

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

***OPEN GAIT ANALYTICS* - 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

***OPEN GAIT ANALYTICS* - 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 , 01 DE Setembro DE 2015

FICHA CATALOGRÁFICA

Roberto Aguiar Lima

OPEN GAIT ANALYTICS - IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA, [Distrito Federal] 2015. NUMERAÇÃO . 73 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 marchar. 2. aprendizado de máquina

3. joelho. 4. simulação

I. FGA UnB Gama/ UnB. II. *OPEN GAIT ANALYTICS* - IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA

CDU: N° da CDU (biblioteca)

REFERÊNCIA BIBLIOGRÁFICA

LIMA, R. A. (ANO). TÍTULO. Dissertação de Mestrado em Engenharia Biomédica, Publicação NO./ANO, Programa de Pós-Graduação em Engenharia Biomédica, Faculdade Gama, Universidade de Brasília, Brasília, DF, 73 p.

CESSÃO DE DIREITOS

AUTOR: Roberto Aguiar Lima

TÍTULO: *OPEN GAIT ANALYTICS* - 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.

ENDEREÇO.

CEP: ..., Brasília, DF – Brasil

DEDICATÓRIA

Para ..., com amor.

AGRADECIMENTOS

...

O Mestre na arte da vida faz pouca distinção entre o seu trabalho e o seu lazer, entre sua mente e seu corpo, entre sua educação e sua recreação. Ele simplesmente persegue sua visão de excelência em tudo o que faz, deixando para os outros a decisão de saber se está trabalhando ou se divertindo. Ele acha que está sempre fazendo as duas coisas simultaneamente.

Texto Budista

RESUMO

***OPEN GAIT ANALYTICS* - 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

Texto corrido sem parágrafo. 1 página.

Palavras-chaves: análise de marchar, aprendizado de máquina, joelho, simulação.

ABSTRACT

OPEN GAIT ANALYTICS - 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, Month of Year.

Texto corrido sem parágrafo. 1 página.

Key-words: gait analysis, machine learning, knee, simulation.

SUMÁRIO

1	INTRODUÇÃO	15
1.1	CONTEXTUALIZAÇÃO E FORMULAÇÃO DO PROBLEMA	15
1.2	OBJETIVOS	15
1.2.1	Objetivo Geral	15
1.2.2	Objetivo Específicos	15
1.3	REVISÃO DA LITERATURA	15
1.4	ORGANIZAÇÃO DO TRABALHO	16
2	FUNDAMENTAÇÃO TEÓRICA	17
2.1	ANÁLISE DE MARCHA	17
2.1.1	BREVE HISTÓRICO	17
2.1.2	FUNDAMENTOS	18
2.1.3	MÉTODOS DE COLETA DE DADOS PARA ANÁLISE	19
2.2	MÉTODOS ÁGEIS	21
2.2.1	SCRUM	25
2.2.2	HISTÓRIAS DO USUÁRIO	26
2.3	SOFTWARE COMO SERVIÇO	28
2.4	COMPONENTES, FRAMEWORKS E FERRAMENTAS	29
2.4.1	APLICAÇÕES WEB COM <i>ANGULARJS</i>	29
2.4.2	ORGANIZAÇÃO DE PROJETOS WEB COM <i>ANGULAR-SEED</i>	30
2.4.3	SOLUÇÃO DE DESIGN WEB COM <i>ANGULAR-MATERIAL</i>	31
2.4.4	GRÁFICOS E ANIMAÇÕES 3D NA <i>WEB</i> COM <i>THREEJS</i>	31
2.4.5	SERVIÇOS REST COM <i>FLASK</i>	32
2.4.6	SERVIÇO DE BASE DE DOCUMENTOS COM <i>MONGODB</i>	33
2.5	CMAC	35
2.5.1	TREINAMENTO DA CMAC	38
2.5.2	LICENÇA MIT	39
3	METODOLOGIA	40
3.1	AMBIENTE DE ESTUDO	40
3.2	DELIMITAÇÃO DO ESTUDO	42
3.3	VISÃO	43
3.4	MODELO DE GESTÃO	43
3.5	MODELO DE ARQUITETURA	43
3.5.1	CAMADA DE APLICAÇÃO WEB	43
3.5.2	CAMADA <i>REST WEB API</i>	46

