

UnB - UNIVERSIDADE DE BRASÍLIA
FGA - FACULDADE UNB GAMA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA
BIOMÉDICA

**IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO
PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA**

Roberto Aguiar Lima

ORIENTADORA: Dra. Lourdes Mattos Brasil
COORIENTADORA: Dra. Vera Regina Da Silva Marães

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA BIOMÉDICA

PUBLICAÇÃO: NUMERAÇÃO / 2015
BRASÍLIA/DF : Setembro - 2015

UnB - UNIVERSIDADE DE BRASÍLIA
FGA - FACULDADE UNB GAMA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA
BIOMÉDICA

**IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO
PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA**

Roberto Aguiar Lima

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA DA FACULDADE GAMA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA BIOMÉDICA.

APROVADO POR:

Prof. Dra. Lourdes Mattos Brasil
(Orientadora)

Prof. Dra. Vera Regina Da Silva Marães
(Coorientadora)

Prof. Dra. Aline Araujo do Carmo
(Examinador Externo)

Prof. Dr. Jairo Simão Santana Melo
(Examinador Externo)

BRASÍLIA/DF , 30 DE Setembro DE 2015

FICHA CATALOGRÁFICA

Roberto Aguiar Lima

IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA, [Distrito Federal] 2015.

NUMERAÇÃO . 101 p., 210 x 297 mm (FGA/UnB Gama, Mestre, Engenharia Biomédica, 2015). Dissertação de Mestrado - Universidade de Brasília. Faculdade Gama. Programa de Pós- Graduação em Engenharia Biomédica.

1. Análise de Marcha. 2. Software como Serviço

3. Sistemas Inteligentes. 4. Simulação

I. FGA UnB Gama/ UnB. II. IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA

REFERÊNCIA BIBLIOGRÁFICA

LIMA, R. A. (2015). TÍTULO. Dissertação de Mestrado em Engenharia Biomédica, Publicação 38A/2015, Programa de Pós-Graduação em Engenharia Biomédica, Faculdade Gama, Universidade de Brasília, Brasília, DF, 101 p.

CESSÃO DE DIREITOS

AUTOR: Roberto Aguiar Lima

TÍTULO: IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA

GRAU: Mestre

ANO: 2015

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

2015.

Av. Central, Bl. 191, N. 10, N. Bandeirante.

CEP: 71710010, Brasília, DF – Brasil

DEDICATÓRIA

Para todos os programadores e engenheiros, que como eu amam o que fazem. Que consigam escapar das garras das atividades burocráticas administrativas que a sociedade brasileira nos obriga a encontrar. Que encontrem a coragem e a inteligência necessárias a ativar a energia inovadora dentro de cada uma delas, energia esta, que no final se traduz em produtos e serviços que vão ajudar milhões de pessoas.

AGRADECIMENTOS

Agradeço a professora Dra. Lourdes Mattos Brasil, que sem dúvida é a pessoa mais paciente e persistente que encontrei na vida, tenho certeza que sem sua ajuda e estes atributos maravilhosos este trabalho não chegaria ao fim.

Também agradeço a professora Dra. Vera Regina Da Silva Marães, que foi minha coorientadora e também me ajudou no desenvolvimento do software me descrevendo funcionalidades interessantes para serem implementadas.

Ao meu colega João Paulo Martins, que em nossas discussões, sempre fazia surgir alguma ideia ou despertar algum interesse no nosso campo de estudo.

Ao meu colega Daniel Souza Braga, que me ajudou muito nesta reta final com várias dicas na parte escrita do trabalho.

Aos meus pais, não existem palavras de gratidão suficientes para eles. Sem eles do meu lado, principalmente na atual fase da minha vida, não sei o que teria acontecido comigo.

Também agradeço a Deus, que por muito tempo não O havia procurado, mas no devido tempo veio minha conversão.

A verdadeira miséria é a eterna insatisfação.

Dalai Lama

RESUMO

IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA

Autor: Roberto Aguiar Lima

Orientadora: Profa. Dra. Lourdes Mattos Brasil

Coorientadora: Dra. Vera Regina Da Silva Marães

Programa de Pós-Graduação em Engenharia Biomédica

BRASÍLIA/DF 2015

O presente trabalho tem como objetivo implementar um software como serviço, para análise e simulação de marcha humana, baseado num modelo arquitetural em camadas. A grande vantagem de tal software é sua disponibilidade via *web* e até mesmo em dispositivos móveis. Com esta disponibilidade e o uso crescente do software, surge a possibilidade da geração de uma base de dados com dados de marcha humana. O sistema ainda conta com um módulo de simulação, que tirará proveito desta base. A análise de movimento baseada em dados espaciais, recuperadas de um software de *motion capture*, foi implementada e simulação de sinais usando a rede neural artificial CMAC também. O projeto é *open source* e funcionalidades novas serão adicionadas frequentemente.

Palavras-chaves: Análise de Marcha, Software como Serviço, Sistemas Inteligentes, Simulação.

ABSTRACT

IMPLEMENTING A SOFTWARE AS A SERVICE FOR HUMAN GAIT ANALYSIS AND SIMULATION

Author: Roberto Aguiar Lima

Supervisor: Prof. Dra. Lourdes Mattos Brasil

Co-supervisor: Dra. Vera Regina Da Silva Marães

Post-Graduation Program in Biomedical Engineering

Brasília, September of 2015.

This work has as objective a software as a service implementation, for human gait analysis and simulation, based in a layered architectural model. The software great advantage is his availability at web and mobile devices. With this possibility and the software use, a human gait data base creation will be possible. The software has a simulation module, that will use this data base as data source. Movement analysis based at spacial data, recovered from a motion capture software, was implemented and signals simulations using the artificial neural network CMAC, too. The project is open source and new features will be often implemented.

Key-words: Gait Analysis, Software as a Service, Inteligent Systems, Simulation.

SUMÁRIO

1	INTRODUÇÃO	16
1.1	CONTEXTUALIZAÇÃO E FORMULAÇÃO DO PROBLEMA	16
1.2	OBJETIVOS	17
1.2.1	Objetivo Geral	17
1.2.2	Objetivo Específicos	17
1.3	REVISÃO DA LITERATURA	17
1.4	ORGANIZAÇÃO DO TRABALHO	19
2	FUNDAMENTAÇÃO TEÓRICA	21
2.1	ANÁLISE DE MARCHA	21
2.1.1	BREVE HISTÓRICO	21
2.1.2	FUNDAMENTOS	22
2.1.3	MÉTODOS DE COLETA DE DADOS PARA ANÁLISE	23
2.2	MÉTODOS ÁGEIS	27
2.2.1	SCRUM	30
2.2.2	HISTÓRIAS DO USUÁRIO	32
2.3	SOFTWARE COMO SERVIÇO	33
2.4	COMPONENTES, FRAMEWORKS E FERRAMENTAS	35
2.4.1	APLICAÇÕES WEB COM ANGULAJS	35
2.4.2	ORGANIZAÇÃO DE PROJETOS WEB COM ANGULAR-SEED	36
2.4.3	SOLUÇÃO DE DESIGN WEB COM ANGULAR-MATERIAL	37
2.4.4	GRÁFICOS E ANIMAÇÕES 3D NA WEB COM THREEJS	38
2.4.5	SERVIÇOS REST COM FLASK	39
2.4.6	SERVIÇO DE BASE DE DOCUMENTOS COM MONGODB	40
2.5	CMAC	43
2.5.1	TREINAMENTO DA CMAC	46
2.6	ALGUNS CÁLCULOS CINEMÁTICOS	47
2.7	SIMULAÇÃO	49
3	METODOLOGIA	50
3.1	AMBIENTE DE ESTUDO	50
3.2	DELIMITAÇÃO DO ESTUDO	52
3.3	VISÃO	52
3.4	MODELO DE GESTÃO	53
3.5	MODELO DE ARQUITETURA	54
3.5.1	CAMADA DE APLICAÇÃO WEB	54

3.5.2	CAMADA <i>REST WEB API</i>	57
3.5.3	CAMADA DE BASE DE DOCUMENTOS	60
4	RESULTADOS	62
4.1	Módulo de Análise	62
4.2	Módulo de Simulação	73
4.3	Uso do sistema em dispositivos móveis	76
5	DISCUSSÃO E CONCLUSÃO	80
6	TRABALHOS FUTUROS	82
	REFERÊNCIAS BIBLIOGRÁFICAS	84
	APÊNDICES	89
	APÊNDICE A – ARTIGO COMPLETO WC2015	90
	APÊNDICE B – RESUMO PAHCE 2014	95
	ANEXOS	97
	ANEXO A – PROCESSO NO COMITÊ DE ÉTICA	98
	ANEXO B – LICENÇA MIT	100

LISTA DE TABELAS

Tabela 1 – Pesquisa por palavras-chaves	18
Tabela 2 – Comparando Modelos de Desenvolvimento de Software	29
Tabela 3 – Mapeamento de $s1$	45
Tabela 4 – Mapeamento de $s2$	45
Tabela 5 – Mapeamento para os pesos W	45

LISTA DE FIGURAS

Figura 1 – Atleta iniciando uma corrida. Fonte: Muybridge (1885).	21
Figura 2 – Divisões do Ciclo de Marcha. Fonte: Perry e BurnField (2010).	22
Figura 3 – Câmera Oqus MRI. Fonte: Qualisys (2013b).	24
Figura 4 – Visão do QTM. Fonte: Qualisys (2010).	24
Figura 5 – Configuração de câmeras para captura de dados de marcadores passivos fixados no paciente. Fonte: Qualisys (2013a).	25
Figura 6 – Experimento de captura de dados de <i>IMU</i> em conjunto com uma câmera. Fonte: Leite et al. (2014).	26
Figura 7 – Comparação entre <i>IMU</i> e a câmera. Fonte: Leite et al. (2014).	26
Figura 8 – Eletrogoniômetro. Fonte: Ibrahim et al. (2012).	27
Figura 9 – Valores em comum. Fonte: Alterado de Greene e Stellman (2014).	28
Figura 10 – Exemplo de processo baseado no modelo <i>waterfall</i> . Fonte: Alterado de Pressman e Maxim (2014).	30
Figura 11 – Visão Geral do <i>Scrum</i> . Fonte: Alterado de Schwaber (2004).	31
Figura 12 – Exemplo de <i>story card</i> . Fonte: Alterado de Cohn (2004).	33
Figura 13 – Visões arquiteturais de software como serviço. Fonte: Fox e Patterson (2012).	35
Figura 14 – Diagrama de uma aplicação <i>MVC</i> usando <i>AngularJS</i> .	36
Figura 15 – Layout original de um projeto baseado em <i>angular-seed</i> . Fonte: Google (2015b).	37
Figura 16 – Exemplo de aplicação que utiliza <i>angular-material</i> . Fonte: Google (2015a).	38
Figura 17 – Exemplo de aplicação que utiliza <i>ThreeJS</i> .	39
Figura 18 – <i>Web API REST</i> . Fonte: Masse (2011).	40
Figura 19 – Organização dos dados no <i>MongoDB</i> . Fonte: Plugge, Membrey e Hows (2014).	41
Figura 20 – Listagem de um documento. Fonte: Dayley (2014).	42
Figura 21 – Definindo documentos normalizado com <i>MongoDB</i> . Fonte: Dayley (2014).	42
Figura 22 – CMAC para controle de uma junta. Fonte: Alterado de Albus (1975a).	43
Figura 23 – Quantização de s_1 .	44
Figura 24 – Exemplo hipotético de um ângulo. Fonte: Alterado de Cliparts (2015).	48
Figura 25 – Modelo matemático simplificado de um processo. Fonte: Garcia (2009).	49
Figura 26 – Rede LIS.	50
Figura 27 – Processo de coleta de dados.	51
Figura 28 – Dados disponibilizados pelo QTM.	52
Figura 29 – Processo de desenvolvimento.	53
Figura 30 – Camadas arquiteturais.	54

Figura 31 – Exemplo de uma tela criada com <i>angular-material</i>	55
Figura 32 – Camada <i>web</i>	56
Figura 33 – Exemplo de componentes da camada <i>web</i> funcionando juntos.	57
Figura 34 – Organização do arquivos e diretórios da camada <i>web API</i>	57
Figura 35 – Camada <i>REST WEB API</i>	58
Figura 36 – Exemplo de requisição sendo tratada pela <i>web api</i>	59
Figura 37 – Banco de documentos da aplicação.	60
Figura 38 – Documento da coleção <i>patients</i>	61
Figura 39 – Documento da coleção <i>positionals_data</i>	61
Figura 40 – Tela de seleção dos módulos.	62
Figura 41 – Tela com a listagem de pacientes.	63
Figura 42 – Informações do paciente.	64
Figura 43 – Tela inicial da dados coletados do paciente.	65
Figura 44 – Inclusão de amostra de marcha	65
Figura 45 – Seleção do arquivo no formato <i>MATLAB</i> proveniente do <i>QTM</i>	66
Figura 46 – Dados do arquivo provenientes do <i>QTM</i>	66
Figura 47 – Animação dos marcadores em 3D.	67
Figura 48 – Controle de perspectivas.	67
Figura 49 – Controle de <i>zoom</i>	68
Figura 50 – Controle <i>pan</i>	68
Figura 51 – Controles da animação.	68
Figura 52 – Opção <i>markers</i>	69
Figura 53 – Progressão espacial de um marcador.	69
Figura 54 – Seleção de um marcador pelo <i>mouse</i>	70
Figura 55 – Renomeando um marcador.	70
Figura 56 – Animação mostrando o marcador renomeado.	71
Figura 57 – Opção de visualização e criação de ângulos.	71
Figura 58 – Inclusão de um novo ângulo.	72
Figura 59 – Ângulo de um joelho durante o ciclo de marcha.	72
Figura 60 – Velocidades angulares de um joelho durante o ciclo de marcha.	72
Figura 61 – Módulo de simulação.	73
Figura 62 – Seleção do ciclo de marcha.	74
Figura 63 – Sinais de entrada para a <i>CMAC</i> . No caso posições num plano 3D de marcadores.	74
Figura 64 – Sinais de entrada para a <i>CMAC</i> , ângulos e velocidades angulares.	75
Figura 65 – Exemplo de configuração para uma simulação usando <i>CMAC</i>	75
Figura 66 – Resultado da simulação.	76
Figura 67 – Erro quadrado médio em cada iteração da simulação.	76

Figura 68 – Aplicação <i>Open Gait Analytics</i> rodando num <i>browser Firefox</i> e dispositivo <i>Motorola Xoom</i>	77
Figura 69 – Animação de uma amostra de marcha rodando num <i>browser Firefox</i> e dispositivo <i>Motorola Xoom</i>	77
Figura 70 – Aplicação rodando num <i>iPhone 4</i> com <i>browser Safari</i>	78
Figura 71 – Adaptação da aplicação em telas pequenas.	78

LISTA DE ABREVIATURAS E SIGLAS

ACM DL	<i>Association from Computer Machinery Digital Library]</i>
API	<i>Application Program Interface</i>
CAPES	<i>Coordenação de Aperfeiçoamento de Pessoal de Nível Superior</i>
CPD	Centro de Processamento de Dados
BSON	<i>Binary Object Notation</i>
CSS	<i>Cascade Style Sheet</i>
FCE	Faculdade Ceilândia
FGA	Faculdade Gama
HTML	<i>HiperText Markup Language</i>
HTTP	<i>HiperText Transfer Protocol</i>
GNU	GNU is Not Unix
GUGT	<i>Get Up and Go Test</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
IMU	<i>Inertial Measurement Unit</i>
JSON	<i>JavaScript Object Notation</i>
LIS	Laboratório de Informática em Saúde
LPH	Laboratório de Performance Humana
MIT	<i>Massachusetts Institute of Technology</i>
MLP	<i>Multi Layer Perceptron</i>
MOCAP	<i>Motion Capture</i>
MVC	<i>Model View Controller</i>
PCA	<i>Principal Component Analysis</i>
QTM	<i>Qualisys Track Manager</i>
RNA	Rede Neural Artificial

REST	<i>Representational State Transfer</i>
SPA	<i>Single-Page Applications</i>
SVM	<i>Support Vector Machine</i>
UnB	Universidade de Brasília
URL	<i>Uniform Resource Locator</i>
VM	<i>Virtual Machine</i>

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E FORMULAÇÃO DO PROBLEMA

Quando o autor deparou-se com um problema na área de análise de marcha, ele seguiu toda uma metodologia para resolver este problema. Ao ler artigos desta área, notou que muitas das atividades que ele realizou, provavelmente foram realizadas por aqueles autores também. Na época o autor estava desenvolvendo um software simulador de um joelho, que poderia ser implantado num sistema embarcado para controle de uma possível prótese transfemural ativa. A metodologia usada naquele trabalho pode ser resumida da seguinte forma:

1. Coletar dados de movimentos usando marcadores passivos fixados no corpo de um paciente. Os dados eram coletados por uma série de câmeras;
2. Usar o software *Qualisys Track Manager (QTM)* (QUALISYS, 2010) para converter os dados para o formato *MATLAB*, assim era possível manipulá-los;
3. Criar um programa para extrair os dados em formatos mais amigáveis à sua manipulação;
4. Fazer cálculos de ângulos, velocidades angulares, posicionamentos dos marcadores, velocidades dos marcadores;
5. Fazer gráficos de toda essa informação. Lembrando, tudo isso usando *MATLAB*;
6. Persistir toda essa informação para uso futuro;
7. Criar um algoritmo complexo, que usa certos dados como entrada e algum outro como saída;
8. Executar várias simulações alterando vários parâmetros até achar uma combinação de parâmetros que simule o sinal de uma forma desejada.

Várias das etapas acima, poderiam ser desenvolvidas num software com interface gráfica, facilitando e muito o trabalho do pesquisador da área de análise de marcha. Por exemplo, o novo software receberia o arquivo do *QTM*, criaria todos os dados de movimentos citados automaticamente colocando-os numa base de dados e permitiria imprimir gráficos de todos eles.

Este foi o contexto inicial, que impulsionou o desenvolvimento deste trabalho, mas além deste problema notou-se um potencial a mais, o *QTM* que é o software de captura

de dados. Ele é muito bom mas é de uso genérico. Para utilizá-lo como software de análise de marcha, há um trabalho grande a ser feito. O indício disto é que a maioria dos pesquisadores com que o pesquisador teve contato no laboratório, usavam o mesmo processo, coletavam com o *QTM* e processavam com o *MATLAB*. Geralmente, a outra opção é usar softwares de análise de marcha específicos como o software *Kin Trak* e *Ortho Trak*, descritos em Moraes, Silva e Battistela (2003). Além disso todos, esses softwares, inclusive o *QTM*, são softwares *desktop*, que possuem licenças caríssimas, o que limita o uso dos dados pelos pesquisadores, que tem que ir ao laboratório ou ter um computador com uma licença válida. Daí surgiu outra oportunidade, isto é, criar o novo software como um serviço na *web*, que pode ir evoluindo ao longo do tempo, ou seja, recebendo adições de funcionalidades constantemente, até que seja bom o suficiente para ser usado por qualquer profissional de saúde no globo. Para que a evolução seja constante, optou-se por adotar metodologias ágeis que privilegiem a mudança contínua e o incremento iterativo de funcionalidades.

Com o poder de processamento dos dispositivos móveis de hoje, a nova aplicação também pode resolver problemas de mobilidade, pois pode disponibilizar dados de pacientes onde e quando o profissional de saúde quiser.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O presente trabalho visa iniciar um projeto de desenvolvimento de software como serviço para análise e simulação de marcha humana.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Explicitar uma visão arquitetural inicial do software;
- Escolher componentes de software a serem usados na solução;
- Selecionar um conjunto de funcionalidades suficientes para implementação de uma *release* funcional e de testes para estressarem a arquitetura proposta.

1.3 REVISÃO DA LITERATURA

Foram usados os seguinte serviços *web* para o levantamento bibliográfico deste trabalho:

- *Institute of Electrical and Electronics Engineers (IEEE) Xplore Digital Library* (<http://ieeexplore.ieee.org/>);

- *PubMed* (<http://www.ncbi.nlm.nih.gov/pubmed>);
- Portal de Periódicos CAPES/MEC (<http://periodicos.capes.gov.br>).
- *ACM DL - Association for Computer Machinery Digital Library* (<http://dl.acm.org>).

Todos estes serviços são de grande renome e amplamente utilizados por pesquisadores brasileiros. O *IEEE Xplore* é um dos mais importantes acervos do mundo e para engenheiros de qualquer área, é completamente obrigatório. O *PubMed* é do governo americano e simplesmente não tem concorrente na área de saúde. Já o ACM DL é a principal fonte de pesquisadores na área de tecnologia de informação. O portal de periódicos CAPES é um programa do governo brasileiro que auxilia os acadêmicos do país, bem como ele é um indexador de vários outros portais, nos quais os acadêmicos podem baixar os artigos.

Foram utilizadas as palavras-chaves descritas na Tabela 1 para o desenvolvimento científico deste trabalho em estudo.

Tabela 1 – Pesquisa por palavras-chaves

<i>Palavras-Chave</i>	<i>IEEE</i>	<i>PubMed</i>	<i>ACM</i>	<i>CAPES</i>
Gait Analysis Software	52	203	45	286
Gait Analysis + Software as a Service	0	0	2	0
Gait Analysis + Software Web	0	5	2417832	0
CMAC	775	149	1119	821

Os artigos considerados mais relevantes para o trabalho foram escolhidos, levando-se em consideração, entre outros tópicos, a descrição de características interessantes a serem implementadas no software a ser desenvolvido.

Quando o assunto se trata de análise de marcha, a obra mais aclamada, inclusive citada em muitas das referências pesquisadas, é Perry e BurnField (2010). Como sugerido por Malas (2010), esta é uma obra obrigatória a qualquer um que deseje estudar análise de marcha.

Em Vieira et al. (2015) um sistema de análise e classificação de marcha é proposto como alternativa a soluções de mercado mais caras. A proposta inicial é coletar dados a partir de marcadores posicionados no corpo do paciente, através de câmeras de vídeo, classificando padrões de marcha com aprendizado de máquina.

Em Duhamel et al. (2004) é apresentada uma ferramenta para melhorar a confiabilidade de curvas para um paciente, classificar pacientes em determinadas populações e comparar populações. Trata-se de uma ferramenta estatística para análise de marcha.

Detecções de eventos do ciclo de marcha, são características interessantes para um software de análise de marcha. Em Ghoussayni et al. (2004) são documentados métodos para detecção de 4 eventos: contato do calcanhar, elevação do calcanhar, contato do dedão do pé e elevação do dedão do pé.

Uma comparação entre dois pacotes distintos para análise de marchar foi realizada por Moraes, Silva e Battistela (2003). Neste trabalho dados captados por câmeras e plataformas de força são coletados e passados aos pacotes de software *Kin Trak* e *Ortho Trak*.

Uma amostra de como um software pode ser utilizado para gerar bases de dados de análise de marcha, é visto em Moreno et al. (2009). Neste artigo os autores capturam dados de crianças sadias, afim de obterem padrões para serem utilizados em sistemas de análise de movimentos.

Um sistema de aquisição e análise de marcha, foi desenvolvido e demonstrado em Ferreira, Crisostomo e Coimbra (2009). Neste trabalho, o hardware para captura de dados e o software para análise dos dados, foram desenvolvidos num único projeto. Com os resultados gerados pelas análises feitas por este projeto, foi possível construir um robô bípede, que apresentou resultados satisfatórios caminhando num ciclo de marcha confortável.

A partir da análise de marcha, é possível criar métodos para se estabelecer o grau de desvio do ciclo de marcha que um paciente pode apresentar. Em Beynon et al. (2010) é apresentado o método *Gait Profile Score*. O método em si é um bom candidato a funcionalidade em um software de análise de marcha, pois serviria de auxílio clínico ao profissional da área de saúde. Uma outra funcionalidade inspirada no campo clínico é mostrado em Cippitelli et al. (2015). Neste trabalho, os autores propõem a automatização do método *Get Up and Go Test* (GUGT), que é usualmente utilizado em análise de marcha no campo da reabilitação.

1.4 ORGANIZAÇÃO DO TRABALHO

O Capítulo 1 introduz o trabalho, que trata sobre as motivações que levaram a este projeto e explicita os objetivos do trabalho.

O Capítulo 2 faz um apanhado geral de toda teoria e tecnologias necessárias para a construção do software proposto.

O Capítulo 3 descreve as metodologias adotadas na construção do software, coleta de dados e o modelo arquitetural do mesmo.

O Capítulo 4 mostra os resultados obtidos, com foco no usuário final, no caso pesquisadores e profissionais da área de saúde.

O Capítulo 5 conclui e mostra as visões do autor acerca da obra.

O Capítulo 6 lista os trabalhos futuros que podem ser realizados a partir do produzido até aqui pelo projeto.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ANÁLISE DE MARCHA

2.1.1 BREVE HISTÓRICO

Conforme Baker (2007), Aristóteles (384-322 A.C.) pode ser considerado o primeiro a registrar comentários a respeito de como os humanos caminham. O autor ainda afirma que só na renascença que houveram progressos através de experimentos e teorizações feitas por Giovanni Borelli (1608-1679), e também que Jules Etienne Marey (1830-1904), trabalhando na França e Eadweard Muybridge (1830-1904), trabalhando na América, fizeram grandes avanços na área de mensuração. Ainda conforme Baker (2007), os maiores avanços no início do século vinte foram os desenvolvimentos das placas de força e o entendimento da cinética da marcha.

Na obra de Muybridge (1885), de antes do século vinte, ele busca sistematizar maneiras de se analisar o movimento humano, principalmente usando técnicas de fotografia. A obra apesar de ser o resultado das pesquisas do autor, tem um valor artístico inegável. A Figura 1 dá o tom da obra.

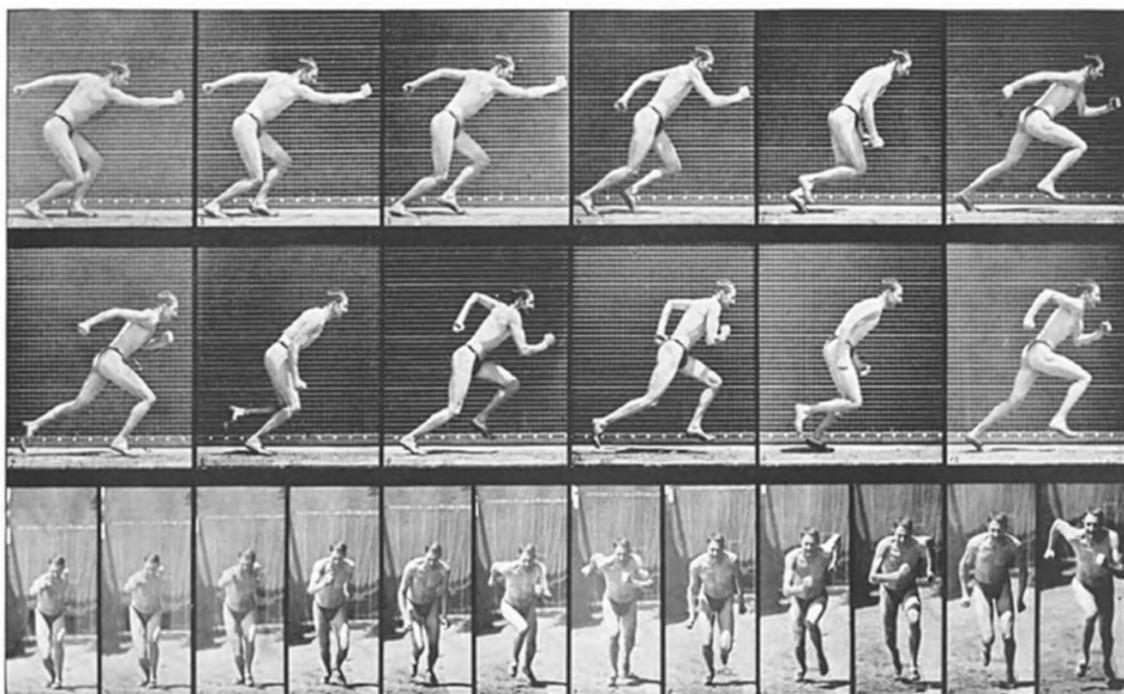


Figura 1 – Atleta iniciando uma corrida. Fonte: Muybridge (1885).

Apesar dos avanços ocorridos na análise de marchar até meados do meio do século vinte, foi só após o advento dos computadores modernos que a análise de marcha clínica tornou-se amplamente disponível (BAKER, 2007).

2.1.2 FUNDAMENTOS

Segundo Perry e BurnField (2010), para se classificar as diferentes divisões da marcha é necessário separá-las em períodos (*Periods*), fases (*Phases*) e tarefas (*Tasks*), conforme a Figura 2.

