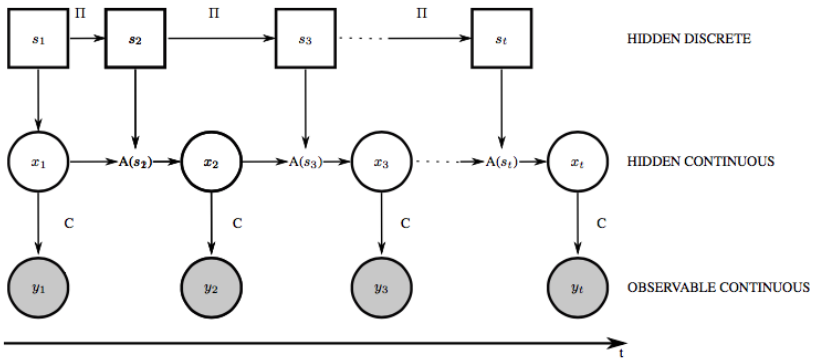

Figura 23 – Autômato de um sistema dinâmico linear chaveado [15].

É claro que o "chute" de estado a que se referiu acima não é de fato um "chute", visto que o estado inicial do sistema é conhecido e que, no evento de um chaveamento de modelo, é sempre possível escolher aquele que seja mais coerente com o comportamento em tempo real das amostras. Então, de fato, esse "chute" será o resultado do processo de

estimação, que será definido matematicamente daqui em diante.

A critério de uma melhor clareza no entendimento do algoritmo, propõe-se aqui explicá-lo à luz de um exemplo multivariável (caso geral) com 3 estados no diagrama principal. Isso quer dizer que foram criados três modelos lineares na etapa de parametrização para prever o comportamento desses estados. Além disso, para visualizar o desenvolvimento das computações e entender a lógica por trás delas, propõe-se também acompanhar através de figuras a estimação de uma das variáveis sendo observadas (as outras terão um desenvolvimento semelhante).

O processo de estimação começa no primeiro ciclo do algoritmo, para $t = 0$. Nesse instante, como sabe-se o estado inicial do sistema e nenhuma estimação ocorreu, a primeira iteração se resume a propagar o estado inicial adiante, como pode ser visto na Figura 24.

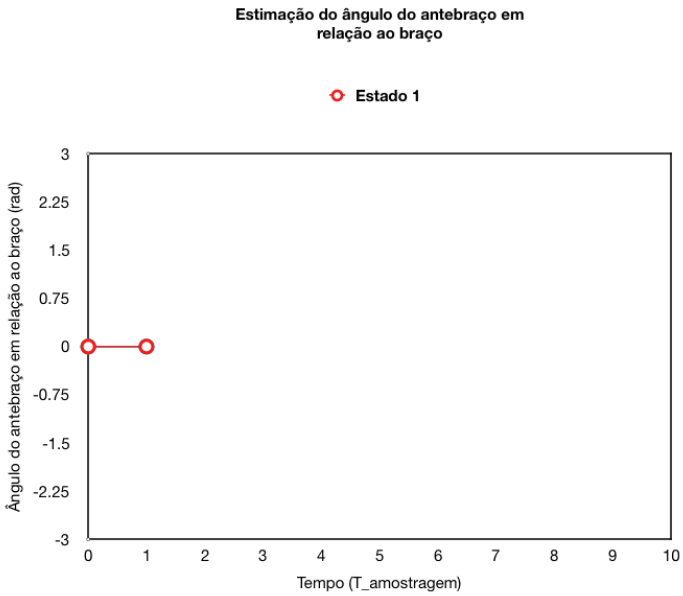

Figura 24 – Primeiro passo da estimação de uma variável (SLDS).

Aqui, antes da chegada da primeira amostra, realiza-se pela primeira vez a etapa de Predição do Filtro de Kalman (equações 1.31 e 1.32). Para cada um dos modelos lineares que o sistema possui, estima-se qual seria o próximo x_t e o próximo Σ_t. Cada modelo, é claro, vai resultar em uma previsão distinta baseada no tipo de comportamento que ele mapeia, o que cria três caminhos distintos (Para um sistema com M estados, M caminhos distintos seriam criados). O resultado desta etapa pode ser visto na Figura 25.

Neste exemplo, o estado 1 está associado a um modelo linear praticamente estático, enquanto os estados 2 e 3 estão associados a dinâmicas ascendentes e descendentes, respectivamente. Assim, o resultado de cada predição segue a lógica por trás de seu modelo associado, supondo que a próxima medida estará mais acima, mais abaixo ou similar ao ponto atual.

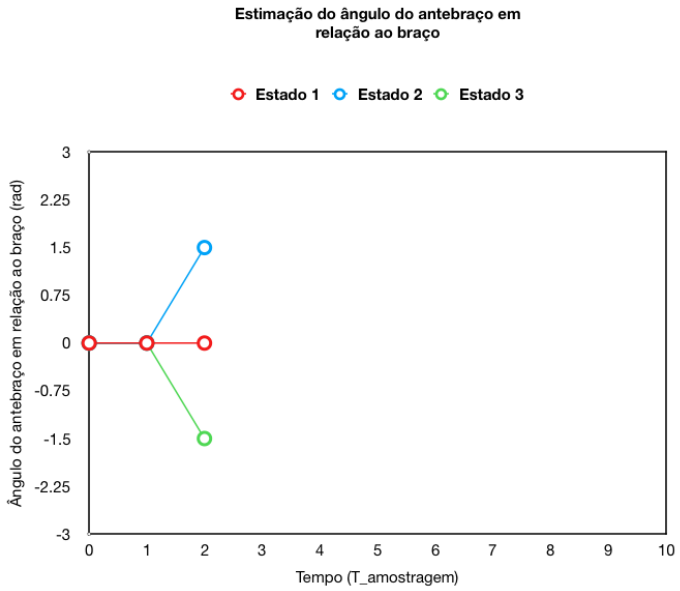

Figura 25 – Predições do Filtro de Kalman (SLDS).

Para a primeira iteração, no entanto, realiza-se ainda uma segunda predição para cada um dos três caminhos criados, gerando assim, ao total, nove estimativas distintas para a variável sendo observada, como na Figura 26.

É importante observar que as figuras dessa seção possuem caráter conceitual e que, para gerar uma melhor visualização, alterou-se a escala da primeira para a segunda predição. Em um gráfico proporcional, todos os segmentos de uma mesma cor teriam a mesma inclinação (velocidade constante). Deste ponto em diante, a escala não será mais alterada.

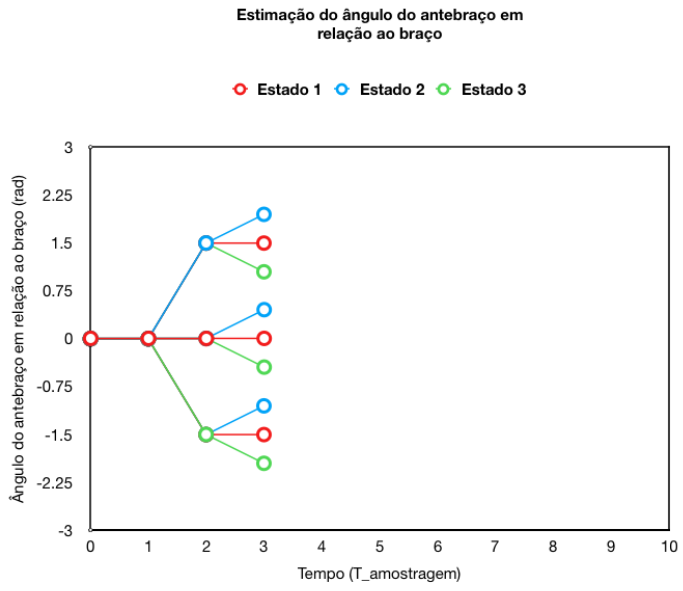

Figura 26 – Segundas predições do Filtro de Kalman (SLDS).

No instante seguinte, a primeira amostra é recebida pela unidade de processamento, como mostra a Figura 27. Imediatamente, o sistema submete suas estimativas à etapa de atualização do Filtro de Kalman (equações 1.33 a 1.38) e, de acordo com a medida recebida, atualiza a posição das nove predições, como na Figura 28.

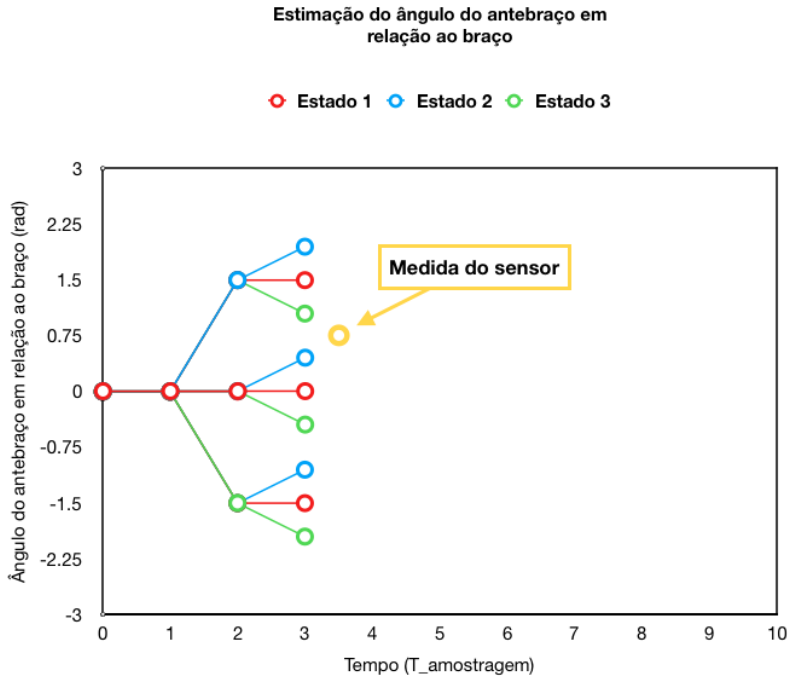

Figura 27 – Recebimento de uma amostra (SLDS).

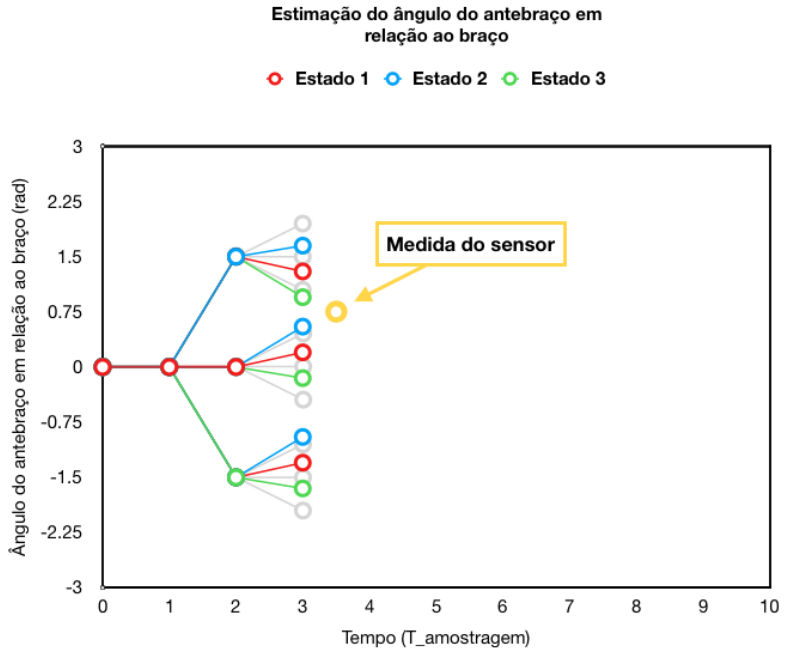

Figura 28 – Atualizações do Filtro de Kalman (SLDS).

Nesse ponto é importante introduzir o conceito de custo. Cada vez que uma etapa

do Filtro de Kalman é finalizada para um dado modelo, ou seja, a cada transição que se cria entre dois pontos na árvore de estimação mostrada nas figuras acima, um custo $C_{t,i,j}$ é a ela associado, em que t é o instante de tempo, i é o estado destino da transição e j é o estado imediatamente anterior, de onde o estado destino foi derivado.

Esse custo $C_{t,i,j}$ é uma função que julga o quão provável é essa transição em vista das respostas do Filtro de Kalman e do HMM. Muitas vezes, principalmente quando ocorrem mudanças de estados, ainda que o HMM diga que é mais provável que o modelo continue o mesmo, o Filtro de Kalman irá dizer que o modelo mudou. Assim, é necessário que haja uma função responsável por "julgar" ambas as informações e balanceá-las de forma a decidir se de fato o estado mudou ou não.

Responsável por esse julgamento, a função custo foi escolhida como sendo uma função Hamiltoniana:

$$C_{t,t-1,i,j} = \frac{1}{2}(y_t - C\hat{x}_{t,t-1,i,j})^T(C\Sigma_{t,t-1,i,j}C^T + R)^{-1}(y_t - C\hat{x}_{t,t-1,i,j}) + \frac{1}{2}\log|C\Sigma_{t,t-1,i,j}C^T + R| - \log\Pi(i,j)). \quad (1.54)$$

Os primeiros dois termos dessa função estão relacionados ao Filtro de Kalman, enquanto o último está relacionado ao HMM. Quanto mais alto o custo de uma transição, menor a chance dela de fato ocorrer.

Quando o erro de inovação ou a variância são baixos, a parte referente ao Filtro de Kalman responde valores pequenos e o termo dominante da equação se torna o do HMM, que geralmente prevê a manutenção do estado. Isso ocorre pois, nesses casos, como $\Pi(i,j)$ é próximo de 1, o *log* mantém-se também em valores baixos e o custo total resulta igualmente em um valor pequeno, tornando essa transição (de um estado para ele próprio) bastante acessível. Dessa forma, pequenas perturbações no Filtro de Kalman são geralmente ignoradas e, como o termo Markoviano impera, o estado se mantém. A probabilidade de transitar de estado nessas condições é muito baixa, uma vez que os valores da matriz de transição do HMM para transições de estados são próximas de zero, o que faz o termo logarítmico explodir, resultando valores muito altos de custo.

Para superar a barreira imposta pelo HMM à transição de estados, é necessário então que haja uma discrepância muito grande entre o modelo do estado atual e as medidas processadas pelo Filtro de Kalman. Assim, quando esses modelos começam a diferir demais, os dois primeiros termos da equação começam a crescer, dominando a função mesmo quando a probabilidade de transitar de estado é muito baixa. Assim, a manutenção do estado se torna muito custosa e chavear os modelos passa a ser uma alternativa atraente para o sistema.

Esse é o primeiro passo de uma aproximação ao algoritmo de Viterbi discutido na seção 1.3.1.1.2, e concentra grande parte da inteligência por trás do algoritmo de SLDS.

Com isso, uma vez atualizadas as 9 (M^2) estimativas realizadas pelo sistema, um custo é calculado através dessa função para cada uma delas. Em seguida, para cada trio de estimativas derivadas do mesmo nó na árvore de estimação, apenas a transição de menor custo é escolhida para se propagar adiante, reduzindo novamente os caminhos a apenas 3 (M), como na Figura 29.

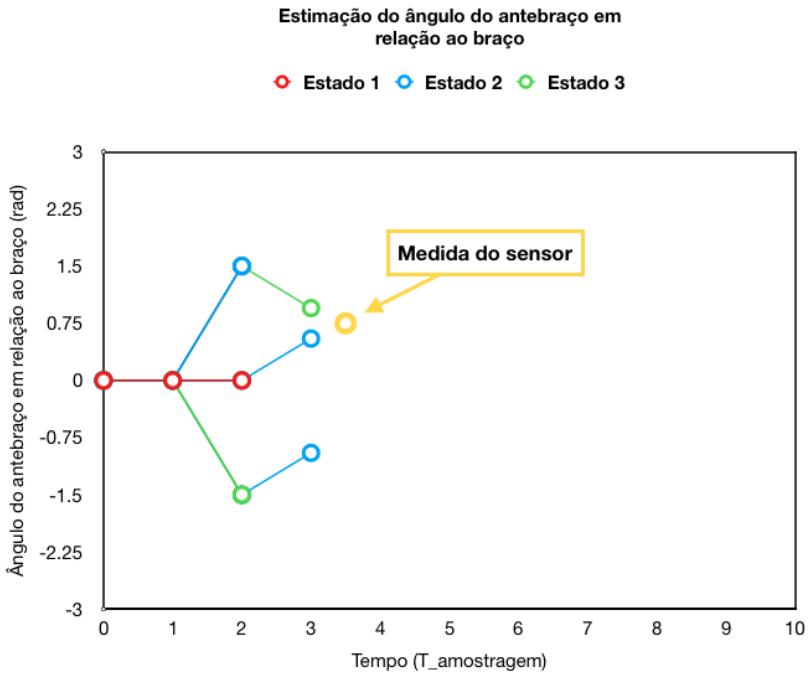

Figura 29 – Poda dos caminhos com base na função custo (SLDS).

Quando as três melhores transições locais são escolhidas, fixam-se então as três novas estimativas na árvore de estimação. Cada nova estimativa fixada, recebe para si também um custo, que será armazenado pelo sistema. Esse custo é definido como

$$C_{t,i} = C_{t,t-1,i,j} + C_{t-1,j}, \tag{1.55}$$

e é a soma do custo da transição calculado no passo anterior (o custo "vencedor") e do custo do estado anterior. Essa portanto, é uma equação recursiva que vai sendo calculada com base nos nós anteriores da árvore. O custo do estado inicial e da primeira transição são definidos como sendo 0.

O próximo passo é repetir novamente, para cada um dos 3 (M) caminhos, o passo de predição do filtro de Kalman. Assim que a próxima amostra chegar, atualizam-se as nove (M^2) predições e, a partir da função custo, novamente selecionam-se as melhores 3 (M) transições locais. Essas etapas estão demonstradas para uma nova iteração na Figura 30.

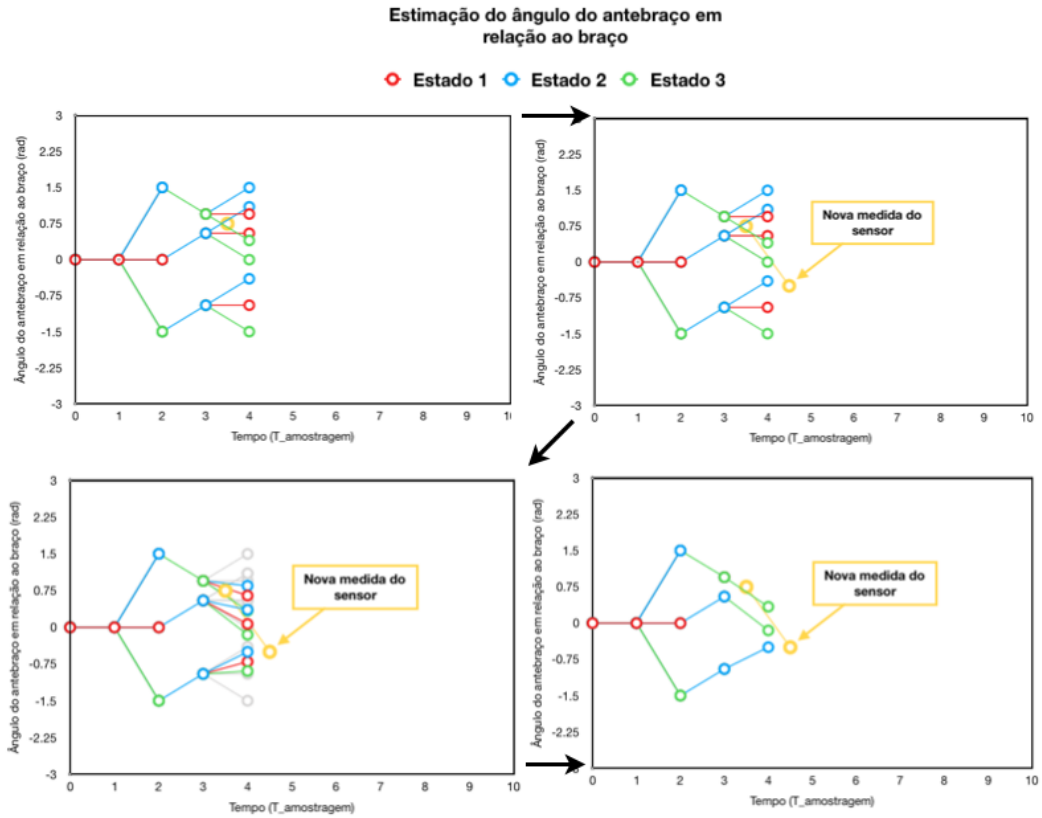

Figura 30 – Iteração completa da etapa de estimação (SLDS).

É interessante perceber aqui que, apesar de cada passo desse processo estar sendo detalhado de forma espaçada, todas essas etapas estão acontecendo centenas de vezes por segundo no intervalo entre sucessivas amostragens da variável observada. Para criar essa referência, o eixo das abscissas dos diagramas de estimação estão normalizados pelo período de amostragem.

A Figura 31 mostra o processo algumas etapas à frente e, a partir dela, é possível identificar visualmente a primeira variável oculta (x_t) sendo rastreada pelo algoritmo. Todavia, falta decodificar a sequência da segunda variável oculta σ_t (s_t no caso univariável).

Ao observar a Figura 31, vê-se que, ao longo do processo de estimação, 3 (M) caminhos vão sendo construídos com as melhores estimativas locais de x_t. Como ao longo do caminho sabe-se o estado ao qual o modelo de cada transição pertence (codificado nas figuras pela cor), sabe-se também, para cada nó na árvore, o estado associado a ele. Isso significa que existem, embutidos na árvore de estimação, 3 (M) vetores contendo uma sequência de estados otimizados localmente para cada transição. Sabendo disso, a pergunta a ser respondida passa a ser como construir o melhor caminho global x_t e a melhor sequência de estados σ_t a partir desses três vetores gerados pela árvore.

Para isso, é necessário realizar um processo *Backward*. Nesse ponto, é interessante observar o algoritmo agindo e perceber que tudo o que foi feito foi avançar através das

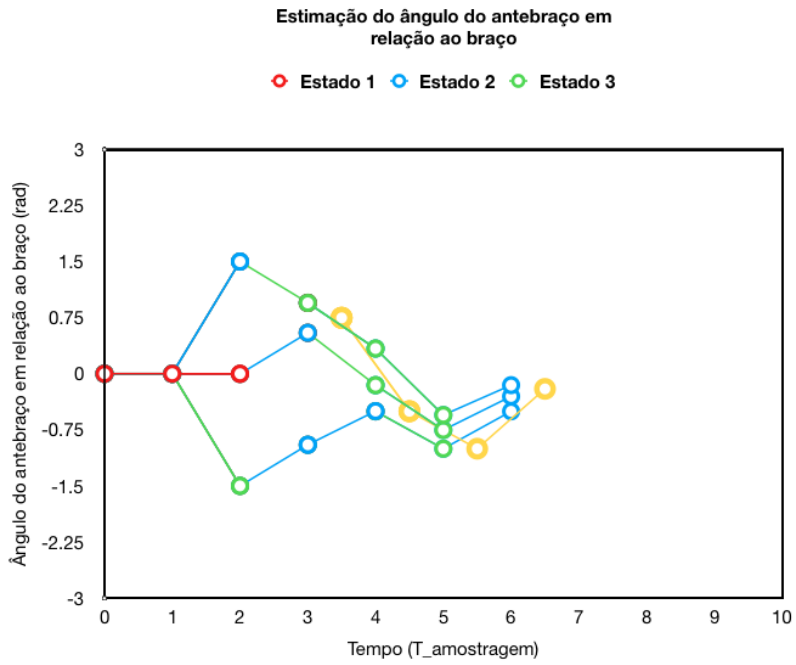

Figura 31 – Árvore de estimações para 4 amostras recebidas.

amostras em uma etapa *Forward*, mas que nunca houve uma etapa de retorno para otimizar as estimativas obtidas, como tipicamente ocorre tanto nos modelos de HMM quanto de LDS.

O processo de retorno (*Backward*) nesse caso é feito da seguinte maneira: em um dado momento da sequência de estimação, antes de voltar para uma nova iteração de avanço (*Forward*), faz-se uma análise de trás para frente de dois dos tipos de vetores armazenados pela árvore de estimações até então: os três (M) vetores de custos $C_{t,i}$ de cada nó da árvore e os três vetores que contêm a sequência de estados estimados para cada um dos três (M) caminhos.

Para cada tempo t em $(t_p, t_a]$ (em que t_a é do instante atual e t_p o instante passado no qual esse retorno foi realizado pela última vez), compara-se os três (M) custos $C_{t,i}$ e, selecionando o menor deles, poda-se os outros dois, firmando o melhor nó dentre os três caminhos para aquele instante de tempo t. Como essa análise é feita independentemente para cada instante de tempo t, pode acontecer de nós vencedores em tempos subsequentes nem mesmo pertencerem ao mesmo caminho. Assim, a melhor sequência de x_t é feita escolhendo os melhores nós para cada instante de tempo desses três (M) vetores. Como sabe-se a que estado cada nó está associado, é imediata a tradução desse resultado para a sequência de estados que melhor descreve a sequência de observações obtida até então.

Supondo que ao alcançar as quatro amostras presentes na Figura 31 o sistema realizou uma etapa do retorno, o resultado seria aquele mostrado na Figura 32, descrevendo a sequência de estados estimados {1, 1, 2, 3, 3, 2} (em que 1 seria o estado em vermelho,

2 aquele em azul e 3 aquele em verde).