3.5.3	CAMADA DE BASE DE DOCUMENTOS	48
4	RESULTADOS	51
4.1	Módulo de Análise	51
4.2	Módulo de Simulação	62
5	DISCUSSÃO E CONCLUSÃO	67
6	TRABALHOS FUTUROS	69
	REFERÊNCIAS BIBLIOGRÁFICAS	70

LISTA DE TABELAS

Tabela 1	– Comparando Modelos de Desenvolvimento de Software	24
Tabela 2	– Mapeamento de $s1$	37
Tabela 3	– Mapeamento de $s2$	37
Tabela 4	– Mapeamento para os pesos W	38

LISTA DE FIGURAS

Figura 1 – Atleta iniciando uma corrida. Fonte (MUYBRIDGE, 1885).	17
Figura 2 – Divisões do Ciclo de Marcha. Fonte (PERRY; BURNFIELD, 2010). . .	18
Figura 3 – Câmera Oqus MRI. Fonte (QUALISYS, 2013b).	20
Figura 4 – Visão do QTM. Fonte: (QUALISYS, 2010).	20
Figura 5 – Configuração de câmeras para captura de dados de marcadores passivos fixados no paciente. Fonte: (QUALISYS, 2013a).	21
Figura 6 – Experimento de captura de dados de <i>IMU</i> em conjunto com uma câmera. Fonte: (LEITE et al., 2014).	21
Figura 7 – Comparação entre <i>IMU</i> e a câmera. Fonte: (LEITE et al., 2014). . . .	22
Figura 8 – Eletrogoniômetro. Fonte: (IBRAHIM et al., 2012).	22
Figura 9 – Valores em comum. Adaptado de (GREENE; STELLMAN, 2014). . . .	23
Figura 10 – Exemplo de processo baseado no modelo <i>waterfall</i> . Adaptado de (PRES- SMAN; MAXIM, 2014).	24
Figura 11 – Visão Geral do <i>Scrum</i> . Adaptado de (SCHWABER, 2004).	25
Figura 12 – Exemplo de <i>story card</i> . Adaptado de (COHN, 2004).	27
Figura 13 – Diagrama de uma aplicação <i>MVC</i> usando <i>AngularJS</i>	29
Figura 14 – Layout original de um projeto baseado em <i>angular-seed</i> . Fonte: (GOO- GLE, 2015b)	30
Figura 15 – Exemplo de aplicação que utiliza <i>angular-material</i> . Fonte (GOOGLE, 2015a).	31
Figura 16 – Exemplo de aplicação que utiliza <i>ThreeJS</i>	32
Figura 17 – Organização dos dados no <i>MongoDB</i> . Fonte (PLUGGE; MEMBREY; HOWS, 2014).	34
Figura 18 – Definindo documentos normalizado com <i>MongoDB</i> . Fonte (DAYLEY, 2014).	35
Figura 19 – CMAC para controle de uma junta. Adaptado de (ALBUS, 1975a). . .	36
Figura 20 – Quantização de <i>s1</i>	37
Figura 21 – Rede LIS.	40
Figura 22 – Processo de coleta de dados.	41
Figura 23 – Dados disponibilizados pelo QTM.	42
Figura 24 – Processo de desenvolvimento.	44
Figura 25 – Camadas arquiteturais.	44
Figura 26 – Exemplo de uma tela criada com <i>angular-material</i>	45
Figura 27 – Camada web	46
Figura 28 – Camada web	46
Figura 29 – Camada <i>REST WEB API</i>	48