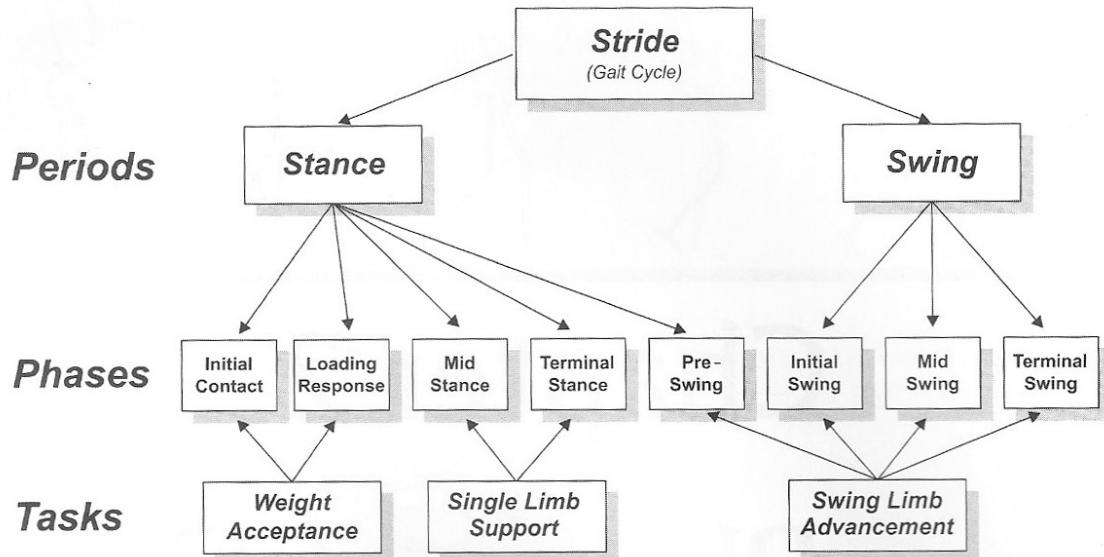


Figura 2 – Divisões do Ciclo de Marcha. Fonte: Perry e BurnField (2010).

A *stride*, que também é um sinônimo para um ciclo de marcha completo, equivale ao momento que, por exemplo, o pé direito toca o chão, sai do chão e o toca novamente. Dentro do ciclo de marcha temos também os períodos que são dois, *stance* e *swing*. O *stance* corresponde ao período que o pé toca o chão pela primeira vez e o deixa durante o ciclo. O *swing* é o período que o mesmo pé deixa o chão e o toca novamente iniciando uma nova *stance* (PERRY; BURNFIELD, 2010).

Além dos períodos, como vemos na Figura 2, temos as fases. As fases representam um intervalo, percentual durante o ciclo da marcha e são muito importantes na avaliação do ciclo. Pois grandes variações nos sinais durante alguma fase, pode representar algum distúrbio a ser diagnosticado. São oito as fases (PERRY; BURNFIELD, 2010):

1. *Initial Contact*, corresponde de 0 a 2% do ciclo de marcha;
2. *Loading Response*, corresponde de 2% até 12% do ciclo de marcha;
3. *Mid Stance*, corresponde de 12% até 31% do ciclo de marcha;
4. *Terminal Stance*, corresponde de 31% até 50% do ciclo de marcha;
5. *Pre-Swing*, corresponde de 50% ate 62% do ciclo de marcha;
6. *Initial Swing* corresponde de 62% até 75% do ciclo de marcha;

7. *Mid Swing* corresponde de 75% até 87% do ciclo de marcha;
8. *Terminal Swing* corresponde de 87% até 100% do ciclo de marcha.

As tarefas são as funções desempenhadas durante o ciclo de marchar e estão relacionadas especificamente com as fases. A Figura 2 mostra o relacionamento entre as três tarefas e suas fases específicas. São elas (PERRY; BURNFIELD, 2010):

1. *Weight Acceptance* - Ela é responsável pela absorção do choque, estabilidade inicial de membro e preservação da progressão;
2. *Single Limb Support* - Esta tarefa é responsável por manter um membro apoiado no chão e progredindo o ciclo de marcha, até que o outro membro toque o chão, ou seja, sua função, é fazer que um membro suporte todo o peso do corpo sozinho;
3. *Swing Limb Advancement* - Sua função é avançar o membro que está no período de *swing*, até que ele esteja pronto para fazer um *Initial Contact*.

2.1.3 MÉTODOS DE COLETA DE DADOS PARA ANÁLISE

Esta seção descreve alguns dos dispositivos que podem ser usados para coletar dados de movimentos. Estes podem ser convertidos e usados pelo software desenvolvido, desde que uma adaptação seja feita. No momento de conclusão deste trabalho o único método adaptado é o por captura de dados por câmeras, usando o sistema da *Qualisys*.

Captura de dados por câmeras

Pode-se começar este assunto pelos métodos de mensuração de movimentos espaciais e de ângulos. O método mais sofisticado hoje para análise de movimentos é o baseado em câmaras de vídeo. Inclusive Grip e Häger (2013), demonstra discorrer sobre as vantagens dos sistemas de câmeras ópticas usando marcadores de superfície em detrimento de outras técnicas de captura de movimento. Neste tipo de técnica, os marcadores podem ser ativos ou passivos, a diferença é que os ativos emitem algum tipo de sinal luminoso ou infravermelho, por exemplo. Este método permite visualizar a posição espacial dos marcadores e a partir daí, calcular velocidades, acelerações, ângulos, velocidades angulares, acelerações angulares, etc. A Figura 3 mostra o modelo *Oqus MRI* da *Qualisys*. Esta é uma câmera muito utilizada no mercado não só para análise clínica mas também para captura de movimentos para serem inseridos em filmes e jogos de computador. Já a Figura 4 mostra o software QTM do mesmo fabricante, já com os dados capturados e animados na tela do computador. A Figura 5 é uma visão de uma possível configuração de câmeras, capturando marcadores de superfície passivos de um paciente.



Figura 3 – Câmera Oqus MRI. Fonte: Qualisys (2013b).

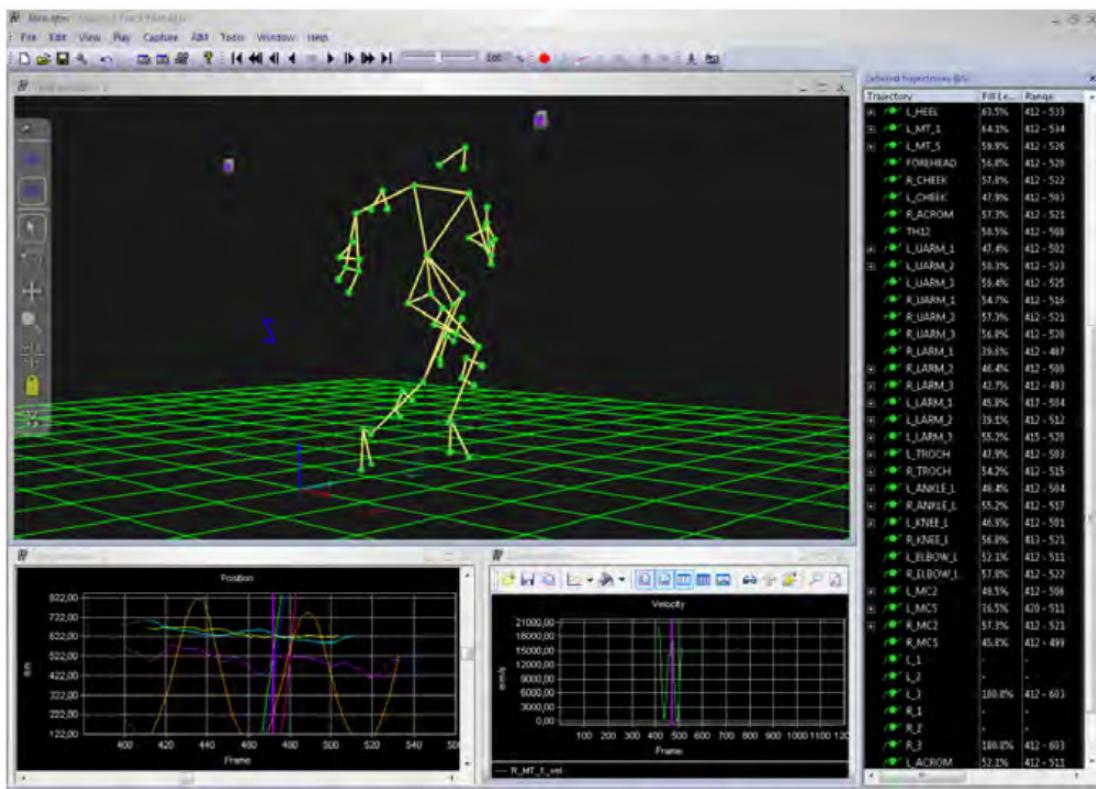


Figura 4 – Visão do QTM. Fonte: Qualisys (2010).

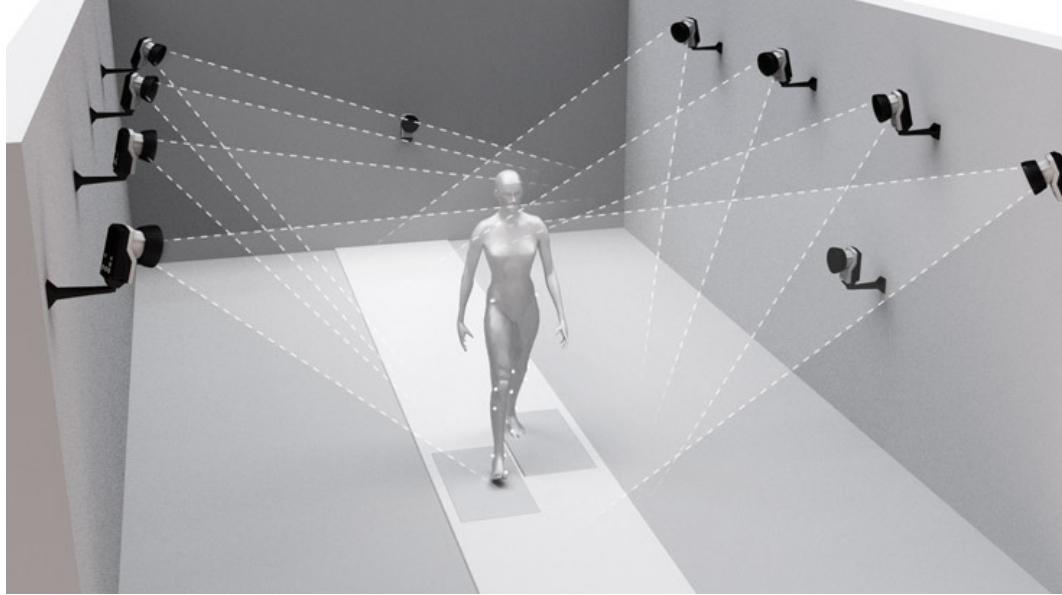


Figura 5 – Configuração de câmeras para captura de dados de marcadores passivos fixados no paciente. Fonte: Qualisys (2013a).

Captura por Unidade de Medida Inercial

Uma outra alternativa, que está sendo desenvolvida na FGA/UnB, é um dispositivo baseado em uma Unidade de Medida Inercial (*Inertial Measurement Unit - IMU*). O IMU é um dispositivo eletrônico provido de acelerômetro e um magnetômetro. Segundo Leite et al. (2014), esta é uma alternativa não visual para extrair parâmetros cinemáticos da marcha humana, trajetória e velocidade. O trabalho foi realizado comparando-se os resultados fornecidos pelo dispositivo e captura de vídeo. A Figura 6 mostra um experimento onde o vídeo e os dados do *IMU* são coletados ao mesmo tempo.

Na Figura 7, é possível visualizar o resultado dos dados coletados do *IMU* e da câmera. Veja que é um resultado bem promissor. Mas a maior vantagem do dispositivo, ainda não foi discutida, seu baixíssimo valor em relação a solução com várias câmeras.

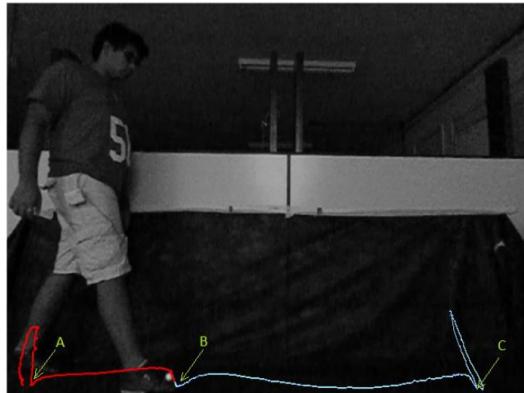


Figura 6 – Experimento de captura de dados de *IMU* em conjunto com uma câmera.
Fonte: Leite et al. (2014).

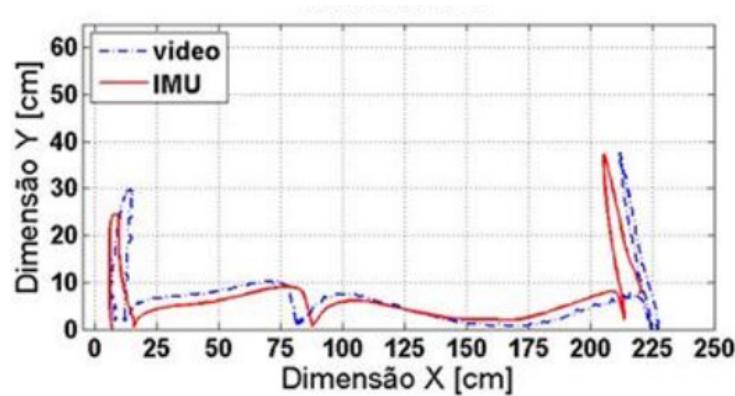


Figura 7 – Comparação entre *IMU* e a câmera. Fonte: Leite et al. (2014).

Captura por Eletrogoniômetros

Eletrogoniômetros são também usados para capturar angulações em articulações, basicamente este aparelho provê uma voltagem que é representativa da mudança de ângulo entre duas superfícies, nas quais o dispositivo é fixado (IBRAHIM et al., 2012). A principal vantagem deste dispositivo é que seu custo é baixo. A Figura 8 mostra um exemplo de eletrogoniômetro. Ele tem como desvantagem sua pouca precisão, mas ainda assim pode ser usado na análise de movimentos por software.



Figura 8 – Eletrogoniômetro. Fonte: Ibrahim et al. (2012).

2.2 MÉTODOS ÁGEIS

A muito tempo vários métodos para desenvolvimento de software são propostos. Em 2001, um grupo de pessoas muito experientes em desenvolvimento de software, juntaram-se em Salt Lake City, Utah, para resolverem problemas de desenvolvimento de software (GREENE; STELLMAN, 2014). Como resultado deste encontro, foi criado o manifesto ágil, que é reproduzido a seguir (BECK et al., 2001):

Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:

Indivíduos e interações mais que processos e ferramentas;

Software em funcionamento mais que documentação abrangente;

Colaboração com o cliente mais que negociação de contratos;

Responder a mudanças mais que seguir um plano.

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Meses após a criação do manifesto, estas pessoas também criaram os princípios ágeis e a Aliança Ágil (LAYTON, 2012).

Os princípios ágeis são um conjunto de 12 itens com o objetivo de auxiliar na implantação de metodologias ágeis. Eles são reproduzidos a seguir a título de ilustração (BECK et al., 2001):

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.

2. Mudanças nos requisitos são bem-vindas, mesmo tardivamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade—a arte de maximizar a quantidade de trabalho não realizado—é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Este manifesto serviu de marco agregador de métodos e técnicas, que já existiam a época, mas não eram amplamente difundidas como *Scrum*, *Extreme Programming*, *kanban*, *lean*, entre outros. Estas técnicas, apesar de anteriores ao manifesto, possuem em seus cernes, muito em comum com os valores ágeis. A Figura 9 tenta ilustrar esta ideia.

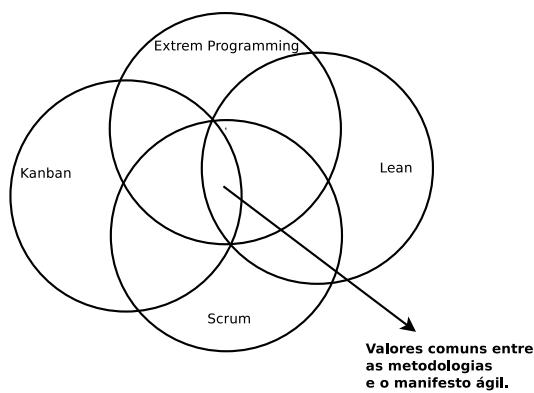


Figura 9 – Valores em comum. Fonte: Alterado de Greene e Stellman (2014).

A adoção dos métodos ágeis hoje é praticamente unanimidade. Isto se deve aos modelos de desenvolvimento adotados até a década de 1990. Esses modelos eram na sua grande maioria baseados no modelo *waterfall*, que imitava uma linha de produção onde o software era desenvolvido em fases. A saída de cada fase era a entrada da próxima. A

Figura 10 mostra um exemplo de processo baseado no modelo *waterfall*. O maior problema do modelo *waterfall*, era que este foi completamente averso a mudanças de requisitos. Hoje, é notório que a grande maioria dos softwares necessitam ser bastante receptivos a mudanças.

O trabalho de Runyan e Ashmore (2014) compara o modelo *waterfall* com o modelo ágil. Esta comparação é resumidamente apresentada na Tabela 2.

Tabela 2 – Comparando Modelos de Desenvolvimento de Software

<i>Waterfall</i>	<i>Ágil</i>
Prescritivo	Abstrato
Documentação extensiva	Mínimo de documentação
Sequencial	Contínuo
Formal	Informal
Foco no processo	Foco na comunicação
Mudança gradual	Mudança rápida

Fonte: Runyan e Ashmore (2014)

Como evolução do modelo *waterfall*, surgiram os processos baseados em modelos iterativos. A diferença agora é que o processo de desenvolvimento passa a ser baseado em ciclos. Cada ciclo passa por cada uma das fases do *waterfall*. Este tipo de processo, apresentava melhorias, mas ainda era concebido sob a forma de um processo que visava resolver problemas determinísticos, como o das linhas de produção das fábricas. Mas, como descrito em Beck (2004), o processo de se construir software é mais parecido com o ato de dirigir. O motorista sabe o destino a que quer chegar, porém, durante o percurso pode haver um acidente e tem-se que mudar um pouco a rota, ou alguém está passando por uma faixa de pedestre e necessita-se parar, ou ainda um sinal de transito pode ficar vermelho. Esta é a principal motivação para que este trabalho privilegie métodos ágeis de desenvolvimento.

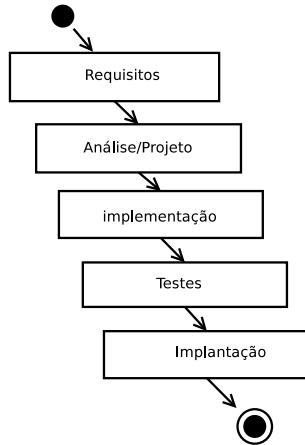


Figura 10 – Exemplo de processo baseado no modelo *waterfall*. Fonte: Alterado de Pressman e Maxim (2014).

2.2.1 SCRUM

O *Scrum*, segundo Rubin (2012), é um método ágil para desenvolvimento de produtos e serviços inovadores. A Figura 11 mostra uma visão geral do *Scrum*.

Basicamente, o fluxo do scrum consiste na criação de um *backlog* de produto. Este *backlog* é uma lista de requisitos a serem implementados no processo de desenvolvimento, e é mantido e priorizado pelo *product owner*. A lista em si pode receber contribuições dos mais diversos envolvidos no processo, mas a última palavra na priorização é sempre do *product owner*. A orientação mais aceita para a priorização dos requisitos, é levar em conta aqueles que mais gerarão valor para o usuário final, no momento em questão (SCHWABER; BEEDLE, 2001).

A próxima etapa no fluxo é a reunião de *sprint*. Esta reunião consiste na demonstração das funcionalidades implementadas na última iteração e na seleção das funcionalidades que serão construídas na próxima iteração. As funcionalidades são escolhidas pelos desenvolvedores, que se comprometem a entregá-las ao final da iteração (SCHWABER, 2004).

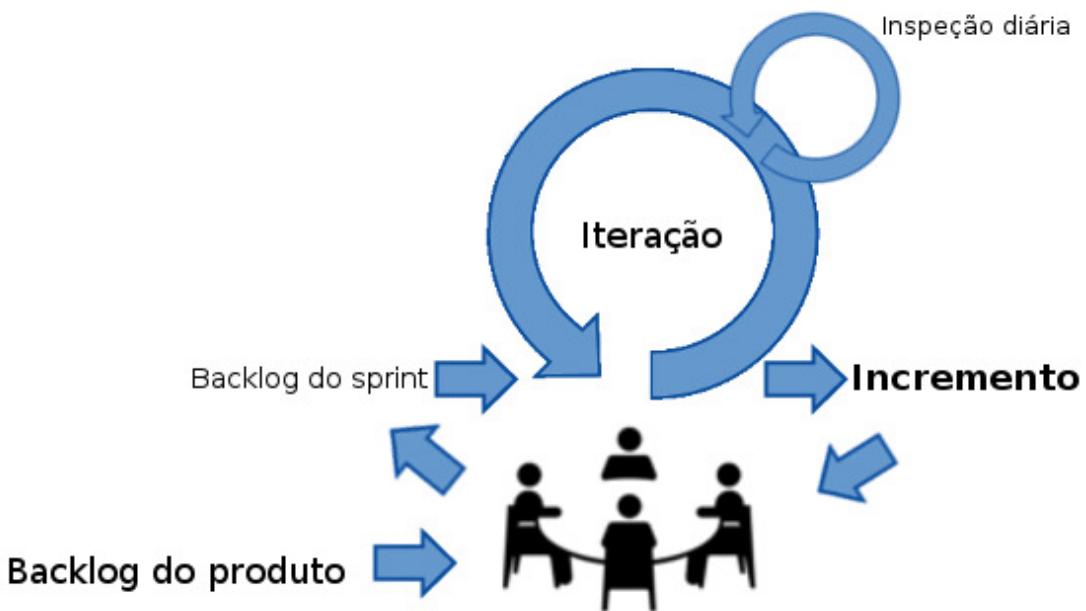


Figura 11 – Visão Geral do *Scrum*. Fonte: Alterado de Schwaber (2004).

Após a reunião do *sprint*, inicia-se a iteração. A iteração é também chamada de *sprint* e consiste num prazo de alguns dias ou alguns meses. Isto depende do projeto em questão. Geralmente *sprints* menores são mais aconselhados, por exemplo, uma semana (SCHWABER, 2004).

Durante a iteração ou *sprint*, ao final do dia preferencialmente, realiza-se uma reunião de inspeção diária. Esta reunião é também chamada de *daily scrum*. Nesta reunião, os envolvidos diretamente no desenvolvimento do produto ou serviço, expõe o andamento de suas atividades pessoais e os impedimentos para a conclusão destas. A reunião é feita com todos de pé, e cada um dispõe apenas de alguns minutos de exposição, por exemplo, 3 minutos. A intromissão de pessoas externas deve ser evitada. A ideia é manter a objetividade e o foco nas tarefas do *sprint*. Ao final da reunião o *scrum master* deve procurar entender todos os impedimentos expostos. A partir deste conhecimento ele deve tomar medidas necessárias para removê-los, assim garantindo o bom funcionamento da equipe de desenvolvimento. Ao final do *sprint* um incremento, ou seja, um pedaço de software com funcionalidades implementadas é construído e preferencialmente implantado. O próximo passo é uma nova reunião de *sprint*.

Os papéis definidos pelo *scrum* são (SCHWABER, 2004):

- *Product Owner*;
- *Scrum Master*;
- Desenvolvedor.

Nas organizações existirão outros papéis, como gerentes, diretores, patrocinadores, usuários finais, entre outros. O importante a se observar, é que apenas os três papéis, *product owner*, *scrum master* e desenvolvedor, é que fazem parte da equipe *scrum*. Os demais papéis participam do projeto, mas é necessário, o entendimento por parte da organização, que demandas de implementação devem ser colocadas no backlog do produto. Só o *product owner* pode priorizar os novos requisitos. Sem este entendimento e comprometimento, fica inviável respeitar prazos e escopo, tornando o processo caótico (SCHWABER, 2004).

Devido ao *Scrum* servir de espinha dorsal a um processo de desenvolvimento ágil, também sendo amplamente utilizado pelo mercado de desenvolvimento de software, ele foi selecionado como método de gestão deste trabalho. Todo o processo do *Scrum* pode ser minunciosamente estudado em Schwaber e Beedle (2001).

2.2.2 HISTÓRIAS DO USUÁRIO

Histórias do usuário é uma técnica que vem em detrimento da produção massiva de requisitos. Até hoje é comum achar projetos de desenvolvimento de software, com documentos de requisitos enormes. Com raras exceções, este não é um modelo muito produtivo. O grande problema é que muitas vezes gasta-se muito tempo, às vezes anos, para se criar estes documentos. Enquanto isso pouco ou nada de software funcional é produzido. Uma consequência comum desta demora, são as prioridades dos usuários mudarem, ou seja, o que foi especificado inicialmente, depois de um ano, por exemplo, já não tem o mesmo valor para os mesmos (PATTON, 2014).

De acordo com Cohn (2004) histórias do usuário descrevem funcionalidades que serão de valor para um usuário ou comprador de um sistema de software. Histórias do usuário são compostas por três aspectos:

- Uma descrição escrita da história usada para planejamento e como lembrete;
- Conversações sobre a história para detalhar a mesma;
- Testes que trazem e documentam detalhes e que podem ser usadas para determinar quando a história está completa.

Histórias do usuário podem ser documentadas usando-se *story cards*. Um exemplo é mostrado na Figura 12.

No *story card* em questão, a parte superior é a descrição do item que é de valor para um usuário. A parte inferior relata testes que devem ser executados. Estes testes também servem para determinar se a história do usuário foi completamente implementada ou não (COHN, 2004).

Dentro de uma metodologia como o *Scrum*, as histórias de usuário podem ser utilizadas para comporem o *backlog* do produto. Elas são excelentes durante a fase de planejamento do *sprint* (HANLEY, 2015).

Como as histórias do usuário são documentos mais informais, é necessário que as mesmas sejam mais detalhadas durante o processo de desenvolvimento. Isto pode ser feito com uma conversa entre o desenvolvedor e o *product owner*. Pode se chegar a conclusão que a história é na verdade um ”épico”, ou seja, é uma história composta de outras histórias. Neste caso a história dever ser quebrada e suas partes enviadas para planejamento novamente (COHN, 2004).

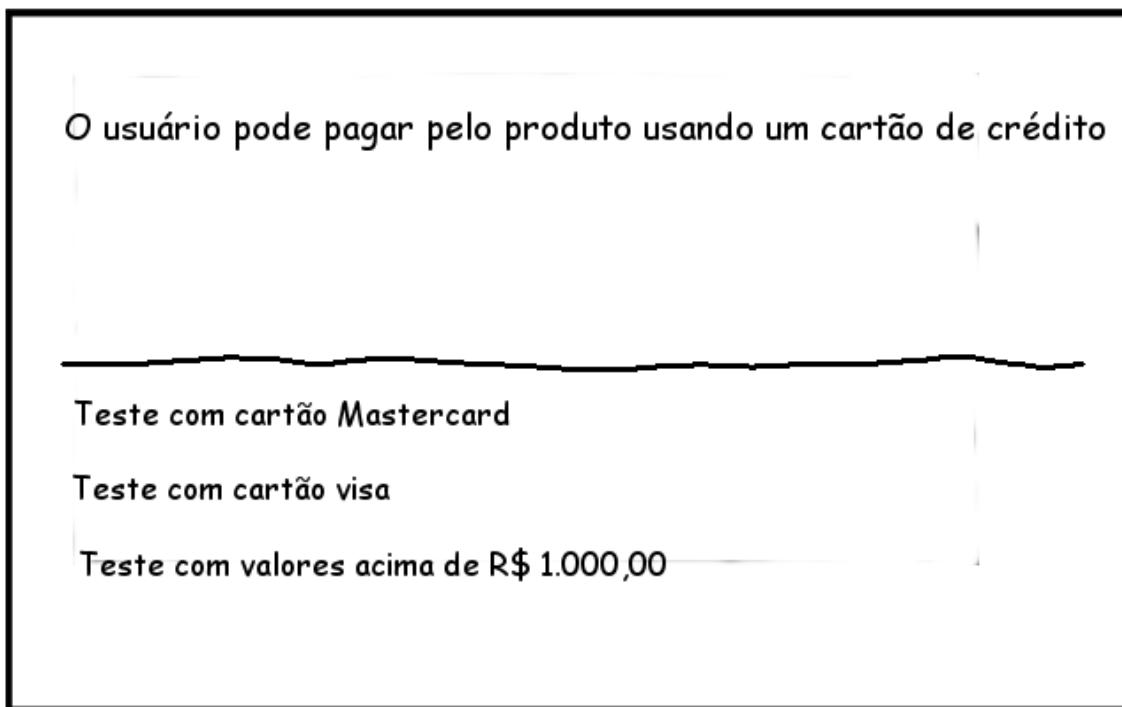


Figura 12 – Exemplo de *story card*. Fonte: Alterado de Cohn (2004).