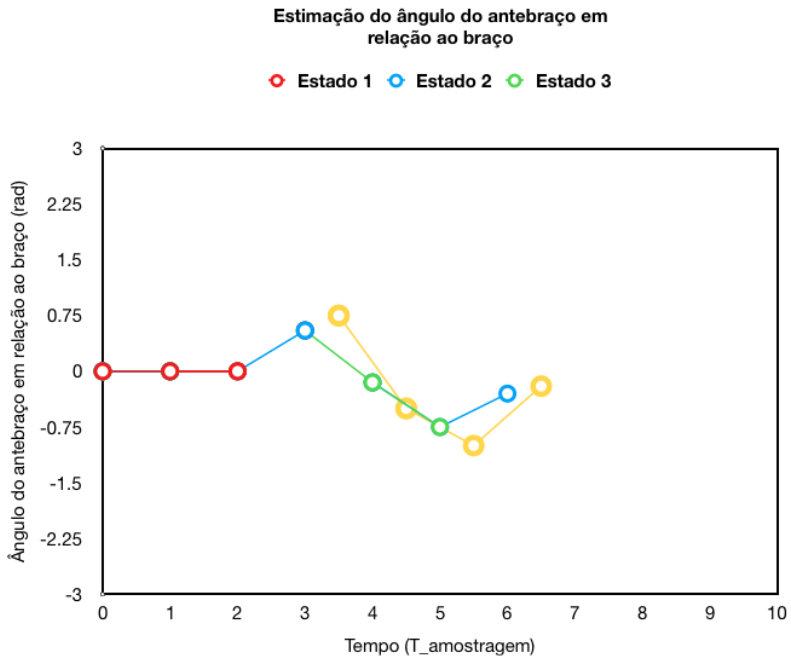

Figura 32 – Melhor caminho para suavização com janela de 4 iterações.

É importante notar que o momento em que se realiza esse procedimento de *Backward* é crucial para o desempenho do sistema em análises mais ruidosas. Como foi explorado no início dessa seção, é possível realizar este retorno em três momentos diferentes: a cada iteração, recebendo o nome de Filtragem; a cada k iterações em uma janela de tamanho fixo, recebendo dessa forma o nome de Suavização; ou apenas ao final da aquisição de dados, realizando então a decodificação completa pelo algoritmo de Viterbi.

Novamente, realizar um processo *online* que utiliza somente Filtragem ou no máximo uma Suavização de janela curta é útil pois permite acessar resultados em tempo real, mas não reune tanta informação quanto métodos de maior *lag* e por isso não fornece estimativas tão acuradas. Para sistemas "comportados", isso não costuma ser um problema; já para sistemas propensos a maiores perturbações é recomendado que se utilize aplicações *offline*.

A partir do resultado mostrado na Figura 32, pode-se ver que, de fato, o que o algoritmo realizou foi uma segmentação automática do movimento. Assim, baseado na segmentação inicial realizada manualmente pelo usuário, o sistema já não precisa mais de um operador para este procedimento e pode então executá-lo por si só para movimentos subsequentes de mesma natureza. Como, além disso, o algoritmo também possui conhecimento acerca do comportamento de cada estado do sistema, então toda a informação necessária para análise está à sua disposição.

Isso resume então a operação de um Sistema Dinâmico Linear Chaveado (SLDS).

Por conta da integração marcante entres as noções de SLDS e do algoritmo de decodificação de Viterbi, esse algoritmo é então chamado SLDS-Viterbi aproximado.

2 Estado da Arte

Este capítulo discursa brevemente sobre os últimos avanços no campo de sistemas automáticos de avaliação de movimento. O estado da arte dessa área será avaliado em dois contextos: um contexto de metodologias utilizadas para o processamento de dados e um contexto de aplicação.

2.1 Contexto de metodologia

Como comentado anteriormente, um grande conjunto de tecnologias emergentes tem surgido no campo de sensoreamento e aquisição de dados. Nesse âmbito, como apontado na seção 1.2.1, os sistemas de captura de movimento têm avançado em duas direções simultaneamente: os sistemas mais profissionais estão se tornado cada vez mais acurados e precisos e, no outro extremo do espectro, sistemas mais simples tem ganhado espaço no mercado residencial ao ficarem cada vez mais baratos sem perder a qualidade. A inteligência por trás do processamento dos dados obtidos por esses sistemas, no entanto, é um tópico ainda em estágio inicial de crescimento, principalmente no que diz respeito à avaliação automática de movimentos como proposta pelo presente trabalho [15].

De forma a avaliar o que se tem feito de mais recente neste campo, é necessário fazer em princípio uma distinção entre os processos de Segmentação e Classificação de movimentos.

A Segmentação está relacionada com a execução singular ou repetida de um movimento previamente conhecido e com a divisão interna desse movimento em pequenas seções das quais se pode extrair parâmetros e informações úteis. Esse processo pode ser feito manualmente com o objetivo de criar movimentos parametrizados (como explicado em 1.3.1.3), mas o real objetivo é que, com base nessa parametrização, os sistemas de estimação possam realizar essa segmentação sem supervisão com movimentos posteriores. É a esse processo automático de análise de movimentos que se refere aqui como Segmentação.

A Classificação pode ser vista, de certo modo, como uma expansão natural da Segmentação. Ela envolve segmentar não um único movimento mas uma sequência de movimentos distintos e desconhecidos de forma a identificar cada execução individualmente e então rotulá-las de acordo com um conjunto conhecido de movimentos individuais.

A divisão que se faz na literatura, no entanto, é entre dois problemas básicos: o problema de Segmentação (com ou sem Classificação) e o problema de Extração de Parâmetros. As soluções atuais na área de avaliação automática de movimentos em geral atacam um desses dois problemas separadamente. A abordagem de [39] e de [15] são espe-

cialmente interessantes uma vez que lidam, de formas diferentes, com ambos os problemas.

O trabalho de [39] utiliza segmentação para identificar os movimentos individuais do teste *Timed Up and Go* (levantar, marcha, giro de 180^o e sentar), vulgo TUG, e então, para cada movimento identificado, aplica modelos extremamente específicos para realizar a extração de parâmetros. Por conta dessa extrema especificidade, a abordagem passou a ser pouco intuitiva e adquiriu baixa usabilidade [15].

Dessa forma, a proposta feita por [15] (baseada no modelo SLDS de estimação) ganha importância própria no campo, destacando-se como a única dentre as referências estudadas capaz de, através do mesmo método, lidar com ambos os problemas de forma geral.

A Tabela 1 mostra diversos estudos realizados no intuito de lidar com cada um desses problemas. Os estudos que trataram do problema de Segmentação foram [15, 39, 40, 41, 42]. Aqueles que se ocuparam também da Extração de Parâmetros foram [15, 39].

Em relação ao problema de Segmentação, alguns dos algoritmos utilizados são:

- DTW (*Dynamic Time Warping*), que encontra um alinhamento ótimo entre duas séries temporais para discernir momentos de transição (utilizado por [40] para segmentação do teste TUG). Requer menos ajustes de parâmetros e um menor conjunto de dados para treinamento, mas necessita de um *template* específico para cada sequência de movimentos;

- Algoritmo de identificação de posições chave, que utiliza classificadores estatísticos para alinhar as séries temporais de movimentos executados com aquelas de *templates* treinados anteriormente (utilizado por [41] para tratar dados obtidos pelo Kinect). Os *templates* são rotulados manualmente e, ao serem passados por uma função específica, geram uma "assinatura" do movimento, que é então utilizada posteriormente para a classificação. Permite refinar os *templates* com novas gravações, mas só modela um movimento por vez;

- Combinação de ZVC tradicional (*Zero-Velocity Crossing*) com métodos de HMM, que utiliza também a lógica de *templates* para segmentação *online* e para classificar diferentes movimentos em uma sequência randômica (utilizado por [42] através de diversos sensores);

- Modelo SLDS (*Switching Linear Dynamic Systems*), que faz uso da gravação, rotulação e parametrização de movimentos em uma fase de treinamento para gerar modelos lineares que, ao serem acompanhados por um Filtro de Kalman durante a estimação, podem ser chaveados pela dinâmica de um HMM no intuito de segmentar o movimento em uma série temporal coerente com aquela parametrizada [15]. Esse foi o método descrito na seção 1.3.1.3 deste texto.

Fonte	Salarian, 2010 [54]	Adame, 2012 [55]	Dios, 2014 [57]	Lin, 2014 [56]	Baptista, 2016 [16]
Sensor	IMU (acelerômetro + giroscópio)	IMU (giroscópio)	Kinect	IMU & MOCAP óptico	IMU (acelerômetro)
Dimensão	Univariável	Univariável	Multivariável	Multivariável	Multivariável
Variáveis analisadas	Tronco, canela e coxa (um para cada movimento)	Inclinação (sentar-levantar) e guinada (giro)	Múltiplos ângulos	Múltiplos ângulos e acelerômetro	Múltiplos ângulos
Movimentos	TUG	TUG	Diversos movimentos discretos	Diversos movimentos discretos	Diversos movimentos discretos
Segmentação	Função matemática projetada de forma específica	DTW	Algoritmo ID de posição chave (classificadores estatísticos + assinatura de movimento)	ZVC	SLDS
Reconhecimento	-	-	-	HMM, DTW	SLDS
Extração de parâmetros	Procedimento específico para cada tipo de movimento	-	-	-	SLDS
Online/ Offline	Offline	Offline	Online	Online	Online

Tabela 1 – Comparação entre métodos de tratamento dos problemas inerentes da avaliação de movimentos [15].

Novamente, pela natureza abrangente da solução proposta e pela conveniência de estar em contato com o autor de [15], decidiu-se que a metodologia do modelo SLDS seria aquela utilizada neste trabalho.

2.2 Contexto de aplicação

Sob uma perspectiva de mercado, é importante ainda entender como os sistemas processados de análise de movimentos estão se desenvolvendo.

Da mesma forma que as ferramentas de aquisição de dados, as plataformas de mercado para análise de movimentos humanos tem crescido muito, principalmente no que diz respeito à portabilidade das soluções. Isso implica que cada vez mais é possível separar a fisioterapia do ambiente clínico e transportá-la para o ambiente pessoal.

Dois sistemas cujo objetivo é exatamente este são desenvolvido pela empresa Sword

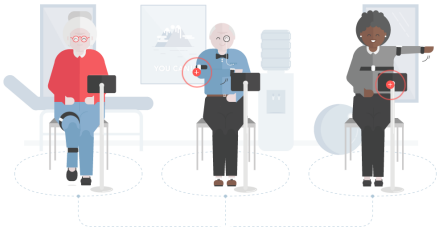

Figura 33 – Prospecto da plataforma "Sword Phoenix"da empresa Sword Health [3].

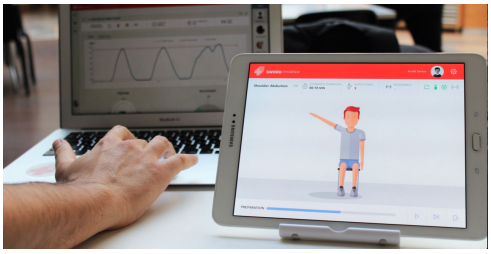

Figura 34 – Plataforma "Sword Arya"da empresa Sword Health [16].

Health [3] e são chamados "Sword Arya"e "Sword Phoenix". Ambos são a idealização de mercado ambicionada a nível de pesquisa por este trabalho, e consistem em uma interface de monitoramento fisioterapêutico aliada ao uso de IMUs. Todo o aparato é disponibilizado para o paciente, que pode realizar os exercícios de sua própria casa em integração com a interface, através da qual um profissional pode então acompanhar o seu desempenho (Figuras 34 e 33). A Sword Health é uma empresa recente que começou a se alastrar pelos Estados Unidos e China em 2017 [16], e que já prevê começar nesse ano a firmar parcerias com clínicas de fisioterapia. O seu moto é a completa reinvenção da fisio-reabilitação [3].

Outra empresa que tem se desenvolvido nesse ramo é a Jintronix, que associa o processo fisioterapêutico a jogos computacionais com detecção de movimento. Esses jogos podem ser jogados em casa pelos pacientes e enviam aos profissionais em tempo real um conjunto de *feedbacks* motores pertinentes ao usuário. Os exercícios propostos pelos jogos são desenvolvidos em conjunto com profissionais da área da saúde e, através de uma plataforma de "telereabilitação" oferecida, é possível estabelecer um laço entre esses profissionais e os pacientes, que recebem sugestões acerca de novos movimentos e respostas acerca dos seus progressos [2]. O intuito é transportar a atmosfera maçante das seções de fisioterapia para um ambiente lúdico e interativo (Figura 35).

Nenhum dos dois sistemas, no entanto, ao menos até onde se pôde entender, utilizam técnicas de avaliação automática do movimento. Em ambos os casos, sempre há um fisioterapeuta responsável por avaliar os dados colhidos pelos exercícios.

Uma empresa que utiliza métodos de avaliação automáticos em um contexto de reabilitação, ainda que não para movimentos, é a CogniFit [43], que oferece um *software* de

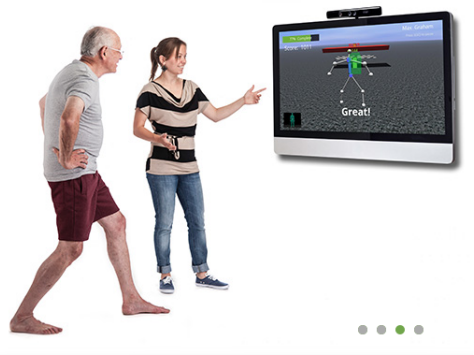

Figura 35 – Imagem de jogo apresentada pelo site da Jintronix [2].

treinamento e reabilitação para habilidades cognitivas. Dezenas de estudos realizados com base nos programas da CogniFit mostram melhoras significativas no desempenho cognitivo de pacientes com diversas condições distintas (desde depressão e insônia a AVCs) [44]. Pareada com um conjunto de associações de neurologia, a empresa já possui milhões de usuários explorando os testes propostos pelo seu *software*.

Todas essas plataformas revelam como não só a pesquisa acerca dos algoritmos de estimação é importante mas também o estudo da sua dimensão de mercado, que, sem dúvidas, tende a aumentar nos próximos anos.

A plataforma que se pretende desenvolver para este trabalho não tem a pretensão de alcançar o patamar de estabilidade e portabilidade daquelas oferecidas por empresas como Jintronix ou Sword Health, mas pretende funcionar como uma prova de conceito para a integração dessas plataformas com um algoritmo robusto de estimação de movimentos.

3 Desenvolvimento

Se os capítulos anteriores abordaram toda a teoria por trás deste trabalho, este será o capítulo responsável por documentar toda a implementação prática da plataforma criada. Ao final do capítulo, o leitor terá entendido como cada elemento do sistema foi construído e integrado para gerar, ao final, a aplicação idealizada.

A título de desenvolver uma melhor linha de raciocínio, apresenta-se aqui a compilação das especificações para a plataforma que se desejava criar:

1. Implementar um *driver* para receber os dados das IMUs YEI 3-Space Sensors (YOST Labs) através do *dongle* sem fio e enviá-los à unidade de processamento;
2. Implementar, através da linguagem Python, uma unidade de processamento em *Back-end* capaz de:
   - 2.1. Indexar no tempo os dados recebidos do *driver*;
   - 2.2. Criar diagramas de ciclo a partir dos dados recebidos do *driver*;
   - 2.3. Segmentar os diagramas de ciclo para extração de parâmetros do movimento;
   - 2.4. Criar modelos lineares referentes a cada segmento dos movimentos parametrizados;
   - 2.5. Inserir os modelos em um *framework* SLDS;
   - 2.6. Avaliar movimentos subsequentes através do algoritmo de SLDS.
3. Implementar, através da linguagem JavaScript, uma interface gráfica em *Front-end* capaz de:
   - 3.1. Apresentar um modelo esqueletal completo do braço (desde o ombro até a mão), em três dimensões, em conformidade com o que é oferecido pelas interfaces de captura de movimento atualmente no mercado;
   - 3.2. Fazer o modelo do braço acompanhar tão fielmente quanto possível o braço do paciente através do rastreamento das IMUs em tempo real durante todo o período de uso da aplicação;
   - 3.3. Apresentar um menu de opções com botões para iniciar, pausar, retomar e cancelar a comunicação com as IMUs, e para conduzir o usuário pelo processo de gravação, segmentação e estimação de movimentos;
   - 3.4. Durante a fase de gravação e segmentação, apresentar na tela o diagrama de ciclos do movimento inicial a ser parametrizado;

3.5. Retornar para o usuário, em tempo real, os resultados da estimação provenientes do SLDS.

4. Estruturar um servidor local em Python para servir a interface gráfica em JavaScript durante a aquisição e o processamento de dados.

É válido ressaltar aqui que, no início deste trabalho, o autor não possuía qualquer experiência em Python, qualquer experiência de comunicação de rede (estrutura servidor-cliente), pouca ou quase nenhuma experiência com as IMUs YEI 3-Space Sensors e média experiência em JavaScript. Por conta disso, nos seis meses de implementação deste projeto foi necessário comprimir também uma curva de aprendizado bastante íngreme a fim de atingir os objetivos propostos.

É evidente que o trabalho de projetar um sistema e implementá-lo na prática é extremamente iterativo, o que significa que quanto mais se desenvolve a plataforma, mais se encontra maneiras de melhorá-la e aprimorá-la em diversos aspectos. O resultado que se apresenta neste capítulo foi aquele obtido ao final dos seis meses de desenvolvimento do sistema. Considerações acerca das próximas iterações da plataforma e de trabalhos futuros que possam ser realizados nesse mesmo âmbito serão discutidas no capítulo de Conclusão.

De forma semelhante àquela proposta na seção 1.2, dividiu-se aqui o desenvolvimento do projeto em duas partes: *Hardware* e *driver* de comunicação; e Desenvolvimento de *Software*. A Figura 36 mostra a visão de mais alto nível do sistema. Cada seção daqui em diante tratará dos aspectos que lhe são pertinentes e, ao longo do capítulo, ambos os blocos da Figura 36 serão destrinchados.

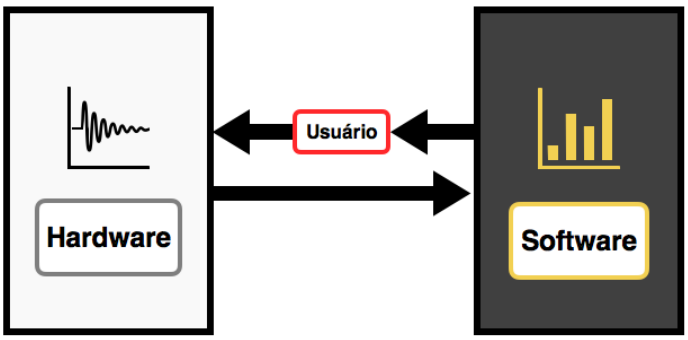

Figura 36 – Diagrama de blocos de mais alto nível do sistema.

3.1 *Hardware* e *driver* de comunicação

Como dito na seção 1.2.1.2, o *hardware* de aquisição de dados escolhido foi o conjunto de IMUs *YEI 3-Space Sensors* da YOST Labs. Essas IMUs possuem duas maneiras

distintas de comunicação: através de um cabo Mini-USB - USB que pode ser conectado entre elas e um computador, ou através de um *dongle* que permite acessar, sem fio, até 15 IMUs distintas.

Como o objetivo da solução envolvia posicionar as IMUs no corpo do paciente, a opção sem fio foi, desde o início, a mais atraente. Assim, o *setup* da aplicação foi definido a partir da utilização de um *dongle* sem fio e de duas IMUs que poderiam então ser posicionadas livremente na terminação distal do braço (logo antes do cotovelo) e na terminação distal do pulso (logo antes da mão). Esse posicionamento das IMUs permitiu, ao fixar a posição do ombro na interface, monitorar a posição do braço e do antebraço em tempo real.

O *dongle* funciona como um concentrador dos dados das IMUs, que são recebidos, compactados e então enviados via USB ao computador a uma *baudrate* de 115200 bps (valor padrão). A comunicação sem fio com as IMUs é feita sob o protocolo DSSS 2.4 GHz (mesma faixa de WiFi ou Bluetooth), que é utilizado para monitorar dados de orientação em atividades com alto nível de mobilidade [10].

O módulo que se comunica com o *dongle* e recebe os dados das IMUs é, como também dito anteriormente, o *driver*. Esse foi o primeiro módulo do projeto e, por conta disso, trouxe consigo algumas decisões importantes. A primeira delas foi a do ambiente de programação, que foi escolhido como sendo o *software* de edição de texto multiplataforma para programação *Sublime Text 2* [45].

A segunda decisão que se teve de tomar em relação a esse módulo foi a de que linguagem utilizar para o seu desenvolvimento. Tipicamente, os dados adquiridos pelo *hardware* são alimentados para o *Back-end* do sistema, onde o processamento ocorre, e daí são distribuídos para quaisquer outros módulos que necessitem deles perifericamente. Assim, uma vez que a linguagem definida para o *Back-end* da aplicação foi Python, essa foi também a linguagem escolhida para implementar o *driver*.

Afortunadamente, a YOST Labs oferece uma biblioteca em Python chamada *threespace api* que já contém a API para os *YEI 3-SPace Sensor*. Isso, somado ao fato de que essas mesmas IMUs já haviam sido utilizadas em outros projetos do LARA (Laboratório de Automação e Robótica da Universidade de Brasília, através do qual este projeto foi desenvolvido), e que já havia um *driver* preliminar implementado, fez com que o desenvolvimento desse módulo levasse bem menos tempo.

Esse *driver* preliminar, no entanto, suportava apenas a aquisição de dados de uma IMU. Apesar de ser o suficiente para testar o *dongle* e ambos os sensores separadamente, era preciso adaptá-lo para alimentar o sistema com ambas as IMUs previstas nas especificações. Para isso, foi necessário estudar a documentação tanto das IMUs [10] quando da API [46].

As alterações realizadas levaram em conta mas não se limitaram a:

- Utilizar uma função de endereçamento presente na API para que o *dongle* pudesse identificar cada uma das IMUs sendo utilizadas com base no seu número serial;
- Vetorizar o buffer de dados para suportar a aquisição de duas IMUs simultaneamente;
- Incluir uma "tara" (referência) às medições das IMUs, de forma que elas seguissem sempre uma mesma orientação;
- Ajustar os eixos de orientação das IMUs;
- Ajustar a frequência de amostragem com a qual se recebia dados das IMUs.

A necessidade de ajustar os eixos de orientação veio na verdade em uma etapa posterior do processo, quando a interface gráfica foi construída e precisou-se adequar os eixos de medição com aqueles usados para representar o braço no espaço tridimensional. O inverso também poderia ter sido feito, mas, por incrível que pareça, essa foi a alternativa mais prática.

Os eixos padrão das IMUs estão mostrados na Figura 35. A nova orientação escolhida foi com o eixo-X apontando para frente e o eixo-Y apontando para a esquerda e com o eixo-Z apontando para cima (considerando a base da Figura 37). Essa mudança foi feita através de um método simples de ajuste de orientação que pode ser encontrado em [10] pelo nome "*setAxisDirections()*".

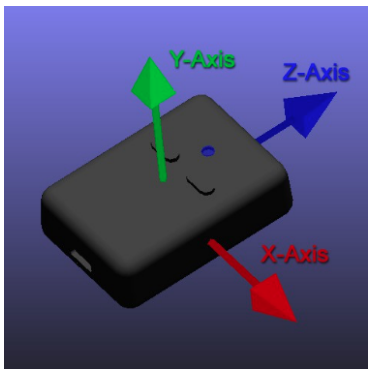

Figura 37 – Eixos de orientação padrão das IMUs (eixo-X apontando para a direita, eixo-Y para cima e eixo-Z para frente [10].

O último item citado na lista de alterações (ajustar a frequência de amostragem de requisição de dados) está relacionado ao fato de que as IMUs operam (quando nenhum filtro específico está selecionado e sem fazer uso de *oversampling*) a uma taxa de aquisição máxima de 1350 Hz, o que quer dizer que não faz sentido pedir dados a uma taxa maior

do que essa. Nesse momento, estabeleceu-se então que a frequência com a qual o *driver* requisita dados às IMUs seria de 1 kHz.

O modo escolhido para requisitar esses dados foi o modo *stream*, no qual só é necessário ajustar os parâmetros das IMUs e do *dongle* uma vez. A partir desse ajuste, sem precisar de outros trechos de comunicação, pode-se deixar as IMUs em modo contínuo de aquisição, requisitando-se os dados de acordo com a necessidade do *software*.

A API oferece uma dezena de tipos de dados diferentes que podem ser requisitados. Inicialmente, escolheu-se trabalhar com ângulos na base Euleriana por motivos de praticidade e conveniência. Percebeu-se posteriormente, no entanto, que essa não era a melhor opção e que havia problemas intrínsecos a essa base no desenvolvimento de aplicações gráficas que precisam lidar com o conceito de espaço tridimensional. Por este motivo, decidiu-se passar a requisitar a orientação das IMUs em uma outra base também suportada pela API, e que é extensamente utilizada nesse tipo de aplicação: quaterniões. Essa decisão e os motivos que levaram a ela serão mais aprofundados na seção 3.2.2.

Diferente dos outros, o *driver* das IMUs foi talvez o módulo menos trabalhoso do projeto. Novamente, isso se deu pelo fato dele já estar parcialmente implementado e por possuir, além de uma biblioteca pronta para Python, uma documentação bastante acessível.

Assim, ao final dessa seção, o diagrama de blocos da Figura 36 (excluindo-se o nó e os ramos do usuário) pode ser expandido para aquele da Figura 38.

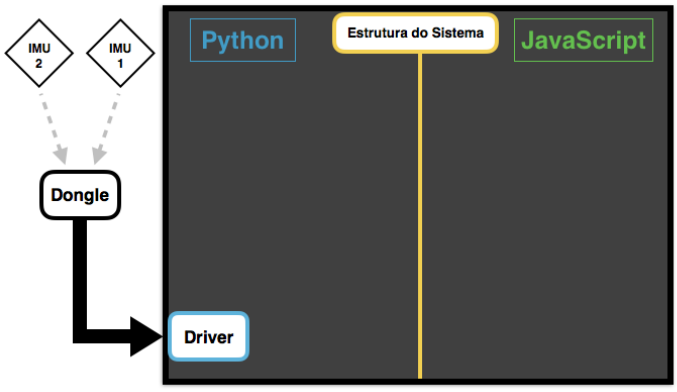

Figura 38 – Diagrama de blocos do sistema com *hardware* e *driver* definidos.