Figura 30 – Banco de documentos da aplicação.	50
Figura 31 – Tela de seleção dos módulos.	51
Figura 32 – Tela com a listagem de pacientes.	52
Figura 33 – Informações do paciente.	53
Figura 34 – Tela inicial da dados coletados do paciente.	55
Figura 35 – Inclusão de amostra de marcha	55
Figura 36 – Seleção do arquivo no formato <i>MATLAB</i> proveniente do <i>QTM</i>	56
Figura 37 – Dados do arquivo provenientes do <i>QTM</i>	56
Figura 38 – Animação dos marcadores em 3D.	57
Figura 39 – Controle de perspectivas.	57
Figura 40 – Controle de <i>zoom</i>	57
Figura 41 – Controle <i>pan</i>	58
Figura 42 – Controles da animação.	58
Figura 43 – Opção <i>markers</i>	58
Figura 44 – Progreção espacial de um marcador.	59
Figura 45 – Seleção de um marcador pelo <i>mouse</i>	59
Figura 46 – Animação mostrando o marcador renomeado.	60
Figura 47 – Opção de visualização e criação de ângulos.	60
Figura 48 – Inclusão de um novo ângulo.	60
Figura 49 – Ângulo de um joelho durante o ciclo de marcha.	61
Figura 50 – Velocidades angulares de um joelho durante o ciclo de marcha.	61
Figura 51 – Módulo de simulação.	63
Figura 52 – Seleção do ciclo de marcha.	63
Figura 53 – Sinais de entrada para a <i>CMAC</i> . No caso posições num plano 3D de marcadores.	64
Figura 54 – Sinais de entrada para a <i>CMAC</i> . Ângulos e velocidades angulares.	64
Figura 55 – Exemplo de configuração para uma simulação usando <i>CMAC</i>	65
Figura 56 – Resultado da simulação.	65
Figura 57 – Erro quadrado médio em cada iteração da simulação	66

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Program Interface</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>
IMU	<i>Inertial Measurement Unit</i>
JSON	<i>JavaScript Object Notation</i>
LIS	Laboratório de Informática em Saúde
LPH	Laboratório de Performance Humana
MOCAP	<i>Motion Capture</i>
MVC	<i>Model View Controller</i>
RNA	Rede Neural Artificial
REST	<i>Representational State Transfer</i>
SPA	<i>Single Page Application</i>
UnB	Universidade de Brasília
URL	<i>Uniform Resource Locator</i>
VM	<i>Virtual Machine</i>

LISTA DE SÍMBOLOS

Símbolos Latinos

F_1, F_2	Força (N)
------------	---------------

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E FORMULAÇÃO DO PROBLEMA

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:

- Definir um processo de desenvolvimento ágil adequado ao projeto;
- Explicitar uma visão arquitetural inicial do software;
- Escolher componentes de software a serem usados na solução;
- Selecionar um conjunto funcionalidades suficientes para implementação de uma *release* funcional e de testes que expressem a arquitetura proposta.

1.3 REVISÃO DA LITERATURA

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

- *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>).

Foram utilizadas as seguintes chaves de pesquisa em cada um dos serviços acima: “*Gait Analysis Software*”.

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 marcha 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 saudáveis, 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

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.