2.3 SOFTWARE COMO SERVIÇO

A ideia de software como serviço é bastante difundida atualmente. Redes sociais, serviços de busca, serviços de *streaming* de vídeo, são amplamente usados por todos. Segundo Fox e Patterson (2012) software como serviço, é definido como o software que entrega software e dados como serviços sobre a *Internet*, usualmente via um programa como um *browser* que roda num dispositivo cliente local, em detrimento de código binário que precisa ser instalado e que roda totalmente no dispositivo.

Várias vantagens são citadas em Fox e Patterson (2012), tanto para usuários quanto para desenvolvedores de software. São elas:

1. Desde de que os usuários não necessitam instalar a aplicação, eles não precisam se preocupar se possuem um hardware específico, ou se possuem uma versão específica de sistema operacional;
2. Como os dados associados ao serviço geralmente são mantidos com o serviço, os usuários não precisam fazer *bakcups*, nem se preocupar em os perderem devido a mal funcionamento, ou até mesmo os perderem devido a extravios;
3. Quando um grupo de usuários querem coletivamente interagir sobre os mesmos dados, software como serviço é um veículo natural;
4. Faz mais sentido manter grandes quantidades de dados centralizados e manter o acesso remoto a estes;
5. Os desenvolvedores podem controlar as versões de software e sistema operacional que executam o serviço. Isto evita problemas de compatibilidade com distribuição do software devido ao grande número de plataformas que os usuários possuem. Além disso, é possível que o desenvolvedor teste novas versões do software usando uma pequena fração dos usuários temporariamente, sem causar distúrbios na maioria dos usuários;
6. Como os desenvolvedores controlam a versão de execução do software, eles podem mudar até mesmo a plataforma dos mesmos, desde que não violem as *Application Program Interfaces (APIs)* de interface com o usuário.

Para quem duvida do poder do software como serviço, é só observar que produtos consagrados como o *Microsoft Office* já possuem versão como serviço, no caso o *Microsoft Office 365*. Outros exemplos seriam o *Twitter*, *Facebook*, entre outros.

Segundo Fox e Patterson (2012), na visão de mais alto nível, de cima para baixo na figura 13, vê-se o software como um site na *web*. O cliente acessa o servidor via *internet* usando um *browser HTML*. Na segunda visão, vê-se uma aplicação *web* em três camadas, apresentação, lógica e persistência. A camada de apresentação serve páginas *HTML* para o usuário. Estas páginas consomem a camada lógica, também conhecida como camada de negócios, que são responsáveis pela lógica de negócio da aplicação. Por último vem a camada de persistência responsável pelo armazenamento e recuperação dos dados da aplicação. A próxima visão é uma extensão da segunda, nela detalha-se mais a camada lógica, que internamente funciona com uma arquitetura *model-view-controller*. Basicamente esta arquitetura define entidades de negócio (*model*), que são manipuladas por controladores (*controller*) e preparadas para apresentação pelo mecanismo de apresentação (*view*). A última camada corresponde as tecnologias e padrões de projetos adotados pela aplicação em si. Por exemplo, na figura seleciona-se o padrão de projeto

active record para implementação do *model*, usando o estilo arquitetural *REST* para implementação do *controller* e o padrão de projeto *template view* para a implementação do *view*. Estas escolhas são dependentes de projeto e geralmente são feitas por um arquiteto de *software* experiente.

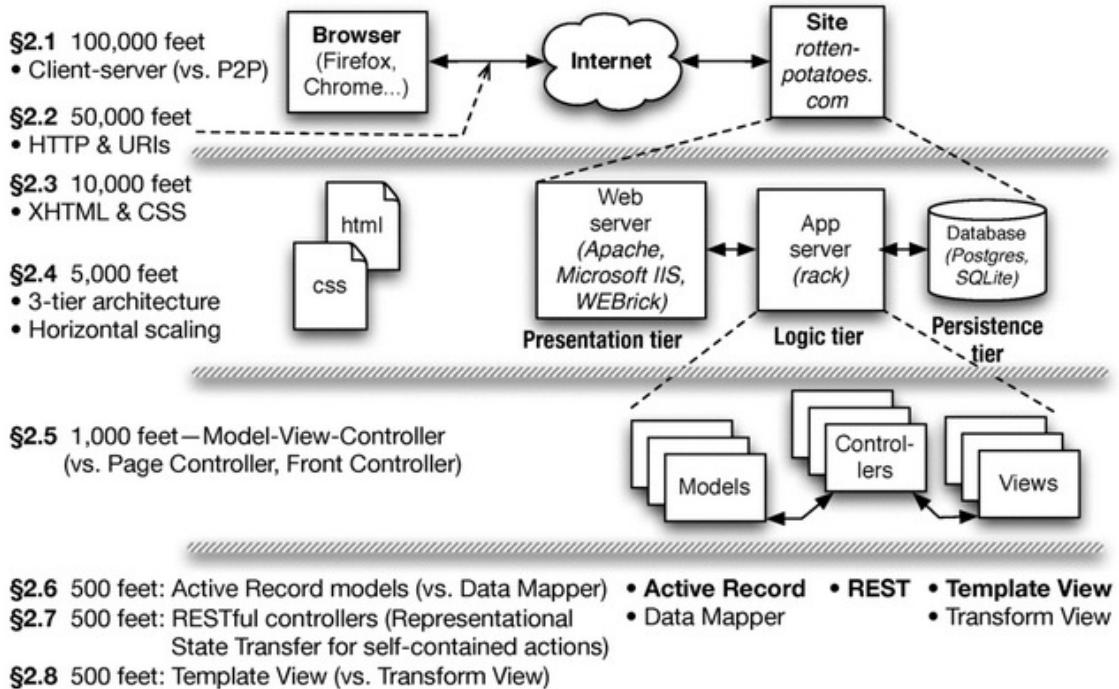


Figura 13 – Visões arquiteturais de software como serviço. Fonte: Fox e Patterson (2012).

2.4 COMPONENTES, FRAMEWORKS E FERRAMENTAS

Neste item serão analisados os componentes, *frameworks* e ferramentas que foram selecionadas para construção do software. Este software será um serviço disponibilizado via *web*, sendo assim, estas tecnologias são próprias para este fim.

2.4.1 APLICAÇÕES WEB COM ANGULAJS

O *AngularJS* é um *framework* de aplicação *web*. Ele foi projetado especificamente para rodar em *browsers* que suportam *HTML 5*. Ele também é adequado a criação de aplicações *web* que rodam em *smartphones*. É um *framework* bastante completo e rico para sua finalidade. A referência Branas (2014) é um guia introdutório conciso no assunto. Segundo Freeman (2014), outro guia no assunto, o *AngularJS* se baseia no padrão de projeto *Model-View-Controller (MVC)* e sua enfase é em permitir a criação de aplicações: extensíveis, manuteníveis, testáveis e padronizadas. A Figura 14 mostra uma representação de uma aplicação fazendo uso do *AngularJS*.

Segundo Williamson (2015), a Figura 14 mostra o diagrama de uma aplicação *AngularJS* e os componentes *MVC*. Uma vez que a aplicação é lançada, os componentes

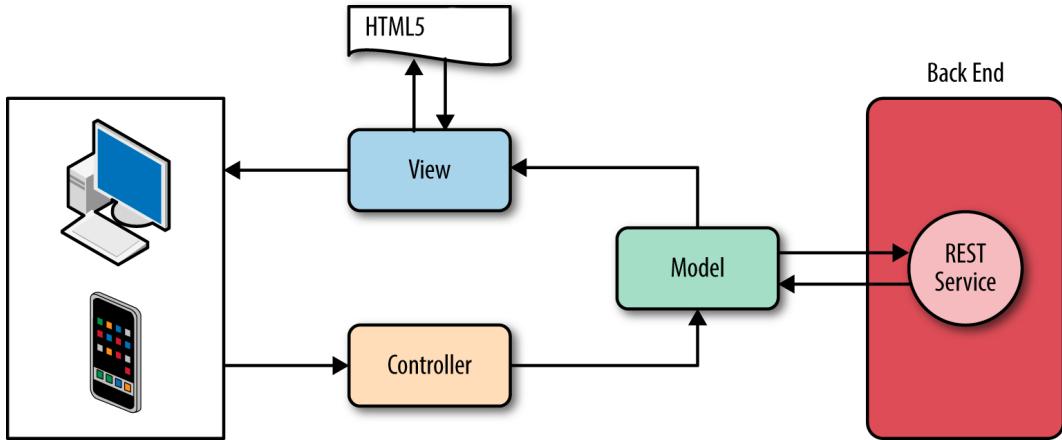


Figura 14 – Diagrama de uma aplicação *MVC* usando *AngularJS*.

Fonte: Williamson (2015).

model, *view* e *controller*, juntamente com todos os documentos *HTML* são carregados no *desktop* ou *smartphone* do usuário e rodam completamente num destes hardwares. A aplicação *AngularJS* conversa com o *backend* via o protocolo *http*. O *backend* é um servidor *web* que mantém chamadas *REST* (explicado na seção 2.4.5), sendo responsável pela execução da lógica e de processos de negócios. Outra característica bastante apreciada pelos desenvolvedores que usam *AngularJS*, é a possibilidade de criar *single-page applications* (*SPAs*). *SPAs* são aplicações que tem uma página *HTML* de entrada. Esta página tem seu conteúdo dinamicamente adicionado e removido da mesma. Esta abordagem permite criar aplicações bastante interativas, lembrando mesmo, aplicações *desktop* escritas em linguagens como *Visual Basic* e *Delphi*.

2.4.2 ORGANIZAÇÃO DE PROJETOS WEB COM ANGULAR-SEED

O *angular-seed*, é um *template* para projetos *web* que utilizam o *AngularJS*. Ele facilita bastante o processo de configuração e padronização do projeto. Ele cria um *layout* de diretórios padronizado, pré-configura as ferramentas de *build* e também já pré-configura o ambiente de testes unitários *web/javascript* usando a ferramenta *JASMINE*. Mais informações sobre o *angular-seed* em Google (2015b).

A Figura 15 mostra o *layout* inicial de diretórios que a ferramenta gera após ser clonada do *Github*.

Antes de começar a usar o *angular-seed* é necessário instalar o ambiente de execução *Javascript Node.JS*. Este ambiente é utilizado para execução de testes e construção de *builds*. Para uma introdução ao *Node.JS* veja Syed (2014).

```

app/          --> all of the source files for the application
  app.css      --> default stylesheet
  components/  --> all app specific modules
    version/   --> version related components
      version.js --> version module declaration and basic
"version" value service
  version_test.js    --> "version" value service tests
  version-directive.js --> custom directive that returns the current
app version
  version-directive_test.js --> version directive tests
  interpolate-filter.js    --> custom interpolation filter
  interpolate-filter_test.js --> interpolate filter tests
view1/
  view1.html        --> the view1 view template and logic
  view1.js          --> the partial template
  view1_test.js     --> tests of the controller
view2/
  view2.html        --> the view2 view template and logic
  view2.js          --> the partial template
  view2_test.js     --> tests of the controller
app.js         --> main application module
index.html     --> app layout file (the main html template file of the
app)
index-async.html --> just like index.html, but loads js files
asynchronously
karma.conf.js   --> config file for running unit tests with Karma
e2e-tests/
  protractor-conf.js --> Protractor config file
  scenarios.js      --> end-to-end scenarios to be run by Protractor

```

Figura 15 – Layout original de um projeto baseado em *angular-seed*. Fonte: Google (2015b).

2.4.3 SOLUÇÃO DE DESIGN WEB COM ANGULAR-MATERIAL

Os usuários profissionais da área de saúde, dificilmente se interessariam por um software complexo sem uma interface atraente e amigável. Uma solução para minimizar este problema é a adoção da biblioteca *angular-material*. Esta biblioteca, como indicado pelo seu nome, é construída com o *AngularJS*. Ela disponibiliza serviços e diretivas que podem ser usados para construir a interface gráfica da aplicação. Diretivas são componentes que podem ser inseridos diretamente no código HTML da aplicação, dando a aparência de estender a própria HTML. Por exemplo, a diretiva *md-button* da biblioteca é um tipo de botão que não é próprio do HTML. Outros exemplos de diretivas são: caixas de diálogos, barras de ferramentas, barras de progresso, grades, *tooltip*, etc. Esta biblioteca é baseada na especificação *Material Design* criada pela empresa *Google*. A especificação discorre sobre padrões de designe gráfico e interação com usuário e é baseada no princípio da metáfora de materiais. Esta metáfora é uma teoria unificada de um espaço racionalizado e sistemas de movimento, isto segundo Google (2015c). Outra vantagem da biblioteca é que ela é projetada para se adaptar a diferentes tipos de dispositivos com telas de tamanhos diferentes. Veja um exemplo de aplicação que usa *angular-material* na Figura 16.

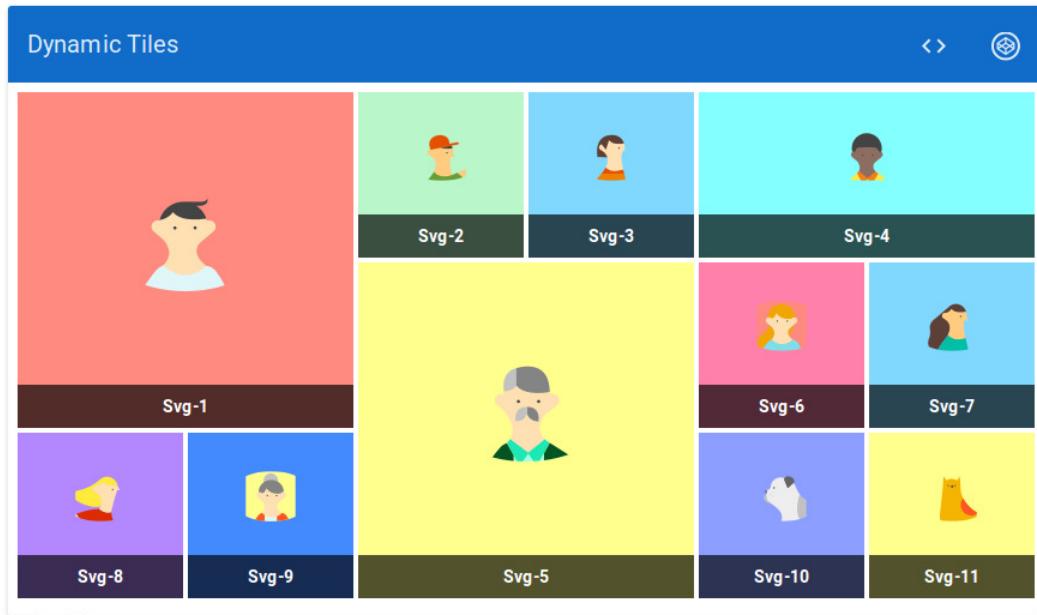


Figura 16 – Exemplo de aplicação que utiliza *angular-material*. Fonte: Google (2015a).

2.4.4 GRÁFICOS E ANIMAÇÕES 3D NA WEB COM THREEJS

Os *browsers* modernos hoje, inclusive os dos *smartphones*, suportam o novo padrão *WebGL*. Este é um padrão *web* multi-plataforma de *API* de baixo nível para gráficos 3D, expostos através de *HTML*. Este padrão suporta o acesso da *API* usando-se a linguagem *GLSL*. Uma vantagem desta *API* é que ela suporta nativamente *GPUs* disponibilizadas pelo hardware que está executando o cliente *web*. Isto torna possível até mesmo a criação de jogos de alta definição em 3D que rodam no *browser* (MATSUDA; LEA, 2013).

O problema com a *WebGL* é que, como dito anteriormente, a *API* é de baixo nível. No contexto de computação gráfica, isto significa que ela fornece primitivas básicas para modelagem 3D e outras opções de otimização do hardware. Para resolver este problema bibliotecas em *javascript* foram desenvolvidas, disponibilizando funções e objetos de alto nível, como cilindros, planos, esferas, animações, entre outros. Uma opção é o *ThreeJS*, descrito em Dirksen (2015). Esta biblioteca apresenta um grande número de funcionalidades e é possível encontrar aplicações e jogos em 3D de nível profissional. Um exemplo de animação renderizada que utiliza *ThreeJS* é mostrada na Figura 17. Mais exemplos podem ser vistos no site *threejs.org*.

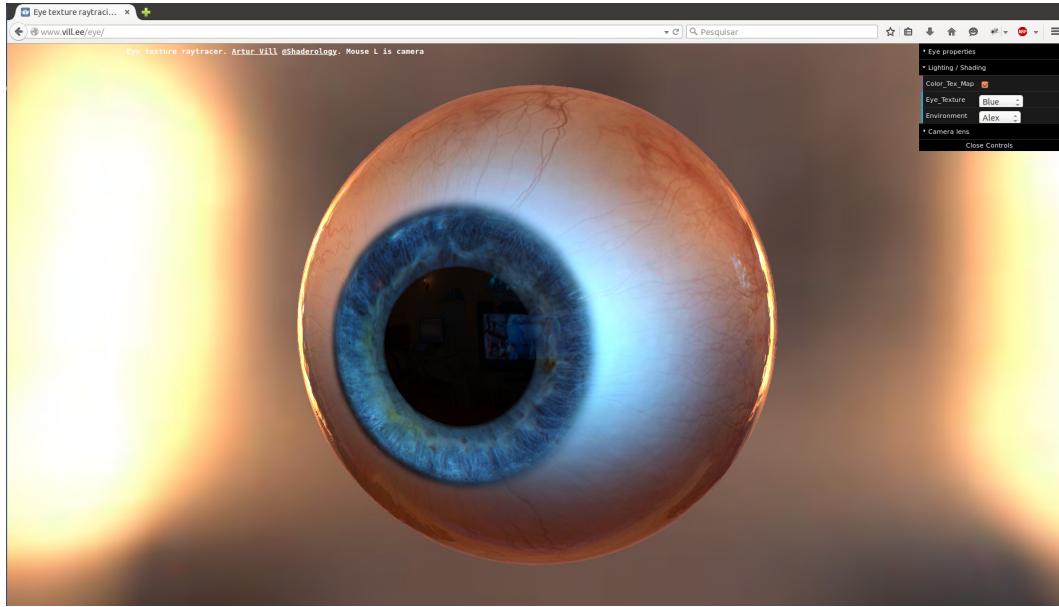


Figura 17 – Exemplo de aplicação que utiliza *ThreeJS*.

Fonte: [\(http://www.vill.ee/eye/\)](http://www.vill.ee/eye/).

2.4.5 SERVIÇOS REST COM *FLASK*

O *Representational State Transfer (REST)* foi primeiramente descrito por Fielding (2000). Ele é definido como um estilo arquitetural de sistemas hipermídia e tem as seguintes características descritas por Grinberg (2014):

1. Cliente-Servidor. Há uma separação clara entre quem consome, cliente, e quem serve e executa a lógica de negócio, servidor;
2. *Stateless*. O cliente precisa incluir nas suas requisições, todas as informações necessárias para o servidor processar o pedido. O servidor não armazena informações de estado do cliente entre as requisições deste;
3. Interface Uniforme. O protocolo que os clientes usam para acessar o servidor precisa ser consistente, bem definido e padronizado. O protocolo comumente usado por serviços *REST* é o *HTTP*;
4. Sistema em Camadas. Servidores *proxy*, *caches* ou *gateways*, podem ser inseridos entre clientes e servidores quando necessários, para melhorar a performance, confiabilidade e escalabilidade;
5. Código Sob Demanda. Clientes podem opcionalmente fazer o *download* de código do servidor e executá-lo em seu contexto.

A principal ideia de serviços *REST* é o fornecimento de recursos. Por exemplo, o cliente requere um usuário, blog, comentários, entre outros. Cada recurso deve possuir uma

URL que o identifica unicamente, por exemplo, <http://www.minhaurl.com.br/usuario/123>, onde 123 é um identificador único do usuário.

O funcionamento de uma *Web API REST* é mostrado na Figura 18. Nesta figura vê-se um cliente, que pode ser um *browser web* ou outra aplicação *mobile* ou uma aplicação web ou qualquer *software* cliente que se comunique pelo protocolo *HTTP*. Este cliente faz requisições para um software servidor, através de uma *API* que foi exposta anteriormente. No lado servidor a chamada a API específica é processada e uma resposta é devolvida ao cliente.

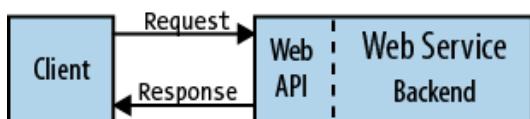


Figura 18 – *Web API REST*. Fonte: Masse (2011).

Para que um serviço *REST* funcione, ele precisa ser implementado para suportar requisições *HTTP*, ou seja, ele teria que ser um servidor *web* completo. A biblioteca escolhida para desempenhar esta função foi a *Flask*. Esta é uma biblioteca escrita na linguagem *Python* e fornece todos as ferramentas necessárias para criar aplicações *webs*. Segundo Maia (2015), o *Flask* vem sendo adotado, por sua filosofia minimalista que não impõe uma arquitetura específica de projeto, assim permitindo que um projeto comece pequeno e simples, evoluindo para um modelo mais complexo.

2.4.6 SERVIÇO DE BASE DE DOCUMENTOS COM *MONGODB*

Conforme Chodorow (2013), o *MongoDB* é poderoso, flexível e um banco de dados de propósito geral bastante escalável. Ele combina a habilidade de alta escalabilidade com índices secundários, pesquisas limitadas, ordenamento, agregações e índices geoespaciais.

O *MongoDB* é classificado como um banco *NoSQL*. Segundo Dayley (2014), o conceito de *NoSQL* consiste em tecnologias que provêm armazenamento e recuperação sem o amarrado modelo tradicional de bancos de dados Relacionais. A motivação para estas tecnologias é o design simplificado, escalabilidade horizontal e controle fino na disponibilidade dos dados.

O *MongoDB* como toda grande ferramenta de software, foi designado baseado em filosofias próprias, que para os novatos, às vezes pode parecer um contrassenso. Plugge, Membrey e Hows (2014) comentam que o pináculo das filosofias do *MongoDB* é a noção de que um tamanho não se adéqua a todos. Por muitos anos bancos de dados relacionais foram usados para armazenar conteúdos de todos os tipos. O principal motivo para isso é que ler e escrever em bancos de dados relacionais é muito mais seguro do que fazer o mesmo em arquivos de sistema operacional. É comum no uso de bancos de dados relacionais, ter

que se criar dezenas de tabelas para se armazenar dados complexos, e ainda tentar fazer todas funcionarem juntas.

Ainda, segundo Plugge, Membrey e Hows (2014), a equipe do *MongoDB* decidiu que não ia criar outro banco de dados para resolver todos os problemas do mundo. Eles criaram um banco que trabalha com documentos ao invés de linhas em tabelas. Essa decisão tornou o *MongoDB* muito rápido, altamente escalável e fácil de usar. Por exemplo, o *MongoDB* não suporta transações, logo você não irá escrever uma aplicação de conta corrente com ele. Sua força está armazenar dados muito complexos. No final o desenvolvedor ainda tem a opção de usar um banco de dados relacional que suporta transação e as vantagens do *MongoDB* para outras partes da aplicação, que se aproveitarão melhor do modelo de documentos.

A Figura 19 mostra como os dados são organizados nesta tecnologia.

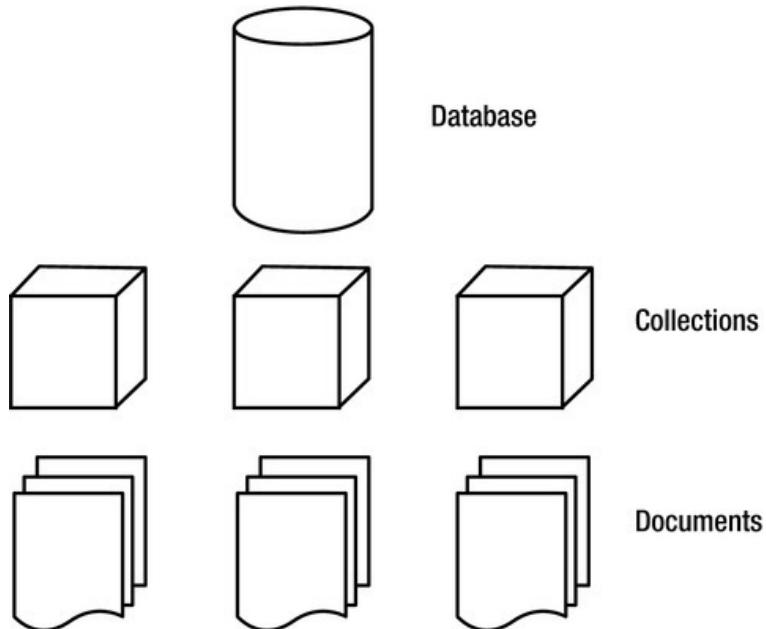


Figura 19 – Organização dos dados no *MongoDB*. Fonte: Plugge, Membrey e Hows (2014).

Segundo a Figura 19, um banco de dados *MongoDB* é composto por coleções de dados. Dentro destas coleções existem documentos. Os documentos dentro de uma coleção não precisam ser do mesmo tipo. Mas esta é uma prática pouco recomendada (PLUGGE; MEMBREY; HOWS, 2014). Os documentos são armazenados no formato *Binary Object Notation (BSON)*, um primo muito próximo do *JavaScript Object Notation (JSON)*. O *BSON* é muito parecido com o *JSON*. Ele foi criado para ser mais otimizado nas leituras e escritas no banco. A Figura 20 retirada de Dayley (2014), mostra como um documento se parece.

```

1 [
2   name: "New Project",
3   version: 1,
4   languages: ["JavaScript", "HTML", "CSS"],
5   admin: {name: "Brad", password: "*****"},
6   paths: {temp: "/tmp", project:"/opt/project",
7           html: "/opt/project/html"}
8 ]

```

Figura 20 – Listagem de um documento. Fonte: Dayley (2014).

Para quem conhece *JSON*, provavelmente compreendeu todo o documento. Um documento fica entre {} e pode conter outros documentos, tipos simples e listas. O documento possui atributos, no exemplo, *name*, *version*, *languages*, *admin*, *paths*. Os atributos *name* *version* são atributos de tipos simples, texto e número no caso. O atributo *languages* é uma lista e os demais são outros objetos.

A questão da normalização dos objetos é sempre uma decisão do desenvolvedor. É perfeitamente possível tratar coleções e documentos, como tabelas. Como cada tabela tem um identificador único gerado pelo sistema é possível inclusive fazer referência entre documentos. Mas o ideal é encontrar um meio termo, onde dados muito acessados juntamente, fiquem numa estrutura hierárquica como a da listagem acima (DAYLEY, 2014). A Figura 21 mostra como se pode fazer a referência entre documentos.



Figura 21 – Definindo documentos normalizado com *MongoDB*. Fonte: Dayley (2014).

2.5 CMAC

A *Cerebellar Model Articulation Controller* (CMAC) foi criada por James Sacra Albus (ALBUS, 1975a). Ele se inspirou no cerebelo dos mamíferos para criá-la. O mesmo autor havia feito um extenso trabalho sobre o funcionamento do cerebelo (ALBUS, 1971). Trabalho este, que resultou numa tese de doutorado (ALBUS, 1972). Aplicações da CMAC podem ser vistas em (ALBUS, 1975b), (ALBUS, 1979), (SABOURIN, 2006) e (LIN; SONG, 2002). Uma aplicação que começou a ser desenvolvida pela UnB é descrita em Andrade et al. (2014). Inclusive a implementação da *CMAC* desenvolvida neste trabalho teve origem nele.

Na Figura 22 é possível ver o funcionamento básico da CMAC. Os sinais S entram no sistema, que mapeiam o mesmo para um conjunto de pesos W^* que devem ser somados para ativação. Note que apenas uma pequena fração de pesos é realmente selecionada para participar na ativação. O conjunto de pesos disponíveis na CMAC é necessariamente maior que o número de pesos ativados W^* .