3.2 Desenvolvimento de *Software*

Uma vez testado o *driver* e conferido que a aquisição de dados de ambas as IMUs estava funcionando perfeitamente, prosseguiu-se então para o grande desafio do projeto:

o desenvolvimento do *software*, no qual reside o coração da aplicação que é o algoritmo SLDS-Viterbi aproximado.

Antes de chegar à implementação da camada de processamento, todavia, é necessário descrever o desenvolvimento do "cartão de visitas" do sistema, o grande motor à frente da intuitividade da solução: a interface gráfica.

Deste ponto em diante, as expressões "camada de processamento" e "unidade de processamento" serão utilizadas intercambiavelmente. Da mesma forma, como já mencionado anteriormente, as expressões "GUI" e "Interface gráfica".

Vale apontar que o sistema foi desenvolvido em duas etapas. Inicialmente, apenas o caso univariável (um único ângulo sendo monitorado) foi tratado. Isso foi muito vantajoso pois, como o desenvolvimento do sistema ocorreu quase simultaneamente com o estudo e compreensão do algoritmo de estimação, a implementação de um caso mais simples em primeira instância foi ideal para uma boa sedimentação dos conceitos. Uma vez validado o funcionamento do sistema para o caso univariável, voltou-se adaptando cada módulo para o caso multivariável.

Apesar disso ter acontecido em dois momentos distintos, as seções a seguir tratarão do desenvolvimento como um único bloco de atividades, dividindo o progresso não exatamente no tempo mas principalmente em relação aos módulos codificados.

3.2.1 Interface Gráfica

O primeiro objetivo de curto-termo assumido após verificar-se que as IMUs estavam operantes foi o de construir o GUI do projeto, o palco onde toda a visualização aconteceria dali para frente. Para isso, a biblioteca "Three.js" foi importada e uma cena foi criada da maneira clássica: uma variável cena, uma variável câmera e uma variável renderizador. Tudo o que isso gera é uma tela escura, como uma folha vazia, preparada para receber os objetos.

Como a principal missão do GUI (além de guiar o usuário pelo processo de estimação) era conter um modelo esqueletal de um braço humano que acompanhasse o braço do paciente em tempo real através das medidas das IMUs, iniciou-se pela tentativa de criar esse modelo de braço.

Quando se cria um objeto pela biblioteca Three.js, são necessárias três coisas: uma geometria, um material e uma malha que una os outros dois em uma entidade única. Quando cria-se mais de um objeto, é possível uni-los através dos comandos *add*, *merge* ou criando um grupo de objetos, o que em teoria permite movê-los juntos pelo espaço.

Ao pensar no modelo de um braço, é quase intuitivo pensar que a melhor aproximação geométrica seja a de um conjunto de segmentos lineares unidos por juntas. De

fato, visualmente, essa é uma boa aproximação. Contudo, há detalhes neste modelo que são mais complexos do que parecem à primeira vista. Cada segmento, por exemplo, deve mover-se em pelo menos alguns graus de liberdade com total independência do segmento anterior; entretanto, ao mover um determinado segmento, todos os outros que estejam conectados ao membro e que sejam posteriores a ele devem mover-se de forma conjunta. Um exemplo disso é que é possível mover o antebraço pelo espaço sem necessariamente mover o braço mas, se o braço está fixo, é impossível mover o antebraço sem mover também a mão.

A primeira abordagem para o modelo do braço foi a de um conjunto de cilindros que funcionassem como os ossos, intercalados por esferas que funcionariam como juntas. Esse grupo de objetos seria então unido usando alguma das técnicas convencionais mencionadas acima. No entanto, os detalhes de movimentação conjunta e organizada inerentes de um braço humano real tornaram essas alternativas impraticáveis. Seria preciso criar dezenas de restrições e equações que regessem o movimento de todos eles ao mesmo tempo de forma a aproximar a dinâmica de um braço humano se se quisesse manter essa abordagem. Por isso, decidiu-se buscar outras opções.

Foi então que encontrou-se, na própria documentação da biblioteca [47], um tipo de objeto chamado *bone* ("osso", em Português). Ao que a documentação indicava, esses ossos podiam ser conectados pelos vértices de forma sequencial para criar modelos corporais na forma de um outro objeto chamado *skeleton* ("esqueleto", em Português). Além disso, um tipo de malha chamado *Skinned Mesh* ("Malha de Pele", em Português), permitia revestir esses ossos com formas geométricas flexíveis que seguiam seus movimentos. Era a ferramenta ideal.

A partir daí, o tempo foi dedicado a entender como os ossos se relacionavam e como conectá-los para construir o esqueleto de um braço. Os resultados progressivos desse processo estão mostrados na Figura 39.

Após construído o modelo, foi o momento de aprender a movimentá-lo. A primeira abordagem foi procurar uma biblioteca que possibilitasse movimentar objetos pelo espaço através de entradas como o mouse ou o teclado (existem dezenas de bibliotecas desse tipo para Three.js), e então adaptá-la para, ao invés de buscar dados dos periféricos do computador, utilizar aqueles provenientes das IMUs.

Uma das bibliotecas encontradas foi a biblioteca "TransformControls.js" que, de forma bastante intuitiva, permite rotacionar, transladar e modificar a escala dos objetos da cena. Apesar de ser bastante interessante, adaptar a estrutura do programa a ela seria contra-producente. Por fim, tudo o que se manteve dessa biblioteca foram alguns aspectos gráficos, o que então não resolveu o problema da movimentação do modelo.

Ao desenvolver aplicações gráficas que lidam com o espaço tridimensional, é inevi-

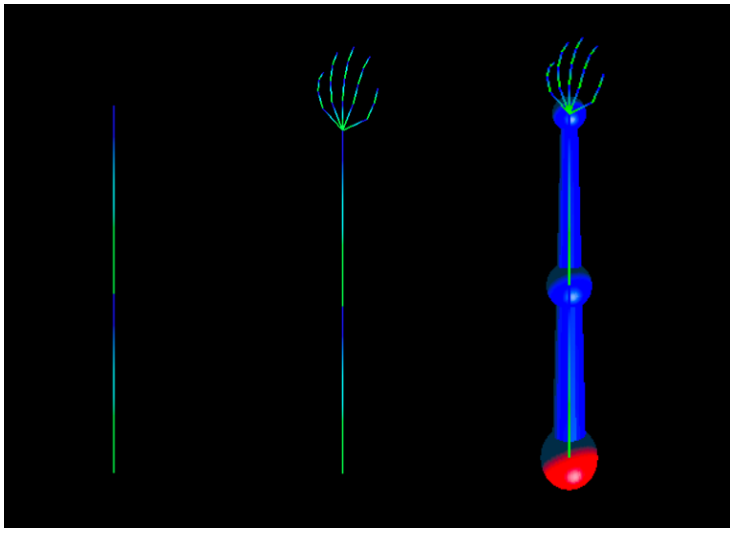

Figura 39 – Etapas do processo de criação do modelo de braço utilizado na interface gráfica. A progressão dos passos está definida da esquerda para a direita, o terceiro modelo sendo aquele utilizado na aplicação.

tável que se passe pelo problema de movimentar e girar objetos no espaço. Isso fica ainda mais marcante quando o objeto é um modelo esqueletal que possui juntas, ossos e diversos eixos distintos, como discutido acima.

A segunda abordagem para gerar a movimentação do modelo foi a abordagem manual, ou seja, a modificação bruta, sem uso de bibliotecas, dos parâmetros espaciais dos objetos na cena. Nesses casos, geralmente a escolha mais óbvia é utilizar ângulos na base Euleriana para denotar as variações angulares entre os objetos. Isso, no entanto, pode causar alguns problemas.

Segundo o teorema da rotação de Euler, é possível descrever uma rotação arbitrária por meio de três ângulos distintos de rotação em torno dos eixos coordenados do sistema. Em outras palavras, a rotação resultante sempre pode ser decomposta em três rotações parciais.

É interessante notar que podem existir diversos caminhos para uma rotação resultante. Em outras palavras, é possível combinar vários conjuntos diferentes de três ângulos que resultem na posição final desejada.

Essa abordagem pode ser vista de forma muito clara em um sistema de *gimbals*, mostrado na Figura 40. Nesse sistema, cada um dos três aros (*gimbals*) pode girar livremente, denotando um dos graus de liberdade do teorema da rotação de Euler.

O problema ocorre quando dois ou mais desses aros se alinham durante uma rotação. Nesse caso, o sistema perde uma das referências pois fica com um grau de liberdade a menos, e isso desorienta o objeto sendo rotacionado. O nome deste fenômeno é *"Gimbal Lock"*, e é um problema em diversos ambientes de programação para jogos, GUIs e

também para modelos reais em engenharia mecânica. Quando essa referência é perdida, é necessário reverter as rotações realizadas e utilizar outra combinação de ângulos que leve para o mesmo lugar. No caso de uma interface gráfica, nesse momento o objeto parece se mover de forma estranha na tela, "contorcendo-se" antes de encontrar o caminho certo.

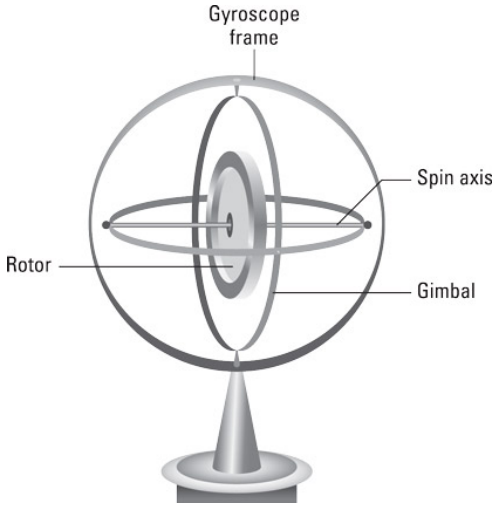

Figura 40 – Sistema de *gimbals* [17].

Para evitar este problema, grande parte dos programadores de jogos e de GUIs tridimensionais preferem deixar os ângulos de Euler de lado e utilizar um outro conceito bastante útil para rotação de objetos: os quaterniões. Foi por isso que mostrou-se (vide seção 3.1) que o tipo de dado de orientação requisitado das IMUs foi escolhido como sendo do tipo *Quaternion*.

Um quaternião nada mais é do que uma expansão dos números complexos que permite representar rotações no espaço tridimensional. Descobertos pelo matemático William Hamilton, eles são definidos por três eixos complexos e um eixo real. Da mesma forma que um número complexo unitário $(cos\theta, isen\theta)$ pode ser utilizado para rotacionar em um ângulo θ vetores na segunda dimensão, quaterniões unitários do tipo $(cos\frac{\theta}{2}, v_x sen\frac{\theta}{2}, v_y sen\frac{\theta}{2}, v_z sen\frac{\theta}{2})$ podem ser utilizados para rotacionar em um ângulo θ vetores na terceira dimensão ao redor do eixo definido pelo vetor unitário (v_x, v_y, v_z).

Por motivos intrínsecos à natureza do processo de rotação dos quaterniões, é possível, através deles, evitar o fenômeno de *Gimbal Lock*. Quaterniões, no entanto, possuem uma matemática específica e, em primeira instância, imaginou-se ser necessário utilizar essa matemática para aplicar os dados sobre o modelo do braço presente na interface. As equações foram estudadas e, com o apoio de [48], construiu-se uma pequena biblioteca para utilização da matemática de quaternião. Eventualmente, entretanto, descobriu-se que os objetos da biblioteca Three.js possuíam métodos que suportavam dados de orientação representados por quaterniões. Assim, a aplicação dos dados recebidos pelas IMUs para o modelo de braço seria imediata. Após alguns testes de aplicação manual de quaterniões

para rotacionar o modelo do braço, constatou-se que de fato eles funcionavam muito bem.

A necessidade que surgiu em seguida, então, foi a de transportar os dados que estavam sendo obtidos pelo *driver* das IMUs para o programa JavaScript, de forma a testar a movimentação da interface não através da aplicação manual de quaterniões, mas através da aplicação dos próprios dados dos sensores ao modelo. Depois de pesquisar diferentes métodos (vide seção 1.2.4), percebeu-se que a melhor alternativa era a de implementar uma estrutura do tipo cliente-servidor e servir os dados adquiridos em Python para a aplicação em JavaScript.

Para subir essa estrutura, foi preciso selecionar um *framework* de rede através do qual se pudesse trabalhar. Com base em [49] e [50], selecionou-se um *micro-framework* de rede chamado "Flask", bastante conhecido por servir Python para páginas *web*. Para utilizá-lo, basta (após a instalação), importá-lo para o arquivo Python desejado e então utilizar suas funcionalidades.

Com as ferramentas oferecidas pelo Flask, e após realizar um estudo sobre estrutura de rede, foi possível começar a a configurar rotas de acesso ao servidor. Essas rotas são caminhos através dos quais o cliente pode solicitar diversos tipos de serviços distintos. Elas funcionam como pontes que podem ser utilizadas pelo programa em JavaScript para acessar as funções e dados descritos em Python.

Cada rota responde a um tipo de requisição diferente (GET, POST, PUT, DELETE...), que são de praxe da comunicação HTTP. Os únicos dois tipos de requisição utilizadas neste trabalho foram GET e POST. A requisição GET serve para solicitar dados do servidor, enquanto a requisição POST serve, tipicamente, para criar ou executar algum processo que rode no servidor. Quando cria-se uma rota no servidor, especifica-se a que tipo de requisição ela responde. Quem realiza essas requisições é o cliente, e isso é possível através da importação de uma biblioteca contendo os métodos da arquitetura Ajax.

Neste momento surgiram então dois novos arquivos no projeto: um arquivo escrito em Python cujo objetivo era rodar o servidor no *framework* Flask e outro arquivo escrito em JavaScript responsável por realizar para a interface todas as requisições a este servidor. Daqui em diante, as palavras "servidor" e "cliente" serão usadas para designar esses dois arquivos, respectivamente.

É importante frisar aqui que, como essa estrutura servidor-cliente foi erguida apenas com base na necessidade de criar uma camada de comunicação entre as linguagens Python e JavaScript, o servidor foi mantido em um endereço local, referenciado explicitamente pelo cliente. Isso significa que ainda não seria possível rodar o servidor em uma máquina e o cliente em outra. Ambos devem rodar na mesma máquina para que os sistema possa funcionar, apenas por uma questão de endereçamento. Como o sistema é

razoavelmente enxuto, isso não foi considerado um problema.

Ao rodar o comando "app.run()" no arquivo Python criado, o servidor então é erguido com todas as rotas definidas e a seguinte mensagem surge no console: "* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)". Este é o endereço IP no qual o servidor está rodando, juntamente com o número da sua porta de acesso [50]. O que aparece após a última barra ("/") no endereço é o identificador da rota. Neste caso, a rota chamada é a rota padrão (apenas "/"), e geralmente é a rota que chama a página inicial da aplicação.

A partir deste momento, todas as "pontes" de comunicação estão abertas, aguardando pedidos do cliente. Um detalhe que vale mencionar é que, para que essas "pontes" todas pudessem operar independentemente umas das outras, gerando processos que não concorrem por recursos e que não são interrompidos uns pelos outros, optou-se por utilizar uma estrutura de *threads*: caminhos paralelos e independentes de processamento para cada rota estabelecida. Isso pode ser feito a partir da configuração de uma opção do *framework* Flask.

A primeira rota implementada foi justamente a solução para o problema que deu origem à necessidade por um servidor em primeiro lugar: uma rota para servir os dados brutos colhidos das IMUs.

Através da execução contínua de uma requisição GET ao servidor, pôde-se então puxar os dados, em formato json, do *driver* em Python diretamente para o cliente em JavaScript. A partir desse ponto, foi simples extrair esses dados da variável em formato json e aplicar os quaterniões de orientação das IMUs nos objetos do modelo do braço.

Para criar uma ambientação na cena, inseriu-se uma malha gradeada (*grid*) para servir de referência e posicionou-se o modelo do braço sobre ela. De modo a obter um bom ângulo de visualização do modelo, foi preciso rotacionar a malha e o modelo de formas adversas, mas o resultado foi satisfatório, como mostrado no primeiro quadro da Figura 41.

Enfim, o primeiro teste de movimentação a partir dos dados das IMUs ocorreu, e os resultados foram desastrosos. O braço da interface se movia de forma completamente diferente do braço do usuário. Foi apenas depois de um razoável processo de *debug* em toda a geometria da cena que se descobriu o problema: ao gerar a visualização ideal do modelo havia-se alterado o posicionamento dos eixos do sistema de coordenadas cartesianas. De forma a não desarranjar a cena, a solução proposta foi simples: ajustar os eixos do sistema de orientação das próprias IMUs para ser compatível com aquele da interface (como mencionado na seção 3.1).

Uma vez ajustados os eixos, o resultado foi perfeito. A partir da atualização contínua do renderizador da cena, a cada instante os dados de orientação de cada uma das

juntas dos ossos eram mapeados com aqueles recebidos das IMUs, levando o modelo a mover-se exatamente como o braço do usuário (Figura 41).

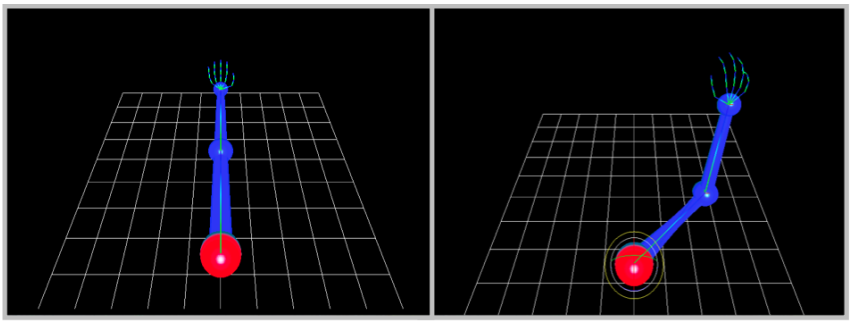

Figura 41 – Dois *frames* referentes à representação espacial do modelo do braço, o primeiro parado na posição inicial e o segundo durante um movimento de acenar.

Assim, estava finalizado o problema do rastreamento. Neste ponto, o diagrama de blocos do sistema já poderia ser representado como na Figura 42.

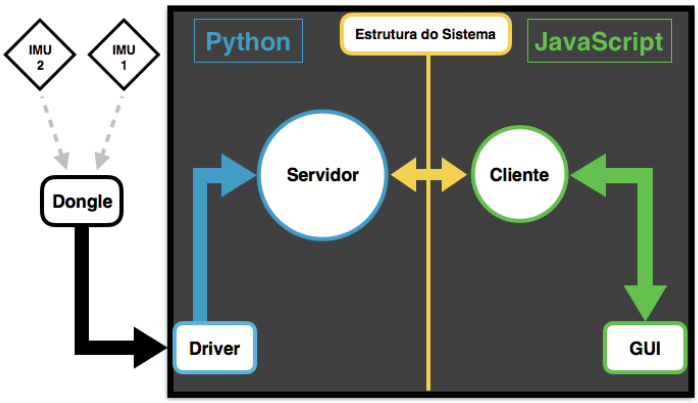

Figura 42 – Diagrama de blocos do sistema.

O próximo passo lógico era então criar um menu para iniciar a organização do processo de gravação, segmentação, parametrização e estimação dos movimentos. Após uma rápida pesquisa, percebeu-se que o modelo mais utilizado de menu para aplicações em Three.js é um menu chamado "dat.GUI", que permite criar botões de vários tipos para implementar funções no código.

Na realidade, ao importar a biblioteca do dat.GUI os primeiros botões criados foram quatro botões de controle para a interface:

- **Iniciar *Stream*:** Inicia o laço de requisições GET ao servidor que implementa o rastreamento das IMUs. Clicar neste botão significa ver o modelo do braço começar a acompanhar o braço do usuário.

- **Pausar *Stream*:** Esse botão surge apenas quando o *stream* das IMUs já foi iniciado, e serve para interromper a atualização da orientação do modelo do braço, pausando o rastreamento e congelando o modelo na tela.
- **Retomar *Stream*:** Esse botão surge apenas quando o *stream* está pausado, e serve para retomar o rastreamento.
- **Cancelar *Stream*:** Esse botão surge apenas quando o *stream* já foi iniciado, e serve para encerrar a comunicação com o servidor, saindo do laço de requisições GET que pede os dados das IMUs.

Cada um desses botões chama funções específicas do cliente que manejam o fluxo da informação entre ele e o servidor. Juntamente com esses, implementou-se também um botão chamado "Num Var", que contém uma caixa de seleção para que o usuário escolha o número de variáveis que quer monitorar durante o uso da aplicação (esse valor pode ser alterado posteriormente). As opções presentes nessa caixa de seleção seguem a previsão deste projeto e só vão até duas variáveis. Apesar disso, como dito anteriormente, a camada de processamento está preparada para suportar qualquer número de variáveis. Esse botão também não possui qualquer impacto no rastreamento.

Vale aqui comentar que, independente do número de variáveis escolhidas, o ângulo monitorado para cada junta está definido em código com sendo aquele do segmento corporal que se deseja acompanhar em relação ao eixo do segmento anterior ao qual ele está ligado (para duas variáveis, por exemplo, seria o ângulo do antebraço em relação ao eixo do braço e o ângulo do braço em relação ao eixo do tronco). Essa definição, entretanto, pode ser mudada a qualquer momento de forma bastante simples, podendo associar assim cada variável a outros graus de liberdade do membro. Apesar dessa mudança ainda ter de ocorrer em código, não seria complicado implementar mais uma caixa de seleção no menu para indicar qual dos graus de liberdade de cada junta deseja-se monitorar. Essa opção só não foi explorada por uma questão de prioridade no desenvolvimento.

Após os botões de controle das IMUs e a caixa de seleção do número de variáveis, implementou-se o primeiro botão referente ao processo de estimação: o botão "**Iniciar Gravação**". Antes desse botão ser clicado, os dados propagados das IMUs para a interface faziam o modelo do braço se mover de acordo com o braço do usuário, mas não eram armazenados nem processados de maneira alguma. A partir do momento em que esse botão é apertado, os dados começam a ser gravados durante uma janela de aproximadamente 15 segundos.

Esses dados são aqueles que serão utilizados para criar o diagrama de ciclos do movimento. Isso significa que, ao apertar o botão "**Iniciar Gravação**", um usuário saudável deve executar o movimento prescrito durante os próximos 15 segundos. É dito que o indivíduo a executar este movimento deve ser "saudável" pois é desse diagrama de ciclos

que serão extraídos os parâmetros do movimento, o que implica que o sistema aprenderá a julgar os movimentos subsequentes com base nessa execução.

Para visualizar graficamente essa gravação, decidiu-se construir um pseudo-gráfico que fosse registrando a curva das medidas em tempo real durante a janela de gravação, e que dispusesse essa curva ao operador do sistema após o fim do processo.

Inicialmente, pensou-se em utilizar bibliotecas específicas para criação de gráficos. No entanto, como era desejável que esse fosse um gráfico interativo, sobre o qual o usuário pudesse agir durante o processo de segmentação, essa opção foi então descartada. A grande maioria das bibliotecas gráficas para análise de dados ou geram gráficos a partir de um conjunto de dados pré-definido ou até desenvolvem gráficos em tempo real, mas sem grande potencial de atuação do usuário.

Assim, de forma pouca ortodoxa, apesar de criativa, decidiu-se construir (literalmente) o gráfico a partir de elementos da própria biblioteca Three.js. O pano de fundo foi então feito com um objeto de malha gradeada, exatamente como o do "piso" da interface, e os pontos foram de pequenas esferas vermelhas posicionadas longe o suficiente para dar a impressão de pontilhado.

Todavia, após as primeiras tentativas de coordenar esses elementos, percebeu-se um grande problema: como a câmera da cena estava posicionada acima do modelo, apontando para um ponto pouco abaixo do horizonte e diferente da origem do sistema de coordenadas, todos os objetos estavam em perspectiva. Por conta disso, ao posicionar o plano gradeado na vertical, ele aparecia inclinado na imagem, como se de fato estivesse sendo visto ligeiramente de cima e de lado. Os pontos, quando, posicionados na malha em coordenação com a gravação do movimento, apareciam nas posições geometricamente corretas, mas visualmente distorcidas pela perspectiva tridimensional da cena. O que estava acontecendo era que se queria criar uma estrutura bidimensional em uma cena tridimensional observada por uma perspectiva muito específica.

Atingir esse objetivo não foi uma tarefa fácil. Para isso, foi necessário construir uma função de transformação geométrica para o tamanho da grade de fundo do gráfico e para o posicionamento em tempo real dos pontos. Como, para dar a impressão de que o gráfico estava reto (na vertical), foi necessário posicioná-lo inclinado, então se os dados em um determinado instante dissessem que o ângulo medido era $\frac{\pi}{2}$, não bastava posicionar o ponto na marca de $\frac{\pi}{2}$ do gráfico. Era necessário antes passar esse valor pela função de transformação, que envolvia transladá-lo não só no eixo do tempo e no eixo dos ângulos como também na profundidade da cena, para que ele pousasse sobre o ponto correto do plano inclinado do gráfico. Apesar de ter sido uma tarefa complicada, o resultado foi bastante satisfatório, como pode-se ver na Figura 43.