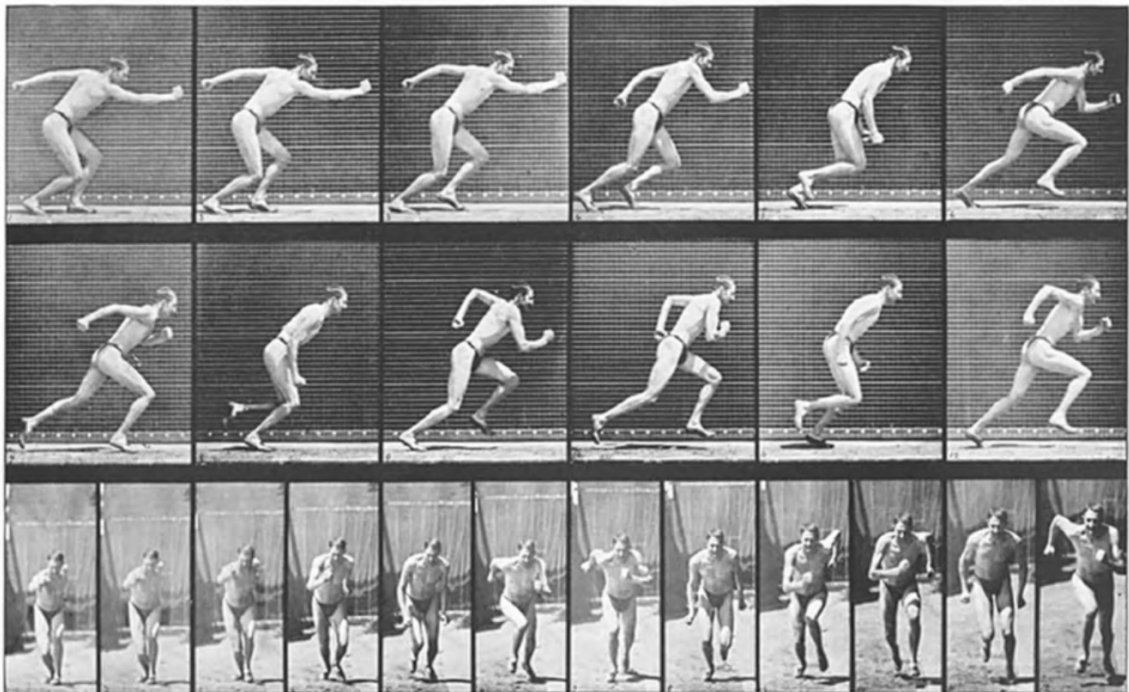


Figura 1 – Atleta iniciando uma corrida. Fonte (MUYBRIDGE, 1885).

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.

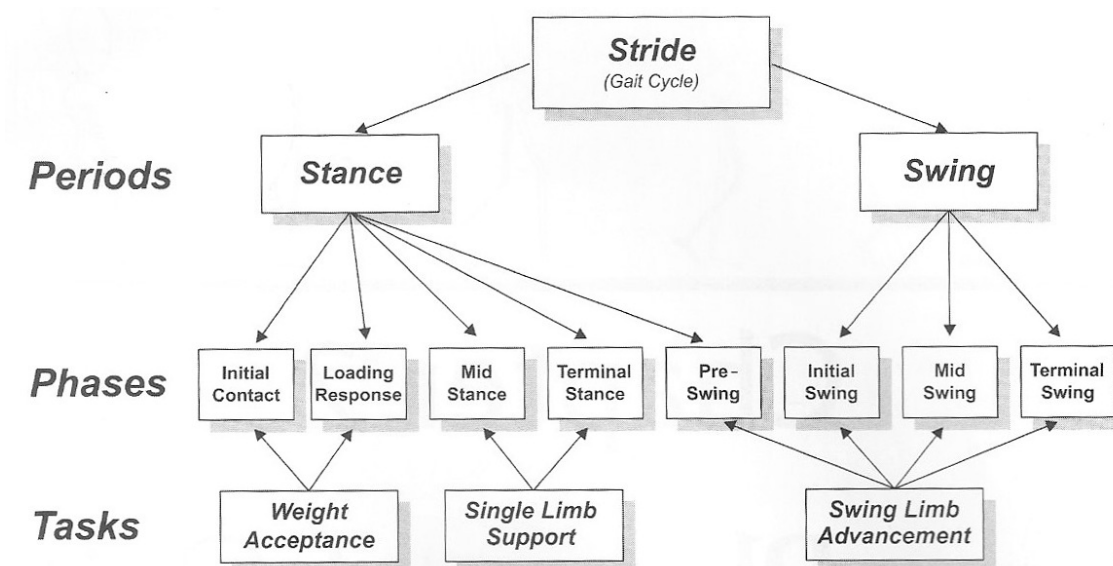


Figura 2 – Divisões do Ciclo de Marcha. Fonte (PERRY; BURNFIELD, 2010).

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 representa 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% até 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

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 discorre 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*, está 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.

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*). 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 da *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.

Captura por Eletrogoniômetros

Eletrogoniômetros, também são 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.



Figura 3 – Câmera Oqus MRI. Fonte (QUALISYS, 2013b).