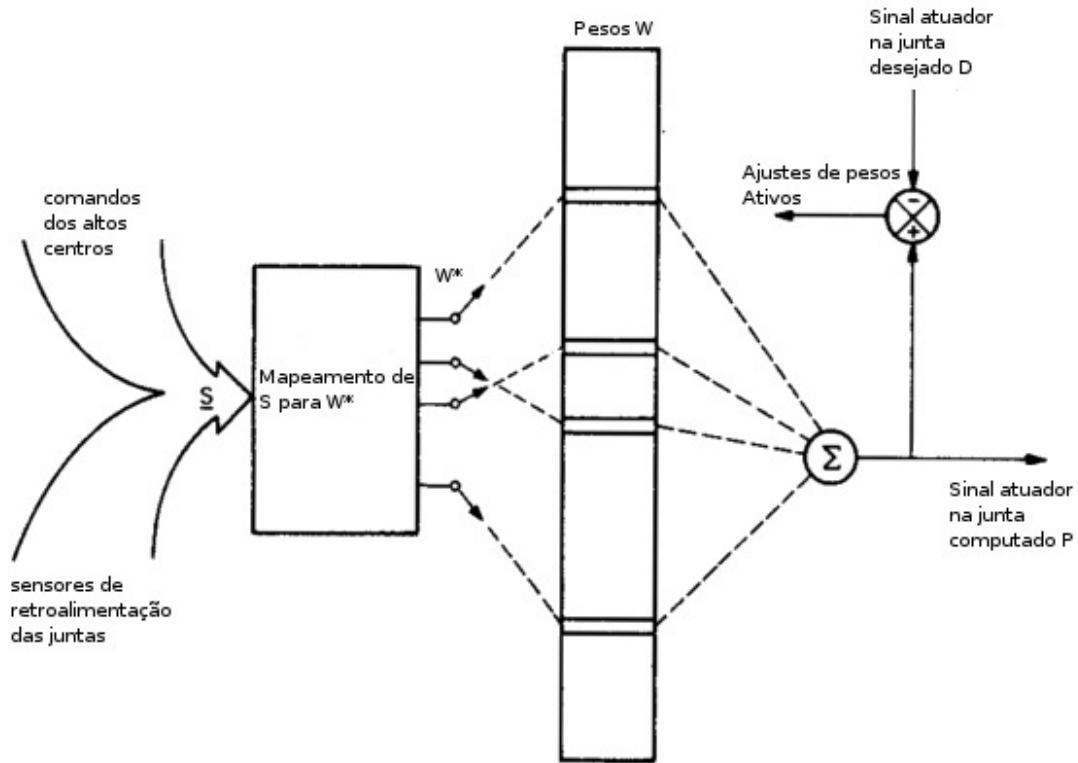


Figura 22 – CMAC para controle de uma junta. Fonte: Alterado de Albus (1975a).

A CMAC da Figura 22 também pode ser classificada como um sistema *Multiple Input Single Output* (MISO), ou seja, suporta a entrada de vários sinais de entrada e processa um sinal de saída. Para se produzir uma CMAC *Multiple Input Multiple Output* MIMO, bastaria implementar várias MISOs, compartilhando as mesmas entradas.

Os passos para que o sinal seja computado são descritos a seguir. Estes passos são os mesmos descritos em Albus (1975a).

Primeiramente, define-se o número de pesos $NW*$ a serem ativados para comporem a saída da CMAC.

O segundo passo é quantizar os possíveis valores para cada item do vetor de entrada S . Por exemplo, se o primeiro item $s1$ de S aceita valores de -1 até 1 e se quer 5 valores possíveis, quantiza-se então os valores de -1 até 1, conforme a Figura 23.

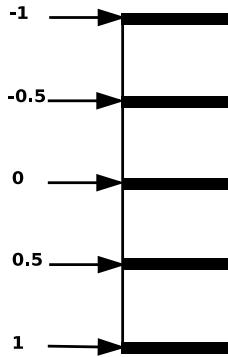


Figura 23 – Quantização de $s1$.

Isto significa que quaisquer que sejam os valores de $s1$ os mesmos devem ser convertidos para -1, -0,5, 0, 0,5 e 1. Por exemplo, se o valor de $s1$ for 0,75, será convertido para o valor 1, se for -0,75 será o valor 0 e se for 0,25 será o valor 0,5. A esta quantização dá-se o nome de resolução da CMAC (ALBUS, 1975a).

O próximo passo é criar uma tabela para cada um dos sinais discretizados de entrada do vetor S . Supondo que o vetor S possui 2 sinais de entrada $s1$ e $s2$ e um número de ativações $NW*$ igual a 3, deve-se criar duas tabelas, uma para cada sinal, conforme se apresentam na Tabela 3 e na Tabela 4. Para facilitar o entendimento, irá se considerar os valores possíveis de $s1$ iguais aos inteiros de 1 até 6 e os valores possíveis de $s2$ iguais aos inteiros de 1 até 4. Cria-se uma coluna com os valores do sinal em questão. Para cada valor, cria-se 3 ($NW*$) valores novos. Para o primeiro valor do sinal, usa-se os 3 primeiros inteiros não negativos. Para o segundo valor do sinal, substitui-se o primeiro dos três itens pelo próximo número após o terceiro item do sinal anterior. Para o terceiro sinal, substitui-se o segundo dos três itens pelo próximo inteiro não negativo. Assim, sucessivamente conforme a Tabela 3 e Tabela 4 (ALBUS, 1975a).

Tabela 3 – Mapeamento de $s1$

<i>Valore de s1</i>	<i>Mapeamento m1</i>
1	0, 1, 2
2	3, 1, 2
3	3, 4, 2
4	3, 4, 5
5	6, 4, 5
6	6, 7, 5

Tabela 4 – Mapeamento de $s2$

<i>Valore de s2</i>	<i>Mapeamento m2</i>
1	0, 1, 2
2	3, 1, 2
3	3, 4, 2
4	3, 4, 5

Depois de mapeado cada valor de cada item de entrada, deve-se combinar os mapeamentos de acordo com a Tabela 5.

Tabela 5 – Mapeamento para os pesos W

<i>s1 / s2</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
1	(0, 0), (1, 1), (2, 2)	(0, 3), (1, 1), (2, 2)	(0, 3), (1, 4), (2, 2)	(0, 3), (1, 4), (2, 5)
2	(3, 0), (1, 1), (2, 2)	(3, 3), (1, 1), (2, 2)	(3, 3), (1, 4), (2, 2)	(3, 3), (1, 4), (2, 5)
3	(3, 0), (4, 1), (2, 2)	(3, 3), (4, 1), (2, 2)	(3, 3), (4, 4), (2, 2)	(3, 3), (4, 4), (2, 5)
4	(3, 0), (4, 1), (5, 2)	(3, 3), (4, 1), (5, 2)	(3, 3), (4, 4), (5, 2)	(3, 3), (4, 4), (5, 5)
5	(6, 0), (4, 1), (5, 2)	(6, 3), (4, 1), (5, 2)	(6, 3), (4, 4), (5, 2)	(6, 3), (4, 4), (5, 5)
6	(6, 0), (7, 1), (5, 2)	(6, 3), (7, 1), (5, 2)	(6, 3), (7, 4), (5, 2)	(6, 3), (7, 4), (5, 5)

O mapeamento da Tabela 5 é feito da seguinte forma (ALBUS, 1975a): Coloca-se cada valor quantizado de $s1$ nomeando cada uma das linhas da tabela. Coloca-se cada valor discretizado de $s2$ nomeando cada uma das colunas das tabelas. Cada célula da tabela é obtida recuperando-se os itens de $s1$ e de $s2$, que estão na Tabela 3 e na Tabela 4, e concatenando-se cada item de acordo com sua posição. Por exemplo, para o valor de $s1$ igual a 5, recuperam-se os itens (6, 4, 5), para o valor de $s2$ igual a 2, recuperam-se os itens (3, 1, 2). Agora é só criar novos itens, na mesma quantidade do valor NW^* , concatenando-se cada item de acordo com sua posição. O lista resultante é ((6, 3), (4, 1), (5, 2)).

Cada vetor de 2 posições da Tabela 5, por exemplo (0, 0), (4, 1), é um rótulo de uma entrada na tabela de pesos W .

Para se calcular a saída do sistema, suponha que os valores de $s1$ e $s2$, ainda sejam 5 e 2, respectivamente, e $NW*$ ainda seja 3. Neste caso, recupera-se o item da linha 5 e da coluna 2, que é ((6, 3), (4, 1), (5, 2)). Agora é acessada a tabela de pesos W e são recuperados os pesos cujas chaves são (6, 3), (4, 1) e (5, 2). Note-se que serão recuperados exatamente 3 pesos, que é exatamente o número de ativações $NW*$ desejadas. Os valores recuperados constituem um vetor chamado $W*$. A saída é calculada somando-se os valores de $W*$.

2.5.1 TREINAMENTO DA CMAC

Sendo a CMAC uma Rede Neural Artificial (RNA) com aprendizado supervisionado, seus pesos podem ser atualizados simplesmente computando-se o erro e atualizando-se apenas os pesos que participaram do computo do sinal P (HAYKIN, 1998). O processo é detalhado nos próximos parágrafos.

Para se treinar a CMAC, primeiro define-se o conjunto de dados para treinamento que devem ser usados. Pode ser usado algo entre 20% e 50% dos dados coletados para desenvolvimento do sistema. Cada item destes dados deve conter um vetor de sinais de entradas S e a resposta desejada D para estes sinais. A tabela de pesos deve ser inicializada com valores randômicos. Para cada vetor S deve ser calculado o vetor $W*$, que contém os endereços dos pesos a serem ativados. Calcula-se, então, a saída da rede P para o vetor S , conforme definido no item 2.5, somando os itens do vetor W . Atualizam-se os pesos sinápticos conforme a Equação 1 adaptada de Sabourin, Yu e Madani (2012).

$$w_i = w_i + \frac{\alpha(D - P)}{NW^*} \quad (1)$$

A variável w_i é um item qualquer dentro da tabela de pesos W . Em cada iteração no conjunto de dados para treinamento, deve-se atualizar apenas os pesos descritos em $W*$ naquele momento. A variável α é o coeficiente de aprendizado, deve ser um valor entre 0 e 1. A variável $NW*$ é o número de valores a serem utilizados para ativação, que consequentemente equivale ao número de itens do vetor $W*$.

Deve-se determinar o número de iterações *batch* para o sistema. Uma iteração *batch* consiste no processamento dos pesos para cada item dentro do conjunto de dados para treinamento (NG, 2015). O sistema deve continuar iterando até atingir o número de iterações *batch*.

Apesar do número de iterações *batch* ser válido como critério de parada para o treinamento da CMAC, nada impede o uso de outros critérios, por exemplo, o erro quadrado médio da saída Y em relação ao desejado D .

2.6 ALGUNS CÁLCULOS CINEMÁTICOS

Para este trabalho ser concluído, alguns cálculos cinemáticos foram necessários. Os cálculos são as velocidades, ângulos e velocidades angulares.

Velocidades entre duas posições adjacentes num espaço 3D, são calculadas conforme a Equação 2 (POOLE, 2011).

$$\vec{v} = \frac{\vec{s2} - \vec{s1}}{\tau} \quad (2)$$

Os vetores $\vec{s1}$ e $\vec{s2}$ são dados espaciais em três dimensões (X, Y, Z) e τ é o tempo transcorrido entre o deslocamento de um ponto ao outro. O vetor \vec{v} possui as velocidades calculadas.

Para o cálculo de ângulos em três dimensões, primeiro os componentes devem ser transladados para uma origem comum, assim é possível usar a Equação 5 como descrita em Edwards (2006). Para tal, usam-se as Equações 3 e 4 (POOLE, 2011).

$$\vec{c1}' = \vec{c1} - \vec{o} \quad (3)$$

$$\vec{c2}' = \vec{c2} - \vec{o} \quad (4)$$

As variáveis $\vec{c1}$, $\vec{c2}$ e \vec{o} , são vetores representando pontos no espaço. O vetor \vec{o} é a origem dos dois seguimentos de reta que chegam a $\vec{c1}$ e $\vec{c2}$ cada um. No contexto deste trabalho, $\vec{c1}$ poderia ser um ponto na coxa, $\vec{c2}$ um ponto na tibia e \vec{o} um ponto no joelho. Assim seria possível calcular o ângulo de um joelho, por exemplo. Veja a Figura 24. As variáveis $\vec{c1}'$ e $\vec{c2}'$, são os vetores transladados e prontos para se calcular o ângulo de acordo com a Equação 5 (EDWARDS, 2006).

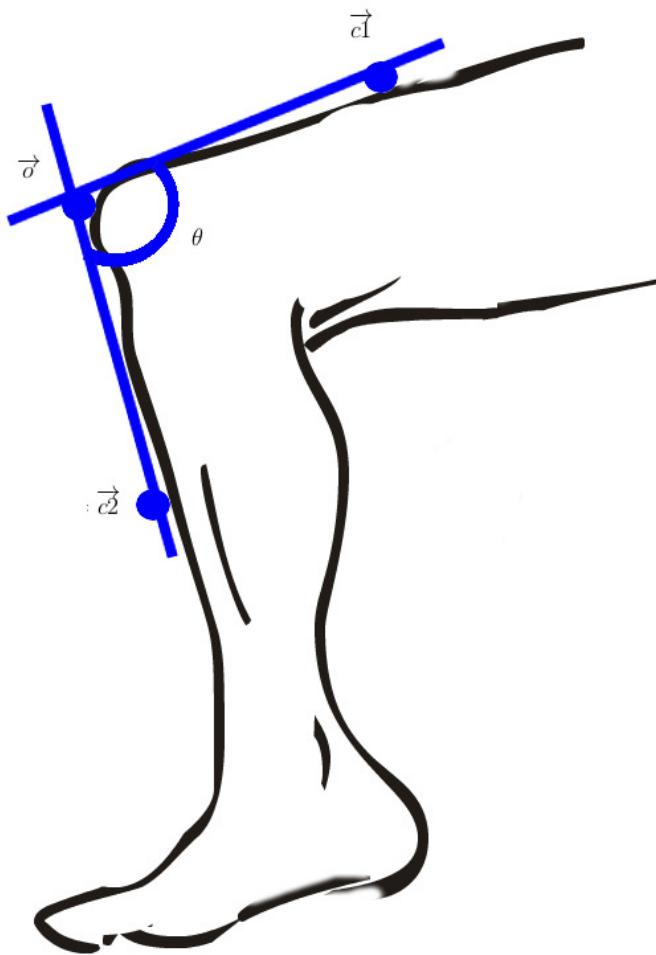


Figura 24 – Exemplo hipotético de um ângulo. Fonte: Alterado de Cliparts (2015).

$$\theta = \arccos\left(\frac{\vec{c1}' \cdot \vec{c2}'}{\|\vec{c1}'\| \cdot \|\vec{c2}'\|}\right) \quad (5)$$

A variável θ é um ângulo em radianos. O operador $\|\cdot\|$ é a distância euclidiana como definida por Poole (2011) e seu cálculo é mostrado na Equação 6.

$$\|\vec{c}\| = \sqrt{\vec{c} \cdot \vec{c}} \quad (6)$$

A velocidade angular é calculada segundo a Equação 7 Poole (2011).

$$\omega = \frac{\theta_2 - \theta_1}{\tau} \quad (7)$$

A variável ω é a velocidade angular em radianos por segundo, θ_1 e θ_2 são ângulos e τ o tempo transcorrido entre a medida do primeiro ângulo e do segundo.

2.7 SIMULAÇÃO

Segundo Garcia (2009) simulação é:

A obtenção da resposta temporal das variáveis de interesse (variáveis dependentes) de um modelo, quando se excita suas variáveis de entrada com sinais desejados e se definem os valores das condições iniciais das variáveis dependentes.

A Figura 25 representa um modelo simplificado de um processo passível de simulação, onde X é conjunto de variáveis de entrada, Y é o conjunto de variáveis de saída e P são os parâmetros do sistema incluindo condições de contorno. Com base neste modelo, Garcia (2009) afirma que é possível simular as seguintes aplicações:

1. Projeto de equipamentos, processos e plantas e seus respectivos sistemas de controle;
2. Pré-operação e operação de plantas;
3. Sistema de controle de processos;
4. Otimização das condições operacionais de plantas.

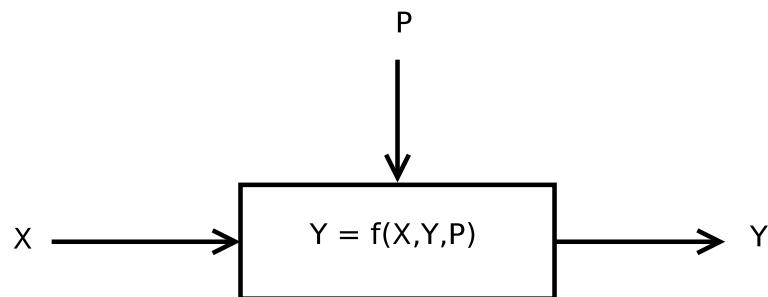


Figura 25 – Modelo matemático simplificado de um processo. Fonte: Garcia (2009).

3 METODOLOGIA

3.1 AMBIENTE DE ESTUDO

Como este trabalho trata de um projeto de desenvolvimento de software na área de análise de marcha, dois ambientes de trabalho distintos foram amplamente utilizados. São estes:

1. Laboratório de Informática em Saúde (LIS) na Faculdade Gama (FGA) da Universidade de Brasília (UnB);
2. Laboratório de Performance Humana (LPH) na Faculdade Ceilândia (FCE) da UnB.

No LIS foram desempenhadas as tarefas relativas a engenharia de software e disponibilização do software. Foram utilizadas estações de trabalho do tipo *Power Mac* com sistema operacional *MAC OS X 10.10.3*, sendo que uma foi preparada para funcionar como servidor de aplicação e roteador de rede. A estação preparada serve de hospedeira de duas máquinas virtuais (*Virtual Machines - VMs*) rodando através do software *Virtualbox 4.3*. Cada VM utiliza sistema operacional *Debian Wheezy GNU/Linux*. Uma destas VMs foi configurada como roteador e *firewall* e a outra como servidor de aplicações. Outra estação de trabalho *Power Mac*, idêntica, e um *notebook* rodando *Ubuntu 14.04 GNU/Linux* foram utilizados como máquinas de desenvolvimento e simulação. Um diagrama da rede criada no LIS é mostrado na Figura 26.

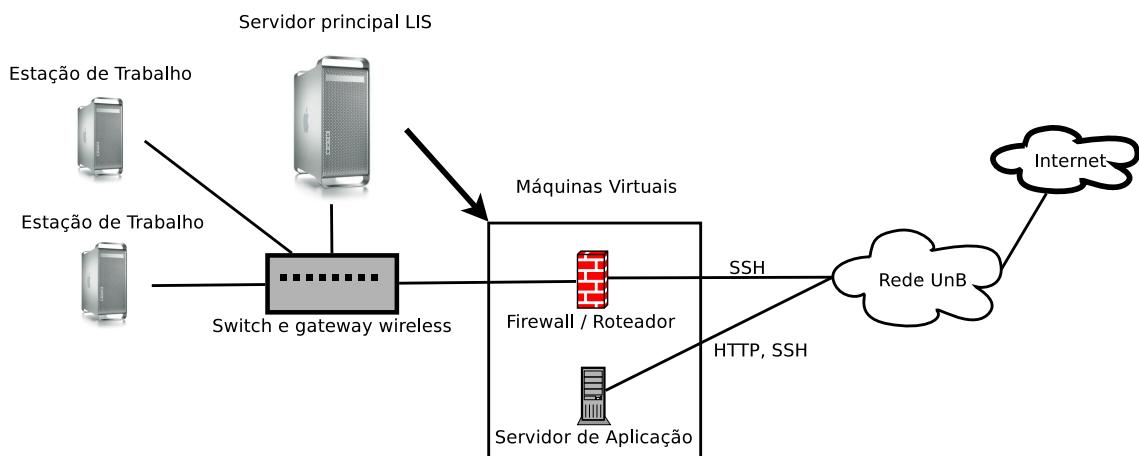


Figura 26 – Rede LIS.

O LPH/FCE foi utilizado para captura de dados de marcha humana. Este laboratório está equipado para coletar dados de plataformas de força, eletromiôgrafos e de marcadores passivos posicionados no corpo do paciente através de câmeras de vídeo *Oqus-MRI*, usando técnicas de (*Motion Capture - MOCAP*), ver Figura 3 na Seção 2.1.3.

Para este trabalho foi utilizado o software *QTM 3.2* da *Qualisys*, ver Figura 4 na Seção 2.1.3, que é responsável pela coleta de dados *MOCAP*.

O processo de coleta é demonstrado na Figura 27.

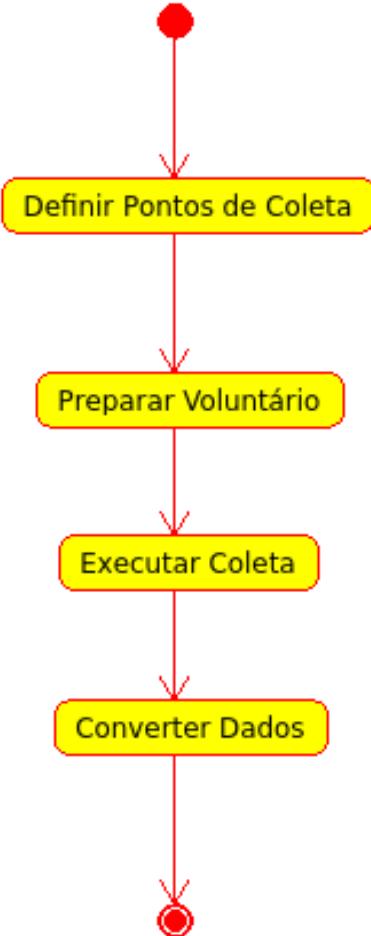


Figura 27 – Processo de coleta de dados.

Primeiro deve-se definir o paciente da coleta e determinar o dia para este processo. Além disso, também é necessário definir quais os pontos no corpo do paciente devem receber marcadores de superfície. O próximo passo se refere a coleta dos dados em si. O paciente deve repetir um ciclo de marcha confortável de aproximadamente 5 segundos, por 5 vezes na frente das câmeras.

Quanto aos dados, estes devem ser convertidos para formato adequado à linguagem Octave, que é a mesma opção para converter para o MATLAB. Esta opção é própria do QTM. O número que o QTM atribui internamente ao marcador é a posição do marcador na matriz gerada durante a exportação. Este número é chamado dentro do QTM de canal. Os dados trazem variáveis espaciais e o erro, com respeito à posição (X, Y, Z) dos marcadores.

A disposição que os dados obtidos neste processo se apresentam, é demonstrada na Figura 28.



Figura 28 – Dados disponibilizados pelo QTM.

São retornados vários dados do QTM, mas os de interesse para o projeto são os que estão na Figura 28. O *Frame Rate* é a taxa de coleta dos dados e está em *frames* por segundos. A matriz de três dimensões, com dados espaciais dos marcadores passivos de superfície, está disposta da seguinte forma:

1. A primeira dimensão tem tamanho variável e representa o número de canais do sistema de coleta, ou seja, cada item representa um marcador;
2. A segunda dimensão é de tamanho 4 e representa a posição num plano 3D (X, Y, Z) do marcador, mais o erro;
3. A terceira dimensão é de tamanho variável e representa o número de *frames* coletados numa caminhada específica.

O projeto no qual ocorreu a coleta foi aprovado pelo Comitê de Ética da Faculdade de Saúde da UnB, processo N11911/12 (ver Anexo A).

3.2 DELIMITAÇÃO DO ESTUDO

Este trabalho tem como foco estabelecer uma metodologia de desenvolvimento inicial a um sistema de análise e simulação de marcha. Ele não busca ser extensivo o suficiente para criar um produto pronto para o mercado, mas pretende, através da implementação de funcionalidades reais, estabelecer uma arquitetura mínima e funcional que sirva de base para a construção do sistema. Como consequência, o projeto também integra os principais componentes desta arquitetura, por exemplo, o serviço de banco de documentos, com a *Application Program Interface (API) web*. Um software como este, robusto o suficiente para ser viável no mercado, seria muito caro.

3.3 VISÃO

Apesar deste trabalho ter um objetivo específico e delimitado, dele nasce um projeto maior, cuja visão é o desenvolvimento de um software como serviço para análise e simulação

de marcha, utilizando o estado da arte em técnicas para este fim. O software deve ser construído utilizando-se métodos ágeis e terá uma arquitetura adaptável que permita evolução contínua.

A estratégia é lançar a versão inicial do software como projeto de código livre, conseguir parceiros e procurar um modelo de negócio sustentável para mantê-lo.

3.4 MODELO DE GESTÃO

O software terá um modelo de gestão baseado no método *SCRUM*, como definido em 2.2.1. O método não é adotado na plenitude, sendo adaptado segundo as limitações de recursos do projeto.

Nesta fase inicial, o processo conta com um desenvolvedor, que também assume o papel de *scrum master*, e um *product owner*. Devido ao tamanho reduzido da equipe e da localização distinta dos membros, não há *daily scrum*, mas problemas de trabalho cotidianos são resolvidos por telefone, *email* ou mensagens instantâneas.

O princípio de *time boxing* é mantido. Ficou definido que o *sprint* consiste do prazo de duas semanas. Ao final do *sprint* uma reunião em duas fases é realizada. A primeira fase consiste na revisão do *sprint* anterior. Já a segunda fase é o planejamento do próximo *sprint*.

Um *backlog* de produto é mantido. Na reunião de final de *sprint* é criado um *backlog* de *sprint*. Os itens de *backlog* são mantidos na forma de estórias de usuários, conforme 2.2.2. Na Figura 29 é apresentada a visão geral do processo.

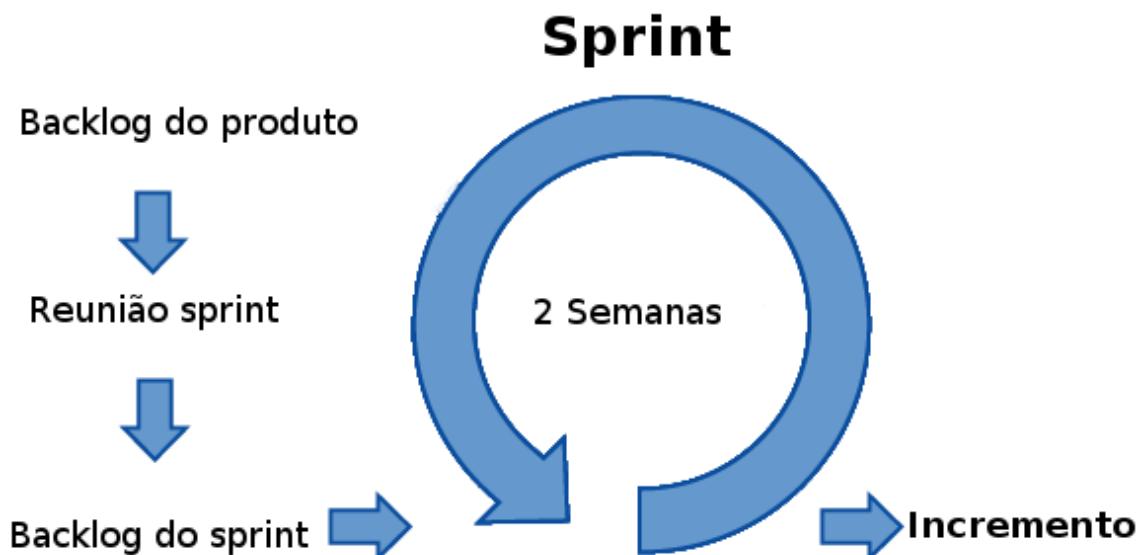


Figura 29 – Processo de desenvolvimento.

3.5 MODELO DE ARQUITETURA

O modelo de arquitetura no seu nível mais elevado, pode ser visto como um modelo de três camadas, conforme a Figura 30.

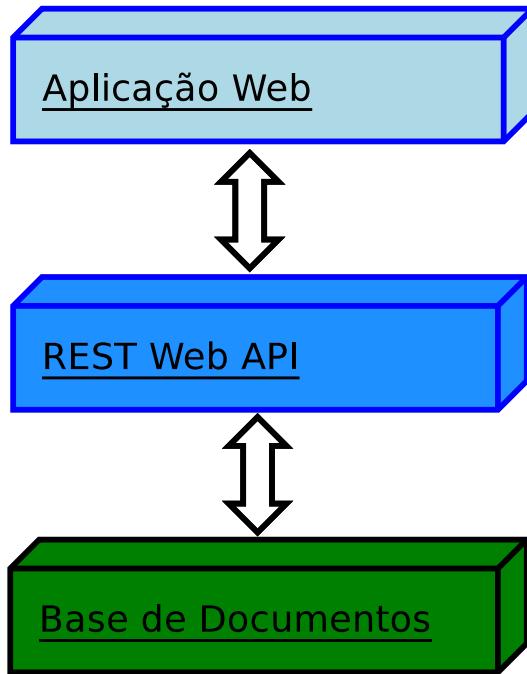


Figura 30 – Camadas arquiteturais.