Como o gráfico foi feito com base nos objetos da biblioteca Three.js, a interface

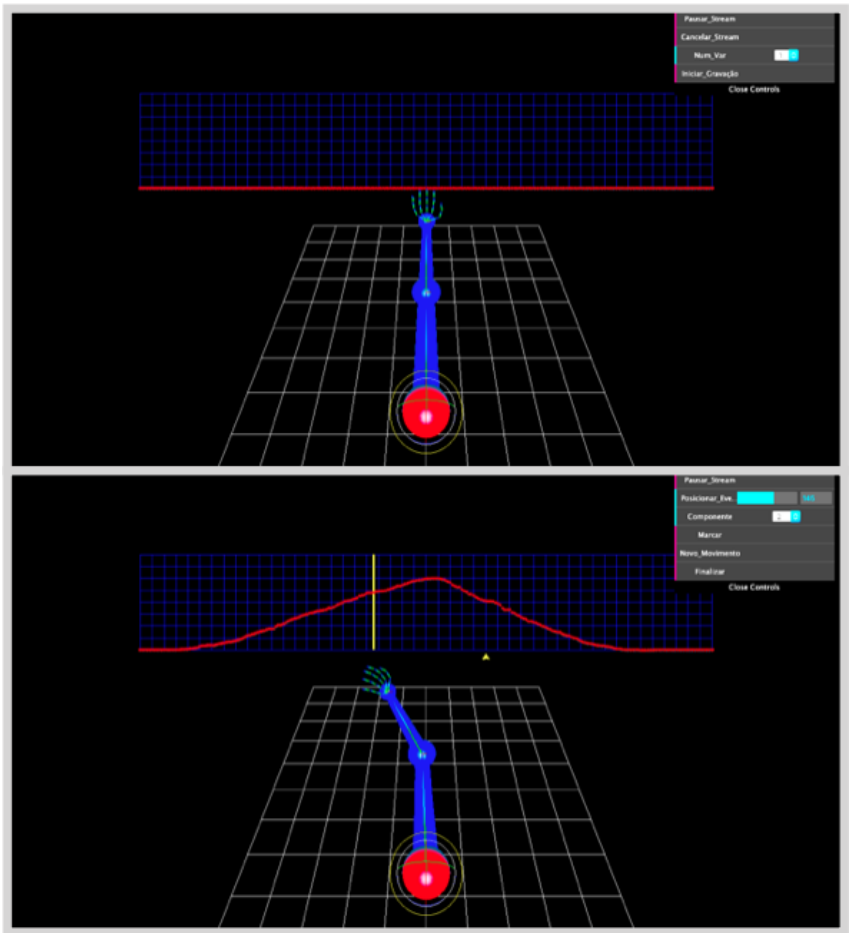

Figura 43 – Imagens do gráfico construído para a gravação de diagramas de ciclo. Acima, logo antes da gravação; abaixo, após o fim da gravação de um movimento de uma variável apenas, no início da etapa de segmentação.

possui total controle sobre ele inclusive durante a gravação do movimento. Por conta disso, como uma característica a mais para dinamizar a cena, pôde-se fazer os pontos se ajustarem em tempo real enquanto o movimento ainda estava sendo gravado, formando o gráfico iterativamente, o que é visualmente bastante interessante.

Assim que a gravação é finalizada, entra-se na etapa de Segmentação do movimento gravado. Baseado na abordagem de [15], propôs-se, para este trabalho, que o operador do aplicativo fizesse manualmente a segmentação dos dados, o que simplifica consideravelmente o modelo matemático proposto mas, por outro lado, pede uma complexidade a mais da interface.

Para que o usuário pudesse segmentar os diagramas de ciclos, no entanto, era necessário que ele pudesse atuar sobre a interface e dizer em que pontos os gráficos deveriam ser seccionados. Além disso, intrínseca a etapa de segmentação está uma etapa de rotulação, na qual deve-se atribuir, a cada segmento criado uma variável de chaveamento para simbolizar o estado descrito.

Após considerar qual seria a melhor maneira de oferecer este controle ao usuário, decidiu-se que essas operações seriam feitas a partir de botões interativos no Menu. Assim, duas coisas principais acontecem ao final dos 15 segundos de gravação: surge um marcador amarelo no canto esquerdo inferior do diagrama e o menu ganha uma série de novos botões:

- **Posicionar Evento:** Esse é um botão deslizante que serve para mover o marcador amarelo na base do gráfico pelo eixo do tempo (abscissas), de forma a escolher um ponto de marcação;
- **Marcar:** Esse botão, quando clicado, insere uma barra amarela sobre o gráfico no ponto onde o marcador está posicionado, delimitando um segmento. A barra amarela simboliza um evento de transição de estados e, portanto, os intervalos entre as marcações são os próprios segmentos do movimento.
- **Componente:** O nome "Componente" segue a definição adotada na seção 1.1, e simboliza a variável de chaveamento do sistema. Este é o símbolo que se vai atribuir ao segmento criado. Aqui, uma caixa de seleção com valores de 1 a 10 é apresentada, para que o usuário escolha qual é o estado do segmento em questão.
- **Novo Movimento:** Esse botão serve apenas para o caso do usuário querer dividir a gravação obtida em mais de um movimento distinto.
- **Finalizar:** Quando já criaram-se todos os segmentos com seus respectivos rótulos, clica-se neste botão para encerrar o processo de segmentação.

Um detalhe importante é que, para o caso em que duas variáveis estão sendo monitoradas, dois conjuntos de pontos de cores diferentes surgem no gráfico, agindo independentemente um do outro mas sendo posicionados no mesmo espaço.

No caso multivariável, há ainda mais um botão que surge no Menu: uma botão de nome "**Variável**". Lembrando daquilo que foi discutido nas seçoes 1.1 e 1.3.1.3.1, quando grava-se o movimento de mais de uma variável ao mesmo tempo, diga-se N variáveis, devem ser gerados não N, mas N+1 diagramas de ciclos: um para cada variável separadamente (diagramas parciais) e mais um com as curvas sobrepostas (diagrama principal). Este botão permite selecionar qual dos diagramas deseja-se visualizar na tela para segmentação, e permite alternar livremente entre eles inclusive durante as marcações. Essa é uma funcionalidade que acrescenta bastante praticidade ao processo de segmentação multivariável, mas que exigiu um tratamento vetorial mais complexo dos dados de cada diagrama bem como das marcações, componentes e segmentos designados a cada um deles.

Um processo de Segmentação multivariável (neste caso de duas variáveis) pode ser visto na Figura 44, onde tanto os diagramas parciais como o diagrama principal podem ser vistos na tela ao alternar a caixa de seleção da variável.

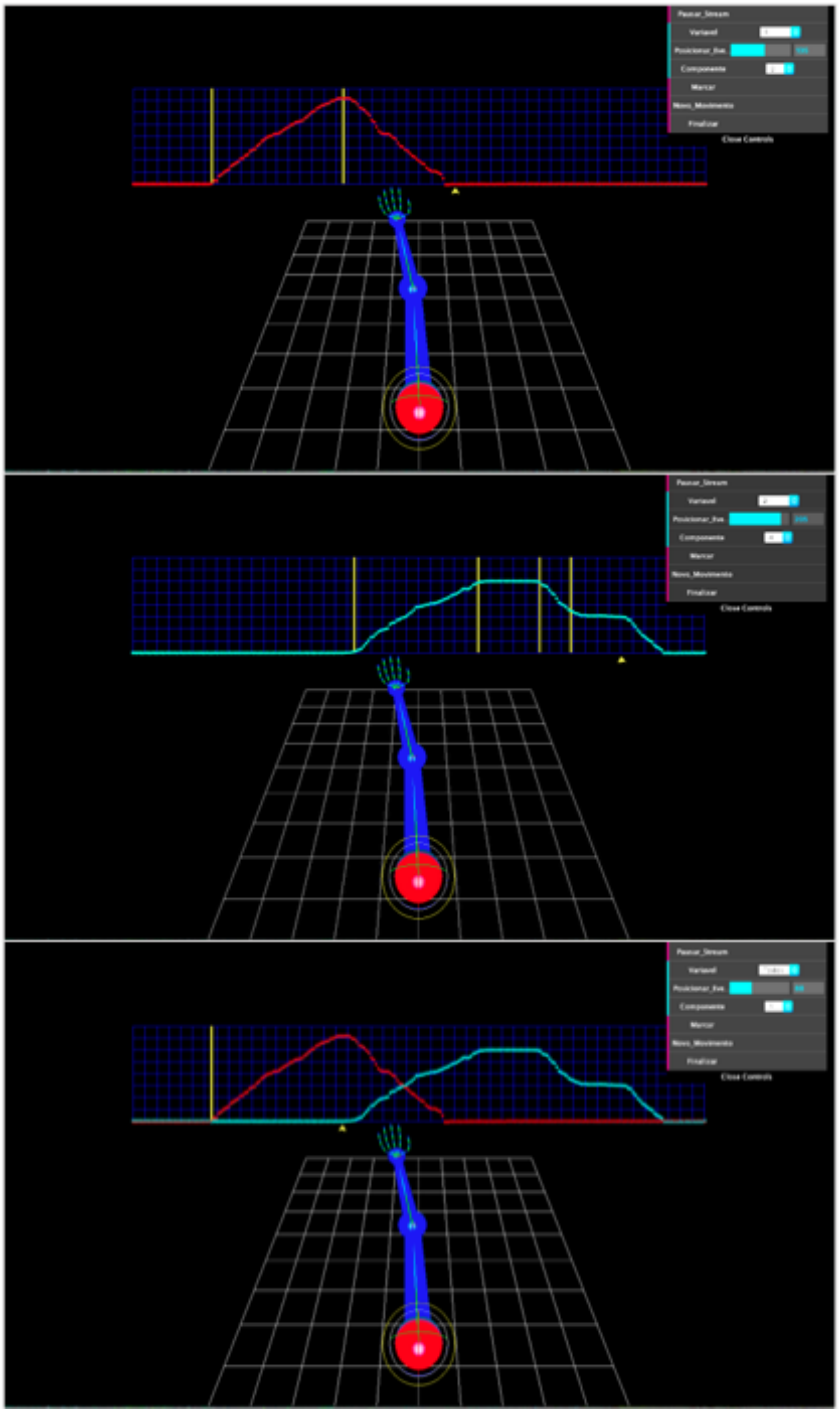

Figura 44 – Imagem obtida durante o processo de segmentação de um caso Multivariável. No topo e no meio, os diagramas parciais do movimento e, por último, o diagrama principal.

Apenas quando todos os diagramas tiverem sido segmentados é que o usuário deve clicar em "Finalizar".

Independente do número de variáveis, o que a interface faz enquanto o usuário segmenta os diagramas é já preparar um conjunto de vetores para a camada de processamento. Esses vetores contêm informações acerca das marcações e dizem, para cada

diagrama de ciclos, qual foi a variável de chaveamento (estado) associada a cada amostra. Assim que o usuário clica o botão de Finalizar, esses vetores são enviados à camada de processamento para Parametrização. Os detalhes do que ocorre aqui serão discutidos na seção 3.2.2.

O último elemento pertinente ao desenvolvimento da interface gráfica que ainda não foi citado foi o *feedback* ao usuário. Foram implementadas duas formas de resposta: uma em tempo real e outra ao final da estimação.

A resposta em tempo real é dada por um elemento de texto em três dimensões que surge na tela durante a fase de Estimação, e que mostra um grande número vermelho de 1 a 10. Baseado nas respostas do SLDS, esse número identifica, ao mesmo tempo que o usuário move seu braço, o estado em que o movimento está. Apesar de ser um *feedback* bastante técnico, ele funciona como a prova de conceito da validade do algoritmo. Através dessa resposta, existem diversos desdobramentos de análise que podem ser feitos, mas que por uma questão de tempo e de objetivos, não foram implementados para este trabalho. Mais sobre isso será discutido no capítulo de Conclusão.

O *feedback* gráfico entregue pela interface ao usuário pode ser visto na Figura 45.

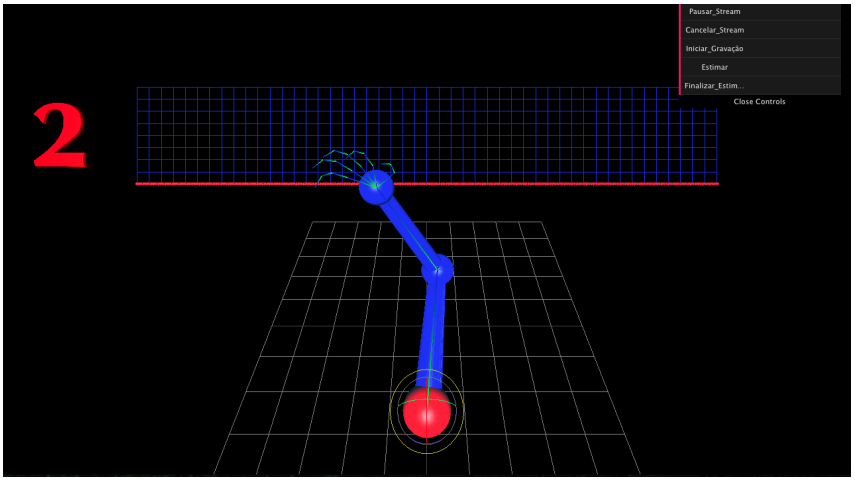

Figura 45 – *Feedback* em tempo real acerca do estado do movimento.

Como o GUI cresceu rapidamente, ultrapassando em muito o que havia sido previsto inicialmente, decidiu-se dividi-lo em três módulos (arquivos): o módulo "Menu", responsável por implementar todas as funções associadas aos botões do menu; o módulo "Visual", responsável pela construção de todos os objetos contidos na cena; e o módulo "Renderer", responsável pela renderização em tempo real da interface.

Apesar desse ser o relato principal do desenvolvimento da interface gráfica, existem diversos detalhes a mais, especialmente no que diz respeito às nuances do menu, que não foram aqui mencionados. Esses detalhes serão tratados mais à frente na seção 3.3 ou em integração com tópicos específicos da camada de processamento, discutidos na seção 3.2.2.

3.2.2 Camada de Processamento

O desenvolvimento da camada de processamento do sistema foi, sem sombra de dúvidas, a parte mais desafiadora deste trabalho. No início, o conhecimento da linguagem Python era inteiramente baseado no conhecimento prévio que se tinha da linguagem C. Isso, somado ao fato de que não se sabia absolutamente nada sobre Filtros de Kalman, cadeias de Markov ou conceitos mais profundos de processamento de sinais, fez deste deste trabalho uma jornada trabalhosa e de muito aprendizado.

O primeiro passo para a implementação da camada de processamento foi um estudo profundo de [15]. Aqui vale ressaltar que essa foi a tese de base para todo o desenvolvimento dessa camada, da qual toda fundamentação matemática foi derivada. A real compreensão do algoritmo aconteceu gradualmente, e só veio a ser completada ao final do projeto. Isso fez com que o estudo de [15] fosse um fator constante durante toda a fase de implementação deste módulo.

A tarefa inicial da camada de processamento é a da Parametrização. Essa etapa ocorre ao final da Segmentação e, através os dados obtidos dos diagramas de ciclos, é responsável por criar os parâmetros para o modelo de Markov (a matriz de transição de estados Π) e para os modelos lineares referentes a cada estado do sistema (pares {velocidade, variância}). Novamente, é importante lembrar que a matriz Π é calculada apenas para o diagrama de ciclos principal, enquanto os pares {velocidade, variância} são calculados para cada estado dos diagramas parciais.

Como esses parâmetros estão intimamente ligados aos diagramas de ciclos gerados pela interface e ao *input* feito pelo usuário através do menu, e uma vez que calculá-los é um processo computacionalmente simples que só ocorre uma vez a cada processo de estimação, foi decidido fazer uma alteração na estrutura do sistema: ao invés de implementar toda a camada de processamento em Python, essa etapa de Parametrização seria codificada em JavaScript.

Codificar a Parametrização na mesma linguagem da Segmentação foi bastante útil, e permitiu acessar diretamente as variáveis geradas pelo GUI, sem precisar enviá-las através do servidor para um arquivo Python que realizasse esse processamento. Assim, foi criado um novo módulo em JavaScript chamado "Parameters", que ficou responsável por realizar, em comunicação direta com o GUI, a etapa de Parametrização.

Para esta etapa, três funções foram então implementadas: uma função para realizar o cálculo da matriz Π, uma função para o cálculo dos pares {velocidade, variância} de cada estado e uma função auxiliar que gera, a partir do conjunto de todos os diagramas de ciclos, um vetor contendo quais estados s dos diagramas parciais estão relacionadas a quais estados σ do diagrama principal. Essa função aproxima a dinâmica da função $\phi(s)$ descrita em 1.3.1.3.2.

Tendo em vista os métodos propostos em 1.3.1.3.2, imaginou-se em um primeiro momento que o cálculo desses parâmetros seria bastante simples e direto. Entretanto, como cada um deles envolve somatórios de elementos específicos da etapa de segmentação, foi necessário um tratamento cuidadoso dos dados para que os resultados fossem corretos. Todas as funções foram implementadas à mão e não houve uso de qualquer biblioteca neste módulo.

O desenvolvimento matemático da Parametrização levou em conta a construção de dois vetores auxiliares: um vetor que diz, para cada movimento, todas as variáveis de chaveamento utilizadas para rotular seus segmentos, em ordem de aparição; e um vetor que guarda o número de ocorrências de cada um desses estados no espaço amostral, isto é, que diz quantas amostras no movimento estão associadas a uma determinada variável de chaveamento.

Uma vez construídos esses vetores, e através dos vetores gerados pela segmentação (vetores de dados dos ângulos gravados e seus respectivos vetores de chaveamento), foi possível então calcular os parâmetros do algoritmo.

A matriz Π foi calculada da seguinte forma: para cada amostra do diagrama associada a um estado j, se a amostra anterior está associada a um estado diferente i, soma-se $\frac{1}{occ(i)}$ ao índice {i,j} da matriz, em que occ(i) é o número de ocorrências do estado i no espaço amostral; caso a amostra anterior esteja associada ao mesmo estado, soma-se então ao índice {i,i} da matriz. Ao final, o resultado de cada índice da matriz é a soma do número de ocorrências da transição designada por aquele índice, dividida pela número total de ocorrências do estado de origem, que é exatamente a definição da matriz Π proposta em 1.3.1.3.2.

Para as velocidades, foi utilizada uma abordagem um tanto alternativa: para cada segmento dos diagramas parciais, somou-se as diferenças entre as amostras consecutivas e dividiu-se a soma total pelo número de amostras naquele segmento. Isso resultou em um parâmetro de velocidade normalizado com base na frequência de amostragem, ou seja, um valor de variação angular por amostra, o que dissociou o sistema do tempo e associou-o à taxa de amostragem. Como essa taxa é conhecida (para o processo de gravação pertence à faixa de 20-30 Hz), então a qualquer momento é possível converter os valores de volta para o tempo.

De fato, um detalhe interessante aqui é que como o renderizador da cena roda a 60 fps, existe um grande desencontro das frequências do sistema, causando uma subutilização dos recursos disponíveis. Enquanto o servidor é capaz de servir os dados das IMUs a uma taxa de aproximadamente 1 kHz (menos um *overhead* de comunicação), a interface só é capaz de processá-los a uma taxa de 60 Hz. No contexto da aplicação deste trabalho, e considerando os testes realizados (vide capítulo 4), essa redução no processamento não prejudicou a performance do sistema de forma grave.

Já durante o cálculo dos parâmetros de velocidade, uma média das amostras é feita recursivamente e então, sob uma nova passagem pelas medidas adquiridas, as variâncias são calculadas para cada segmento, seguindo a definição padrão em 1.3.1.3.2.

Ao final do processo de Parametrização, a resposta é então uma matriz $(N \times M_n)$ de velocidades, em que N representa o número de variáveis monitoradas e M o número de estados nos diagramas parciais de cada uma; uma matriz $(N \times M_n)$ de variâncias seguindo o mesmo princípio; uma matriz Π $(M^* \times M^*)$ de transição de estados, em que M^* representa o número de estados do diagrama principal; e um vetor que diz quais são os estados $\sigma \in \{1, 2, ..., M^*\}$ do diagrama principal, e quais são os estados $s_n \in \{1, 2, ..., M_n\}$ de cada diagrama parcial que a ele estão associados. Esses resultados são então devolvidos para a Interface gráfica e ela, em retorno, os repassa através do cliente ao servidor da aplicação.

Assim, pode-se perceber que a decisão de escrever a Parametrização em JavaScript também trouxe uma maior organização ao fluxo de operação do algoritmo: os passos preliminares (pertinentes ao fisioterapeuta/ operador do sistema/ usuário saudável) de gravação, segmentação manual, rotulação e parametrização são todos processados pelo lado JavaScript da aplicação; os passos referentes ao processo de estimação em tempo real, como será mostrado daqui em diante, são todos função da camada de processamento em *Back-end*, restando à interface apenas mostrar os resultados ao usuário e continuar rastreando as IMUs.

Faz-se necessário notar que, ao longo do desenvolvimento do projeto, as noções de *Back-end* e *Front-end*, Python e JavaScript, unidade de processamento e Interface, foram aos poucos se tornando mais fluidas. Ao fim do deste capítulo, será possível ver as definições finais dessas fronteiras.

Enfim, havia chegado o momento de implementar o coração do processamento do sistema: o algoritmo SLDS-Viterbi aproximado para estimação dos movimentos. Seguindo a proposta deste trabalho, um novo módulo foi criado do lado Python da aplicação para realizar essa tarefa.

Para melhor estruturar o processamento de dados, decidiu-se utilizar conceitos de orientação a objeto para a construção desse módulo. Apesar de já possuir certa experiência com orientação a objeto, foi necessário estudar as peculiaridades da linguagem Python para a aplicação desses conceitos.

Enfim, o módulo foi dividido em três seções: uma seção de funções auxiliares e inicialização, uma seção de modelos e uma seção contendo as funções próprias do SLDS. Como para rodar o algoritmo SLDS era necessário antes construir os modelos lineares associados a cada um dos estados do diagrama de ciclos, a primeira seção desenvolvida foi a de modelos.

Nessa seção, duas classes foram criadas: uma classe "Kalman" e uma classe "Markov". A seguir, detalha-se o significado dos objetos criados por essas duas classes.

Quando criado, um objeto da classe Kalman recebe em seu construtor um estado σ (s no caso univariável) e o vetor de velocidades referente a este estado (que seria um vetor unitário no caso univariável). Com base nessas duas entradas, constroem-se as matrizes de transição "A" e de observação "C" que definem o modelo linear desse estado. Assim, um objeto da classe Kalman é o próprio modelo linear de um estado do sistema, indexado pela sua variável de chaveamento. Esses objetos são a célula básica do SLDS e, ao inseri-los em uma lista, é possível chaveá-los através de seus índices durante o processo de estimação.

Os métodos da classe Kalman (na realidade métodos do objeto) são dois, como definido na seção 1.3.1.2.1: Predição e Atualização, implementados matricialmente como pela definição da mesma seção. Ambos retornam um vetor com a posição e a variância preditas ou atualizadas, respectivamente.

A noção de orientação a objeto é extremamente útil durante a etapa de estimação, pois permite chamar esses métodos para cada modelo criado a partir da indexação desses modelos.

A classe Markov possui como único parâmetro a matriz Π de transição de estados e, como único método, a etapa de previsão do HMM que diz, para um vetor de entrada {i,j}, qual é a probabilidade da transição $i \rightarrow j$. Uma vez que não mais do que um objeto da classe Markov é criado, sua definição no modelo orientado a objeto é mais uma questão de organização e formalismo do que de necessidade.

Uma vez construídas essas classes, foi necessário então criar uma função de inicialização para criar os objetos de cada uma delas com base na resposta obtida da Parametrização. A essa função foi atribuída a tarefa de, além de criar um objeto da classe Kalman para cada estado do diagrama principal e um objeto da classe Markov para o diagrama como um todo, inicializar todos as matrizes do processo de estimação. São elas:

- **"Xpath", "COVpath", "Spath"e "COSTpath":** Matrizes (M estados x T instantes) responsáveis por por guardar os dados das estimações de cada um dos N caminhos que são construídos iterativamente no decorrer do algoritmo (vide seção 1.3.1.3.3). Respectivamente, essas matrizes guardam as árvores de estimação das medidas de posição, das variâncias, das sequências de estados e das sequências de custos dos estados estimados para cada movimento.

- **"ANGLEpath":** Matriz (1 x T instantes) responsável por por guardar os dados reais das medições realizadas durante o processo de estimação para cada instante de tempo.

- **"Xestimates", "COVestimates", "PROBestimates"e "COST":** Matrizes qua-

dradas (M caminhos x M estados) responsáveis por por guardar o resultado de cada uma das M^2 estimações realizadas a cada iteração *Forward* do Filtro de Kalman. Respectivamente, essas matrizes guardam as estimativas das medidas de posição, das variâncias, as probabilidades de cada transição e os custos associados a elas. A estrutura dessas matrizes permite fácil indexação e comparação desses dados.

- **"Xtrue", "Strue":** Matrizes (1 x T instantes) responsáveis por por guardar, a cada passo de *Backward* efetuado, o resultado final da estimação da posição e da sequência de estados, respectivamente.

A partir dessas matrizes, a seção de implementação das funções do SLDS pôde enfim ser construída. As três funções de base para essa parte codigo foram as funções:

- **"forwardP":** Realiza uma etapa de Predição do SLDS, na qual os métodos de "predict" dos objetos da classe Kalman e da classe Markov são invocados. Preenche as matrizes Xestimates, COVetimates e PROBestimates. Nas primeiras duas iterações do problema, também inicializa a árvore de estimação.
- **"forwardU":** Acrescenta um elemento à matriz ANGLEpath e Resume a etapa de Atualização do Filtro de Kalman, invocando o método "update" dos objetos da classe Kalman. Atualiza as matrizes Xestimates e COVestimates e preenche a matriz COST através do cálculo do custo de cada uma das M^2 transições criadas na função "forwardP". Através do julgamento dos elementos dessa matriz COST, acrescenta um elemento às matrizes Xpath, COVpath, Spath e COSTpath. Por fim, a cada l (variável de janelamento) estimações, chama a função "backward".
- **"backward":** Realiza uma etapa Filtragem, acrescentando l elementos às matrizes Xtrue e Strue e podando o sistema para obter o melhor caminho que pode ser obtido pelo método escolhido.

Ao testar essas funções ao longo da implementação do módulo SLDS, três dificuldades principais foram encontradas.