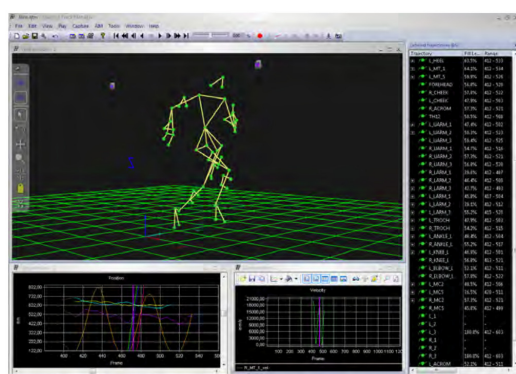


Figura 4 – Visão do QTM. Fonte: (QUALISYS, 2010).

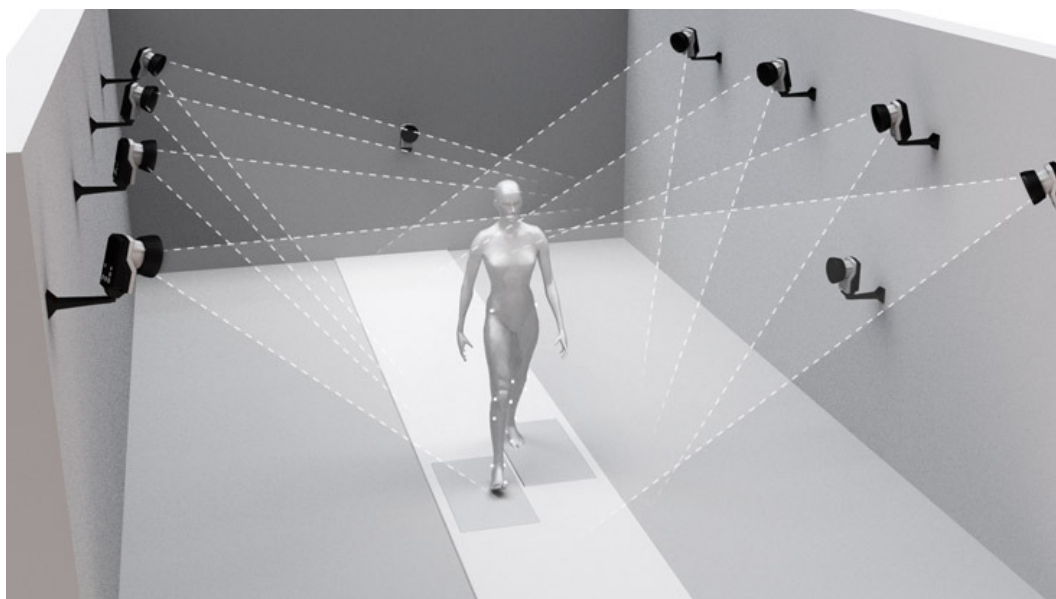


Figura 5 – Configuração de câmeras para captura de dados de marcadores passivos fixados no paciente. Fonte: (QUALISYS, 2013a).

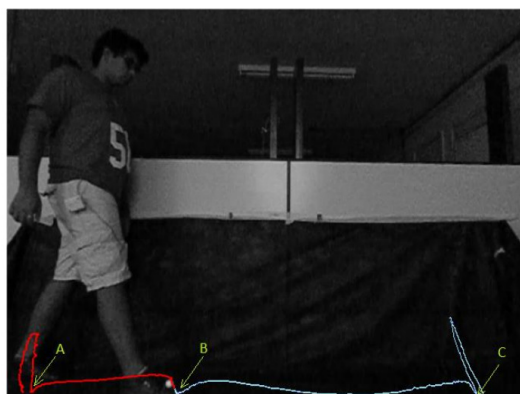


Figura 6 – Experimento de captura de dados de *IMU* em conjunto com uma câmera. Fonte: (LEITE et al., 2014).