A camada *web* é responsável pela interação com o usuário. A camada *Web API* é responsável pela lógica de negócio. A camada de base de documentos é responsável pela persistência dos dados da aplicação.

3.5.1 CAMADA DE APLICAÇÃO WEB

Esta camada foi projetada para rodar em *browsers* que suportam *HTML 5*. Ela é desenvolvida usando-se *Javascript*, *CSS* e *HTML*. Além disso, adotou-se o *framework* de desenvolvimento *web AngularJS*, ver 2.4.1.

Como o projeto não possuía recursos adequados a criação de uma equipe de desenvolvimento *web* completa, afim de se minimizar os problemas com *design web*, optou-se por usar a biblioteca *angular-material*, ver 2.4.3.

A Figura 31, mostra um exemplo de uma tela criada com as diretivas do angular-material. Note que todo o *look and feel* da tela é determinado pelo comportamento padrão da biblioteca.

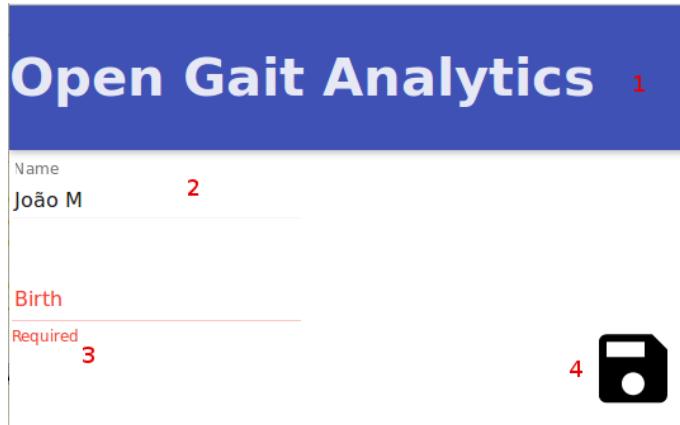


Figura 31 – Exemplo de uma tela criada com *angular-material*. 1) Diretiva *md-toolbar*; 2) Diretiva *md-input-container*; 3) Diretiva *ng-messages* em conjunto com a *md-input-container*; 4) Diretiva *md-button*.

ORGANIZAÇÃO DO CÓDIGO FONTE

A criação de um ambiente de desenvolvimento *web*, para um software de média para grande complexidade, não é uma tarefa trivial de ser resolvida. É necessário criar padrões de organização de arquivos, configurar e instalar pacotes de software para desenvolvimento, testes, implantação, construção de *builds*, entre outros. Para facilitar esta tarefa, optou-se em utilizar o projeto *angular-seed*, ver 2.4.2. A ideia deste projeto é servir de esqueleto de projetos *web* que utilizam o framework *AngularJS*. Para usar este projeto, basta cloná-lo diretamente do seu repositório *git* no site [github.com](https://github.com/angular/angular-seed.git), conforme o comando abaixo.

```
git clone https://github.com/angular/angular-seed.git
```

VISÃO ARQUITETURAL DA CAMADA WEB

A Figura 32 mostra o funcionamento e os padrões mínimos a serem seguidos para implementação da camada *web*.

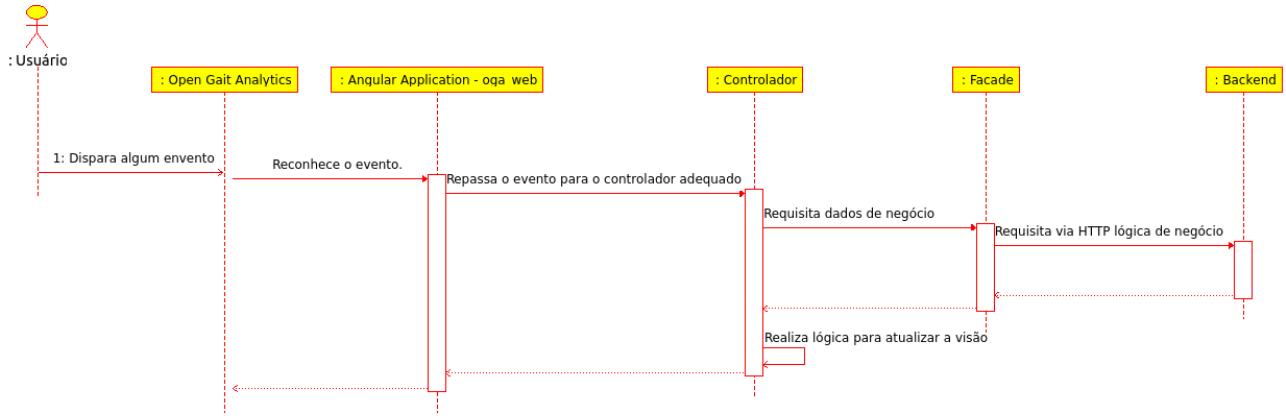


Figura 32 – Camada web.

Este esquema funciona da seguinte forma: O usuário usando seu *browser*, acessa o *site* do software. Ao realizar qualquer evento na aplicação, por exemplo, clicar num botão, ou selecionar um item em uma lista de seleção, a aplicação *angularjs* detecta o evento e seleciona um componente de software chamado controlador. Existem vários destes controladores no sistema, cada evento do usuário é redirecionado para um que seja adequado. No controlador é onde grande parte da programação acontece. Ele é associado a um *template HTML*. Este template funciona como componente de apresentação para o usuário, e é o que o controlador manipula para mostrar informações ao usuário. Quando o controlador precisa executar lógica de negócios, ou requisitar dados persistidos, ele deve chamar um componente do tipo *Facade*. A aplicação *web* roda no *browser* do usuário e não persiste dados nem executa lógica de negócios. Este é um estilo arquitetural escolhido para o projeto. A *Facade* é um *proxy* que se comunica com um *backend* via *HTTP* no estilo *REST* de comunicação via *web*.

A Figura 33, mostra um exemplo de componentes escritos em *JavaScript* respondendo a um evento disparado pelo usuário. Este evento poderia ser oriundo de um clique num botão ou na chamada de uma *URL* específica. Depois de reconhecido o evento, no caso um pedido para ver a lista de pacientes, o componente *PacientesCtrl* e o *template patients.html* são carregado pelo *framework AngularJS*. Neste momento o *framework* acessa o componente *PacientesFacade* através do seu método *getPatients*. Este método invoca o método *get* do componente *\$HTTP* que faz parte do *framework*. Uma requisição é feita para o *backend*, que retorna os dados no formato *JSON*. Finalmente, o *framework* detecta a resposta e atualiza a tela para o usuário.

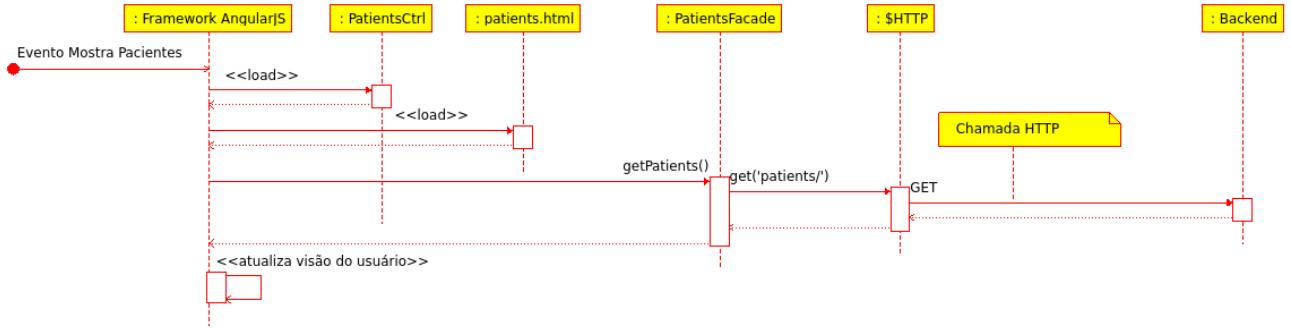


Figura 33 – Exemplo de componentes da camada *web* funcionando juntos.

3.5.2 CAMADA REST WEB API

Esta é a camada responsável por expor toda a lógica de negócio e acesso a dados persistidos. Escolheu-se que esta camada fornece suas funções via *web API* no estilo REST. Uma das vantagens desta escolha é o alto desacoplamento, entre camada *web* e lógica de negócio. Além disso, pode-se integrar mais facilmente a aplicação com outras, já que a lógica de negócio é toda exposta como *API web*. Veja a seção 2.4.5 para um melhor entendimento deste estilo *REST*.

ORGANIZAÇÃO DO CÓDIGO FONTE

A Figura 34 mostra a configuração básica dos arquivos e diretórios da aplicação.

<code>oga_api/</code>	// Diretório com o núcleo da <i>web API</i> .
<code>api_0_0/</code>	// Diretório de uma versão específica da <i>web API</i> .
<code>views.py</code>	// Arquivo de programa com implementação da <i>web API</i> .
<code>etl/</code>	// Programas para extração de dados. Exemplo dados do QTm.
<code>ml/</code>	// Algoritmos de aprendizado de máquina. Exemplo CMAC.
<code>physics/</code>	// Algoritmos para cálculos físico. Exemplo velocidades, etc.
<code>test_rest.py</code>	// Programa para testes automatizados da <i>web API</i> .
<code>commands.py</code>	// Comandos de gestão do projeto
<code>config.py</code>	// Config. de ambientes de desenvolvimento, testes e produção
<code>manage.py</code>	// Programa de gestão do projeto
<code>requirements.txt</code>	// Lista de bibliotecas utilizadas pelo projeto

Figura 34 – Organização do arquivos e diretórios da camada *web API*.

A linguagem de programação *Python* foi escolhida para implementar esta camada. Vários foram os motivos: a biblioteca *Flask* para implementar a *web API*, a biblioteca *NumPy*, que é excelente para cálculos, comparável ao Matlab, a biblioteca *Matplotlib* para criação de gráficos científicos, a baixa curva de aprendizado da linguagem, sua fama com desenvolvedores ao redor do mundo.

Para diminuir os problemas de ambientes, oriundos de um projeto complexo como este, optou-se por utilizar o programa *virtualenv*. Este programa cria um ambiente *python*

virtual, baseado num arquivo de configuração. Isso faz com que todos os desenvolvedores envolvidos no projeto, possuam ambientes muito semelhantes. Ao se clonar o projeto basta entrar no diretório da API e digitar o comando:

```
virtualenv env
```

Este comando irá criar uma diretório chamado *env*. Para poder ativar o ambiente virtual é necessário o comando no unix:

```
. env/bin/activate
```

As bibliotecas necessárias a execução da aplicação estão listadas no arquivo *requirements.txt*. Para instalá-las no novo ambiente virtual é necessário o comando:

```
pip install -r requirements.txt
```

Neste ponto, o código já pode ser editado e executado. Para facilitar um pouco as coisas, foi criado o programa *manage.py*. Para rodar um servidor web local respondendo na porta 5000, com fins de desenvolvimento, basta digitar o comando:

```
python manage.py runserver
```

Já para executar testes automatizados:

```
python manage.py test
```

Vale lembrar que o servidor *MongoDB*, deve estar configurado, rodando e suas configurações editadas no arquivo *config.py*.

VISÃO ARQUITETURAL DA CAMADA REST WEB API

A Figura 35 mostra um *blueprint* de como funciona e como deve ser desenvolvida esta camada.

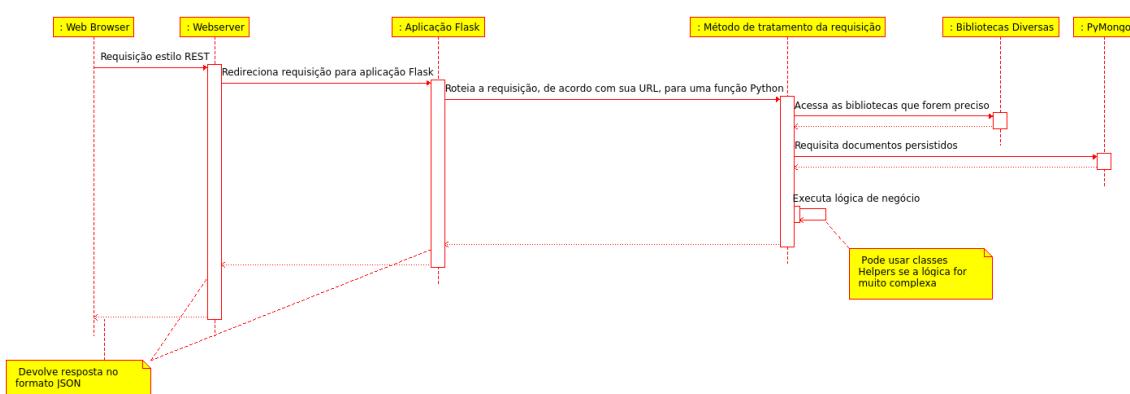


Figura 35 – Camada REST WEB API.

Tudo começa quando um *browser web* faz uma requisição do tipo *HTTP* a um servidor *web*. O servidor *web* identifica se a requisição é para a aplicação *flask* do sistema em questão. Se for, esta requisição é repassada para a aplicação *flask*. Agora as coisas começam a ficar mais interessantes. A aplicação *flask* analisa a *URL* da requisição, verifica o método da requisição e repassa os dados da requisição, num formato amigável ao *python*, para uma função *python*. Por padrão os parâmetros das requisições via método “*GET*” são repassadas ao *python* como *string*. Para os demais métodos, padronizou-se receber o *payload* da requisição *HTTP*, como objetos *JSON*, que são facilmente convertidos para dicionários *python*.

São nas funções *python*, que tratam as requisições, que a lógica de negócio é executada. Aqui bibliotecas como a *NumPy* podem ser chamadas, ou mesmo bibliotecas criadas pelos desenvolvedores da aplicação. É a partir deste ponto que dados podem ser acessados do banco de documentos pela biblioteca *PyMongo*. Ao final da execução uma resposta é gerada no formato *JSON* para que seja consumida pela camada *web*.

A Figura 36, mostra um exemplo de componentes escritos em *Python*, tratando uma requisição *HTTP*, no caso uma chamada a *URL* *http://«myurl»/patient*. Depois que o servidor *web* repassou a requisição para a aplicação que usa o *framework Flask*, o *framework* executa sua rotina de roteamento e descobre qual função *Python*, contida no componente *views*, deve ser executada, no caso a função *get_patients*. Esta função acessa um objeto do tipo *DataBase*, pertencente a biblioteca *PyMongo*, e executa o método *find* da coleção *patients* pertencente ao *DataBase*. O resultado desta chamada são os dados contidos no banco de dados retornados no formato *BSON*. O componente *json_util*, da biblioteca *PyMongo*, tem seu método *dumps* chamado. Este método converte os dados de *BSON* para *JSON*. Finalmente, o *framework Flask* responde a requisição.

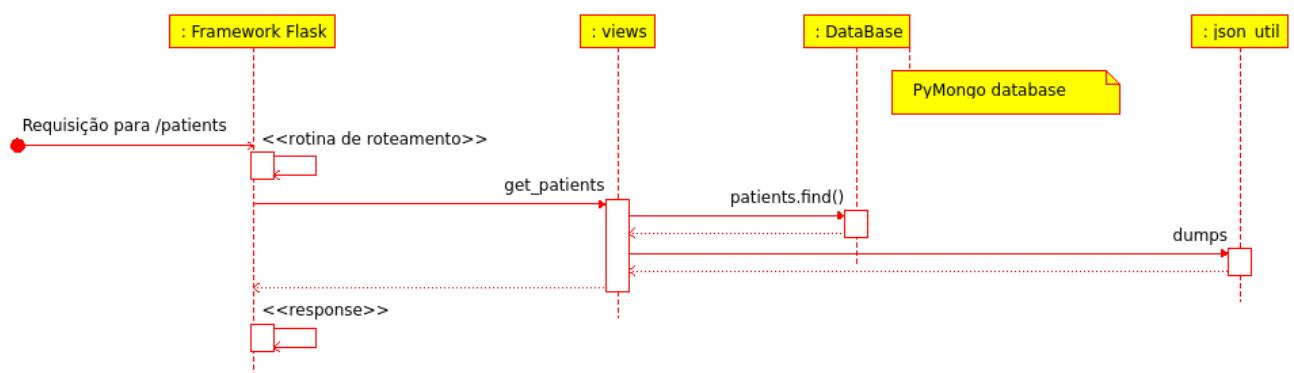


Figura 36 – Exemplo de requisição sendo tratada pela *web api*.

3.5.3 CAMADA DE BASE DE DOCUMENTOS

Para esta camada foi escolhido o banco de documentos *MongoDB*, ver a seção 2.4.6. Há várias vantagens no uso desta tecnologia, mas a determinante foi a facilidade de uso e criação de estruturas de dados. No início do projeto, foi usado um banco de dados relacional e um *framework* de mapeamento objeto-relacional. Devido a natureza altamente complexa dos dados, dados espaciais provindos de marcadores de superfície capturados por câmeras, eles são multidimensionais também. Sem dúvida isto ajudou a tornar possível criar esta primeira versão do software em tão pouco tempo.

A estrutura do banco da aplicação é mostrada na Figura 37. O banco é composto por duas coleções: os dados dos pacientes na coleção *patients* e os dados recuperados do *QTM positionals_data*.

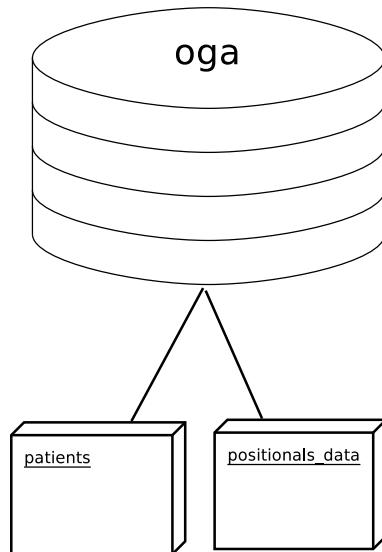


Figura 37 – Banco de documentos da aplicação.

A Figura 38 mostra um exemplo de documento da coleção *patients*. Já a Figura 39 mostra um exemplo de documento da coleção *positionals_data*.

```

1 {
2     "\_id" : ObjectId("5601ed9e54bac75828a154e4"),
3     "city" : "Brasilia",
4     "name" : "Beto Jamais",
5     "weight" : 75,
6     "profession" : "Professor",
7     "zipCode" : "70000000",
8     "sex" : "male",
9     "phone" : "61 5555-5555",
10    "state" : "DF",
11    "race" : "white",
12    "birth" : "2000-01-01T02:00:00.000Z",
13    "address" : "Rua da Serra",
14    "maritalStatus" : "married",
15    "height" : 1.8
16 }
17

```

Figura 38 – Documento da coleção *patients*.

```

1 {
2     "\_id" : ObjectId("55a8f9874f9336e37084be92"),
3     "patient\_id" : ObjectId("55a8f97654bac7124a76ec4d"),
4     "gait\_sample\_index" : 0,
5     "original\_filename" : "Walk1.mat",
6     "angles" : [
7         {
8             "origin" : "0",
9             "component\_b" : "2",
10            "component\_a" : "1",
11            "description" : "Angulo Joelho Direito"
12        },
13        {
14            "origin" : 4,
15            "component\_b" : 6,
16            "component\_a" : 5,
17            "description" : "teste"
18        }
19    ],
20    "initial\_frame" : 300,
21    "number\_markers" : 3,
22    "markers" : [
23        "Joelho Direito",
24        "Tibia Direita",
25        "Trocanter Direito"
26    ],
27    "frame\_rate" : 315,
28    "frames" : 1491,
29    "final\_frame" : 1300
30    "trajectories":[ Aqui deveria aparecer uma lista com
31                    milhares de linhas numa matriz de 3 dimensoes]
32 }
33

```

Figura 39 – Documento da coleção *positionals_data*.

4 RESULTADOS

O software construído por este projeto, que está na versão 0.1, foi inteiramente construído pelo autor desta obra, e está liberado como software livre no site https://github.com/rob-nn/open_gait_analytics, sob a licença *Massachusetts Institute of Technology (MIT)* (ver Anexo B). A intenção do autor é aumentar as chances de que futuras versões do software sejam construídas, não importando se serão comerciais ou gratuitas.

O software em si é composto por dois módulos que são apresentados na Figura 40. O principal objetivo neste momento é atrair pesquisadores da área de análise de marcha para o software. Por isso há o módulo simulação, no qual o pesquisador poderá simular sinais. O módulo de análise visa atender tanto a pesquisadores, quanto a profissionais da área clínica. Neste momento, o software é mais um protótipo funcional do que algo pronto para o mercado. Portanto, o uso por profissionais da área clínica não é recomendado ainda.

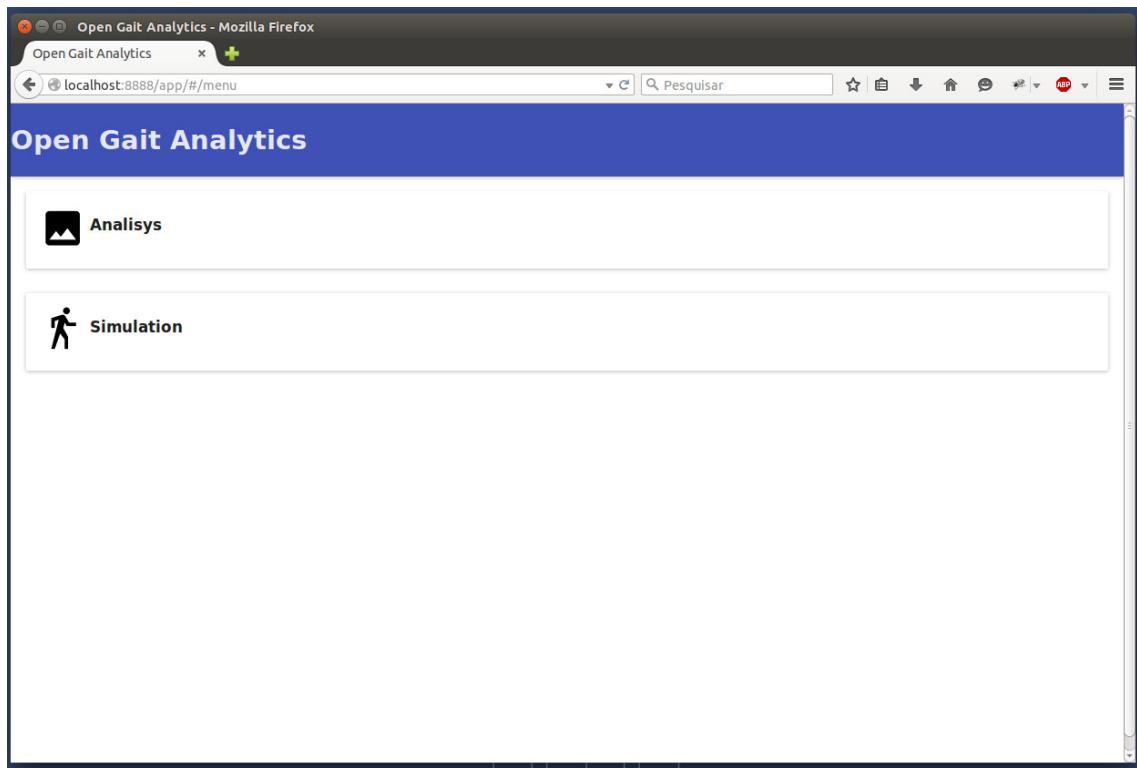


Figura 40 – Tela de seleção dos módulos.

4.1 MÓDULO DE ANÁLISE

Neste fase o software conta apenas com análise de movimentos, com sinais oriundos de marcadores passivos de superfície, captados por câmeras, usando o software *QTM*. No *QTM* é feita a conversão dos dados para o formato *Matlab* que é reconhecido pelo sistema

construído.

A primeira tela deste módulo pode ser vista na Figura 41. Este tela apresenta a listagem de pacientes, cadastrados no sistema. O botão abaixo a direita, é a função para adicionar novos pacientes.

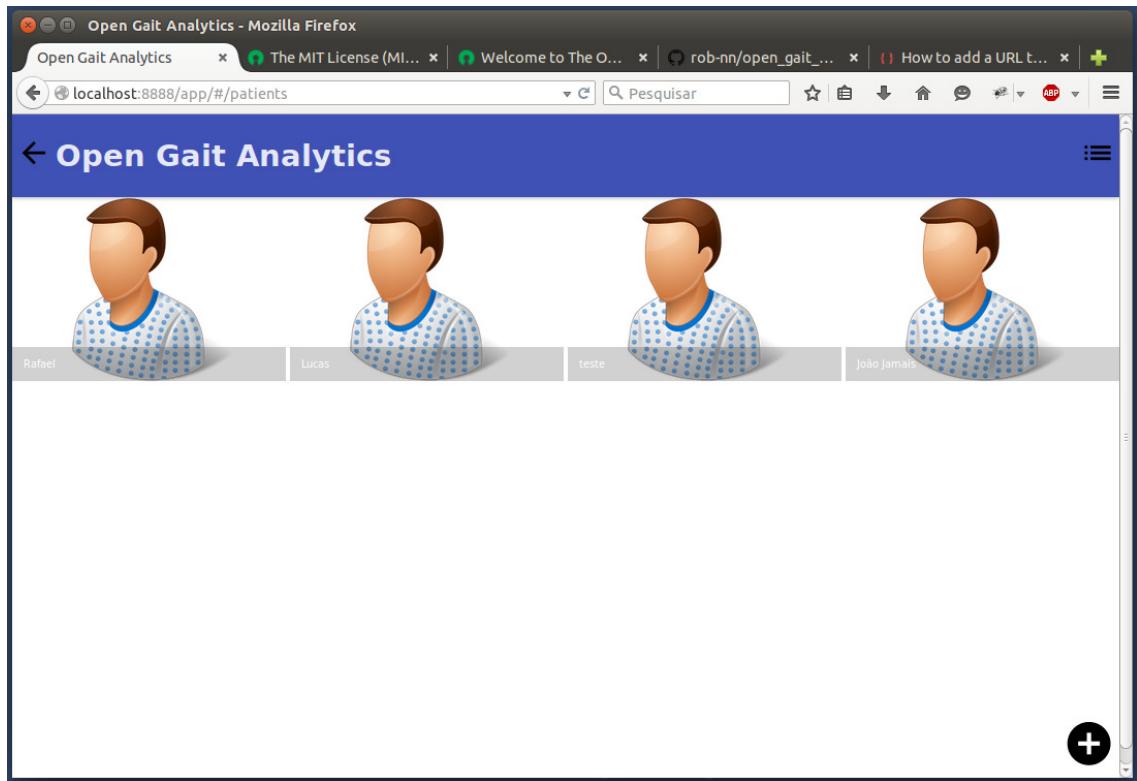


Figura 41 – Tela com a listagem de pacientes.

A Figura 42 mostra as informações do paciente que devem ser preenchidas ao se executar a função adicionar paciente. Esta tela é uma adaptação da ficha de avaliação que está na obra Maraes (1999).

The screenshot shows a Mozilla Firefox browser window with the title 'Open Gait Analytics - Mozilla Firefox'. The address bar displays 'localhost:8888/app/#/patient_new'. The main content area is titled 'Open Gait Analytics' and contains a form for entering patient information. The fields and their values are:

- Name: Roberto Aguiar Lima
- Sex: Male (radio button selected)
- Address: Av. Central, Bl. 191 N. 10, Núcleo Bandeirante
- City: Brasília
- State: DF
- ZIP Code: 71720005
- Phone: 61 81543841
- Birth: 1977-11-19
- Race: White (radio button selected)
- Profession: Programador
- Marital Status: Single (radio button selected)
- Weight: 87
- Height: 1.75

A save icon (floppy disk) is located in the bottom right corner of the form area.

Figura 42 – Informações do paciente.

Ao se selecionar um paciente da tela mostrada na Figura 41, a tela da Figura 43 aparece. No caso em questão, nenhuma coleta de dados foi carregada para o paciente. Logo, o próximo passo é adicionar uma nova amostra de marcha. O usuário deve então informar a descrição da coleta e a data que a mesma ocorreu e salvar estas informações, conforme a Figura 44.