A primeira delas foi que a função custo (equação 1.54) possuía um problema computacional intrínseco: ao calcular o logaritmo de números muito próximos de zero (como de fato a variância e as probabilidades de transição de estados muitas vezes eram), a função retornava erro de domínio. De fato, o logaritmo de um número que tende a zero, tende a menos infinito, que é uma definição adversa para o sistema. Entretanto, era necessário calcular esse valor, ou não haveria custos para comparar e o algoritmo não iria adiante.

Para solucionar essa questão, utilizou-se um truque matemático: ao invés de escrever $(\log a)$, escreveu-se $((log1000 * a) - 3)$, que é equivalente e não resulta, no cálculo explícito da função log, em um erro computacional.

Uma vez resolvida esta questão, o algoritmo rodou por completo pela primeira vez, dando margem para o segundo problema: por algum motivo o resultado da estimação estava extremamente instável, flutuando entre diversos valores. Era possível ver que o algoritmo respondia ao movimento, mas a saída era muito imprecisa, submetida a grandes oscilações.

Foi apenas depois de um longo processo de *debug* que finalmente entendeu-se a origem do problema: a taxa de atualização do Filtro de Kalman estava muito alta.

É importante definir aqui ao menos parte do fluxo de informações. Apesar do *driver* receber os dados das IMUs já no *Back-end*, no lado Python da aplicação, o módulo SLDS roda com as informações dos ângulos da interface gráfica. Isso quer dizer que primeiro os quaterniões obtidos das IMUs são enviados à interface gráfica e apenas quando eles já foram mapeados pelo modelo do braço é que extrai-se dele os ângulos na base Euleriana que são usados como "posição" pelo algoritmo de processamento.

Assim, como a etapa de Atualização só pode ser realizada quando uma amostra é recebida de volta da Interface, já no formato angular, uma pequena função de controle de fluxo havia sido implementada no intuito de sempre aguardar por este dado antes de prosseguir com o algoritmo. O problema, no entanto, estava sendo que essas função estava perdendo o controle dada a taxa com que o renderizador devolvia o dado angular e requisitava o avanço da estimação.

Para resolver este problema, o que se fez foi simplesmente diminuir a taxa de Atualização do Filtro de Kalman, e o sistema passou a responder bem aos movimentos, sem oscilar e entregando resultados coerentes.

No entanto, percebeu-se que o resultado ainda não era perfeito. Apenas algum tempo depois entendeu-se o motivo: a taxa com que o movimento fora amostrado no processo de gravação da interface gráfica, e que por consequência gerara os parâmetros de velocidade dos modelos lineares, era diferente da taxa com que os movimentos subsequentes estavam sendo estimados. Isso significa que apesar do movimento estar perfeitamente coerente com aquele que havia sido parametrizado, o modelo dizia que as amostras deveriam variar uma taxa b_0 e o algoritmo de estimação percebia a varição uma taxa $b_1 < b_0$. Como todas as velocidades do sistema estão normalizadas com a frequência, os modelos não batiam.

Considerando esse desencontro de taxas, foi necessário incluir um fator de ajuste ao parâmetro de velocidade dos modelos lineares que compensasse pela diferença. Como a razão entre as taxas era de 1:2, o fator de ajuste foi escolhido como sendo 0,5. Uma vez incluído este fator, os resultados do sistema finalmente, pela primeira vez, foram tão excelentes quanto podiam ser.

Enfim estabelecida a validação do sistema após uma bateria de testes simples,

partiu-se para enfrentar o que viria a ser talvez o maior desafio do desenvolvimento da camada de processamento.

Como a primeira implementação do código só tratou o caso univariável, os elementos de cada uma das matrizes do módulo SLDS (e do módulo de Parametrização em JavaScript) eram de fato apenas elementos. Contudo, ao adaptar este módulo para o caso multivariável, aquela que talvez tenha sido uma das maiores complexidades de todo o projeto surgiu: cada elemento do sistema se tornou uma matriz, e as matrizes que costumavam ter duas dimensões passaram a ter três ou até mesmo quatro dimensões (linearizadas para três).

A partir desse momento, o sistema ganhou uma complexidade que não se previra em primeira instância, e foi necessário começar a pensar na dinâmica do próprio algoritmo tridimensionalmente. Uma das bibliotecas mais importantes utilizadas nesse contexto foi a biblioteca "numpy", mencionada na seção 1.2.2. Através dela, o tratamento de vetores e matrizes foi facilitado imensamente, o que foi importante no momento de transpor esse problema para o caso multivariável.

O processo mais árduo foi o de entender como as dimensões de cada matriz do problema cresciam para mais de uma variável. Como o modelo linear no espaço de estados e o algoritmo de previsão e atualização do Filtro de Kalman dependem rigidamente da estrutura das matrizes, e como cada uma traz dimensões específicas do problema, essas matrizes precisaram ser adaptadas individualmente de acordo com a sua natureza para entregar o mesmo resultado do caso univariável, apenas em uma formatação vetorizada.

Após um bom número de horas realizando e testando essas adaptações, estudando a documentação da biblioteca "numpy" e implementando expansões para cada método desenvolvido, finalmente o sistema ficou pronto, tornando-se capaz de suportar tanto o caso univariável quanto o caso multivariavel.

Quando o desenvolvimento do sistema geral foi finalizado, decidiu-se ainda implementar três funcionalidades. A primeira foi a de resetar o algoritmo ao final de todo processo de estimação e, assim, permitir que o usuário repetisse o procedimento quantas vezes lhe fosse necessário. A implementação dessa função foi direta e imediata, apesar de ter exigido uma avaliação detalhada de todas as variáveis que precisavam voltar ao seu estado original antes de iniciar um novo ciclo de gravação.

A segunda funcionalidade que decidiu-se implementar foi a inclusão de um segundo tipo de *feedback*, oferecido ao usuário após finalizado o processo de estimação. Esse *feedback* vem na forma de um arquivo no formato "txt" que armazena as matrizes ANGLEpath e Xtrue para cada uma das variáveis analisadas e a matriz Strue para o movimento estimado. Através deste arquivo, é possível utilizar um script em Python, também fornecido por este projeto, para gerar os gráficos do movimento, contrastando as medidas das IMUs, o

melhor caminho estimado e a melhor sequência de estados obtida pelo algoritmo. Através da análise desses gráficos, é possível extrair diversas informações, como tempo de duração das fases do movimento, amplitude máxima e mínima, e uma série de outros dados que podem ser pertinentes à avaliação de um profissional. Os gráficos são obtidos através da utilização de uma biblioteca chamada "matplotlib", também mencionada em 1.2.2. A cada estimação feita (entre consecutivos *resets* do sistema), um novo arquivo .txt é gerado.

De forma a potencializar a análise posterior dos movimentos executados durante a etapa de estimação, decidiu-se implementar ainda uma funcionalidade secundária na aplicação: ao clicar no botão "Finalizar Estimação", responsável por encerrar a atividade do algoritmo SLDS-Viterbi, o botão de "Estimar" surge novamente no menu. Isso permite que o usuário realize diversas estimações com base na mesma gravação, gerando assim múltiplos arquivos .txt de saída. Ao dar-se por satisfeito, o usuário pode então clicar o botão "Reiniciar", que enfim reinicializa o programa e permite realizar uma nova gravação de movimento.

A terceira e última funcionalidade adicional foi a chamada única do sistema. Antes, era necessário rodar o arquivo "server" que continha o servidor em Python e, separadamente, rodar o arquivo JavaScript com o GUI para levantar a aplicação. Através de uma biblioteca chamada "webbrowser", no entanto, pôde-se implementar uma chamada para a página web de dentro do próprio servidor. Assim, basta então rodar um arquivo e todo o sistema esta pronto para funcionar, o que o aproxima de uma aplicação convencional.

Feito isso, estava então completo o desenvolvimento de todas as funcionalidades propostas pelas especificações deste projeto. Neste ponto, o sistema pode ser descrito como na Figura 46. A título de esclarecimento, tudo à exceção do GUI pertence ao chamado *Back-end* do *software*. O GUI é o módulo atribuído ao *Front-end*.

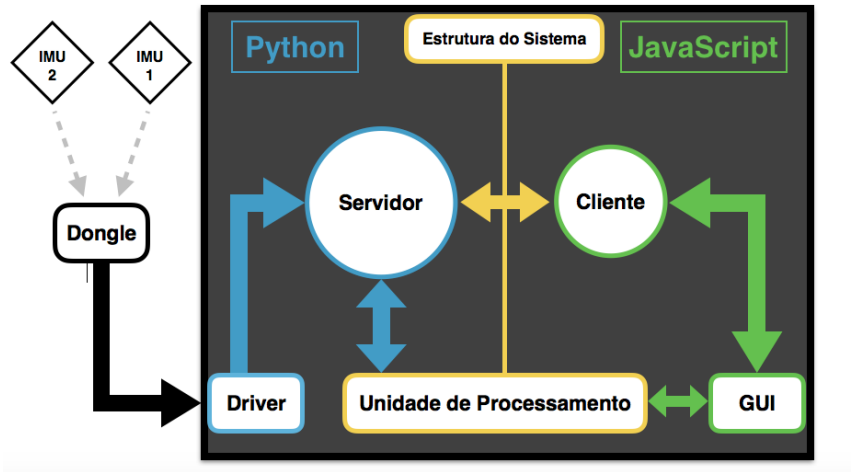

Figura 46 – Diagrama de blocos final do sistema implementado.

3.3 Fluxo de Operação

Após o relato individual do desenvolvimento de cada módulo do aplicação, essa seção apresenta um resumo passo a passo da operação do sistema, enfim explicando, cronologicamente, como as trocas de informações são feitas e como a estimação é posta em prática.

Sequência de passos:

1. Fixam-se as duas IMUs no braço do usuário (palma virada para cima, uma IMU no braço, logo acima do cotovelo, e a outra no pulso, logo antes da mão), e liga-se o *dongle* no computador;
2. Roda-se o arquivo "server";
3. A interface surge na tela do navegador, em seu estado inicial: modelo parado na posição inicial e menu apenas com a opção de "Iniciar Stream";
4. Clica-se no botão de "Iniciar Stream" do menu. O modelo ganha vida e passa a rastrear as IMUs no braço do usuário. Opções de "Pausar Stream", "Cancelar Stream", "Iniciar gravação" e "Número de variáveis" surgem na tela. O usuário pode pausar/retomar ou cancelar/iniciar o *stream* de dados das IMUs quantas vezes quiser antes de clicar o botão de "Iniciar gravação", mas deve selecionar o número de variáveis desejadas antes disso;
5. Clica-se no botão de "Iniciar Gravação". O número N de variáveis selecionado é usado para inicializar o sistema de processamento. Caso $N > 1$, um novo conjunto de pontos de cor diferente aparece no gráfico. Os conjuntos de pontos começam a se mover de acordo com a(s) variável(is) sendo monitorada(s);
6. Ao fim de um período de aproximadamente 15 segundos, a gravação acaba e as curvas obtidas ficam congeladas na tela. Um marcador amarelo surge no canto esquerdo inferior do gráfico e, no menu, surgem as opções "Variável", "Posicionar Evento", "Componente", "Marcar", "Novo Movimento" e "Finalizar";
7. Escolhe-se o rótulo (variável de chaveamento) do primeiro estado do movimento através da caixa de seleção "Componente". Através do botão deslizante "Posicionar Evento", arrasta-se o marcador amarelo até a primeira transição de estados vista na curva e clica-se no botão "Marcar". Uma divisão amarela surge nesse ponto do gráfico, demarcando o final do primeiro estado. As amostras dessa marca para trás serão rotuladas com o Componente escolhido.
8. A cada marcação, as amostras entre a divisória inserida e a anterior recebem o rótulo selecionado em "Componente". Segue-se segmentando e rotulando o diagrama até que uma divisória tenha sido colocada no último ponto do gráfico.

9. Caso mais de uma variável esteja sendo analisada, repete-se o processo dos dois itens acima para todas as outras variáveis. Repete-se ainda o mesmo processo para o diagrama principal do movimento, que estará indicado com o título "Todos" na caixa de seleção de variáveis.

10. Uma vez terminada a segmentação e rotulação de todos os diagramas, clica-se em "Finalizar". Nesse momento o módulo "Parameters" é chamado e ocorre a etapa de Parametrização, através da qual são calculados os vetores de velocidade, variância, a matriz de transição de estados e o vetor auxiliar de combinação de estados do sistema. Isso ocorre em uma fração de segundos, sem que o usuário perceba, e esses dados são enviados ao servidor. O módulo SLDS é chamado e os modelos de Kalman e Markov são inicializados. Os pontos no gráfico voltam à posição original (todos em zero), e as opções de marcação do Menu são substituídas pelo botão "Estimar".

11. Ao clicar o botão "Estimar", ele some do menu e é substituído pelo botão "Finalizar Estimação". A função "forwardP" do módulo SLDS é chamada pela primeira vez e um laço de estimação é estabelecido em que:

    a) O módulo SLDS roda uma iteração da Predição;

    b) O servidor continua enviando os quaterniões de orientação obtidos das IMUs para a Interface Gráfica e ela os aplica ao modelo do braço para manter ativo o rastreamento;

    c) O cliente envia uma amostra do(s) ângulo(s) (Euleriano(s)) monitorado(s) para o servidor.

    d) O módulo SLDS roda uma iteração da Atualização;

    e) O módulo SLDS roda uma iteração da Filtragem;

    f) O cliente requisita o resultado da estimação (estado atual estimado pelo sistema) e mostra-o na tela da interface;

12. O laço prossegue, mostrando as fases do movimento na tela até que o usuário clique no botão "Finalizar Estimação". Neste momento o menu volta a apresentar a opção de "Estimar" e ganha também a opção "Reiniciar". O arquivo de saída .txt é gerado na pasta "Output" do diretório do projeto, podendo ser analisado posteriormente pelo profissional ou pelo próprio paciente;

13. Caso o usuário decida rodar o algoritmo de estimação novamente, clicando em "Estimar", o sistema volta ao passo 11. Caso o botão "Reiniciar"seja clicado, o Menu volta então à configuração padrão inicial, com as opções de cancelar ou pausar o stream, com a caixa de seleção para o número de variáveis e com o botão "Iniciar Gravação", retornando assim para o passo 5.

14. A plataforma continua funcionando até que se derrube o servidor ou até que se feche o navegador.

4 Testes e Resultados

Para fins deste trabalho, objetivou-se realizar uma bateria de testes para verificar o desempenho e a consistência do modelo SLDS implementado. O desempenho da interface gráfica ou das IMUs não foi alvo dessa bateria de testes, mas será discutido brevemente na seção 4.3.

De qualquer maneira, o principal foco deste capítulo é avaliar se o modelo SLDS proposto (da forma como foi desenvolvido) é capaz de atender às especificações em cenários diversos. Para isso, foram propostos quatro tipos de testes:

- **Teste de invariância à velocidade:** Avaliar se o algoritmo é capaz de estimar a posição e o estado corretos para a amostragem de um movimento realizado com diferentes velocidades;
- **Teste de invariância à amplitude:** Avaliar se o algoritmo é capaz de estimar a posição e o estado corretos para a amostragem de um movimento realizado com diferentes amplitudes;
- **Teste de robustez:** Avaliar se o algoritmo é capaz de estimar a posição e o estado corretos para a amostragem de um movimento realizado com diferentes níveis de instabilidade (tremor);
- **Teste de invariância ao usuário:** Avaliar se o algoritmo é capaz de estimar a posição e o estado corretos para a amostragem de um movimento realizado por diferentes usuários;

Foi desejável, antes de mais nada, estruturar os testes da seguinte maneira: primeiramente, eles seriam realizados em um caso base univariável com poucos estados a título de validação. Então, para avaliar a escalabilidade dos resultados obtidos, os mesmos testes seriam realizados nos dois sentidos em que se pode acrescentar complexidade ao sistema: em um caso com mais estados (univariável mais complexo) e em um caso com mais variáveis (multivariável). Assim, validando o caso mais simples e mostrando como os resultados desse caso variam ao escalar o sistema, cria-se um panorama geral do desempenho do sistema.

Para o modelo univariável mais simples, escolheu-se o movimento de Flexão-Extensão de braço, segmentado em quatro estados (estendido - flexionando - flexionado - estendendo). Para o modelo escalado em relação ao número de estados, escolheu-se o movimento Flexão seccionada-Extensão, que é uma adaptação da Flexão-Extensão com

uma pausa durante a flexão, sendo segmentado então em seis estados (estendido - início da flexão - semi-flexionado - fim da flexão - flexionado - extensão). Por fim, para o modelo escalado em relação ao número de variáveis escolheu-se o movimento de Alongamento de Tríceps, convenientemente também segmentado em quatro estados (Tríceps estendido - braço levantando - alongando tríceps - braço abaixando) para isolar a variação de parâmetros ao número de variáveis.

Para os dois primeiros movimentos (univariáveis), o ângulo monitorado foi o ângulo entre o antebraço e o braço. Para o último (multivariável), foram monitorados o ângulo entre o antebraço e o braço e o ângulo do braço em relação ao tronco.

O movimento de Flexão seccionada-Extensão, especialmente, foi o movimento escolhido para a realização do teste de invariância ao usuário, no qual três sujeitos externos ao desenvolvimento do projeto foram convidados a testar o processo de estimação.

Apesar de formalmente não ter havido um teste único para a variação da taxa de atualização do filtro de Kalman, esse conceito é explorado em diversos pontos das seções a seguir, mostrando o efeito que aumentar ou diminuir essa taxa tem sobre os resultados da estimação.

Todos os testes foram realizado com as mesmas configurações e posicionamento das IMUs. Para analisar os resultados, gráficos foram gerados com as ferramentas de plotagem oferecidas na pasta "Output" do sistema.

4.1 Flexão-Extensão (Univariável)

4.1.1 Teste de Velocidade

O algoritmo de estimação foi rodado a uma taxa de atualização de aproximadamente 20 Hz para esse teste, e os resultados estão mostrados na Figura 47.

Ao analisar a progressão dos estados para cada uma das três velocidades, pode-se perceber que o chaveamento do modelo SLDS foi extremamente acurado. Seguindo as marcações de referência, vê-se que o instante no tempo em que a troca de estados ocorre é exato a menos de uma diferença de no máximo uma amostra.

Em relação ao desempenho do Filtro de Kalman, pode-se ver que a fidelidade da estimação diminuiu levemente com o aumento da velocidade do movimento, mas de forma alguma o suficiente para perder a dinâmica da curva. Isso implica em uma manutenção da acurácia e uma leve perda em precisão na estimação da posição para velocidades mais altas.

Ao analisar o movimento mais rápido (mostrado em verde na Figura 47), pode-se ver que estados "estáticos", de menor variância, são acompanhados mais fielmente pelo

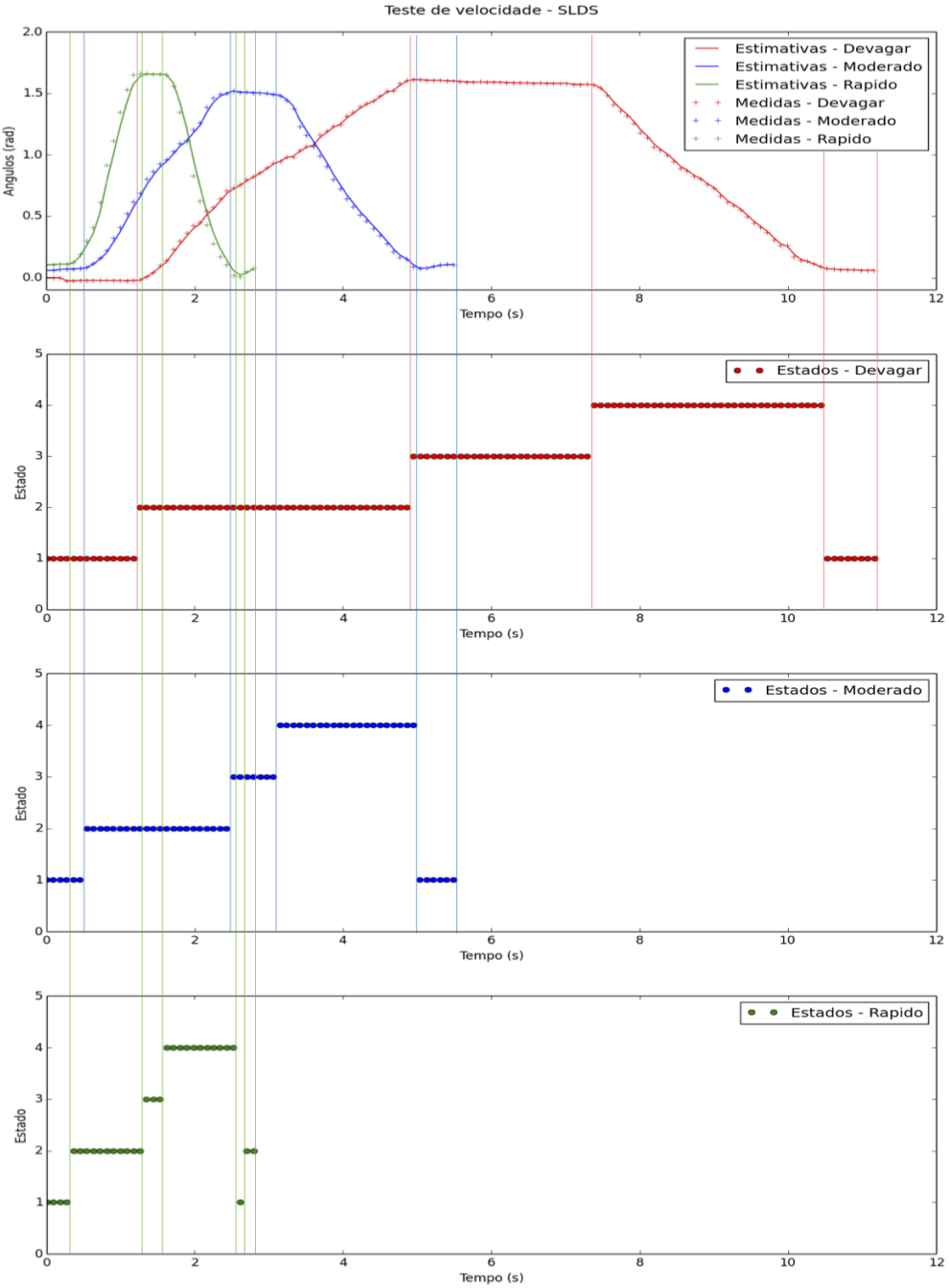

Figura 47 – Teste de velocidade univariável para movimento de quatro estados.

filtro do que estados que envolvem a movimentação do membro e que, por consequência, ao menos neste estudo, possuíram variância maior. A curva mais lenta (em vermelho), mostrou excelente acurácia e precisão na estimação da posição angular do membro, e a curva moderada (em azul), também alta acurácia e uma precisão já levemente defasada.

Vale lembrar aqui que para determinar essa sequência de estimações (de ambas as variáveis ocultas - posição contínua e estado discreto) foi utilizado o método *Backward* de Filtragem, que só faz uso a informação colhida até o instante presente para julgar

o melhor caminho percorrido até então. Isso significa as curvas da Figura 47 (tanto de estados quanto da posição) são construídas com informação limitada do processo e em tempo real na aplicação, o que mostra que apesar da pequena perda em precisão, o teste mostrou um resultado inclusive melhor do que se esperava, indicando boa resiliência ao lidar com velocidades distintas no caso univariável com movimentos de até quatro estados.

4.1.2 Teste de Amplitude

Também a uma taxa de atualização de aproximadamente 20 Hz, este teste rendeu os resultados mostrados na Figura 48.

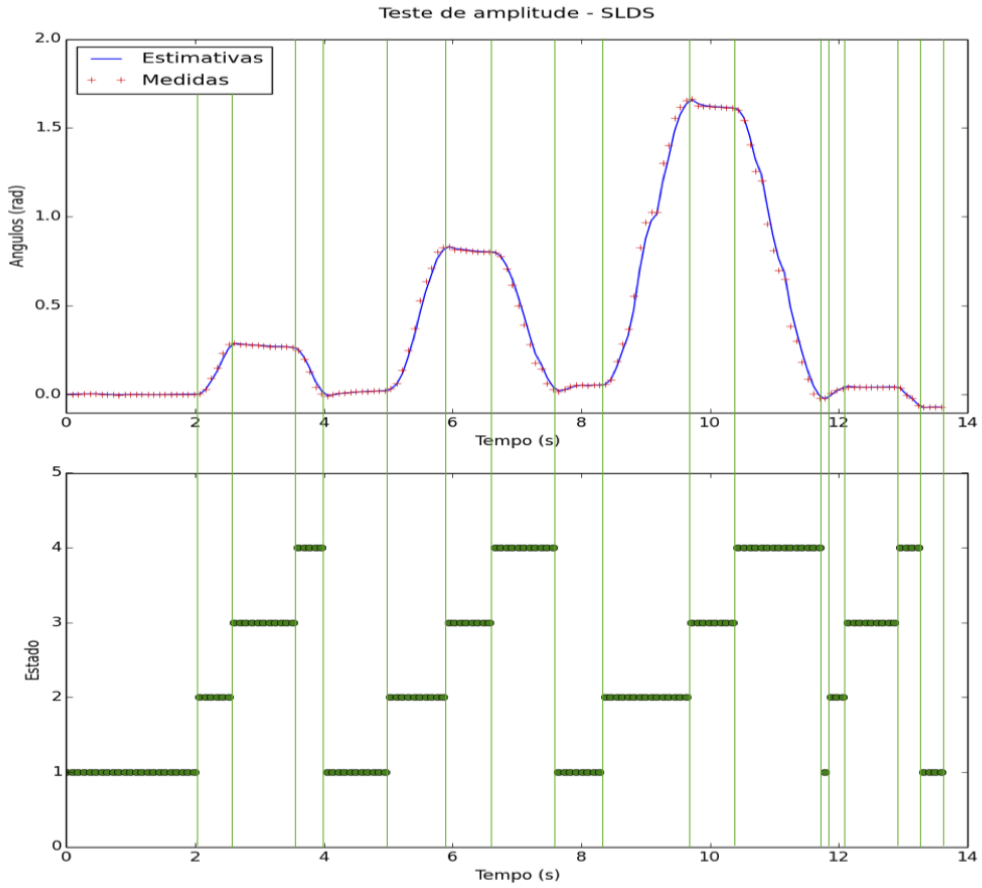

Figura 48 – Teste de amplitude univariável para movimento de quatro estados.