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;

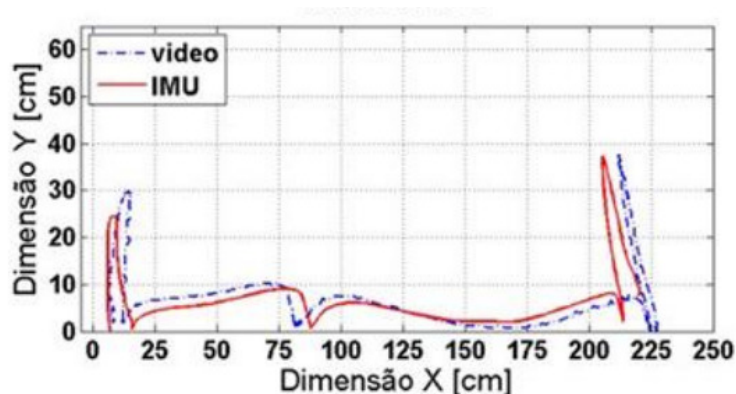


Figura 7 – Comparação entre *IMU* e a câmera. Fonte: (LEITE et al., 2014).



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

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 tardiamente 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.

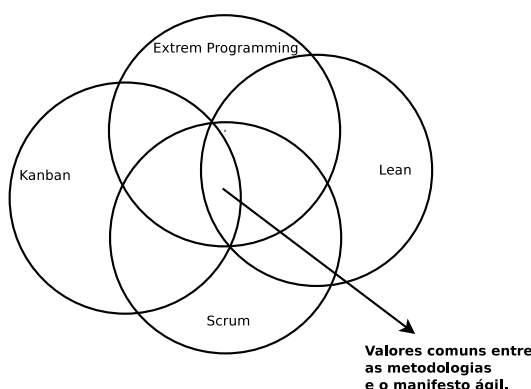


Figura 9 – Valores em comum. Adaptado de (GREENE; STELLMAN, 2014).

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 Runyan e Ashmore (2014) compara o modelo *waterfall* como o modelo ágil. Esta comparação é resumidamente apresentada na Tabela 1.

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 melhoras, 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 trânsito pode ficar vermelho. Esta é a principal motivação para que este trabalho privilegie métodos ágeis de desenvolvimento.

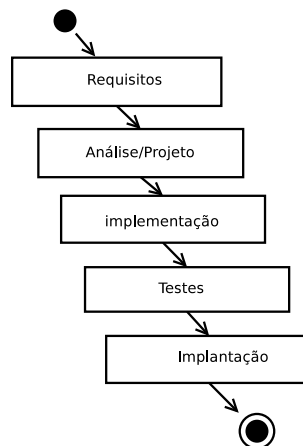


Figura 10 – Exemplo de processo baseado no modelo *waterfall*. Adaptado de (PRESSMAN; MAXIM, 2014).

Tabela 1 – 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; ASHMORE, 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*.