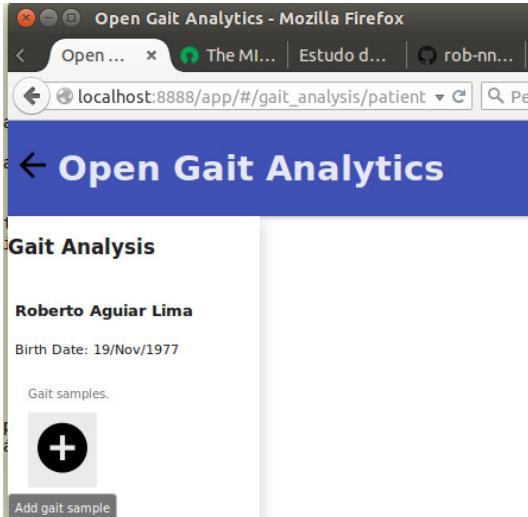


Figura 43 – Tela inicial da dados coletados do paciente.

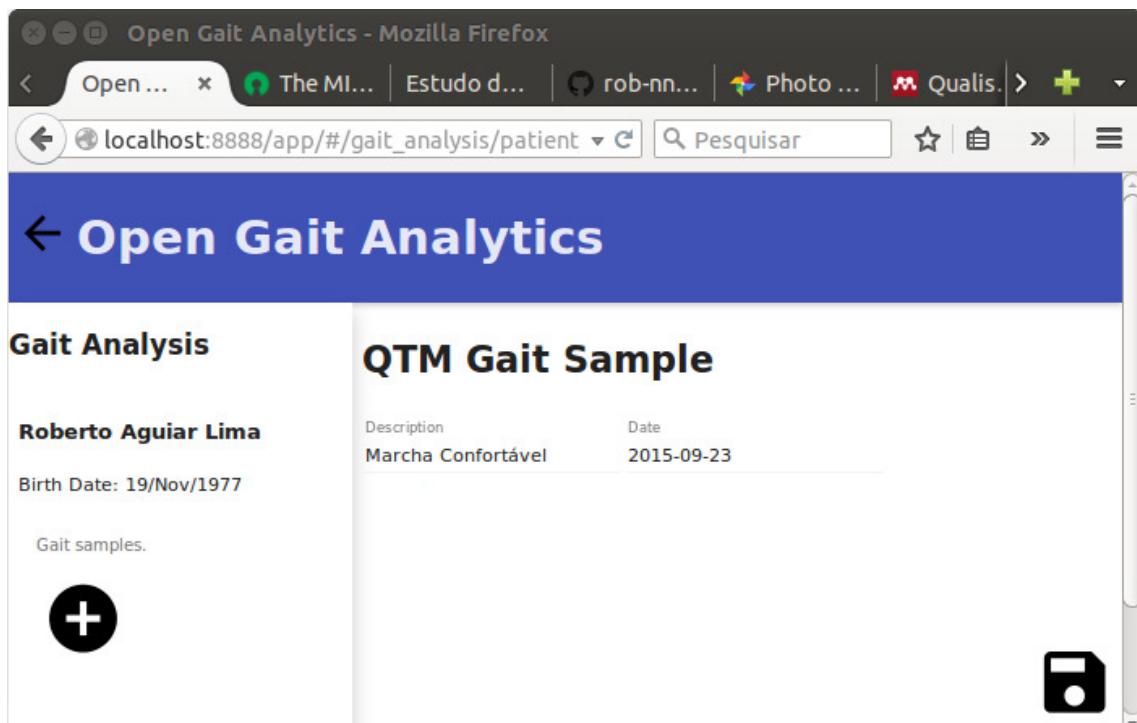


Figura 44 – Inclusão de amostra de marcha

Depois de salva as informações o sistema pede para que o usuário selecione o arquivo proveniente do *QTM*, conforme a Figura 45. Depois de selecionado o arquivo com os dados da marcha, seus dados são mostrados para o usuário, conforme a Figura 46.

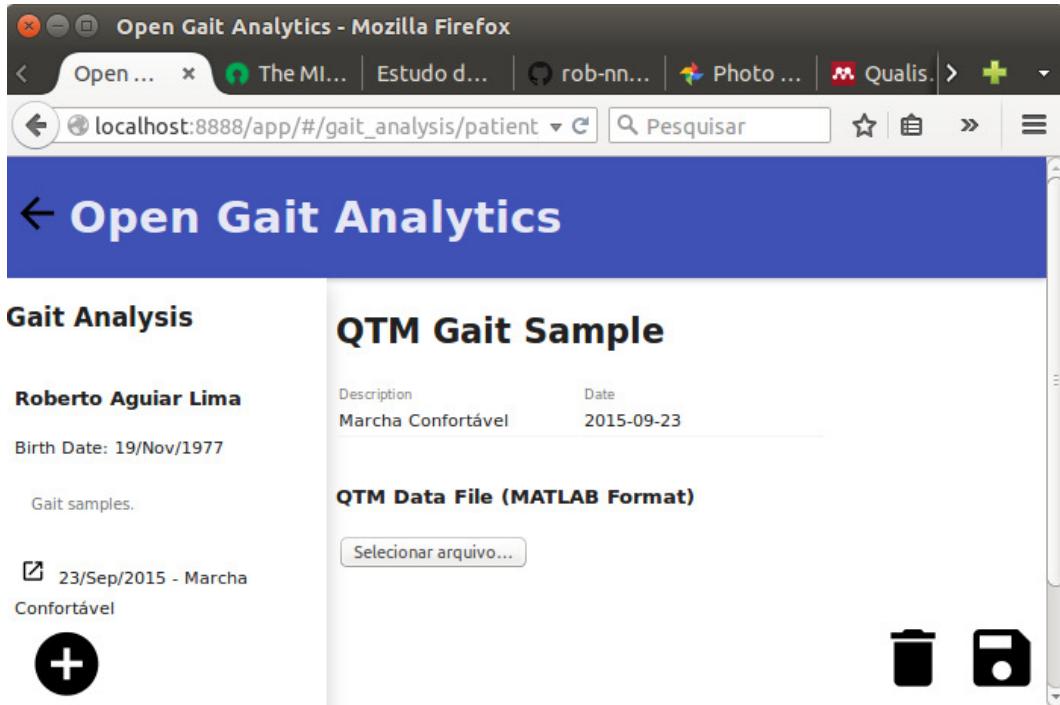


Figura 45 – Seleção do arquivo no formato *MATLAB* proveniente do *QTM*.

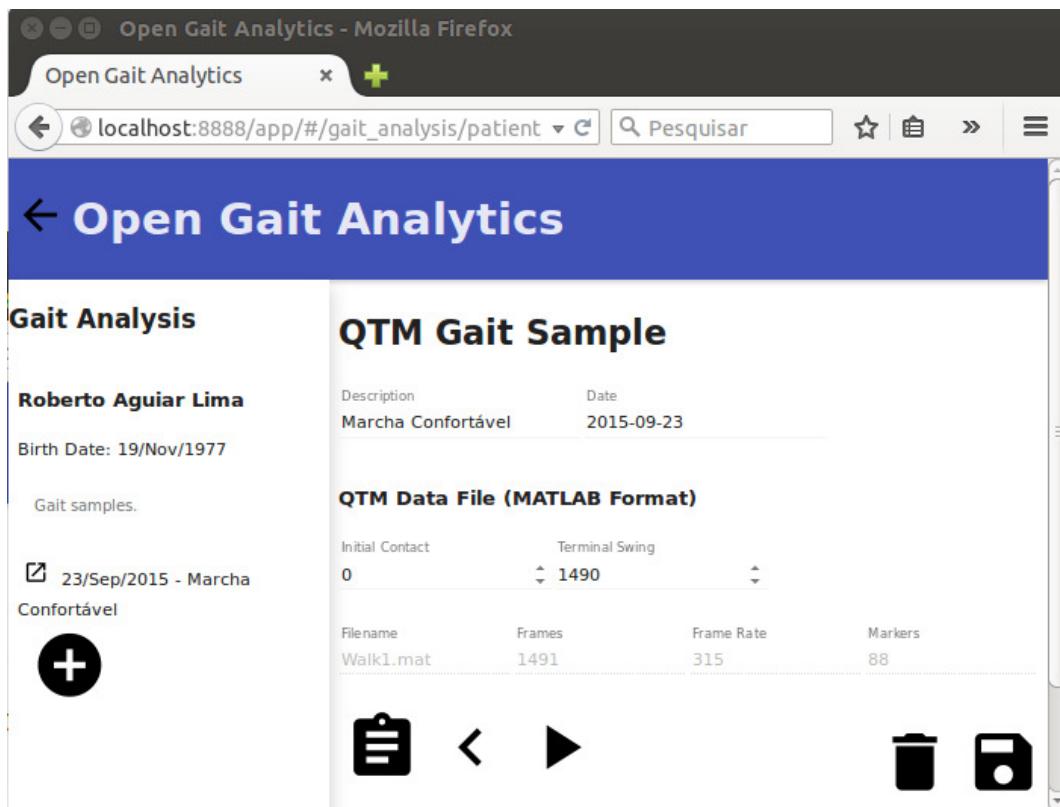


Figura 46 – Dados do arquivo provenientes do *QTM*.

Neste momento, se o usuário quiser visualizar uma animação dos dados, basta

clicar na seta negra que aponta para a direita que uma animação será mostrada conforme a Figura 47. Esta animação foi construída usando a tecnologia *ThreeJS*, resumida na seção 2.4.4. Uma das grandes vantagens desta característica do software em relação ao *QTM*, que também a possui, é o fato de que a animação está rodando num *browser web* moderno, ou seja, qualquer um com um *browser* assim pode vê-la sem precisar do *QTM* instalado. Além do mais, como o projeto pode continuar, fica a critério dos usuários decidirem que novas características seriam interessantes, não somente nas animações, mas em todo o software.

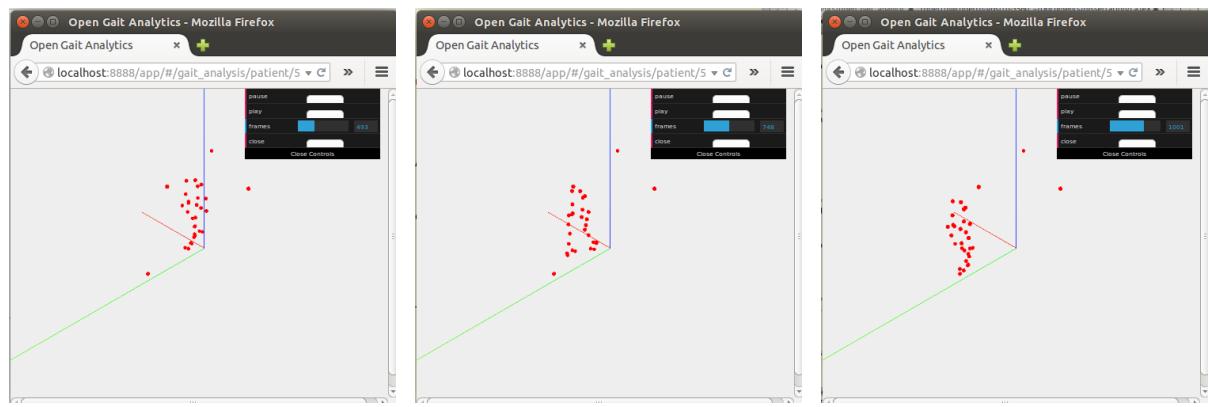


Figura 47 – Animação dos marcadores em 3D. A primeira figura mostra o contato inicial de uma perna. A segunda um momento no período de instância. A terceira o balanço terminal.

A tela da animação também possui controle de perspectivas (Figura 48), controle de *zoom* (Figura 49), e controle *pan* (Figura 50). Também foram implementados, até o momento, botões de *play*, *pause*, fechar e um contador de *frames* (Figura 51).

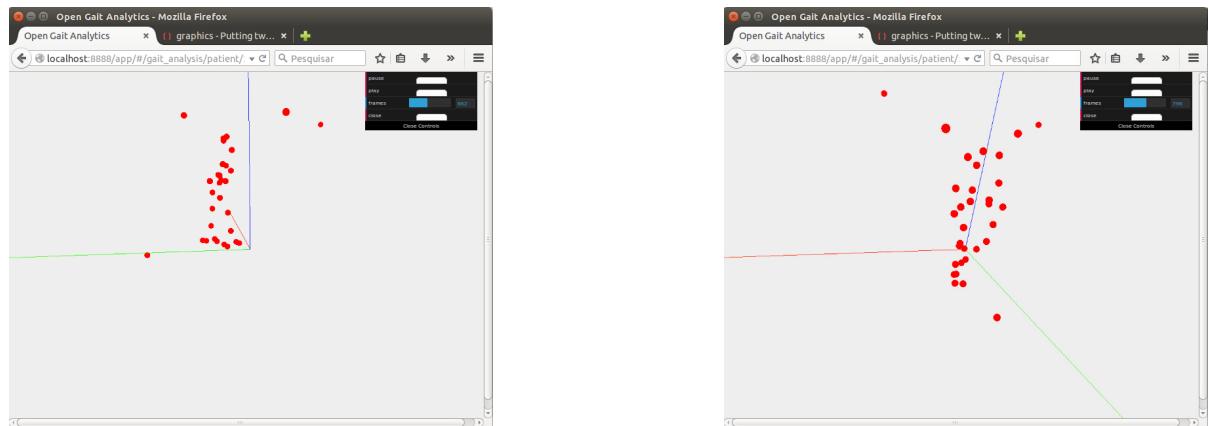


Figura 48 – Controle de perspectivas. Primeira figura mostra uma perspectiva lateral do paciente. A segunda figura mostra uma perspectiva diagonal.

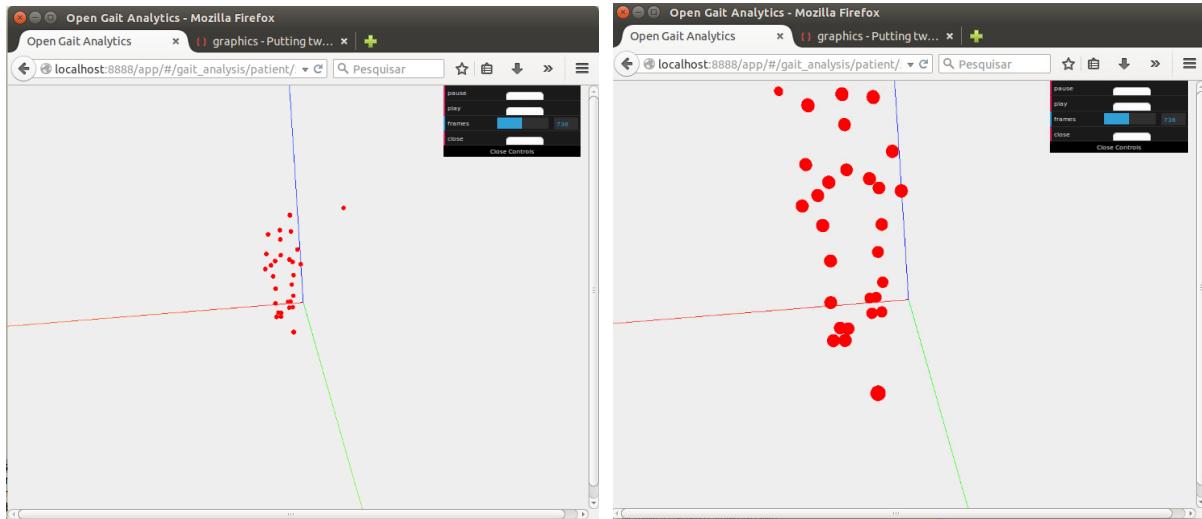


Figura 49 – Controle de *zoom*. Primeira figura *zoom out*. Segunda figura *zoom in*.

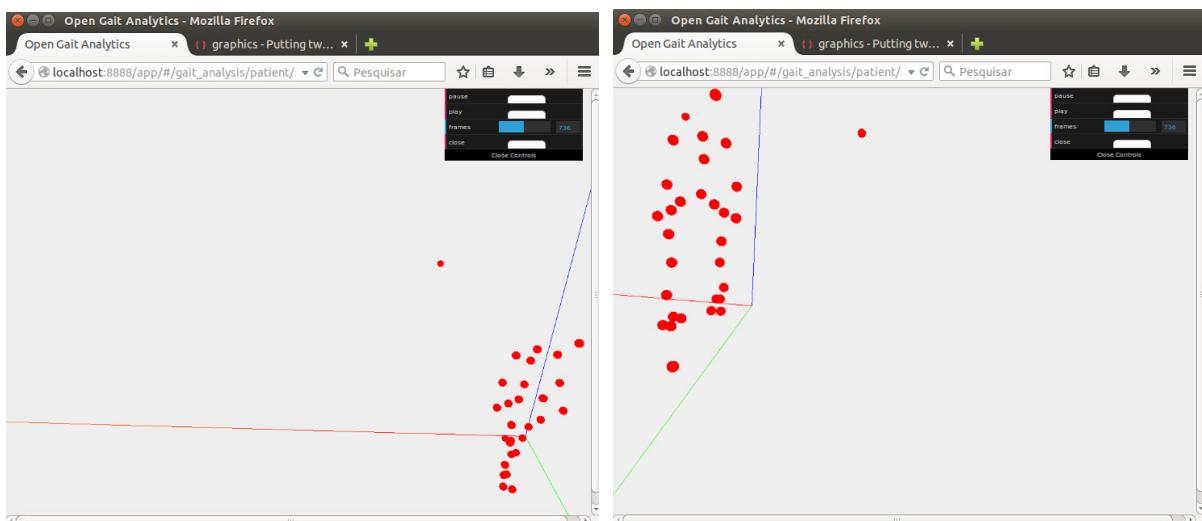


Figura 50 – Controle *pan*. Primeira figura mostra o paciente mais a direita e embaixo. A segunda figura mostra o paciente mais a esquerda e acima.



Figura 51 – Controles da animação.

É de fundamental importância que o usuário configure os parâmetros *Initial Contact* e *Terminal Swing* mostrados na Figura 46. Sem estes parâmetros os gráficos, vão mostrar os sinais nas fases erradas do ciclo de marcha. A técnica que se recomenda é

inicializar a animação e quando o usuário perceber o *initial contact*, pressionar o botão *pause* e anotar o *frame*. Fazer a mesma coisa para o *terminal swing*.

Outra opção disponível na Figura 46 é a opção *Markers*, esta opção permite nomear os marcadores e visualizar sua progressão espacial. A Figura 52 mostra o resultado de se selecionar esta opção. Ao clicar no botão ao lado de algum marcador, sua progressão no espaço é mostrada num gráfico como o da Figura 53. O domínio é o percentual do ciclo de marcha, já a imagem são dados espaciais brutos oriundos do *QTM*.

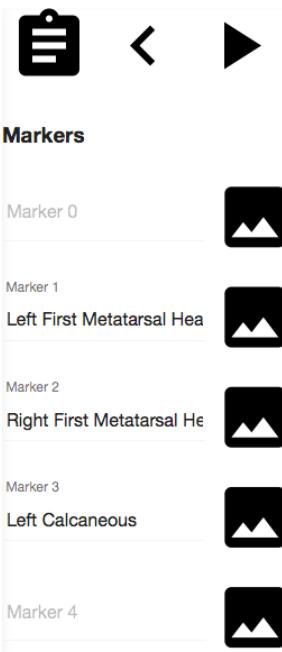


Figura 52 – Opção *markers*.

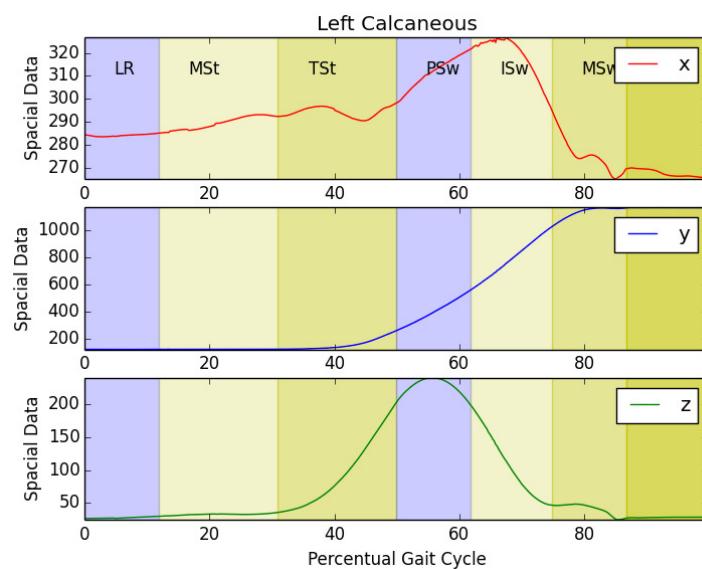


Figura 53 – Progressão espacial de um marcador.

A nomeação dos marcadores, não é uma tarefa trivial. Para isso foi criada uma ferramenta dentro da animação para ajudar com esta tarefa. Primeiro, deve-se entrar na animação, depois pausá-la, e posicionar a visualização de uma forma que ajude a detectar o marcador procurado. Veja a Figura 54, nela um marcador foi clicado com o *mouse*, o marcador ficou azul e ao seu lado ele mostra o índice 30. Agora é só voltar na opção de marcadores, procurar o índice 30 (*Marker 30*) e colocar o nome desejado. No caso deste marcador o nome é joelho esquerdo (*left knee*, Figura 55). Agora para o sistema o marcador 30 é sempre o joelho esquerdo (Figura 56).

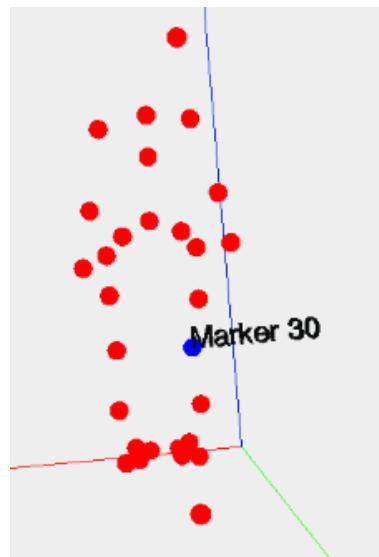


Figura 54 – Seleção de um marcador pelo *mouse*.

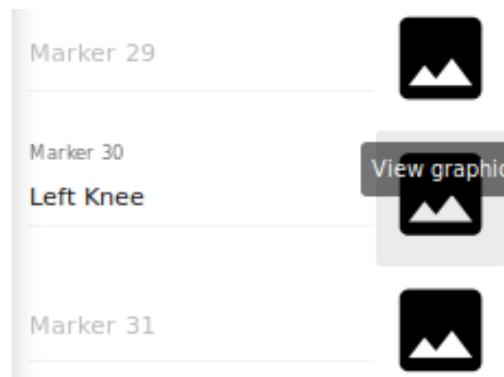


Figura 55 – Renomeando um marcador.

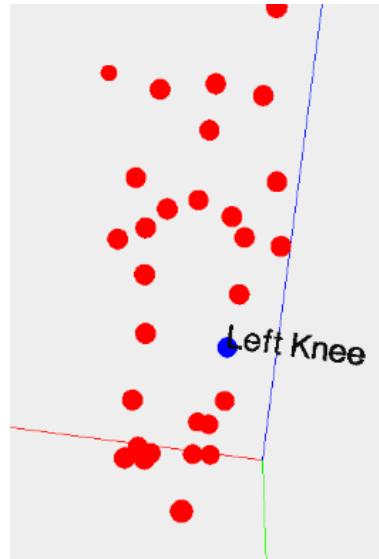


Figura 56 – Animação mostrando o marcador renomeado.

Outra funcionalidade importante é o gerador de ângulos. Esta opção está disponível na Figura 46. E após selecionada é mostrada na Figura 57. Para se gerar um ângulo, o usuário necessita selecionar a opção de inclusão de ângulo e preencher os dados da Figura 58. O usuário precisa indicar a origem do ângulo, por exemplo, o joelho, o componente A, por exemplo, algum músculo da coxa, e o componente B, por exemplo, a tibia. Estes pontos poderiam representar o ângulo de um joelho, por exemplo.

Description	Origin	Component A	Component B
Right Knee Angle	Right Trochanter	Right Knee	Right Tibia
Left Knee Angle	Left Trochanter	Left Knee	Left Tibia

Figura 57 – Opção de visualização e criação de ângulos.

New Angle

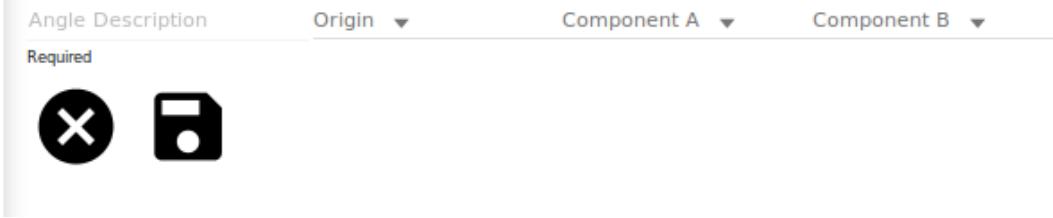


Figura 58 – Inclusão de um novo ângulo.

Depois dos ângulos criados é possível ver seus valores durante o ciclo de marcha ou suas velocidades angulares, conforme as Figuras 59 e 60. Os ângulos são criados conforme as fórmulas que estão na Seção 2.6. O número de pontos usados para o cálculo depende dos parâmetros informados e da quantidade de *frames* por segundo configurada no *QTM*.

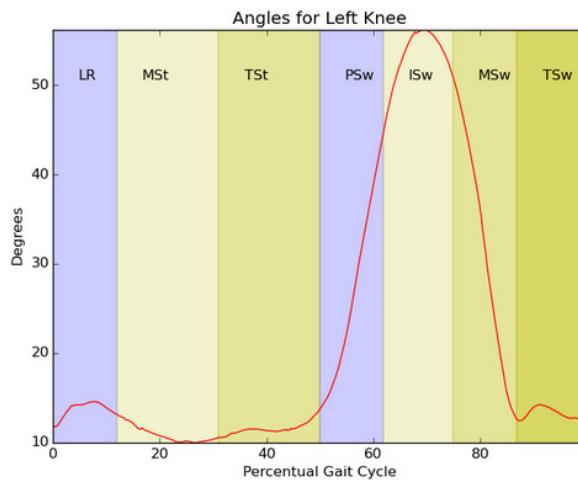


Figura 59 – Ângulo de um joelho durante o ciclo de marcha.

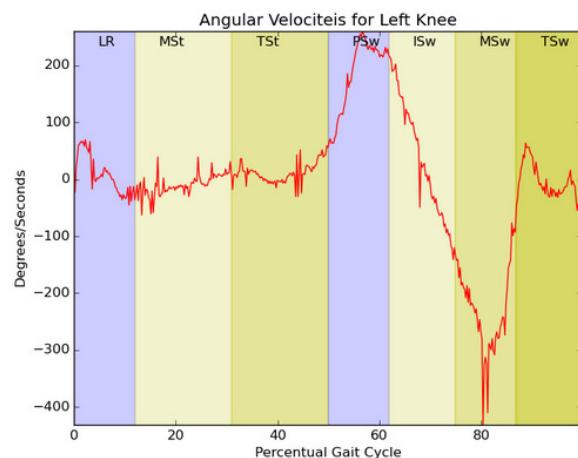


Figura 60 – Velocidades angulares de um joelho durante o ciclo de marcha.

4.2 MÓDULO DE SIMULAÇÃO

Este módulo é sem dúvida o que mais vai contribuir para os pesquisadores, pois estes contarão com a base integrada gerada pelo módulo de análise e com estas informações será possível realizar simulações de sinais e classificações. A versão atual do software ainda é muito pobre neste quesito, mas o objetivo neste momento é mostrar a viabilidade de se usar algoritmos de sistemas inteligentes, integrados na base que vai sendo gerada. Acredita-se que a ferramenta vai ser mais apreciada por pesquisadores que tem como formação a área de saúde, pois estes não precisarão de todas as técnicas exigidas para fazer simulações num ambiente como o do *MATLAB*.

A primeira funcionalidade de simulação que foi implantada foi a simulação pela *CMAC*, em que esta RNA foi descrita na seção 2.5. Ao se selecionar o módulo simulação, a tela da Figura 61 é apresentada. A opção *CMAC* já aparece selecionada. Para efeitos da demonstração das funcionalidades deste módulo foi realizada uma simulação das velocidades angulares de um joelho, a partir de outros sinais disponíveis.