Mais uma vez, é possível ver que a alta acurácia no chaveamento dos estados proporcionou segmentações exatas para os quatro casos de amplitudes testadas. Isso mostrou que, ao menos em princípio, o algoritmo funciona de forma independente da amplitude do movimento, entregando resultados coerentes em todos eles.

Além da acurácia no tempo, pode-se ainda realizar uma análise qualitativa dos resultados deste teste ao observar a progressão dos estados separadamente do gráfico de

posição. Excetuando o eixo do tempo, é possível ver que o padrão na sequência dos estados é perfeitamente consistente durante todo o teste, sem qualquer oscilação. A sequência {1,2,3,4,1,2,3,4,...} se mantém por todo o intervalo, o que mostra também o bom funcionamento do modelo.

É interessante observar, em contraste com o teste anterior (teste de velocidade), que os movimentos desse teste foram executados sem se preocupar com a velocidade, desempenhando-os no tempo que seria natural do movimento. Pode-se ver, nesse caso, que a estimação da posição seguiu as medidas de forma muito semelhante à da curva de velocidade "moderada" do teste anterior, com acurácia e precisão bastante altas mas com uma leve degradação da precisão.

Em suma, esse teste mostrou de forma satisfatória que o modelo SLDS da forma como foi implementado funciona de forma invariável com a amplitude do movimento.

4.1.3 Teste de Robustez

À mesma taxa de atualização 20 Hz, esse teste foi realizado da seguinte forma: primeiro movimento de forma natural; segundo movimento com tremores leves porém perceptíveis no membro; terceiro movimento com tremores exagerados no membro. Objetivou-se aqui analisar de o algoritmo era capaz de continuar respondendo de forma consistente com o aumento do ruído no movimento. Obteve-se desse teste os resultados mostrados na Figura 49.

No primeiro movimento, o chaveamento realizado pelo algoritmo foi ótimo, segmentando as componentes nos instantes corretos. No segundo movimento já pode-se notar a primeira alteração: apesar de ter segmentando quase o movimento inteiro também de forma ideal, por volta de 12,5 s da gravação o filtro de Kalman encontrou um pequeno platô e assumiu que o movimento tivesse estabilizado, chaveando do quarto estado para o primeiro. Como não existiam outros modelos descendentes além do quarto, e como a matriz de transição do HMM elevou demais o custo para retornar do primeiro para o quarto estado, foi mais rentável para o algoritmo permanecer no primeiro estado até que, de fato, no terceiro movimento, a curva recomeçasse a subir.

Ao observar o platô que causou o erro no julgamento do algoritmo, pode-se perceber que ele realmente é muito pequeno, da ordem de um quarto de segundo. Aqui surge a primeira consideração acerca da taxa de atualização do filtro de Kalman: taxas mais elevadas (maiores ou iguais a 20 Hz) geram estimações exageradamente responsivas a ruído.

Esse fato é corroborado pela análise do terceiro movimento (o mais ruidoso dos três). Nele, é possível ver que picos locais muitas vezes minúsculos são interpretados como movimentos individuais e completos. Como o HMM garante consistência na máquina de

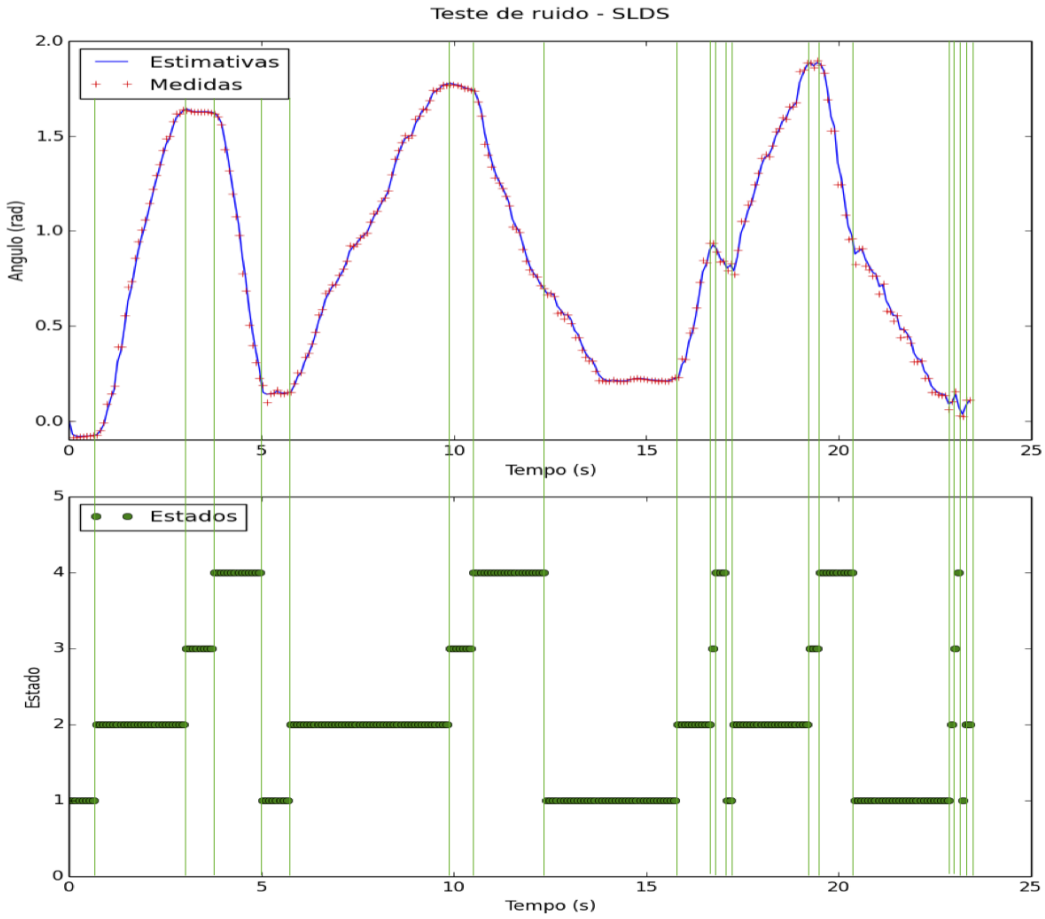

Figura 49 – Teste de robustez univariável para movimento de quatro estados.

estados que é percorrida ao longo do algoritmo, esse comportamento de alta frequência gera uma micro-sequência de estados dentro do movimento, como de fato pode ser visto correndo três vezes no último movimento da Figura 49.

A solução para esse tipo de comportamento é a implementação de algo que atue como um filtro passa-baixas no processo de estimação do Filtro de Kalman. De forma bastante simples, diminuiu-se então a taxa com que o processo de atualização era feito e, a título de contraste, projetou-se um conjunto de curvas submetidas também a níveis exagerados de tremor (até mais do que no caso original) sob a taxa de 8,5 Hz. Alguns dos resultados estão mostrados na Figura 50

Em primeiro lugar, é importante frisar que a extrema agitação a que foram submetidos os movimentos nesse caso não é aparente pois como a unidade de processamento só recebe as amostras a cada passo de atualização do filtro de Kalman, o vetor de medidas que é plotado na curva só é construído a cada ciclo de atualização. Isso quer dizer que diminuir a taxa de atualização significa aplicar o filtro passa-baixas não só na estimação como também na visualização das medidas obtidas dos sensores, o que suaviza a curva apresentada. Possibilidades alternativas para corrigir essa divergência de visualização se-

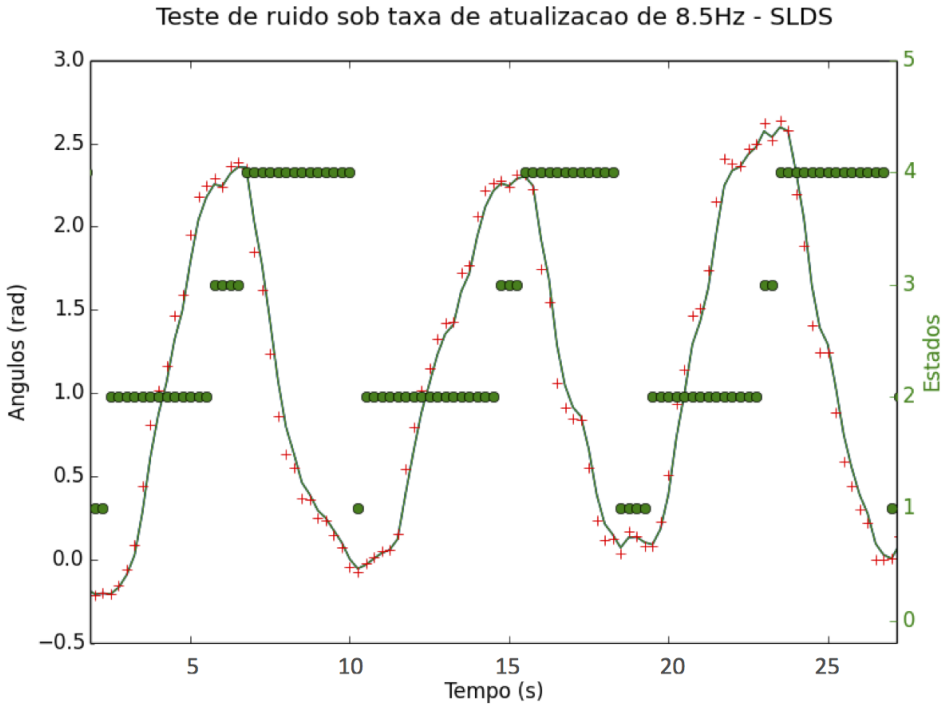

Figura 50 – Teste de robustez univariável para movimento de quatro estados sob taxa de atualização de 8,5 Hz.

rão discutidas no capítulo de Conclusão.

De qualquer forma, pode-se perceber na Figura 50 que a diminuição da taxa de atualização do Filtro de Kalman corrigiu a análise exagerada que fora obtida anteriormente. Logicamente, isso vem a um preço que está relacionado a um pequeno atraso da ordem de uma ou mais amostras nas transições de estados.

O desafio fica restrito então a balancear a taxa de atualização do Filtro para um valor que seja adequado à granularidade que se deseja obter na medição em questão. Independente disso, essa análise mostrou que é possível deixar o algoritmo robusto o suficiente a ruído motor de forma a responder de maneira satisfatória ao processo de estimação de um movimento univariável de até quatro estados.

4.2 Flexão seccionada-Extensão (univariável)

Este movimento, apesar de similar ao primeiro, possui um diferencial importante: a pausa durante a flexão do braço leva à criação de dois estados a mais na segmentação do movimento. Isso significa, para o sistema, uma adesão de complexidade para a qual vale a pena reavaliar o desempenho dos testes anteriores.

4.2.1 Teste de Velocidade

Os resultados desse teste, realizado à taxa mais baixa de 8,5 Hz, estão apresentados na Figura 51 a seguir.

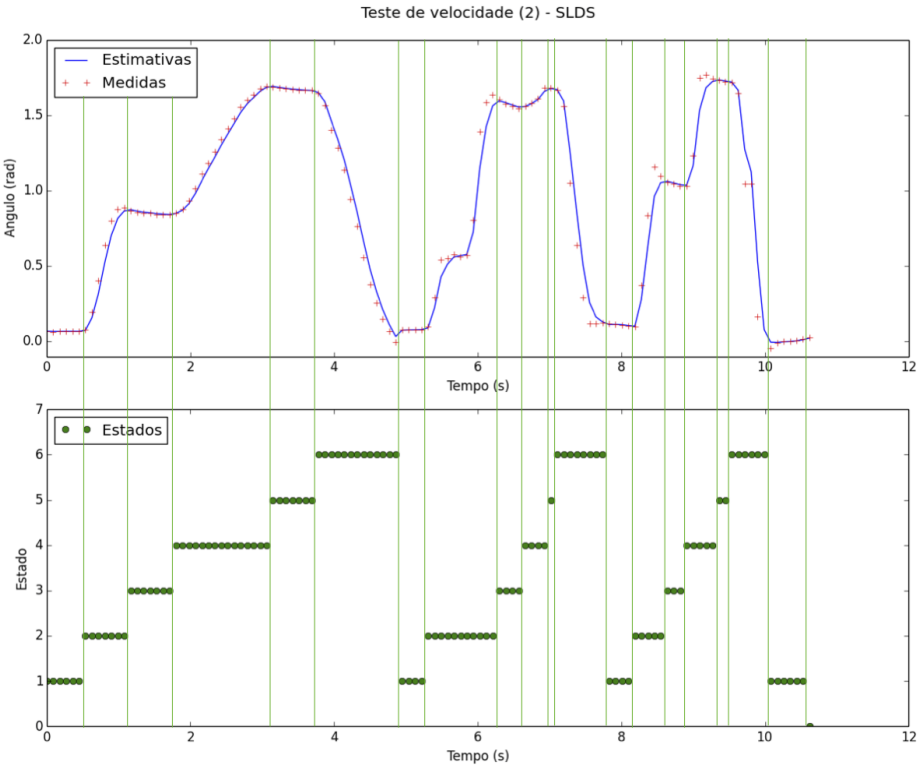

Figura 51 – Teste de robustez univariável para movimento de seis estados sob taxa de atualização de 8,5 Hz.

Baseado no que se pode observar, vê-se que, para o movimento mais lento e para o movimento mais rápido, o algoritmo retornou um chaveamento ideal dos estados, decodificando perfeitamente as componentes da curva. Para o segundo movimento, no entanto, houve um pequeno desvio: a uma taxa mais baixa de atualização, a pequena pausa do meio da flexão de braço não foi reconhecida, resultando em uma segunda componente prolongada e um mapeamento errôneo dos estados 3, 4 e 5. Apesar de terem se manifestado de forma condizente com seus respectivos modelos (3 aproximadamente estático, 4 crescente e 5 também aproximadamente estático), esses três estados tomaram o lugar do que deveria ser apenas o estado 5. A moderada perturbação no topo da curva apenas contribuiu para isso.

De fato, essa ocorrência comprova o fato de que 8,5 Hz é, possivelmente uma taxa muito baixa para a atualização. Com relação a isso, pode-se perceber que as "quinas" das medidas também foram atenuadas consideravelmente na transição entre estados (o que não é um problema em si). Dessa forma, percebeu-se que, de fato, abaixar demais a taxa de Atualização do Filtro de Kalman também não é a melhor solução. Portanto, deve-se

encontrar uma ponderação para que taxa não seja nem muito alta (e responsiva a ruído) nem muito baixa (e tolerante demais).

Escolheu-se então a taxa de 12 Hz. A 52 mostra o resultado do teste de velocidade moderada para essa taxa mais elevada.

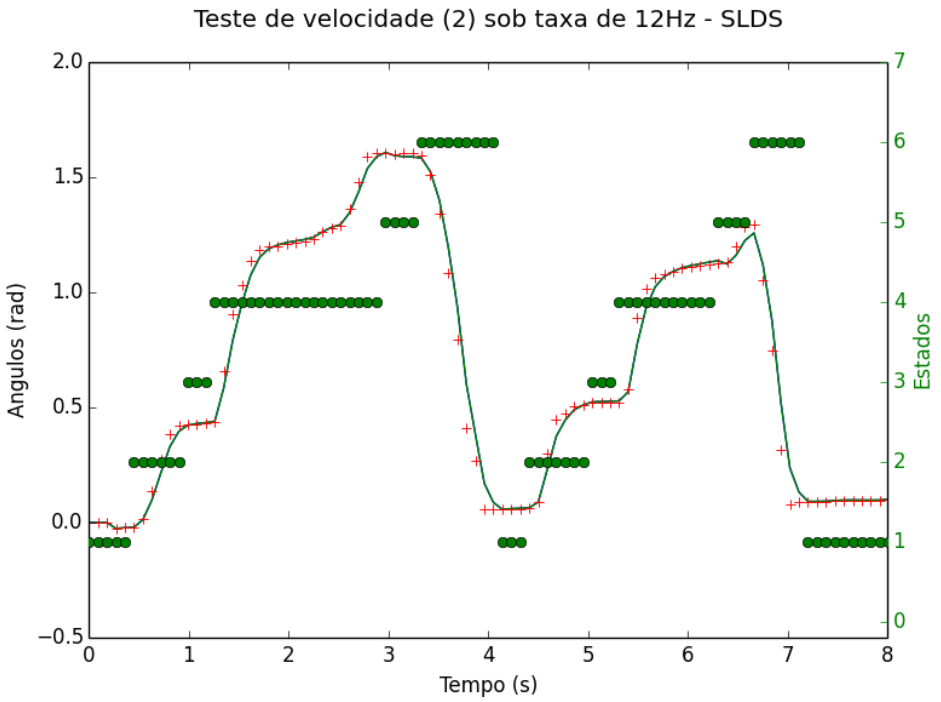

Figura 52 – Teste de velocidade univariável para movimento de seis estados sob taxa de atualização mais elevada (12 Hz).

Esse teste foi bastante revelador. Ao inserir componentes espúrios no movimento para estressar o sistema, percebeu-se que o algoritmo fez exatamente o que deveria ter feito. O procedimento foi o de, no momento da estimação, provocar estados semi-flexionados (3) mais curtos e então fazer uma transição gradual do estado de finalização da flexão (4) para o estado flexionado (5), com pequenas variações de velocidade de forma a aproximar a forma de "patamar" que poderia confundir o algoritmo como na Figura 51.

Assim, no primeiro movimento da Figura 52, vê-se que, após um terceiro estado bastante curto (menos de 0,5s de duração), produziu-se uma variação de velocidade no final do quarto estado em que um patamar levemente inclinado foi criado antes de voltar a acelerar e então assentar de fato. Sem grandes problemas, o sistema identificou o terceiro estado e considerou de maneira correta o patamar inclinado como parte do quarto estado, chaveando para o quinto estado apenas quando o movimento de fato se estabilizou.

No segundo movimento da figura 52, um terceiro estado um pouco mais longo foi gerado e, dessa vez sem oscilar muito a velocidade do quarto estado, foi-se diminuindo-a gradualmente. Ao final, antes de voltar a estender o braço, provocou-se um pequeno pico.

Novamente, no entanto, o algoritmo conseguiu segmentar o movimento de forma satisfatória, identificando o terceiro estado e estendendo o quarto estado até pouco antes do pico, quando de fato o movimento estabilizou. O pico não representou qualquer problema.

Resolvida a questão da taxa de atualização e voltando para a Figura 51, pode-se enxergar que o Filtro de Kalman se comporta basicamente da mesma forma que na seção 4.1.1. Para cada movimento mais rápido, a estimação da posição sofre um pouco em precisão. Comparando-se a duração de cada curva desse movimento com aquelas do movimento de Flexão-Extensão, vê-se que o desvio das amostras com relação à velocidade manteve-se praticamente o mesmo. De qualquer forma, isso não prejudicou significativamente o desempenho das tarefas de segmentação.

Em suma, os resultados desse teste mostraram que aumentar o número de estados não prejudica a flexibilidade do algoritmo em relação à velocidade do movimento.

4.2.2 Teste de Amplitude

Esse teste foi realizado da mesma forma que na seção 4.1.2, e seus resultados estão mostrados na Figura 53.

Ao observar a figura, vê-se sem muita dificuldade que a segmentação do movimento realizada pelo algoritmo foi realizada de maneira exemplar. Para esse teste não houve qualquer problema e, de maneira bastante direta, o sistema foi capaz de estimar sem maiores complicações e com alta acurácia a sequência de estados percorrida pelo movimento.

A conclusão foi que, mais uma vez, a alteração de amplitude não impactou o desempenho do sistema de forma alguma. Além disso, os resultados do teste sugeriram que um aumento de estados do movimento analisado não prejudica o julgamento referente a diferentes amplitudes de entrada.

4.2.3 Teste de Robustez

Realizado também da mesma forma que na seção 4.1.3, esse teste retornou resultados como aqueles mostrados na Figura 54.

Similar ao teste de amplitude, os resultados na Figura 54 foram bastante satisfatórios. Todos os estados foram julgados adequadamente com as estimações de posição e não houve grandes problemas.

É interessante perceber que mesmo no caso de maior perturbação, para o qual os tremores foram executados de maneira realmente exagerada, o sistema se comportou adequadamente, mapeando os eventos no instante correto.

Pode-se aqui questionar esse resultado ressaltando que, por sorte, as maiores per-

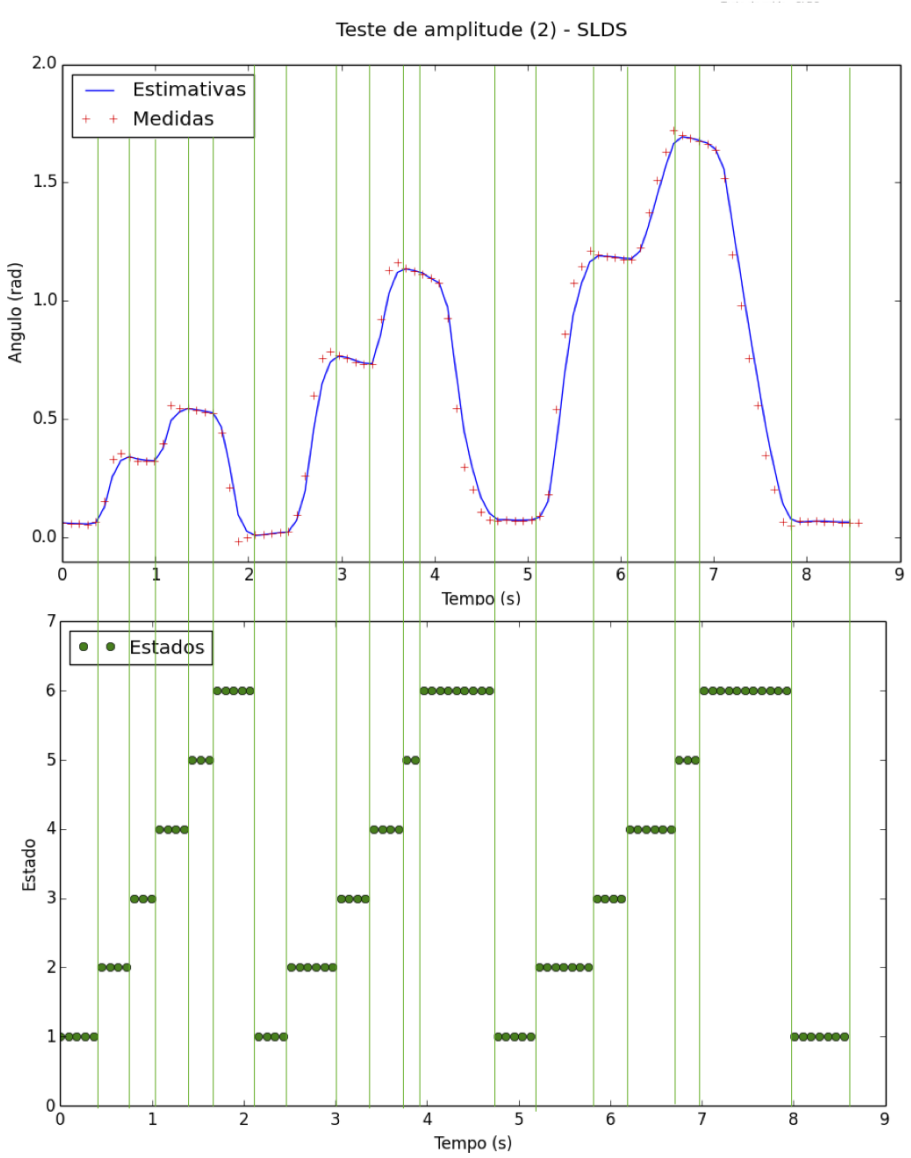

Figura 53 – Teste de amplitude univariável para movimento de seis estados.

turbações no sistema foram todas no sentido contrário das variações propostas pelo estado seguinte (por exemplo o vale criado no terceiro estado, que é descendente enquanto o quarto estado é ascendente). De fato, se há, por exemplo, uma variação descendente no estado 3, mesmo com uma resposta divergente do filtro de Kalman acaba sendo mais custoso para o algoritmo chavear para o quarto estado do que permanecer no terceiro, uma vez que a dinâmica do quarto estado é praticamente oposta a uma dinâmica descendente. Isso faz com que perturbações dessa natureza funcionem como um bloqueio ao chaveamento de estados, o que é uma "faca de dois gumes": se por um lado não se permite chavear o sistema para um estado que seria ainda mais divergente do que o atual, também não se chaveia o estado atual para nenhum outro mais adequado porque a máquina de estados administrada pelo HMM não define essa transição (por exemplo do estado 3 para o estado 6 na Figura 54).

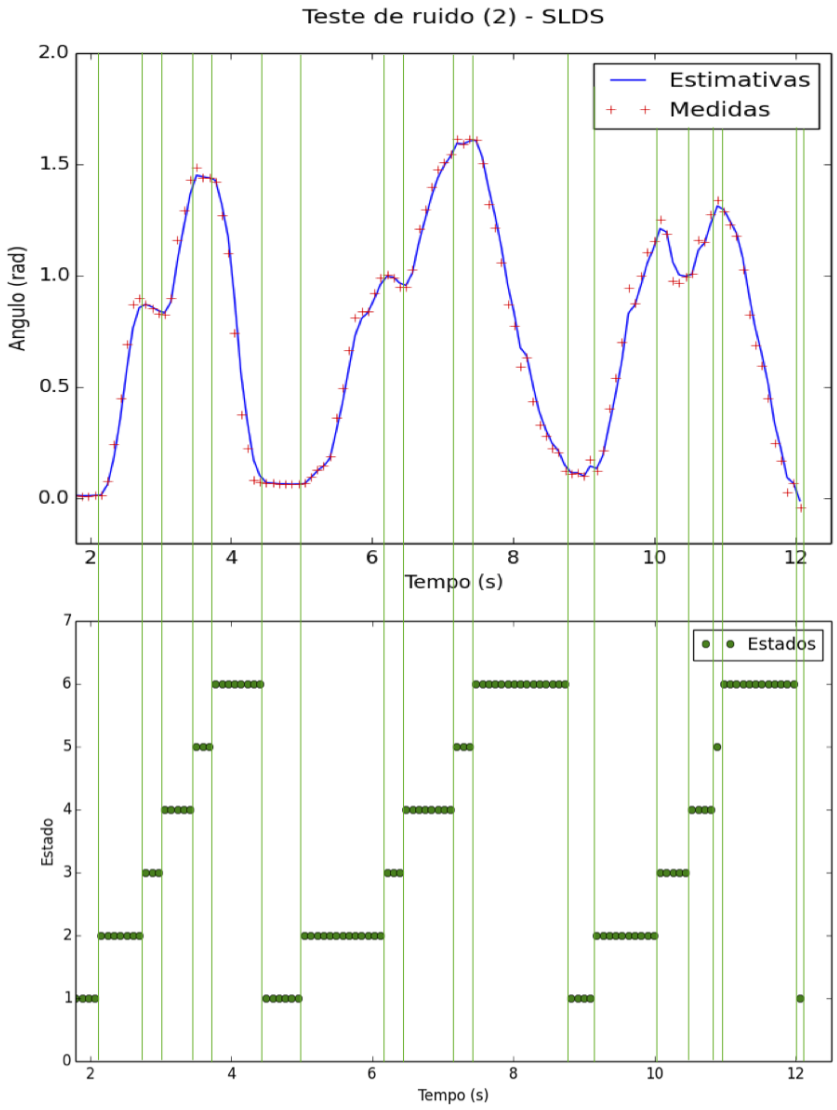

Figura 54 – Teste de robustez univariável para movimento de seis estados.

Essa característica que garante a palavra final ao HMM e que não permite, em última instância, que o Filtro de Kalman chaveie os estados de uma forma que a matriz de transição de estados não permita, ocorre por conta de um parâmetro utilizado no cálculo do custo de cada transição. Aos elementos zerados de Π, ou seja, às transições de estados que nunca foram observadas na etapa de gravação e parametrização, é associado o maior custo computacionalmente possível de se atribuir. Assim, ainda que o Kalman cresça muito, o máximo que seus fatores podem chegar a ser na equação do custo é iguais a esse custo máximo, o que ainda não é suficiente para permitir o chaveamento de estados.

Sendo desejável mudar essa dinâmica e permitir que o Filtro de Kalman argumente com o HMM mesmo quando a probabilidade de transição é 0 na Matriz Π, basta ajustar esse parâmetro.

De qualquer maneira, este teste mostrou que o algoritmo continua robusto ao

adicionar mais estados e testá-los contra níveis altos de ruído.

4.2.4 Teste de Invariância ao Usuário

Este teste foi realizado com dois objetivos: analisar o comportamento do algoritmo para diferentes usuários além do autor, de forma a provar que seus vícios motores não impactaram no desenvolvimento da aplicação; e observar os resultados da estimação em um ambiente "descontrolado".

O teste foi realizado com três sujeitos de idades e características antropomórficas ligeiramente distintas. Cada sujeito foi orientado a realizar o movimento de Flexão seccionada-Extensão algumas vezes livremente, sugerindo-se a possibilidade de realizá-los mais rápido ou mais devagar, com tremor ou sem, mais curtos ou mais longos, da forma como os sujeitos preferissem.

Os resultados mais interessantes obtidos das três observações está mostrado na Figura 55 abaixo. Dos resultados obtidos, pode-se perceber que aqueles mais acurados foram os do sujeito #1, representados pelo primeiro quadro da Figura 55. Aqui é possível ver que todos os chaveamentos foram executados no instante correto, resultando em uma segmentação perfeita do movimento.

Já para o caso do sujeito #2, os primeiros erros puderam ser percebidos. A primeira nuance que pode-se observar é que o primeiro movimento realizado por este sujeito foi diferente daquilo que era esperado pelo algoritmo. Na tentativa de se adaptar à natureza do movimento, a estimação julgou um pequeno pico na base do movimento como sendo o início da flexão de braço, prosseguindo com a sequência de estados daí em diante. Apesar de não estar errado no que diz respeito à natureza de cada estado, sabe-se, no entanto, que o pequeno pico percebido como os estados 2 e 3 não passam de um ruído no início do movimento. De qualquer maneira, provavelmente essa era de fato o melhor palpite que o sistema poderia dar nessa situação.

O segundo movimento executado pelo sujeito #2 também possui um erro na segmentação: por conta de outro pequeno pico na transição do estado 4 para o estado 5, o algoritmo atribuiu o estado estático que deveria ser o quinto ao primeiro, julgando que a pequena descida do pico já fosse o sexto estado (estendendo braço) e que já se havia retornado à condição original. Assim, toda a verdadeira extensão de braço ficou mascarada pelo primeiro estado.

Outro problema com a estimação para este usuário foi o fato de que a zona de ruído entre a segunda e terceira execução de movimentos, por possuir um perfil levemente semelhante ao do movimento parametrizado, foi considerada como sendo um movimento à parte. Isso ocorreu pois, ao retornar do movimento anterior, a extensão do braço ultrapassou a posição angular inicial, fazendo com que o antebraço "ricocheteasse" alguns

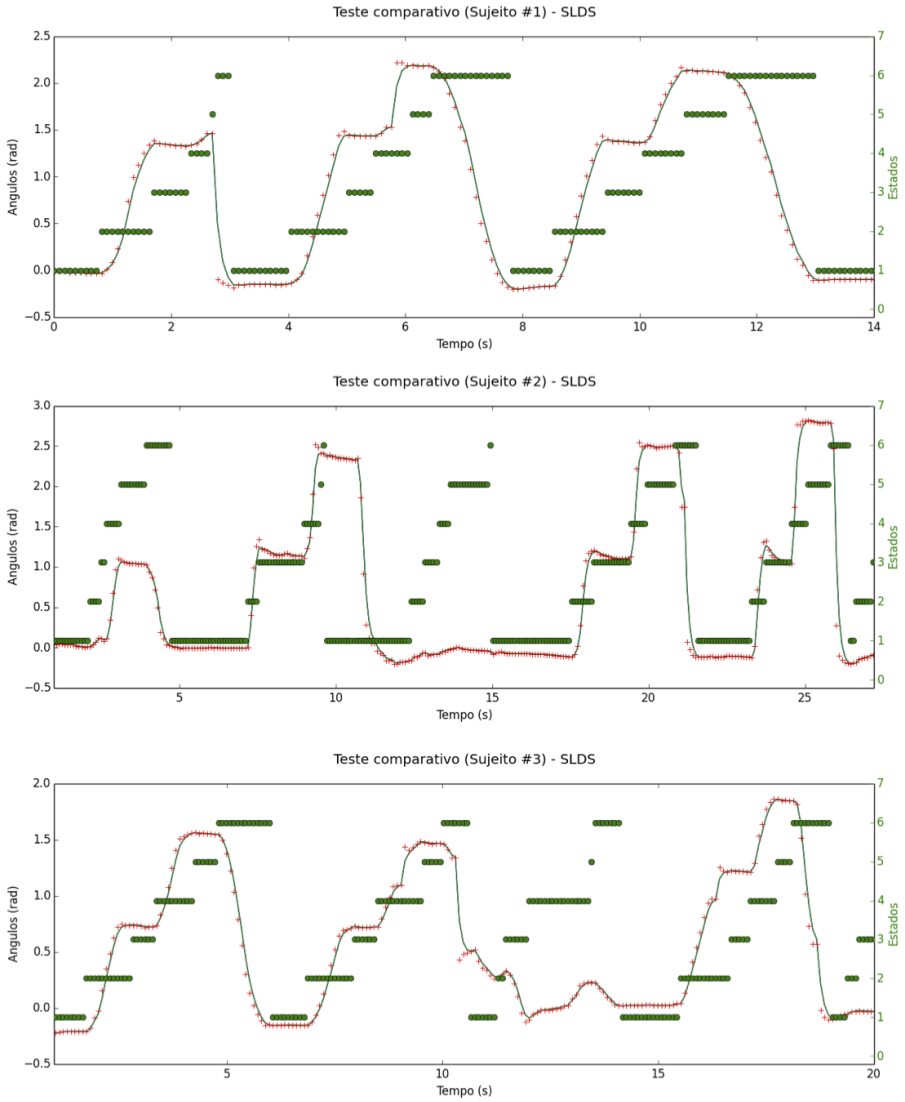

Figura 55 – Teste de invariância ao usuário realizado com três indivíduos distintos.

graus ao final do movimento, oscilando sutilmente antes de se estabilizar.

Por fim, os últimos dois movimentos do segundo sujeito foram segmentados com precisão e acurácia.

Para o sujeito #3, o primeiro e o último movimentos geraram respostas bastante coerentes, sendo segmentados nos instantes corretos. O segundo movimento realizado, no entanto, só foi segmentado corretamente até o início do sexto estado (estendendo braço). A partir daí, o surgimento de um pequeno platô na descida do ângulo levou o algoritmo a acreditar que o movimento já havia se estabilizado, chaveando para o primeiro estado. Como não existem estados descendentes definidos após o primeiro, o algoritmo manteve-se nesse estado até que o primeiro indício de ascendência surgiu, dando margem para o segundo estado. Entretanto, mesmo esse perfil de subida ainda fazia parte de uma perturbação no estado de extensão do braço. Como logo em seguida houve a execução

do movimento para uma amplitude extremamente baixa, o sistema considerou toda a sua fase de ascensão (estados 2, 3, e 4) como sendo apenas pertinente ao estado 4 em uma continuação aos três primeiros estados gerados erroneamente na curva anterior.

Observando os resultados para os três sujeitos, vê-se que, quando fora de um ambiente controlado, o algoritmo pode responder de forma incorreta. Das 74 transições analisados na Figura 55, 59 foram corretas, enquanto o resto foi mapeado de maneira errada. Isso computa uma taxa de acertos aproximada de aproximadamente 79.7% para ambientes ditos descontrolados.

É claro que os resultados analisados na Figura 55 foram obtidos de apenas uma parte dos dados colhidos. Analisando mais profundamente o restante dos dados, seria possível construir uma estatística mais exata. Houve resultados melhores do que aqueles mostrados e também houve resultados piores. Aquilo que foi discutido nesse teste, no entanto, resume o comportamento geral das outras medições.

É difícil afirmar se esse teste de fato diz alguma coisa sobre a invariância do sistema aos usuários. O que ele revela, com certeza, é que em ambientes não controlados os espúrios nas medidas podem causar todo tipo de reação dos modelos chaveados. Sugere-se realizar mais testes em campo e com outros usuários de forma aprimorar a robustez do sistema a esses comportamentos.

Em um contexto de fisioterapia, é importante lembrar que há um profissional responsável por guiar o paciente pelo processo do exercício, o que significa que para o objetivo deste trabalho ainda existe um certo controle sobre o movimento. Isso, contudo, não elimina a necessidade de tratar a reação do algoritmo a essas perturbações motoras durante os processos de estimação.

Em suma, acredita-se que este foi um dos testes mais importantes do projeto, e talvez aquele que mais se aproxime de um cenário real de aplicação.

4.3 Alongamento de Tríceps (Multivariável)

De forma a testar a escalabilidade do algoritmo no sentido do número de variáveis monitoradas, decidiu-se testar o movimento de Alongamento de Tríceps. Esse movimento leva em conta o monitoramento não apenas do ângulo entre braço e antebraço como também o do ângulo entre braço e tronco, totalizando então duas variáveis observáveis.

Uma das maneiras de se dividir o movimento do alongamento de tríceps é em quatro estados para cada variável (diagramas parciais) e em quatro estados no movimento composto (diagrama principal), onde em geral o primeiro estado simboliza a posição relaxada, o segundo estado simboliza o movimento de trazer o braço para trás da cabeça, o terceiro estado a posição de alongamento e o quarto estado o movimento de relaxar o

braço novamente, levando-o para a frente do corpo. Analisando-o como um todo, vê-se que é a expansão lógica do movimento de Flexão-Extensão tratado na seção 4.1.

Trabalhar com uma análise multivariável permitiu ainda avaliar a capacidade do sistema de executar o que de fato se chama Segmentação, combinando diversos estados para segmentar o movimento em função de todas as variáveis sendo observadas.

Vale ressaltar que todos os testes detalhados abaixo foram realizados à taxa de atualização de 20 Hz.

4.3.1 Teste de Velocidade

De forma análoga à seção 4.1.1, os resultados do teste de velocidade para três ritmos distintos estão detalhados na Figura 56 a seguir.

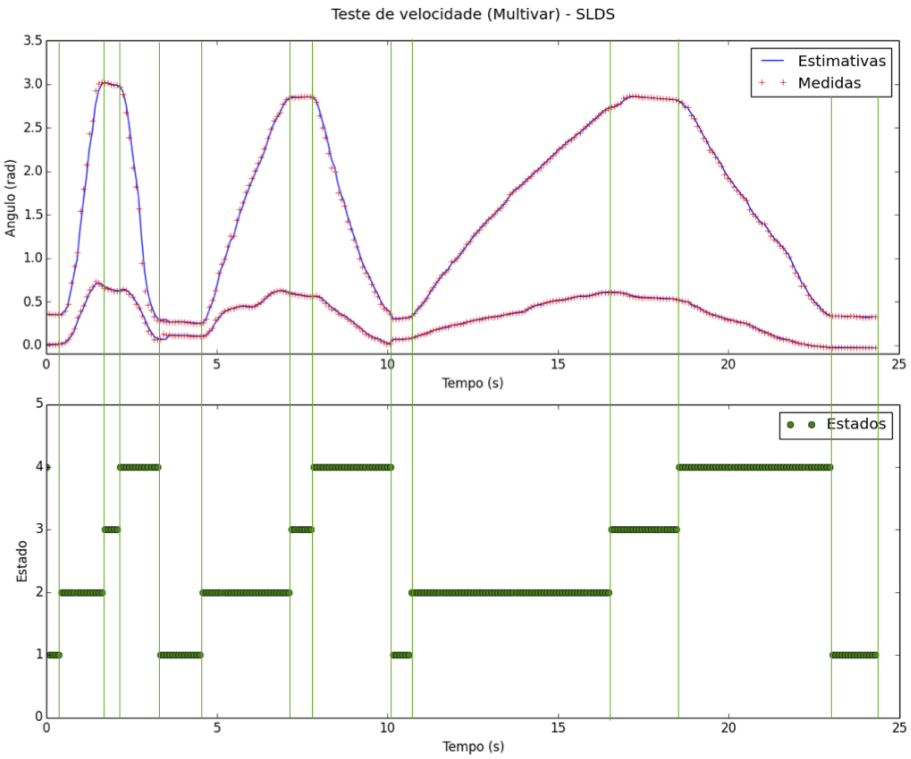

Figura 56 – Teste de velocidade multivariável para movimento de quatro estados. A curva inferior representa o ângulo entre braço e tronco; a curva superior representa o ângulo entre braço e antebraço.

Ao analisar as curvas resultantes desse teste, a primeira coisa que salta aos olhos é que a resposta da estimação de posição está bastante afinada, o que está relacionado aos testes terem sido feitos a uma taxa de atualização mais alta (20 Hz). Em segundo lugar, vê-se que os estados parecem estar sendo chaveados com bastante acurácia, acertando as transições sem dificuldades.

Ao observar os dois movimentos individualmente, no entanto, percebe-se que as transições possuem um comportamento ligeiramente diferente dos casos univariáveis: em alguns pontos, elas parecem acertar perfeitamente o instante do chaveamento, enquanto em outros instantes parecem sutilmente deslocadas para um lado ou para o outro.

Por incrível que pareça, são justamente esses pequenos desencontros aparentes das transições que escondem a verdadeira beleza do caso multivariável: como os estados são composições (ortogonais) dos modelos lineares de ambas as variáveis, os eventos de chaveamento sempre estão orientados à combinação do comportamento de todas as variáveis observadas.

Tomando o segundo movimento (velocidade moderada) como exemplo, vê-se que, para o ângulo entre braço e tronco (curva inferior), parece que a transição do segundo para o terceiro estado só ocorre algumas amostras depois que o ângulo já estabilizou. Todavia, ao comparar esse comportamento simultaneamente com o do ângulo entre braço e antebraço (curva superior), vê-se que até a transição ocorrer, esse ângulo ainda estava aumentando. Em termos mais simples, foi como se, apesar do braço já estar estático, o algoritmo tivesse esperado o usuário terminar de dobrar o cotovelo para declarar que alcançara-se a posição de alongamento de tríceps. Esse é um resultado de extrema beleza, ainda que não se possa perceber à primeira vista.

Esse fenômeno se repete algumas vezes em diversos pontos das curvas, e só é possível pois a decisão de chavear ou não de modelo sempre depende de ambos os movimentos. Essa decisão, por depender da função custo que balanceia as respostas do Filtro de Kalman e do HMM, não é perfeitamente discreta, e sempre vai depender de qual modelo se ajusta melhor às condições observadas.

Outra consequência interessante que isso traz pode ser vista no segundo estado do segundo movimento (velocidade moderada). Nesse segmento do movimento, vê-se que o ângulo entre braço e tronco (curva inferior) sofreu uma perturbação que, no caso univariável, talvez tivesse sido considerada como uma transição de estado. Como ambas as variáveis estão amarradas uma à outra no modelo, entretanto, pequenas flutuações em uma delas não tendem a impactar profundamente o sistema caso a outra também não varie significativamente. Isso significa que, de maneira geral, o sistema multivariável possui uma natureza mais robusta que o sistema univariável.

Com o sucesso desse teste, mostrou-se que a adição de variáveis ao sistema, ao menos em pequena escala, não tende a impactar a flexibilidade da análise em relação a diferentes velocidades de entrada.

4.3.2 Teste de Amplitude

Novamente, a metodologia do teste de amplitude foi seguida como a proposta de 4.1.2. Os resultados foram como mostrado na Figura 57.

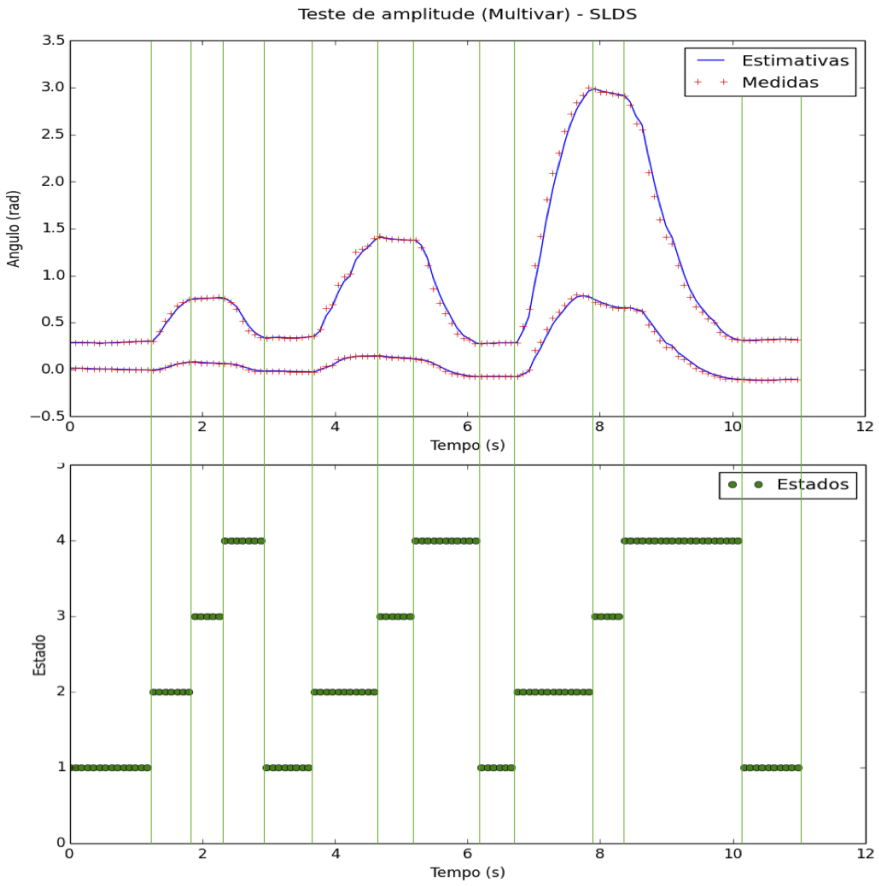

Figura 57 – Teste de amplitude multivariável para movimento de quatro estados. A curva inferior representa o ângulo entre braço e tronco; a curva superior representa o ângulo entre braço e antebraço.

A resposta desse teste mostra, mais uma vez, a especialidade do caso Multivariável discutida no teste de velocidade. Aqui é ainda mais nítida a tendência do ângulo entre braço e tronco estabilizar algumas amostras antes do ângulo entre braço e antebraço, e pode-se observar mais claramente o chaveamento "esperando" para ocorrer apenas ao início da estabilização desse segundo ângulo. Na saída do estado 3 também pode-se observar que a transição ocorre assim que esse ângulo de antebraço começa a diminuir novamente, ainda que o braço ainda esteja estático em relação ao tronco.

Em resumo, o resultado da estimação para esse teste foi bastante positivo, indicando pouca dependência do desempenho do algoritmo com a velocidade quando da adição de mais variáveis ao sistema.

4.3.3 Teste de Robustez

O teste de robustez para o caso multivariável foi executado da mesma maneira que aquela detalhada na seção 4.1.3. As curvas e as sequências de estados resultantes estão apresentadas na Figura 58.

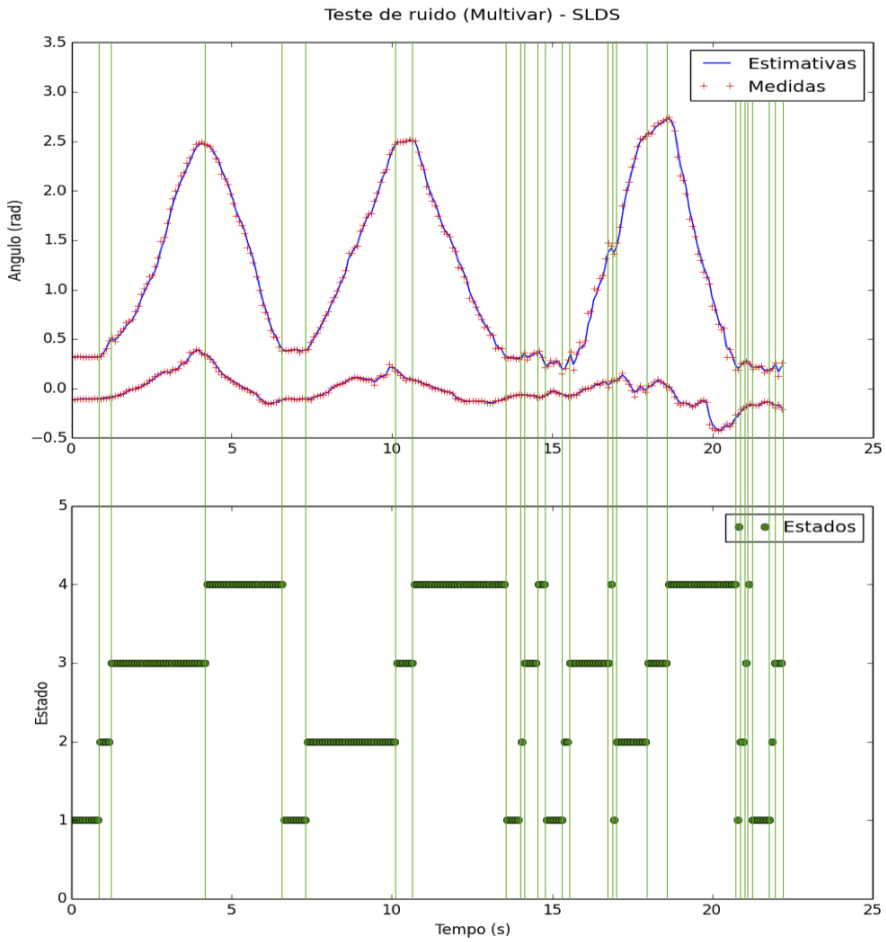

Figura 58 – Teste de robustez multivariável para movimento de quatro estados. A curva inferior representa o ângulo entre braço e tronco; a curva superior representa o ângulo entre braço e antebraço.

A partir desse resultado, é possível mais uma vez entender que a taxa de 20 Hz talvez seja alta demais para o processo de atualização do caso multivariável (como já havia sido mostrado no caso univariável). Ao inserir ruído mecânico ao sistema, e quando de fato esse ruído é exagerado, o sistema perde o controle.

Apesar do último movimento (aquele com mais ruído) ser o mais chamativo, vê-se que, em primeiro lugar, a alta taxa de atualização levou a um erro de chaveamento no primeiro movimento, transitando do segundo para o terceiro estado muito antes do instante correto por conta de uma pequena falha na seção ascendente da curva. Como não havia nenhum estado ascendente definido após o terceiro, o sistema caiu novamente

na própria armadilha de manter um estado atual errôneo até a próxima transição definida para o estado (no caso a transição para um modelo descendente).

Depois desse primeiro erro, a segunda curva, na qual já havia sido inserido um nível moderado de ruído mecânico, mostrou um resultado bastante acurado na estimação da sequência de estados.

O que mais perturbou o sistema, de fato, foi o terceiro movimento. Deve-se admitir aqui que a quantidade de ruído inserido para esse movimento foi extrema e que, novamente, o problema da visualização das perturbações impede a real apreciação do que foi esse processo. Ainda assim, percebe-se que o algoritmo foi capaz de suavizar o ruído e, ao menos a nível do Filtro de Kalman, delinear o movimento pretendido. Por conta disso, acredita-se que a estimação da sequência de estados também deveria reportar bons resultados.