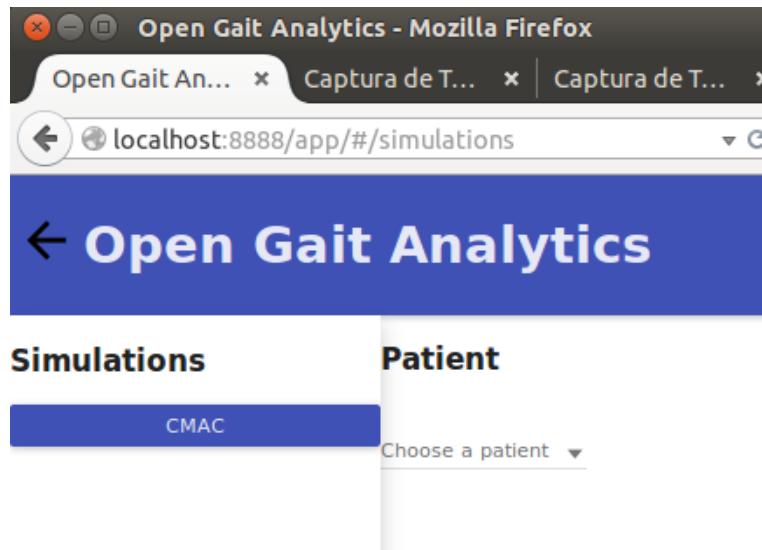


Figura 61 – Módulo de simulação.

Após ser selecionado o nome de um paciente, deve-se selecionar uma amostra de ciclo de marcha (Figura 62). Após a seleção do ciclo de marcha, uma lista com vários sinais são mostrados, basicamente os sinais são coordenadas de posições de marcadores, ângulos e velocidades angulares. As Figuras 63 e 64 mostram uma fração dos sinais possíveis para seleção. Ao se selecionar um sinal de entrada é necessário também informar sua quantização.

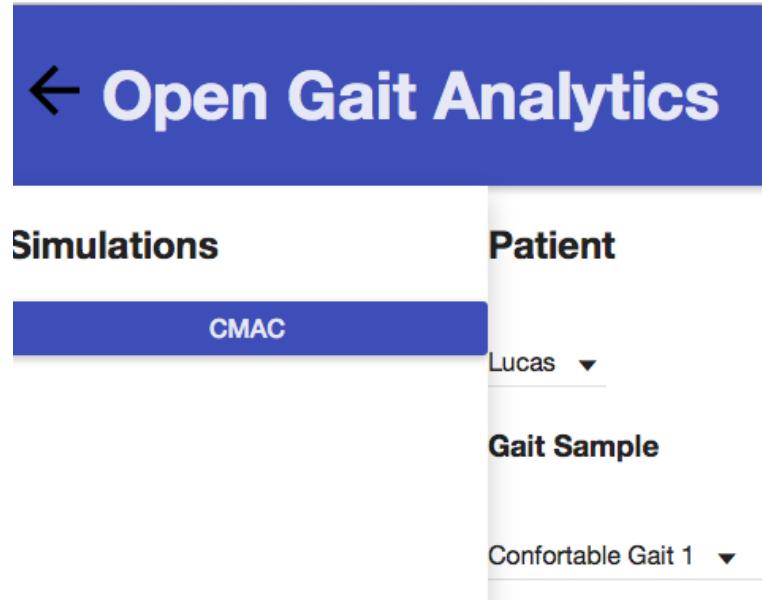


Figura 62 – Seleção do ciclo de marcha.

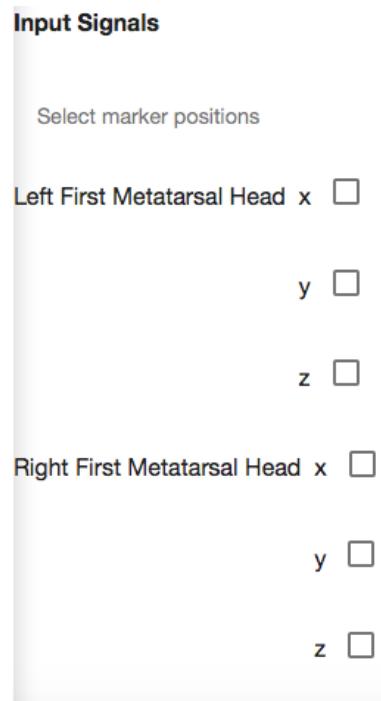


Figura 63 – Sinais de entrada para a *CMAC*. No caso posições num plano 3D de marcadore.

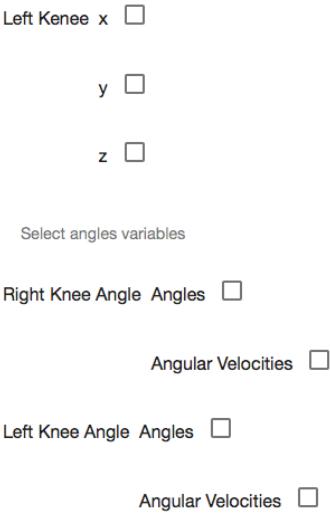


Figura 64 – Sinais de entrada para a *CMAC*, ângulos e velocidades angulares.

A Figura 65 mostra uma configuração que gera a saída da Figura 66. O gráfico mostrado é a saída da RNA *CMAC*. A Figura 67 mostra o erro quadrado médio da execução da simulação ao longo das iterações.

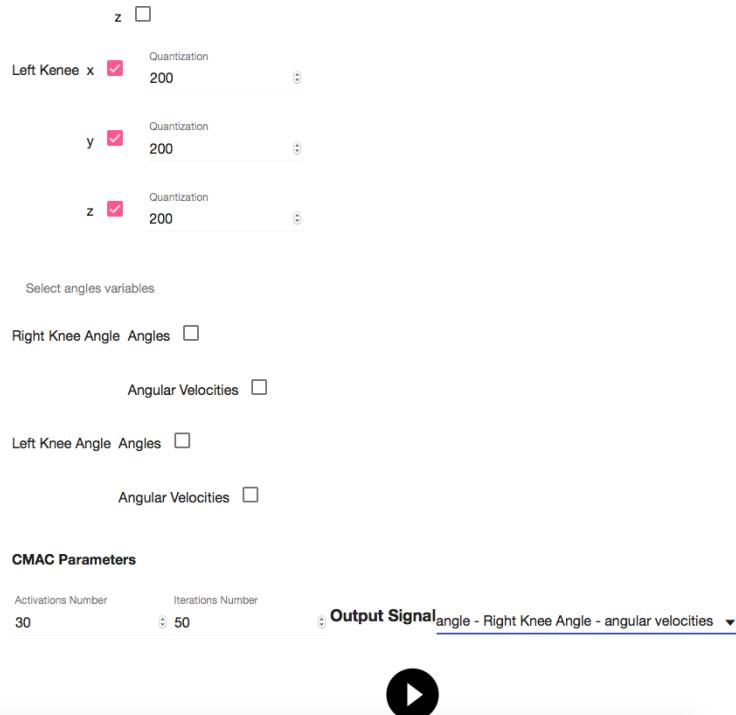


Figura 65 – Exemplo de configuração para uma simulação usando *CMAC*.

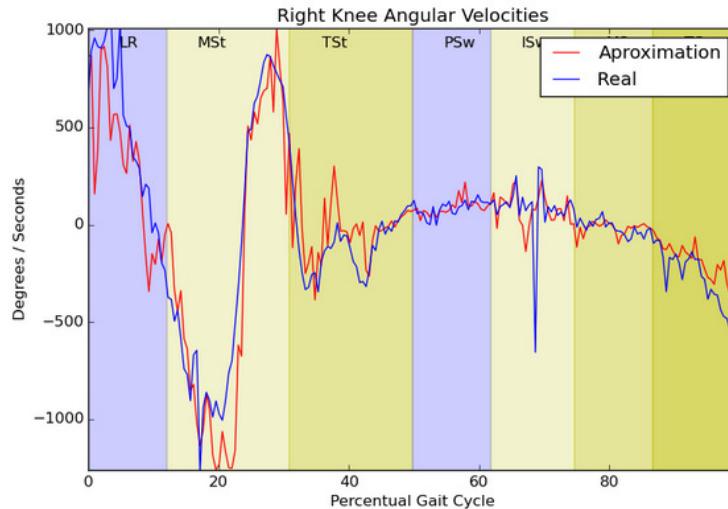


Figura 66 – Resultado da simulação.

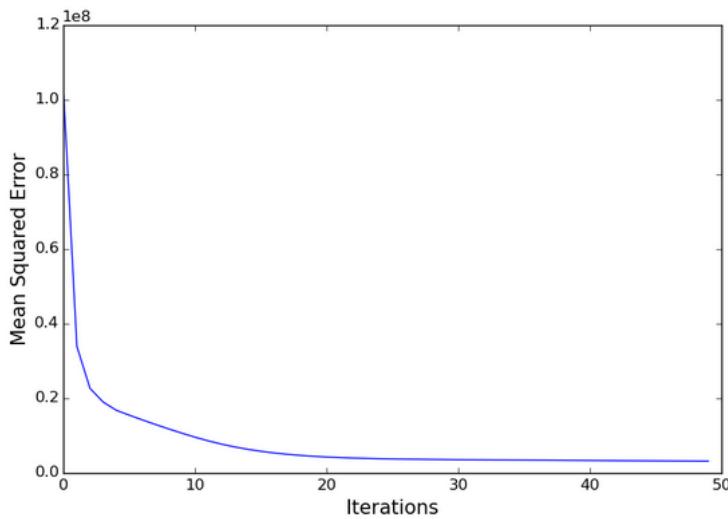


Figura 67 – Erro quadrado médio em cada iteração da simulação.

4.3 USO DO SISTEMA EM DISPOSITIVOS MÓVEIS

Uma das maiores vantagens nas tecnologia escolhidas para compor a camada web, é sua total compatibilidade com dispositivos móveis capazes de executar *browsers* modernos como *Firefox*, *Chrome* ou *Safari*. O *angular-material* já apresenta comportamentos muito bons em dispositivos de pequenas telas. Veja um exemplo nas Figuras 68 e 69 da aplicação rodando no *browser Firefox*, num dispositivo *Motorola Xoom*. A Figura 70 mostra a aplicação rodando em um *iPhone4* com *browser Safari*.

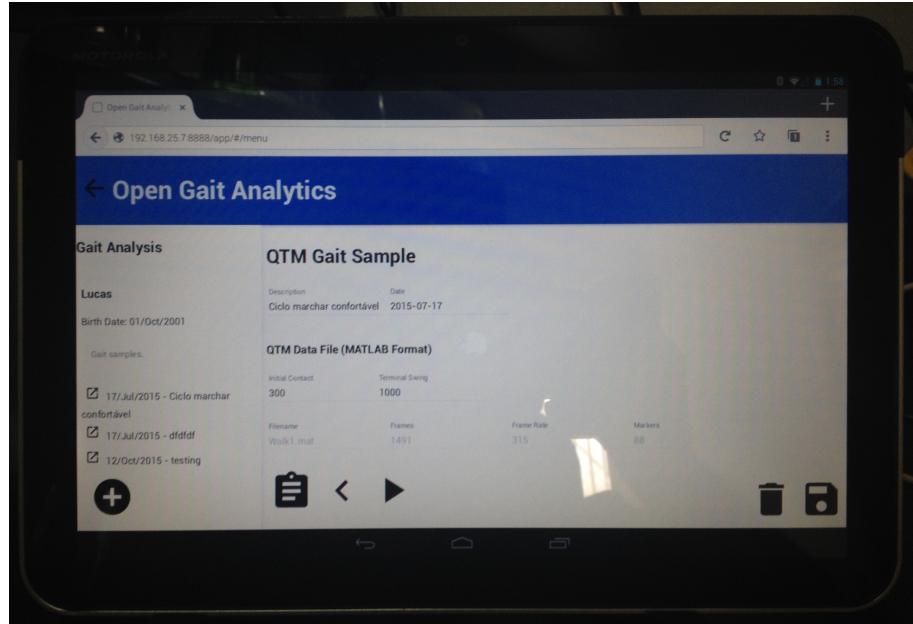


Figura 68 – Aplicação *Open Gait Analytics* rodando num *browser Firefox* e dispositivo *Motorola Xoom*.

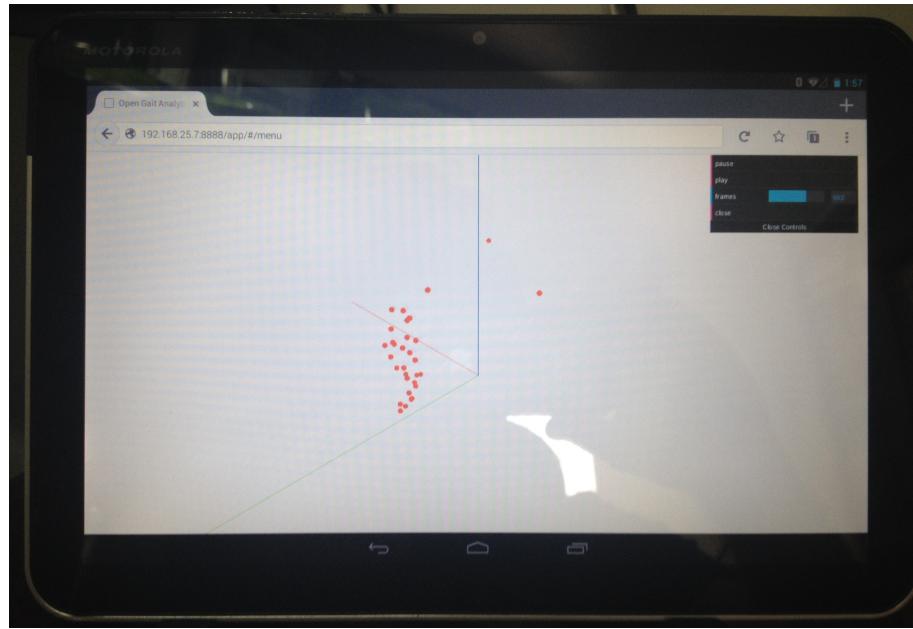


Figura 69 – Animação de uma amostra de marcha rodando num *browser Firefox* e dispositivo *Motorola Xoom*.



Figura 70 – Aplicação rodando num *iPhone 4* com *browser Safari*.

Para frisar como a aplicação se adapta inteligentemente ao dispositivo que está rodando, compare a Figura 68 com a Figura 71. Veja que a barra lateral desaparece na tela menor do *iPhone 4*. Em cima da tela aparece a opção *open side* que quando clicada mostra a barra lateral por cima do formulário.

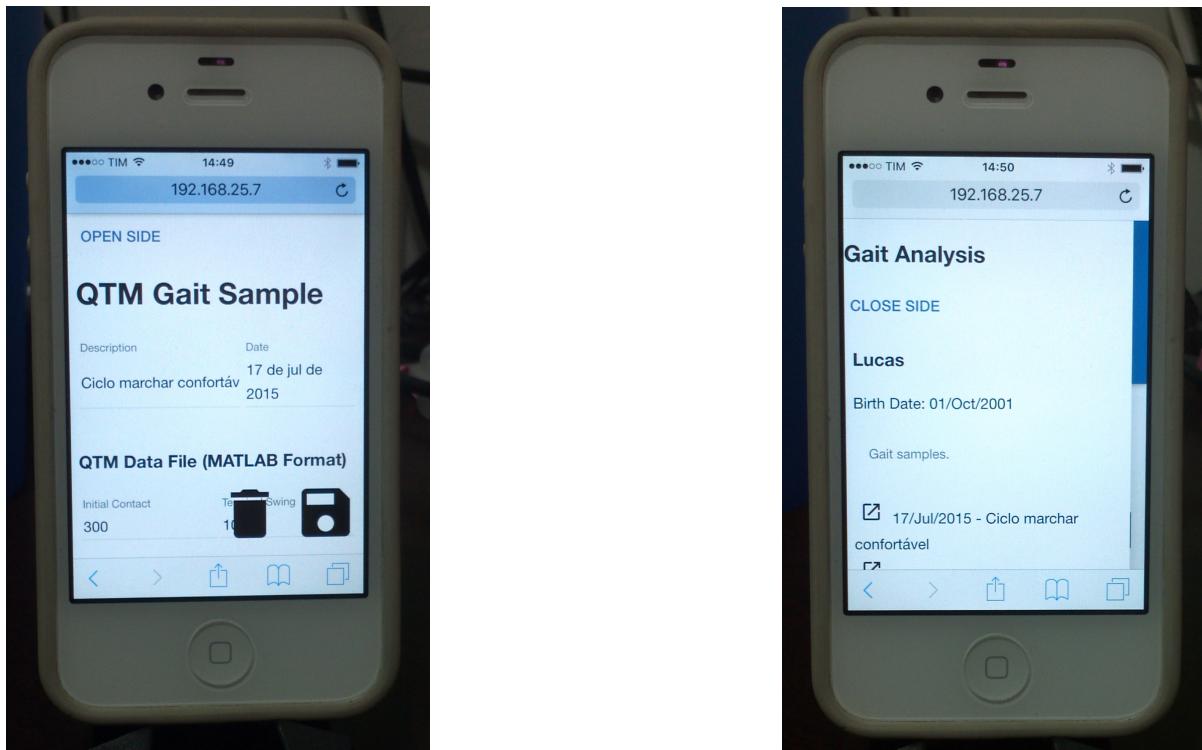


Figura 71 – Adaptação da aplicação em telas pequenas.

Para o profissional de saúde, este é um recurso a mais, pois agora do seu próprio celular e em qualquer lugar ele poder ver dados dos seus pacientes.

5 DISCUSSÃO E CONCLUSÃO

Há vários softwares de análise de marcha no mercado, mas a ideia de criar um totalmente na *web* tem suas vantagens. A mais óbvia é a disponibilidade, já que a aplicação fica num servidor na *internet*, isto é, o usuário só precisa acessar um *site* com um *browser*. Outra não tão óbvia e que diz respeito principalmente ao módulo de simulação, é a possibilidade de escalar a aplicação para necessidades de processamento gigantescas, usando infraestrutura como serviço de fornecedores como o *Microsoft Azure* ou *Amazon Webservices*. Como a camada *web* se comunica via *HTTP* no estilo *REST*, usando uma *facade* para isto, basta fazer a *facade* do módulo de simulação apontar para uma *web farm* alocada sob demanda num dos serviços mencionados. A vantagem óbvia é que quem serve a aplicação, pode alocar estes serviços sob demanda, não necessitando possuir um Centro de Processamento de Dados (CPD) caríssimo. O custo é repassado ao cliente que deseja usar os serviços de simulação que demandam muito poder de processamento.

Outro ponto interessante com a implantação da aplicação é a criação de uma base de dados de marcha humana, que pode receber dados de todo o globo. As vantagens para pesquisadores da área de análise de marcha seriam inimagináveis.

Claro que nada disto é gratuito, os responsáveis pelo projeto terão de almejar meios para produzi-lo, achar nichos de mercado e colocá-lo em produção.

Ressalta-se, novamente, que o software que está sendo entregue não está pronto para a produção. O prazo disponível para desenvolvê-lo, inclusive adquirindo conhecimentos sobre muitas das tecnologias adotadas, foi de aproximadamente cinco meses. Isto ocorreu porque durante o programa de mestrado resolveu-se mudar o tema, devido a uma série de intempéries. No entanto, o software foi feliz em mostrar a integração de vários componentes nas diferentes camadas da aplicação. O ato de stressar a arquitetura é muito importante para se avaliar a viabilidade técnica de um projeto de engenharia de software.

Outro problema com o software entregue, é seu modesto poder de processamento com relação a simulações. Facilmente uma simulação em um grande *Data Center* pode demorar dias. A solução para este problema é desenvolver um método para disparar a simulação assincronamente, bem como criar uma tela de gestão da execução da mesma. Para isso, o software terá de ser integrado a componentes de computação distribuída como *Apache Spark* ou o *Hadoop*, fazendo uso de infraestrutura como serviço conforme mencionado anteriormente. O potencial para este projeto se tornar algo inovador e, principalmente, muito útil na área de marcha humana é imenso.

Talvez o objetivo que tenha sido mais prejudicado, foi a implantação do método ágil. Como o método escolhido foi o *SCRUM* e não foi possível criar as reuniões diárias

(*daily scrum*), a dinâmica da equipe não foi a mesma em que o autor teve a oportunidade de trabalhar com outras equipes. Recomenda-se também, que uma equipe de especialistas clínicos e pesquisadores da área da marcha humana sejam adicionados ao projeto e ajudem ao *product owner* do projeto a definir novas funcionalidades para o sistema. Isso certamente vai acelerar a adoção do software por estes profissionais, já que são as necessidades deles que serão atingidas. Talvez um projeto de *crowdfunding* possa trazer estes profissionais. É comum nestes projetos os clientes pedirem funcionalidades para o software. O problema é que este é um software para um nicho muito especializado, pode ser que não seja uma boa ideia.

A questão dos testes também foi um pouco prejudicada, mas não abandonada. Na verdade esta foi uma escolha do autor, que devido ao pouco tempo para desenvolvimento, preferiu dar ênfase na produção de novas características. Mesmo assim todos os principais componentes, como a *web API* e os componentes de telas da análise de marcha, possuem testes automatizados. A consequência é que *bugs* menores como listas de seleção podem aparecer fora dos locais adequados, o componente de controle de animação às vezes fica com um tamanho inadequado, mas não é nada que com os recursos adequados não se resolva.

Vale lembrar também que o projeto não vai se limitar a análise de movimento. Outros métodos de coleta de dados para análise serão inseridos ao longo do tempo, por exemplo, IMU.

Concluindo, os objetivos foram alcançados, foi definida uma metodologia ágil, foi criado um modelo de arquitetura, um software foi construído, integrado com os principais componentes e, na medida do possível testados, e mais uma vez, frisando que toda a arquitetura foi estressada. Além disso, a aplicação ainda apresenta características para serem executadas em dispositivos móveis. Uma ferramenta de análise de movimento com várias funcionalidades foi construída, assim como uma simulação usando a *C MAC*. Também o código fonte com todo o histórico de desenvolvimento do projeto, inclusive histórias do usuário, estão no site (https://github.com/rob-nn/open_gait_analytics), lembrando que o código está sob licença MIT (Anexo B), e qualquer um pode usá-lo conforme a necessidade.

6 TRABALHOS FUTUROS

Como este foi um projeto com um intuito de plantar uma semente, não faltarão trabalhos futuros para complementá-lo, aumentá-lo ou mesmo expandi-lo para outras áreas além da análise de marcha. Além disso, o projeto conta com um *backlog* de produto sempre evoluindo, o que é uma fonte de trabalhos futuros constante.

1. O trabalho mais urgente a ser feito é usar a versão atual, testá-la o máximo possível, corrigir os *bugs* encontrados e disponibilizá-la na *web*;
2. Inserir o padrão ouro nos gráficos para efeito de comparação;
3. Criar no módulo de simulação uma ferramenta para modelagem de sinais de entrada, métodos de processamento e sinais de saída, baseados nos dados da base de documentos;
4. Habilitar o protocolo *HTTP Auth* nas requisições feitas a *web API*;
5. Habilitar protocolo *HTTPS* entre servidor *web* e *browser* cliente;
6. Implementar um detector automático de ciclo de marcha, assim não será necessário o usuário informar o início e o fim do ciclo;
7. Permitir cadastrar protocolos de coleta por câmeras e fazer a detecção automática dos mesmos, assim o usuário não necessitará nomear marcadores;
8. Permitir coletar dados de plataforma de força e criar gráficos;
9. Permitir coletar dados de *IMUs* e criar gráficos;
10. Permitir coletar dados de *EMGs* e criar gráficos;
11. Permitir coletar dados de eletrogoniômetros;
12. Criar suporte a várias língua, começando com português e inglês;
13. Implementar outros algoritmos de sistemas inteligentes, como *Principal Component Analysis (PCA)*, *Kmeans*, *Support Vector Machine (SVM)*, *Multi Layer Perceptron (MLP)* (HAYKIN, 1998), entre outros, integrando estes algoritmos a ferramenta de modelagem de sinais, permitindo se fazer classificações e regressões. Por exemplo, usando-se *SVM* é possível fazer a detecção de quedas, ou usando o *Kmeans* é possível detectar os momentos distintos no uso de uma plataforma de força. Aqui o que vai imperar é a criatividade do pesquisador, que terá nas mãos uma ferramenta visual para fazer estas simulações. A evolução desse módulo acontecerá quando as

simulações forem úteis o suficiente para auxiliar nos diagnósticos de patologias na marcha. Com a tecnologia atual de aprendizado de máquina, as possibilidades são muito atrativas.

14. Integrar o módulo de simulação com plataformas de infraestrutura com serviço, usando para isso ferramentas como *Hadoop* ou *Spark*;
15. Tornar a função de execução de simulações assíncronas;
16. Criar um módulo gestor da execução das simulações, com opções de parar, pausar, continuar, alocar mais recursos, enfileiramento;
17. Criar uma ferramenta para extração de dados com possibilidade de criar filtros nos dados espaciais.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALBUS, J. A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC). ... *Systems, Measurement, and Control*, n. SEPTEMBER, p. 220–227, 1975. Disponível em: <<http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?articleid=1402197>>. 11, 43, 44, 45
- ALBUS, J. Data Storage in The Cerebellar Model Articulation Controller (CMAC). ... *Systems, Measurement, and Control*, 1975. Disponível em: <<http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?articleid=1402199>>. 43
- ALBUS, J. Mechanisms of planning and problem solving in the brain. *Mathematical Biosciences*, v. 293, 1979. Disponível em: <<http://www.sciencedirect.com/science/article/pii/0025556479900634>>. 43
- ALBUS, J. S. A robot conditioned reflex system modeled after the cerebellum. *Proceedings of the December 5-7, 1972, fall joint computer conference, part II on - AFIPS '72 (Fall, part II)*, ACM Press, New York, New York, USA, p. 1095, 1972. Disponível em: <<http://portal.acm.org/citation.cfm?doid=1480083.1480144>>. 43
- ALBUS, S. A Theory of Cerebellar Function. v. 10, p. 25–61, 1971. 43
- ANDRADE, J. A. A. et al. Proposta de estudo de redes neuro- fuzzy para aplicações em um controle de prótese ativa transtibial. In: *XXIII Congresso Brasileiro em Engenharia Biomédica*. [S.l.]: CBEB, 2014. 43
- BAKER, R. The history of gait analysis before the advent of modern computers. *Gait and Posture*, v. 26, n. 3, p. 331–342, 2007. ISSN 09666362. 21
- BECK, K. *Extreme Programming Explained: Embrace Change*. 2. ed. [S.l.]: Addison-Wesley Professional, 2004. ISBN 9780321278654. 29
- BECK, K. et al. *Manifesto para Desenvolvimento Ágil de Software*. 2001. Disponível em: <<http://www.agilemanifesto.org/iso/ptbr/>>. 27
- BEYNON, S. et al. Correlations of the Gait Profile Score and the Movement Analysis Profile relative to clinical judgments. *Gait and Posture*, Elsevier B.V., v. 32, n. 1, p. 129–132, 2010. ISSN 09666362. Disponível em: <<http://dx.doi.org/10.1016/j.gaitpost.2010.01.010>>. 19
- BRANAS, R. *AngularJS Essentials*. Birmingham: Packt Publishing Ltd., 2014. ISBN 978-1-78398-008-6. 35
- CHODOROW, K. *MongoDB: The Definitive Guide*. 2. ed. [S.l.]: O'Reilly Media, Inc., 2013. ISBN 9781449344689. 40
- CIPPITELLI, E. et al. Kinect as a Tool for Gait Analysis: Validation of a Real-Time Joint Extraction AlgorithmWorking in Side View. *Sensors*, v. 15, n. 1, p. 1417–1434, 2015. ISSN 1424-8220. Disponível em: <<http://www.mdpi.com/1424-8220/15/1/1417/>>. 19
- CLIPARTS. *Leg Clip Art*. 2015. Disponível em: <<http://cliparts.co/legs-clip-art>>. 11, 48

- COHN, M. *Users Stories Applied*. Crawfordsville: Addison Wesley, 2004. ISBN 978-0-321-20568-1. 11, 32, 33
- DAYLEY, B. *Node.js, MongoDB, and AngularJS Web Development*. [S.l.]: Addison-Wesley Professional, 2014. 11, 40, 41, 42
- DIRKSEN, J. *Learning Three.js - the JavaScript 3D Library for WebGL*. 2. ed. [S.l.]: Packt Publishing, 2015. ISBN 9781784392215. 38
- DUHAMEL, a. et al. Statistical tools for clinical gait analysis. *Gait and Posture*, v. 20, n. 2, p. 204–212, 2004. ISSN 09666362. 18
- EDWARDS, L. *Calculus*. 9. ed. Belmont: Brooks / Cole, 2006. 784 p. ISBN 978-0-547-16702-2. 47
- FERREIRA, J. a. P.; CRISOSTOMO, M. M.; COIMBRA, a. P. Human gait acquisition and characterization. *IEEE Transactions on Instrumentation and Measurement*, v. 58, n. 9, p. 2979–2988, 2009. ISSN 00189456. 19
- FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — UNIVERSITY OF CALIFORNIA, 2000. Disponível em: <<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>>. 39
- FOX, A.; PATTERSON, D. *Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing*. [S.l.]: Strawberry Canyon LLC, 2012. ISBN 0984881212. 11, 33, 34, 35
- FREEMAN, A. *Pro AngularJS*. [S.l.]: Apress, 2014. ISBN 9781430264484. 35
- GARCIA, C. *Modelagem e Simulação*. 2. ed. São Paulo: edUSP, 2009. ISBN 9788531409042. 11, 49
- GHOUSSAYNI, S. et al. Assessment and validation of a simple automated method for the detection of gait events and intervals. *Gait and Posture*, v. 20, n. 3, p. 266–272, 2004. ISSN 09666362. 19
- GOOGLE. *Angular Material Demo Grid List*. 2015. Disponível em: <<https://material.angularjs.org/latest/demo/material.components.gridList>>. 11, 38
- GOOGLE. *Angular-Seed*. 2015. Disponível em: <<https://github.com/angular/angular-seed>>. 11, 36, 37
- GOOGLE. *Material Design*. 2015. Disponível em: <www.google.com.br/design/spec/material-design/introduction.html>. 37
- GREENE, J.; STELLMAN, A. *Learn Agile*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2014. ISBN 9781449331924. 11, 27, 28
- GRINBERG, M. *Flask Web Development*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2014. ISBN 9781449372620. 39
- GRIP, H.; HäGER, C. A new approach to measure functional stability of the knee based on changes in knee axis orientation. *Journal of Biomechanics*, v. 46, n. 5, p. 855–862, 2013. ISSN 00219290. 23

HANLEY, J. *Scrum - User Stories: How to Leverage User Stories For Better Requirements Definition (Scrum Series) (Volume 2)*. [S.l.]: CreateSpace Independent Publishing Platform, 2015. ISBN 978-1512368031. 33

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. 2. ed. [S.l.]: Prentice Hall, 1998. ISBN 978-0132733502. 46, 82

IBRAHIM, B. K. et al. An Approach for Dynamic Characterisation of Passive Viscoelasticity and Estimation of Anthropometric Inertia Parameters of Paraplegic's Knee Joint. In: *Viscoelasticity - From Theory to Biological Applications*. InTech, 2012. Disponível em: <<http://www.intechopen.com/books/viscoelasticity-from-theory-to-biological-applications/an-approach-for-dynamic-characterisation-of-passive-viscoelasticity-and-estimation-of-anthropometri>> 11, 26, 27

LAYTON, M. C. *Agile Project Management For Dummies*. 1. ed. [S.l.]: For Dummies, 2012. ISBN 9781118235850. 27

LEITE, W. V. et al. Avaliação cinemática comparativa da marcha humana por meio de unidade inercial e sistema de vídeo. In: *XXIV Congresso Brasileiro de Engenharia Biomédica-CBEB 2014*. Uberlândia: CBEB, 2014. p. 1–4. 11, 25, 26

LIN, J.-n.; SONG, S.-m. Modeling gait transitions of quadrupeds and their generalization with CMAC neural networks. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, v. 32, n. 3, p. 177–189, ago. 2002. ISSN 1094-6977. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1097731>>. 43

MAIA, I. *Building Web Applications with Flask*. 1. ed. [S.l.]: Packt Publishing, 2015. ISBN 9781784396152. 40

MALAS, B. *Book Review - Gait Analysis: Normal and Pathological Function, 2nd Edition*. 2010. Disponível em: <<http://www.oandp.org/reading/gaitfunction.asp>>. 18

MARAES, V. R. F. d. S. *Estudo da Variabilidade da Frequência Cardíaca Durante o Exercício Físico Dinâmicos em Voluntários Sadios*. 174 p. Tese (Doutorado) — UNICAMP, 1999. Disponível em: <<http://www.bibliotecadigital.unicamp.br/document/?code=vtls000178242&fd=y>>. 63

MASSE, M. *REST API Design Rulebook*. [S.l.]: O'Reilly Media, Inc., 2011. ISBN 9781449310509. 11, 40

MATSUDA, K.; LEA, R. *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL (OpenGL)*. [S.l.]: Addison-Wesley Professional, 2013. ISBN 978-0321902924. 38

MORAES, J.; SILVA, S.; BATTISTELA, L. Comparison of two software packages for data analysis at gait laboratories. *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE Cat. No.03CH37439)*, v. 2, p. 1780–1783, 2003. ISSN 1094-687X. 17, 19

MORENO, a. et al. Development of the spatio-temporal gait parameters of Mexican children between 6 and 13 years old data base to be included in motion analysis softwares. *2009 Pan American Health Care Exchanges - PAHCE 2009*, n. 2, p. 90–93, 2009. 19

- MUYBRIDGE, E. *The Human Figure In Motion*. London: Chapman & Hall, 1885. 11, 21
- NG, A. *Machine Learning by Stanford University*. 2015. Disponível em: <<https://www.coursera.org/learn/machine-learning/home/welcome>>. 46
- PATTON, J. *User Story Mapping: Discover the Whole Story, Build the Right Product*. 1. ed. [S.l.]: O'Reilly Media, 2014. ISBN 978-1491904909. 32
- PERRY, J.; BURNFIELD, J. M. *Gait Analysis Normal and Pathological Function*. 2nd. ed. [S.l.]: SLACK Inc., 2010. ISBN 978-1-55642-766-4. 11, 18, 22, 23
- PLUGGE, E.; MEMBREY, P.; HOWS, D. *MongoDB Basics MongoDB Basics*. 1. ed. [S.l.]: Apress, 2014. ISBN 9781484208953. 11, 40, 41
- POOLE, D. *Linear Algebra: A Modern Introduction*,. 3. ed. [S.l.]: Books / Cole, 2011. 233,575 p. ISBN 978-0-538-73545-2. 47, 48
- PRESSMAN, R.; MAXIM, B. *Software Engineering: A Practitioner's Approach*. 8. ed. [S.l.]: McGraw-Hill Education, 2014. ISBN 978-0078022128. 11, 30
- QUALISYS. Qualisys Track Manager – QTM Motion Capture software for tracking all kind of movements. *Qualisys.Com*, 2010. Disponível em: <<http://www.qualisys.com/wp-content/uploads/2012/11/pi\qtm.pdf>>. 11, 16, 24
- QUALISYS. *Gait analysis & Rehabilitation*. 2013. Disponível em: <<http://www.qualisys.com/applications/biomechanics/gait-analysis-and-rehabilitation/>>. 11, 25
- QUALISYS. *Oqus MRI*. 2013. Disponível em: <<http://www.qualisys.com/products/hardware/oqus-mri/>>. 11, 24
- RUBIN, K. S. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. [S.l.]: Addison-Wesley Professional, 2012. ISBN 9780137043293. 30
- RUNYAN, K.; ASHMORE, S. *Introduction to Agile Methods*. [S.l.]: Addison-Wesley Professional, 2014. 29
- SABOURIN, C. Control Strategy for the Robust Dynamic Walk of a Biped Robot. *The International Journal of Robotics Research*, v. 25, n. 9, p. 843–860, set. 2006. ISSN 0278-3649. Disponível em: <<http://ijr.sagepub.com/cgi/doi/10.1177/0278364906069151>>. 43
- SABOURIN, C.; YU, W.; MADANI, K. Gait Pattern Based on CMAC Neural Network for Robotic Applications. *Neural Processing Letters*, v. 38, n. 2, p. 261–279, nov. 2012. ISSN 1370-4621. Disponível em: <<http://link.springer.com/10.1007/s11063-012-9257-6>>. 46
- SCHWABER, K. *Agile Project Management with Scrum*. [S.l.]: Microsoft Press, 2004. ISBN 9780735619937. 11, 30, 31, 32
- SCHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. [S.l.]: Pearson, 2001. ISBN 0130676349. 30, 32
- SYED, B. A. *Beginning Node.js*. [S.l.]: Apress, 2014. ISBN 9781484201879. 36

VIEIRA, A. et al. Software for human gait analysis and classification. In: *4th Portuguese BioEngineering Meeting*. Porto: [s.n.], 2015. 18

WILLIAMSON, K. *Learning AngularJS*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 9781491916759. 35, 36

Apêndices

APÊNDICE A – ARTIGO COMPLETO WC2015

Human Knee Simulation Using CMAC ANN

R. A. Lima¹, V. R. F. S. Marães¹, J. P. Martins¹ and L. M. Brasil¹

¹ University of Brasília-UnB at Gama-FGA, Brasília, Brazil

Abstract— This paper aims to show the use of a CMAC (Cerebellar Model Articulation Control), a kind of ANN (Artificial Neural Network). The CMAC is based on cerebellum of mammals, but despite this characteristic, actually, what promotes its use, is its very fast operation, which makes it suitable for adaptive control in real time. This type of control is needed, for example, to control an active transfemoral prosthesis. Simulation of knee angular velocities, based on collected data from the contralateral knee, is presented. The simulation is available as open source software.

Keywords— CMAC, machine learning, knee, simulation.

I. INTRODUCTION

The quest to improve the life quality is constituted as one of the prerogatives of Biomedical Engineering. Rehabilitation discipline is supported by biomedical engineering, for example, in prostheses constructions. Prostheses can be passives or actives [1]. Passive form use intrinsically passive actuators and active form have automatic actuators.

The construction of an active transfemoral prosthesis is not a trivial problem solution. To build them is required before to develop or to use a biomechanical model. From this, one must create a control method using control engineering techniques and / or intelligent systems.

The rise of intelligent systems brought a new landscape for Biomedical Engineering. This kind of solution allows previously unimagined system classes. It is possible to build intelligent systems which are not solvable easily by traditional PID (Proportional-Integral-Derivative), with many variables, many parameters and nonlinearities. Intelligent systems are everywhere, in digital cameras, Internet search engines, speech recognition systems, cell phones, car brake systems and countless other devices.

Among the techniques for building intelligent systems, there are the ANNs (Artificial Neural Networks). This kind of system is too classified as machine learning based system [2]. They are systems which learn to figure out complex problems during the learning phase, through historical collected data, without the necessity of complex physical models. The CMAC (Cerebellar Model Articulation Control) [3] is a ANN inspired on cerebellum of mammals [4], which compared with another kind of ANN called MLP (Multi-Layer Perceptron)[5], with at least one hidden layer,

has the advantage of needing much less calculations to update their weights during training [6].

Lin shows in [7], the effectiveness of CMAC model, by preliminary studies of kinematic control and gait synthesis. After training an ANN based on CMAC, to learn multivariable and nonlinear relationships kinematics of quadruped gait, the ANN was used to control straight and uphill walk of quadruped robots.

The [8] depicts the robustness of a CMAC in a biped robot running in conditions presenting disorders. The [9] presents strategies for using the CMAC in same type of robot.

Before one can build a prosthesis controller, is advisable first to simulate their behavior via software. This idea can facilitate prototypes production. In this context, the paper illustrates a human knee angular velocities simulation, using machine learning based CMAC model for possible knee control in a transfemoral active prosthesis.

II. MATERIALS AND METHODS

The simulation process is summarized in Figure 1.

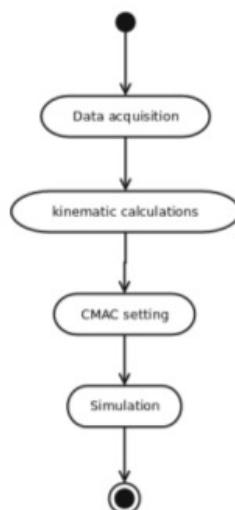


Fig. 1: Simulation Process

A. Data Acquisition

Knee kinematic data was utilized for CMAC training. This data was captured through motion capture techniques, using 12 cameras Qualisys Oqus MRI, passive markers and software package Qualisys QTM 3.2. Data were converted to the appropriate format supported by Octave 3.8.1 language (MATLAB format). Data were captured from a subject in Human Performance Laboratory at Faculty UnB Ceilândia. A healthy male subject was selected. He repeated a walk of approximately 5 seconds for 5 times. This process generated spatial variables, regarding the position of the markers. The markers were distributed along 34 positions at inferior limbs. As only knee flexion and extension were necessary for this work, only the markers of tibias, knees and trochanters were utilized. It was possible with these data, to calculate knees angles, angular velocities and angular accelerations. The initial and final data of each sample had to be eliminated, because they constitute the comfortable gait cycle beginning and end.

The data acquisition was approved by the UnB Health Faculty Ethics Committee, protocol N11911/12.

B. Kinematic Calculations

Kinematic calculations were made using the Octave programming language which is compatible with MATLAB. This choice was made, because Octave is open source and because the QTM export data to MATLAB format.

Angles were obtained using the equation (1) from [10].

$$\theta = \arccos\left(\frac{u \cdot v}{\|u\| \|v\|}\right) \quad (1)$$

The u variable is equivalent to the vector at the trochanter point; v is the vector at the tibia point. The knee point must be the origin of u and v , so these must be translated to the new origin [11].

The angular velocities were obtained using adjacent points of collected data following equation (2):

$$\omega = \frac{\theta_2 - \theta_1}{t} \quad (2)$$

The variable t is the time between adjacent calculated angles θ_1 and θ_2 . For this paper t is equal 1/315 seconds.

The source code to extract markers position and to generate knee angles, angles velocities and angles accelerations, is available in http://github.com/rob-n/gait_data_loader.

C. CMAC setting

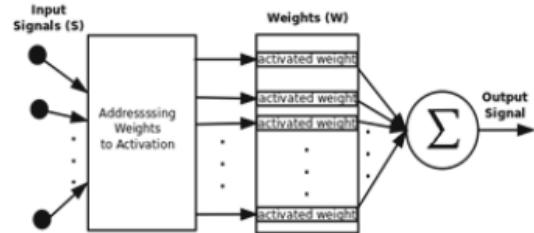


Fig. 2: Simulation Process

The CMAC model proposed by [3] is depicted in Figure 2. This type of ANN receives a signal vector S , and then uses this vector to calculate addresses. These addresses correspond to positions in weight vector W . Only activated weights will be summed to compose the CMAC output signal.

The input signals for a CMAC must have their enter space known. For example, for a vector S , with signals s_1 , s_2 and s_3 , the signal s_1 could have a value between 0 and 1, the signal s_2 a value between -500 and 100 and s_3 a value between -0.1 and 0.1. Furthermore, each signal must be converted to an exact quantity of discrete values. For example, a signal s_x , which is between 0 and 2, has five discrete values, such that their possible values are 0, 0.5, 1, 1.5 and 2. If for example, it has the value 1.6, then s_x takes the value 2, because there is not the value 1.6 in the list of discrete values. The quantity of discrete values by signal defines the signal resolution. More discrete values more resolution. Less discrete values less resolution. See Figure 3. The calculation of addresses has been implemented using the same algorithm proposed by [3].

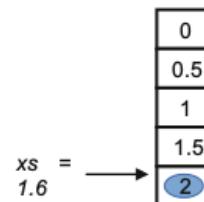


Fig. 3: Input value discretization for a signal s_x

An essential parameter in CMAC is the number of activations. As the name says, it defines the number of weights which will compose the output. How much higher this pa-

parameter, higher will be the generalization encompassed by CMAC.

ANNs as machine learning algorithms must be trained. Learning can be supervised or not supervised. CMAC's case is supervised learning. During this phase input signals and the desired output signal must be given for CMAC's learning algorithm. The last parameter to inform is the iterations number of the CMAC. This parameter is the number of times the training algorithm will update the weights after calculating the output for all input signals previously collected.

D. Simulation and implementation details

For CMAC's implementation and knee's simulation was adopted the Python 2.7 programming language, the software packages NumPy and Matplotlib were used too. The operating system is Mac OS X Yosemite version 10.10.1. The hardware is a Mac Pro (Mid 2010), processor 2.8 GHz Quad-Core Intel Xeon, RAM 3GB. In addition, the source code to the created simulations is available in <http://github.com/rob-nn/motus>.

As input signals for the simulation, velocities vectors in a 3D plane (x, y, z) of the left knee were chosen. As output, was chosen to predict the signal corresponding to the right knee angular velocities. These choices were made based on work [12].

Before running the simulation, the RNA CMAC must be trained. For accomplish this task only 50% of the data collected in a single walk were used. The other 50% were used to make the prediction of the output signal and compare them with the real signal captured.

III. RESULTSS

Figure 4 shows the simulation performed. For this simulation was informed: 50 iterations, 30 activations and 200 discrete values per input signal. The input signals are left knee angular velocities in a 3D plane (x, y, z). The data are 50% of a 5 seconds walk. Parts of the first and last moments were discarded. The output is the angular velocities of the right knee.

In addition to the generated simulation, an open source project, called Motus was created. See Figure 5 This project aims to create a tool for basic gait analysis and signal simulation of human lower limbs. Figure 5 shows the version 0.1 of this tool, which is available on GitHub site at <http://github.com/rob-nn/motus.py> address. The available version is capable, a while, to work with the following signals:

1. Knee angles (flexion, extension);
2. Knee angular velocities (flexion, extension);
3. Knee angular accelerations (flexion, extension);
4. Knee velocities in a 3D plane.

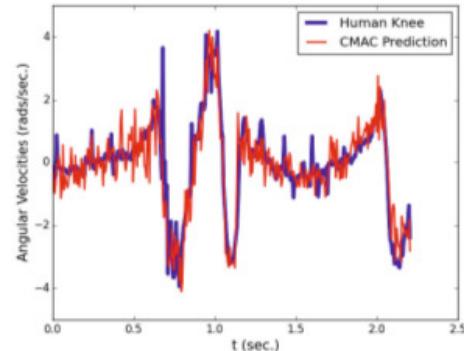


Fig. 4: Human knee angular velocity prediction

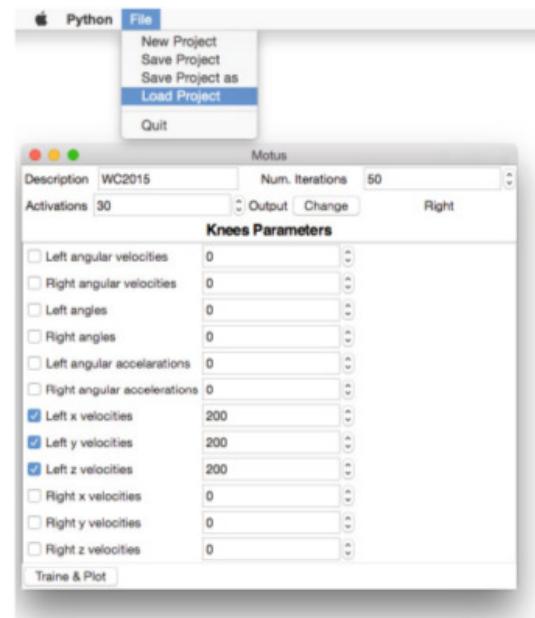


Fig. 5: Motus Version 0.1

Motus development is powered by Healthy Informatics Lab (*Laboratório de Informática em Saúde – LIS*).

IV. DISCUSSION

The trained CMAC can now be used for testing in a real prototype for a transfemoral active prosthesis. The advantage is that it is easily implemented in embedded systems. It requires only simple tables and calculations. Maybe more data is required, but a tool for training the CMAC is already done.

Others joints (hip, ankles) and order degree of freedoms can be simulated too. It is necessary however, to collect data from the other joints and calculate new angles, velocities and accelerations (using the same implemented methods), which isn't done yet. Motus is a working in progress.

V. CONCLUSIONS

The development of a transfemoral active prosthesis requires building control systems. These systems should be implemented as embedded systems, which typically have limited computing power. ANNs can be used for building control systems, but ANNs require a process called training. This usually requires a high computational power and is best accomplished in desktop systems or computers clusters.

This paper has shown the simulation approximation of a signal relating to a knee angular velocity, based on contralateral knee signals. All this simulation was performed using a CMAC, a type of ANN, which showed very good results graphically.

Besides the simulation, all necessary code to build and train the CMAC and also to simulate signals is available as open source. This project is on GitHub site and is accessible to anyone. The same is under continuous development.

VI. ACKNOWLEDGMENTS

The authors thank the resources provided by the LIS and the Movement Lab of UNB Ceilândia, also CAPES for the financial support given to the first author.

VII. CONFLICT OF INTERESTS

The authors declare that they have no conflict of interest.

VIII. REFERENCES

1. Pickle N T, Wilken J M et al. (2014) Whole-body angular momentum during stair walking using passive and powered lower-limb prostheses. *J Biomechanics* 47:3380–3389
2. Bishop C N (2006) Pattern recognition and machine learning. Springer, Singapore
3. Albus J S (1975) A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *J of Dynamic Systems, Measurement, and Control* 220–227
4. Albus J S (1975) A theory of cerebellar function. *Mathematical Biosciences* J 10:25–61
5. Haykin S (2008) Neural networks and learning machines. Pearson Prentice Hall, New York
6. Smith R L (1998) Intelligent motion control with an artificial cerebellum. University of Auckland, Auckland
7. Lyn Y (1992) A CMAC neural-network-based algorithm for kinematic control of a walking machine. *Engrn. Appl. Artif. Intel.* J 6:539–551
8. Sabourin C, Bruneau O (2005) Robustness of dynamic walk of a biped robot subjected to disturbing external forces by using CMAC neural networks. *Robotics and Autonomous Systems* J 51:81–99
9. Sabourin C, Bruneau et al. (2006) Control strategy for the robust dynamic walk of a biped robot. *The International J of Robotics Research* 25:843–860
10. Edwards L (2010) Calculus. Books Cole, Belmont
11. Poole D (2011) Linear algebra a modern introduction. Brooks Cole, Boston
12. Vallery H., Burgkart R et al. (2011) Complementary limb motion estimation for the control of active knee prostheses

Author: Roberto Aguiar Lima

Institute: UnB Gama / Mestrado em engenharia biomédica

Street:

City: Brasília

Country: Brazil

Email: aguiarlima.roberto@gmail.com

APÊNDICE B – RESUMO PAHCE 2014



Using a Fuzzy-CMAC Network for Control of a Transtibial Prosthesis

Uso de uma Rede Fuzzy-CMAC para Controle de uma Prótese Transtibial

R. A. Lima¹, L. M. Brasil¹, V. R. F. Silva¹, J. A. A. Andrade¹

¹Universidade de Brasília, Campus Gama, DF, Brasil

Prostheses are objects for studies in human rehabilitation. Active prostheses applied to amputees have found their place in researches, but for this, a robust control model must be implemented to ensure their reliability of operation. In this context it is possible use AI (Artificial Intelligent) to resolve this problem. This paper seeks the implementation of a Fuzzy-CMAC (Cerebellar Model Articulation Controller) network to control a transtibial prosthesis.

Keywords — Fuzzy-CMAC, Prostheses Control, Artificial Neural Networks.

Próteses são objetos de estudo em reabilitação humana. As próteses ativas aplicadas a amputados têm encontrado seu lugar nas pesquisas, mas para tanto, um modelo de controle robusto deve ser implementado para garantir a confiabilidade de funcionamento das mesmas. Neste contexto é possível utilizar a Inteligência Artificial (IA) para resolver tal problema. O presente trabalho busca a implementação de uma rede Fuzzy-CMAC (Cerebellar Model Articulation Controller) para controlar uma prótese transtibial.

Palavras-Chave — Fuzzy-CMAC, Controle Próteses, Redes Neurais Artificiais.

Os sistemas Neuro-Fuzzy tem aplicações em diversas áreas, desde modelagem térmica para o processo de produção de tubos laminares até o controle das posturas das pernas num robô bípede. Dentre as inúmeras possibilidades deste tipo de sistema, encontram-se as redes Fuzzy-CMAC, que têm como capacidade, a memorização de características do ambiente em que operam [1]. O presente trabalho busca a implementação do controle de uma prótese para indivíduos amputados abaixo do joelho. Este controle se dará por uma rede Fuzzy-CMAC, que levará em consideração, parâmetros cinematográficos, dinâmicos da prótese e também sinais bioelétricos do usuário. A prótese terá três graus de liberdade, dorsiflexão e flexão plantar, eversão e inversão, abdução e adução, isso, comparando-se com os movimentos de um tornozelo real.

Inspirada no cortéx cerebral dos mamíferos, a CMAC (*Cerebellar Model Articulation Controller*) foi criada por [2]. Seu funcionamento se dá através de memórias associativas, que relacionam as entradas às saídas da rede. Verificou-se que a CMAC é um sistema adequado ao controle em tempo real, o que é bem vindo para o controle de vários graus simultâneos de liberdade, e também um sistema de gerenciamento de memória pois para entradas semelhantes, obtém-se saídas semelhantes. Uma característica indesejável da rede CMAC é que sua saída apresenta muitas descontinuidades. A técnica que se utilizou primeiramente para diminuir este problema, era fazer uso de um grande número de funções de ativação, o que consequentemente aumentava a complexidade do algoritmo e comprometia a sua execução em tempo-real. Atualmente, para se resolver este problema, é melhor usar uma rede Fuzzy-CMAC que adota funções de pertinência. Assim são usadas funções mais suaves que as originais binárias (CMAC), para ativação. [3]

O uso da RNA (Rede Neural Artificial) Fuzzy-CMAC permitiu mapear, com sucesso, a posição de um tornozelo, no plano sagital, durante um ciclo de marcha, ou seja, do ângulo formado entre o pé e a perna. Comparando-se o resultado da RNA com o método de aproximação por Jacobianos, a RNA se mostrou superior pois o segundo método apresentou regiões de instabilidade que prejudicaram seus resultados [4]. Em [5] também é demonstrada uma metodologia para o projeto de um padrão de marchar de um robô bípede, que além de tudo é ainda autoadaptativo. Assim, como próxima etapa deste trabalho será realizado justamente esse projeto de padrão marcha, levando-se em conta ainda os dados bioelétricos do paciente.

REFERENCES

- [1] J. A. A. Andrade, L. M. Brasil, E. H. Diniz, K. C. Borges, S. S. R. F. Rosa and R. C. Silva, "Proposta de Estudo de Redes Neuro- Fuzzy para Aplicações em um Controle de Prótese Ativa Transtibial," XXIII Congresso Brasileiro em Engenharia Biomédica – XXIII CBEB, pp. 1730-1732, 2012.
- [2] J. S. Albus, *A new approach to manipulator control: The cerebellar model articulation controller*. Trans. ASME, J. Dyn. Syst., Meas. Control, v. 97, p. 220-227, Sept. 1975.
- [3] S. O. Rezende, *Sistemas Inteligentes – Fundamentos e aplicações*. São Paulo: Manole, 2002, pp. 221-222.
- [4] J. A. A. Andrade, K. C. Borges, L. M. Brasil, E. H. Diniz, J. F. Figueiredo and R. C. Silva, "Estudo de uma RNA FCMA como Base de Controle para uma Prótese Ativa Transtibial", IV Encontro de Ciência e Tecnologia da Faculdade UnB – Gama, pp. 232-235, Dezembro, 2012.
- [5] C. Sabourin, W. Yu and K. Madani, *Gait Pattern Based on CMAC Neural Network for Robotic Applications*. Neural Process Let, vol. 38, pp. 261-279, 2013.

Anexos

ANEXO A – PROCESSO NO COMITÊ DE ÉTICA



PROCESSO DE ANÁLISE DE PROJETO DE PESQUISA

Registro do Projeto no CEP: **119/11**

Título do Projeto: "Tecnologias avançadas de próteses para amputados de membro inferior".

Pesquisadora Responsável: Geovany Araujo Borges

Data de Entrada: 31/08/11

Com base na Resolução 196/96, do CNS/MS, que regulamenta a ética em pesquisa com seres humanos, o Comitê de Ética em Pesquisa com Seres Humanos da Faculdade de Ciências da Saúde da Universidade de Brasília, após análise dos aspectos éticos e do contexto técnico-científico, resolveu **APROVAR** o projeto 119/11 com o título: "Tecnologias avançadas de próteses para amputados de membro inferior". Área Temática Especial – "Pesquisa Grupo 1 Novos Procedimentos, Novos Equipamentos" analisado na 3ª reunião ordinária realizada no dia 12 de março de 2013.

O pesquisador responsável fica, desde já, notificado da obrigatoriedade da apresentação de um relatório semestral e relatório final sucinto e objetivo sobre o desenvolvimento do Projeto, no prazo de 1 (um) ano a contar da presente data (item VII.13 da Resolução 196/96).

Brasília, 14 de março de 2013.

Natan Monteiro de Sá
coordenador do CEP-FS/UnB

ANEXO B – LICENÇA MIT

The MIT License (MIT)

Copyright (c) year copyright holders

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.