O que pode ser visto no último movimento da Figura 58 é que à medida que se adentra o movimento, o sistema parece começar a se comportar melhor, discernindo bem as fases centrais (final do segundo estado, terceiro completo e início do quarto estado). Os estados que existem na periferia do movimento, no entanto, foram os mais afetados pelo ruído e, por isso, levaram ao problema das micro-sequências de movimentos observado inicialmente na seção 4.1.3.

A proposta para melhorar o desempenho nesse caso é a mesma dos casos univariáveis: diminuir a taxa de atualização do filtro de Kalman. Testes conclusivos acerca do efeito positivo dessa alteração foram realizados mas, a título de não popular mais essa seção com resultados repetidos, decidiu-se omitir a sua conclusão gráfica.

4.4 Outras considerações

Após realizar toda a bateria de testes, não foi apenas possível validar o funcionamento do sistema mas foi também possível entender mais profundamente a relação entre o algoritmo e cada uma das variáveis envolvidas.

Em suma, mostrou-se que o modelo SLDS funciona da mesma forma independentemente da velocidade ou da amplitude do movimento realizado. Com a taxa certa de atualização (encontrada como sendo, para esta aplicação, por volta de 12 Hz), mostrou-se também que é possível tornar o algoritmo robusto a ruído motor, tanto para o caso univariável quanto para o caso multivariável.

A resposta do modelo implementado para casos reais "não controlados", submetidos a diversos usuários, é razoável mas precisa ser estudada melhor a fim de entender como torná-la mais consistente.

Houve ainda um último teste realizado para todos os movimentos: um teste que envolveu estudar se adicionar mais repetições do movimento no processo de gravação e

parametrização impactava positivamente a fase de estimação. O teste, para cada movimento, foi realizado com uma, duas e quatro gravações, mas não mostrou qualquer melhora significativa que justificasse seu suporte.

De maneira ampla, os testes realizados revelaram um bom comportamento do sistema de processamento, indicando um algoritmo capaz de extrair parâmetros e segmentar movimentos em tempo real de maneira bastante satisfatória para o que foi uma primeira implementação da plataforma.

5 Conclusão

A captura e avaliação automática de movimentos tem se mostrado um tópico cada vez mais marcante na pesquisa por aprimoramentos no tratamento de pacientes com debilidades motoras, no treinamento esportivo, na indústria de jogos e em uma série de outros campos da ciência. Saber que é possível, através de algoritmos que já começam a se consolidar, ajudar a melhorar a qualidade de vida de milhares de pessoas é, sem dúvida nenhuma, um grande motor para a contínua pesquisa e desenvolvimento dessas aplicações.

Este trabalho propôs a implementação de uma plataforma de captura e avaliação de movimentos para membro unilateral superior que pudesse gerar *feedbacks* úteis para o processo fisioterapêutico. Para cumprir com esse objetivo, dois elementos foram centrais: a aplicação do modelo SLDS de estimação, inovador por atacar tanto o problema segmentação quanto o de extração de parâmetros do movimento; e a construção de uma interface gráfica capaz de situar o usuário ao redor desse processo.

Ao final deste trabalho, através de um excelente *hardware* de aquisição de dados (os *3-Space Sensors* da YOST labs), implementou-se:

- um *driver* estável, capaz de comunicar plenamente o *hardware* escolhido ao *software* projetado;
- uma interface gráfica tridimensional em JavaScript coerente com os modelos fornecidos em mercado e capaz de rastrear o braço do paciente em tempo real, guiando-o pelo processo de estimação através de um menu e fornecendo resultados *online*;
- uma unidade de processamento em Python, capaz de executar o algoritmo SLDS-Viterbi aproximado e de realizar Extração de Parâmetros e Segmentação de movimentos em tempo real;
- e uma camada de comunicação com base no modelo servidor-cliente que integrou de forma satisfatória as duas frentes do *software*.

A plataforma implementada pôde ser testada de diversas formas, como mostrado no Capítulo 4, e de fato atendeu às especificações propostas, sendo capaz de operar de forma consistente para os três movimentos testados (Flexão-Extensão de braço, Flexão seccionada-Extensão de braço e Alongamento de Tríceps) e de forma invariável à amplitude e velocidade. Com o ajuste da taxa de atualização do sistema de processamento, pôde-se torná-la também invariável a um nível aceitável de ruído, tanto para o caso univariável quanto para o caso multivariável. Testes preliminares com três indivíduos distintos

também foram realizados no sentido de avaliar o desempenho do sistema em cenários reais com múltiplos usuários e, apesar de indicarem um bom comportamento do sistema, sugerem a necessidade de realizar mais testes no sentido de tornar o sistema mais resiliente.

Assim, cumpriu-se com os objetivos propostos inicialmente para este projeto e também com aqueles descritos de forma intermediária durante o desenvolvimento do sistema. Ainda que a plataforma tenha atendido às especificações, no entanto, é inevitável que no desenvolvimento de um projeto prático surjam dezenas de ideias referentes à sua melhoria. Na realidade, parece que quanto mais se desenvolve, mais melhorias podem ser feitas para aprimorar o sistema.

Assim, ao final deste trabalho, o sentimento é de que o campo é vasto e de que ainda há muito o que se explorar no sentido de aproximar a plataforma criada de um sistema comercializável ou que possa ser adaptado à indústria médica, principalmente no que concerne a busca por uma sempre maior estabilidade do algoritmo em cenários reais. Ao mesmo tempo, pôde-se perceber sem sombra de dúvidas o potencial que um sistema como o desenvolvido possui para auxiliar no processo de reabilitação não apenas de pacientes que sofreram AVC, mas também de tantos outros casos de deficiência motora, além de abrir diversas portas para melhorias no treinamento esportivo.

Acerca do *hardware*, a conclusão foi que as IMUs escolhidas, apesar de serem um excelente aparato de aquisição, fáceis de usar e de terem uma documentação bastante razoável, foram por vezes causa de frustração durante os testes. Diversas vezes ocorreu da sua comunicação com o *dongle* WiFi falhar por um ou dois pacotes, o que acarretava em um travamento do processo de estimação e consequente arruinamento do conjunto de dados sendo obtidos. Isso possivelmente poderia ser corrigido por código, e na realidade foi tratado internamente em uma das rotinas de controle da plataforma, mas a possibilidade de investigar mais a fundo maneiras de driblar essas perdas de pacote é bastante convidativa.

No contexto de melhora da plataforma de *software*, a seguir estão detalhados alguns dos vários pontos que poderiam se beneficiar de alguns ajustes:

- Como comentado em capítulos anteriores, seria desejável que o usuário pudesse selecionar que grau de liberdade monitorar com cada sensor. Apesar das IMUs estarem previstas para possuir uma posição fixa no corpo do paciente, ele poderia ainda optar por designar à IMU alocada no pulso, por exemplo, o monitoramento do ângulo de guinada (*pitch*), de giro (*roll*) ou de desvio (*yaw*) do antebraço. Essa opção está selecionada em código, mas seria interessante que o usuário pudesse alterá-la através do menu.

- Seria interessante também, no processo de gravação de movimentos, não limitar o usuário a uma janela de tempo pré-estabelecida, mas deixá-lo livre para gravar por

quanto tempo fosse necessário. Outra alternativa é perguntar antes do início da gravação por quantos segundos ele desejaria executar os movimentos e então usar essa janela para a gravação;

- Outro ponto interessante seria não restringir cada estimação a uma gravação apenas, e permitir acrescentar múltiplas gravações de um determinado movimento a um modelo já criado, de forma a aprimorá-lo com o tempo. Isso, no entanto, levaria em conta um módulo de memória não-volátil para o sistema, o que ainda não existe na plataforma atual;

- Em relação ao *feedback* em tempo real, que envolve mostrar o número do componente/ fase do movimento na tela da interface durante a estimação, as duas sugestões principais são as seguintes:

    1. Oferecer ao usuário, durante o processo de rotulação e parametrização, a possibilidade de nomear os componentes do movimento. Dessa forma, seria possível mostrar na tela não "1", "2 " e "3" mas "Relaxado", "Flexão" e "Extensão", o que é mais intuitivo para a análise;
    2. Através dessa entrada de texto, possibilitar também a nomeação do movimento global, de forma a implementar formalmente o passo de Segmentação e, para uma dada combinação de estados (componentes) estimados, mostrar na tela também o nome completo do movimento. Esse foi o objetivo inicial do campo "Novo Movimento", que aparece no menu para a etapa de segmentação manual e rotulação. Por uma questão de tempo e prioridade de tarefas, sua funcionalidade não foi implementada, mas não falta adicionar muito ao código para que passe a estar funcional;

- Tornar o menu mais resistente a erros do usuário. Da maneira como ele está implementado, existe uma boa segurança para a grande maioria dos botões, mas ainda há brechas para, caso o usuário decida utilizar alguns deles em momentos inapropriados, derrubar o fluxo de operação do sistema e travar a aplicação;

- Uma opção bastante atraente que quis-se implementar mas para a qual não houve tempo (e principalmente porque não era uma prioridade) é a de animar novamente o gráfico durante o processo de estimação, mostrando em tempo real não só as estimações dos estados como também da posição;

- Da maneira como a interface foi referenciada ao *driver* das IMUs, e por conta da tara utilizada, foi necessário definir uma posição inicial para o modelo do braço. Para facilitar o desenvolvimento, a posição inicial foi definida como sendo a plena extensão do membro, posicionado na horizontal à frente do corpo e com a palma da mão voltada para cima. Em um contexto de reabilitação, e especialmente em um

contexto de pacientes com espasticidade, muitas vezes não é nem mesmo possível performar essa posição para dar início à fisioterapia. Assim, sugere-se que seja feita uma adaptação no sistema de definição da posição inicial para conformar os casos mais críticos;

- Melhorar o formato do arquivo .txt de saída, que ainda está bastante bruto. Além do mais, algo extremamente simples de se implementar é a impressão, também nesse arquivo, dos tempos de cada fase do movimento e talvez da amplitude máxima de cada curva, de forma a facilitar uma posterior análise;

- Uma possibilidade interessante para tornar o *feedback* temporal mais fácil seria também a de reverter a decisão tomada no início do desenvolvimento do projeto e utilizar a base das medidas no tempo ao invés da frequência. Se esse fosse o caso desde o começo, não teria havido a necessidade de ajustar, por exemplo, os parâmetros de velocidade para módulos do sistema que operam em frequências distintas;

- Outro ponto que foi discutido no texto foi a possibilidade de corrigir o problema de visualização das medidas a uma baixa taxa de atualização do filtro de Kalman, uma vez que nesse caso o filtro também é passado por elas. A solução seria delegar à Interface (ou ao *driver*, o que seria ainda melhor) a responsabilidade de construir o vetor de medidas do movimento, sem condicioná-lo então às taxas do modelo SLDS.

- Por fim, sugere-se implementar de dentro da aplicação um método para derrubar o servidor local ao término do uso da plataforma. No momento, o desligamento do serviço ainda tem ocorrido via terminal e de maneira externa ao sistema.

A fim de permitir a continuação desse projeto e de possibilitar a implementação de tantas dessas melhorias quanto for possível, além de outras mais, uma das conclusões a que se chegou foi que seria interessante disponibilizar todo o código da aplicação no GitHub. De fato, toda a aplicação está versionada por lá na organização "lara-unb" sob o nome de "Interface_SLDS". Dessa forma, as portas ficam abertas para outros desenvolvedores que se sentirem atraídos pela ideia de continuar este trabalho ou de utilizá-lo como inspiração para seus próprios projetos.

A respeito disso, tem-se um comentário a fazer: apesar do foco deste projeto ter sido dado primordialmente aos resultados gerados pelo sistema, percebeu-se, durante a sua implementação, um grande potencial presente na lógica por trás do modelo SLDS para o âmbito educacional. Esse é um algoritmo que exemplifica vários conceitos do processamento de dados voltado para estimação e estudar o seu código bem como as saídas geradas por ele é uma excelente maneira de aprender mais sobre o tema.

Em relação aos potenciais trabalho futuros que podem ser derivados deste projeto, visualiza-se pelo menos algumas possibilidades: a primeira delas seria, como comentado

previamente em outras seções, a de fechar, com o *feedback* gerado pelo sistema, malhas de eletro-estimulação que auxiliassem o paciente a executar movimentos de braço. Nesse contexto, o cenário de órteses inteligentes poderia se beneficiar imensamente. Poder embarcar algoritmos como o de SLDS-Viterbi aproximado em tecnologias desse tipo permitiria, por exemplo, monitorar em tempo real a atividade do membro e então auxiliar, apenas na medida necessária, os movimentos realizados. Além de otimizar a terapia e o exercício muscular nesse caso, essa possibilidade cumpriria com um dos maiores paradigmas das tecnologias de reabilitação, que envolve a criação de um sistema que, quando conectado ao membro lesado do usuário, devolve a ele sua funcionalidade original.

Outros caminhos possíveis envolvem aplicações na indústria esportiva, que sempre possui uma grande demanda por tecnologias que permitam avaliar de forma acurada o movimento e a técnica dos atletas, ou mesmo na indústria do entretenimento, em aplicações como desenvolvimento de jogos com reconhecimento de gestos ou interação homem-máquina com realidades virtuais.

É ainda possível, em um nível mais baixo, realizar expansões da plataforma desenvolvida neste projeto para incluir modelos de outros membros do corpo como tronco e pernas. Além disso, o foco da estimação neste trabalho foi na análise repetitiva de um mesmo movimento. Nesse panorama, e apesar do modelo SLDS suportar uma tarefa desse cunho, não se buscou implementar um classificador de movimentos que soubesse distinguir, dentre várias poses corporais distintas, aquela executada em um determinado instante. Isso de fato é possível através da inclusão de ambos os movimentos no mesmo processo de gravação, mas não foi o foco do projeto. Essa também seria uma abordagem interessante do sistema, que com certeza traria outras possibilidades de aplicação.

Enfim, acredita-se ter, através deste trabalho, desenvolvido uma plataforma simples e eficiente para estimação de movimentos em membro superior, capaz de apresentar *feedbacks* úteis tanto para profissionais quanto para pacientes, direta ou indiretamente através de outras possíveis aplicações.

Ao fim de todo este processo de desenvolvimento, e ao enxergar todo o crescimento obtido, aliado a todas as possibilidades futuras deste campo da Engenharia, não se pode senão ponderar sobre o papel de cada indivíduo no avanço da ciência. Nesse âmbito talvez não haja definição melhor do que aquela dada por Madre Teresa de Calcutá:

"Sei que meu trabalho é uma gota no oceano mas, sem ele, o oceano seria menor".

Referências

1 PINHEIRO, M. O que é espasticidade. Acesso em 06/11/2017. Disponível em: <https://www.tuasaude.com/espasticidade/>. Citado 2 vezes nas páginas 15 e 29.

2 JINTRONIX. Jintronix: melhorando a reabilitação física através de tecnologias de captura de movimento. Disponível em: <http://www.jintronix.com/pt/kinect-rehabilitation-software/>. Citado 5 vezes nas páginas 15, 16, 30, 84 e 85.

3 SWORD HEALTH. Sword health home page. Disponível em: <https://www.swordhealth.com/#/our-solution>. Citado 4 vezes nas páginas 15, 16, 30 e 84.

4 GOBBATO, B. B. Exercícios para reabilitação do ombro. Acesso em 06/11/2017. Disponível em: <http://www.ombroecotovelo.net/videos.html>. Citado 2 vezes nas páginas 15 e 37.

5 THE PIXEL ISSUE. Quantic dream opens new motion capture studio. 2010. Acesso em 08/11/2017. Disponível em: <https://www.pixel-issue.net/quantic-dream-opens-new-motion-capture-studio/>. Citado 2 vezes nas páginas 15 e 42.

6 BT. Motion capture's most embarassing pictures. Acesso em 08/11/2017. Disponível em: <http://home.bt.com/pictures/film-features/motion-captures-most-embarrassing-photos-41363843095816>. Citado 2 vezes nas páginas 15 e 43.

7 RETRO GAMES BRASIL. Kinect microsoft xbox 360 semi novo. Acesso em 08/11/2017. Disponível em: <http://www.retrogamesbrasil.com.br/produto/kinect-microsoft-xbox-360-semi-novo/>. Citado 2 vezes nas páginas 15 e 43.

8 GUEDIM, Z. Leap motion vr hand sensors will replace all controllers. 2017. Acesso em 08/11/2017. Disponível em: <https://edgylabs.com/leap-motion-hand-vr>. Citado 2 vezes nas páginas 15 e 43.

9 SHEIKH, Y. A. Yaser ajmal sheikh web page. 2017. Acesso em 08/11/2017. Disponível em: <http://www.cs.cmu.edu/~yaser/>. Citado 2 vezes nas páginas 15 e 43.

10 YOST LABS. *User's Manual*: 3-space sensor: Miniature attitude and heading reference system. PortsMouth, Ohio, 2014. Citado 6 vezes nas páginas 15, 16, 45, 46, 89 e 90.

11 THALMIC LABS. Myo armband home page. Disponível em: <https://www.myo.com/smartglasses>. Citado 2 vezes nas páginas 15 e 45.

12 YOST LABS. Yost labs 3-space mocap studio. 2016. Acesso em 09/11/2017. Disponível em: <https://yostlabs.com/yost-labs-3-space-mocap-studio/>. Citado 2 vezes nas páginas 15 e 50.

13 MILES, B. Low poly three.js mountain scene. Acesso em 09/11/2017. Disponível em: <https://codepen.io/benjaminmiles/pen/MwNvzE>. Citado 2 vezes nas páginas 15 e 52.

14 DELTAKOSH. [babylon.js]–the train demo. 2013. Acesso em 09/11/2017. Disponível em: <https://www.eternalcoding.com/?p=193>. Citado 2 vezes nas páginas 15 e 52.

15 BAPTISTA, R. de S.; Bó, A. P. L.; HAYASHIBE, M. Automatic human movement assessment with switching linear dynamic system: Motion segmentation and motor performance. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, v. 25, n. 6, p. 628–640, June 2017. ISSN 1534-4320. Citado 17 vezes nas páginas 15, 19, 41, 42, 44, 55, 58, 59, 60, 62, 64, 70, 81, 82, 83, 101 e 105.

16 FERREIRA, R. da R. Sword arya: como alguns sensores e uma aplicação podem transformar a fisioterapia. 2016. Acesso em 22/10/2017. Disponível em: <https://www.futurebehind.com/sword-health-sword-arya-fisioterapia/>. Citado 2 vezes nas páginas 16 e 84.

17 LAFAY, M. How to stabilize your drone camera with a gimbal. Acesso em 11/11/2017. Disponível em: <http://www.dummies.com/consumer-electronics/drones/how-to-stabilize-your-drone-camera-with-a-gimbal/>. Citado 2 vezes nas páginas 16 e 95.

18 POPOVIC, D.; SKINJAER, T. *Control of Movement for the Physically Disabled*. [S.l.]: Springer-Verlag London Ltd., 2000. ISBN 978-1-4471-0433-9. Citado na página 27.

19 FILHO, R. K. do V. M. Abordagem ao avc. Acesso em 06/11/2017. Disponível em: <http://www.cremesp.org.br/pdfs/eventos/eve_29052015_154311_Abordagem%20obre%20AVC%20-%20Dr.%20Rui%20Kleber%20do%20Vale%20Martins%20Filho.pdf>. Citado na página 28.

20 POPULAçãO deve ficar atenta aos riscos do AVC. 2012. Acesso em 06/11/2017. Disponível em: <http://www.brasil.gov.br/saude/2012/10/no-dia-mundial-do-avc-pais-alerta-populacao-contra-a-doenca>. Citado na página 28.

21 WORLD HEALTH ORGANIZATION. Stroke, cerebrovascular accident. Acesso em 06/11/2017. Disponível em: <http://www.emro.who.int/health-topics/stroke-cerebrovascular-accident/index.html>. Citado na página 28.

22 BARROS, A. C. et al. Abordagem inicial e conduta no acidente vascular encefálico isquêmico agudo. 2009. Citado na página 28.

23 FREITAS, G. D. de. Reabilitação neurofuncional em um paciente com hemiplegia espástica como seqüela de avc. estudo de caso. 2011. Acesso em 06/11/2017. Disponível em: <http://www.efdeportes.com/efd155/reabilitacao-neurofuncional-em-um-paciente-com-hemiplegia.htm>. Citado na página 28.

24 REDEBRASILAVC. Acompanhamento após o avc. Acesso em 06/11/2017. Disponível em: <http://www.redebrasilavc.org.br/para-pacientes-e-falimiares/acompanhamento-apos-o-avc/>. Citado na página 29.

25 SILVA, E. de Jesus Alves da. Reabilitação após o avc. 2010. Citado na página 29.

26 VOLL PILATES GROUP. Guia definitivo: Tratamento fisioterapêutico do avc. 2017. Acesso em 06/11/2017. Disponível em: <http://blogfisioterapia.com.br/tratamento-do-avc/>. Citado na página 29.

27 KERR, K. et al. Standardization and definitions of the sit-stand-sit movement cycle. *Gait and Posture*, v. 2, n. 3, p. 182 – 190, 1994. ISSN 0966-6362. Disponível em: <http://www.sciencedirect.com/science/article/pii/096663629490006X>. Citado na página 39.

28 WIKIPEDIA. Motion capture. 2017. Acesso em 08/11/2017. Disponível em: <https://en.wikipedia.org/wiki/Motion_capture#Applications>. Citado 2 vezes nas páginas 42 e 44.

29 VICON. Vicon home page. Disponível em: <https://www.vicon.com>. Citado na página 42.

30 QUALISYS. Qualisys home page. Disponível em: <https://www.qualisys.com/applications/>. Citado na página 42.

31 NICOLAOU, A. The 9 best languages for crunching data. 2014. Acesso em 09/11/2017. Disponível em: <https://www.fastcompany.com/3030716/the-9-best-languages-for-crunching-data>. Citado 2 vezes nas páginas 47 e 48.

32 DEZYRE. Python for data science vs python for web development. 2015. Acesso em 09/11/2017. Disponível em: <https://www.dezyre.com/article/python-for-data-science-vs-python-for-web-development/177>. Citado na página 47.

33 SHAIKH, S. Why python is important for big data and analytics applications? 2016. Acesso em 09/11/2017. Disponível em: <https://www.eduonix.com/blog/bigdata-and-hadoop/python-important-big-data-analytics-applications/>. Citado na página 48.

34 MEIRINHO, J. Infográfico – o sentido da visão: a importância das imagens na comunicação online. 2014. Acesso em 09/11/2017. Disponível em: <http://www.iinterativa.com.br/infografico-sentido-da-visao-importancia-das-imagens-na-comunicacao-online/>. Citado na página 49.

35 VIEIRA, D. Javascript é a linguagem de programação mais usada no mundo; veja ranking. 2017. Acesso em 09/11/2017. Disponível em: <https://www.tecmundo.com.br/software/119217-javascript-linguagens-populares-momento-veja.htm>. Citado na página 52.

36 HEWITSON, J. Three.js and babylon.js: a comparison of webgl frameworks. 2013. Acesso em 09/11/2017. Disponível em: <https://www.sitepoint.com/three-js-babylon-js-comparison-webgl-frameworks/>. Citado na página 52.

37 FRANCESCO, H. D. In simple terms: backend code, frontend code and how they interact. 2017. Acesso em 10/11/2017. Disponível em: <https://hackernoon.com/in-simple-terms-backend-code-frontend-code-and-how-they-interact-2485c5a1bbd2>. Citado na página 54.

38 NGINX. What is http? 2017. Acesso em 10/11/2017. Disponível em: <https://www.nginx.com/resources/glossary/http/>. Citado na página 54.

39 SALARIAN, A. et al. itug, a sensitive and reliable measure of mobility. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, v. 18, n. 3, p. 303–310, June 2010. ISSN 1534-4320. Citado 2 vezes nas páginas 81 e 82.

40 AL-JAWAD, A. et al. Using multi-dimensional dynamic time warping for tug test instrumentation with inertial sensors. In: *2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. [S.l.: s.n.], 2012. p. 212–218. Citado na página 82.

41 DIOS, P. F. de; CHUNG, P. W. H.; MENG, Q. Landmark-based methods for temporal alignment of human motions. *IEEE Computational Intelligence Magazine*, v. 9, n. 2, p. 29–37, May 2014. ISSN 1556-603X. Citado na página 82.

42 LIN, J. F. S.; KULIć, D. Online segmentation of human motion for automated rehabilitation exercise analysis. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, v. 22, n. 1, p. 168–180, Jan 2014. ISSN 1534-4320. Citado na página 82.

43 COGNIFIT. Cognifit home page. Disponível em: <https://www.cognifit.com>. Citado na página 84.

44 COGNIFIT. Scientific validation: where everything starts: peer reviewed publications. Disponível em: <https://www.cognifit.com/neuroscience>. Citado na página 85.

45 SUBLIME TEXT. Sublime text home page. Disponível em: <https://www.sublimetext.com>. Citado na página 89.

46 YOST LABS. *YEI 3-Space Python API*: Api documentation. PortsMouth, Ohio, 2012. Citado na página 89.

47 THREEJS. *three.js*: docs. [S.l.]. Disponível em: <https://threejs.org/docs/>. Citado na página 93.

48 HOSOLMAZ. Rotating coordinate system via a quaternion. 2016. Acesso em 21/08/2017. Disponível em: <https://stackoverflow.com/questions/4870393/rotating-coordinate-system-via-a-quaternion/42180896#42180896>. Citado na página 95.

49 KOCHARYAN, A. Send json data with jquery. 2013. Acesso em 24/08/2017. Disponível em: <https://stackoverflow.com/questions/6587221/send-json-data-with-jquery>. Citado na página 96.

50 COBURN, J. How to get python and javascript to communicate using json. 2017. Acesso em 24/08/2017. Disponível em: <http://www.makeuseof.com/tag/python-javascript-communicate-json/>. Citado 2 vezes nas páginas 96 e 97.