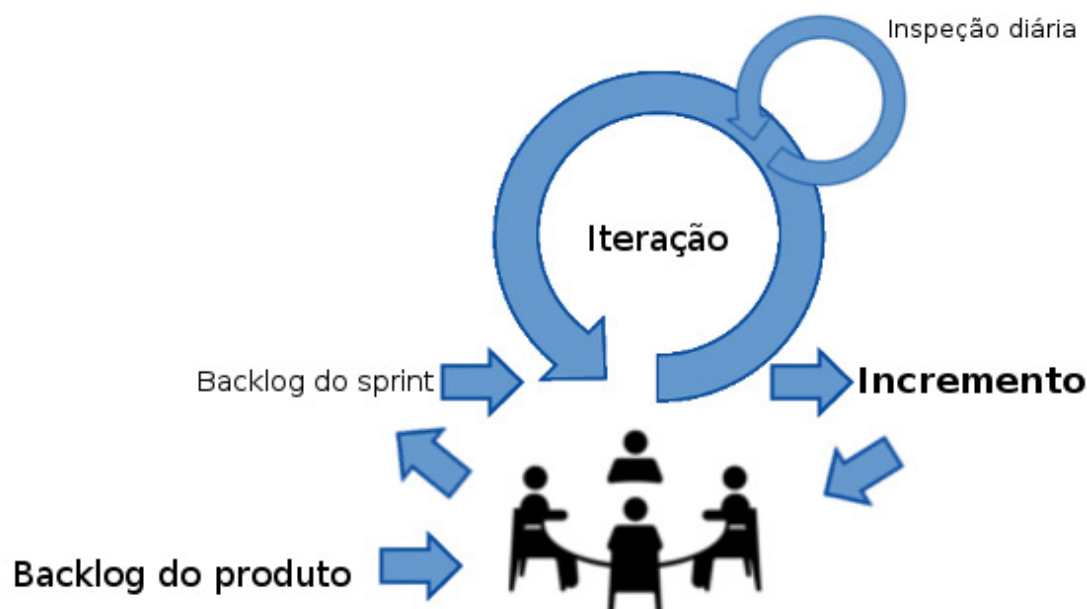


Figura 11 – Visão Geral do *Scrum*. Adaptado de (SCHWABER, 2004).

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).

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, uma semana por exemplo (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 minuciosamente 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.

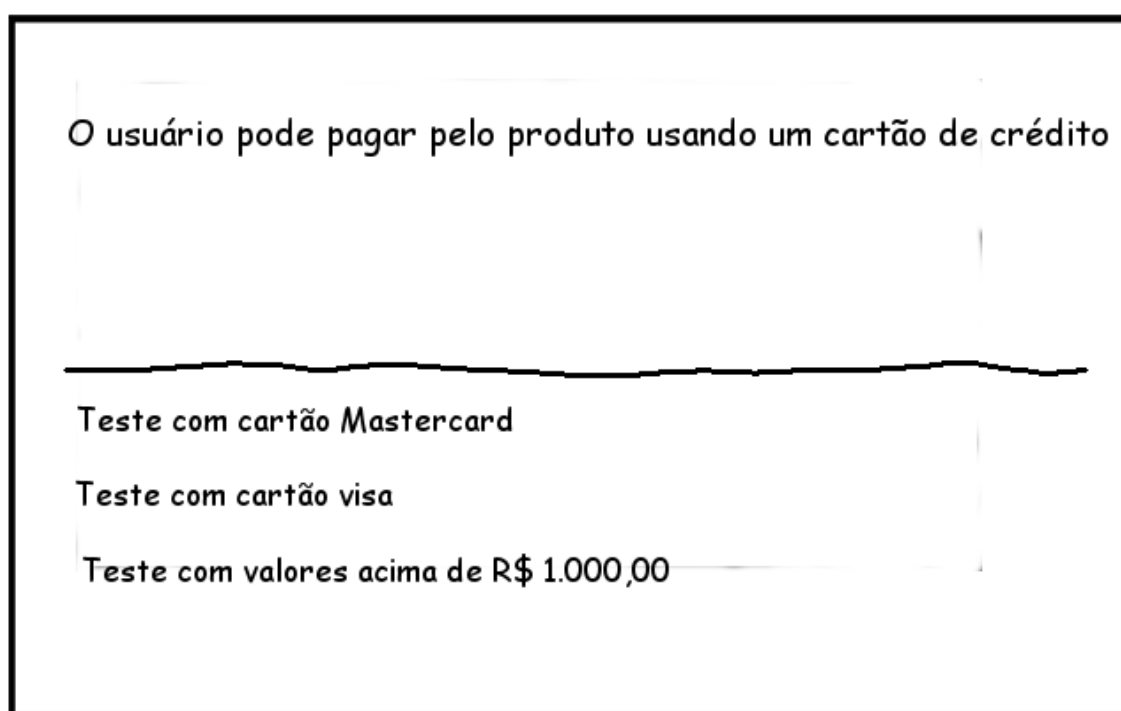


Figura 12 – Exemplo de *story card*. Adaptado de (COHN, 2004).

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).

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 em fazer *backups*, ou os perderem devido a mal funcionamento, ou até mesmo os perderem devido ao 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 APIs (Application Program Interfaces) de interface com o usuário.

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

2.4 COMPONENTES, *FRAMEWORKS* E FERRAMENTAS

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

2.4.1 APLICAÇÕES WEB COM *ANGULARJS*

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 ênfase é em permitir a criação de aplicações: extensíveis, manuteníveis, testáveis e padronizadas. A Figura 13 mostra uma representação de uma aplicação fazendo uso do *AngularJS*.

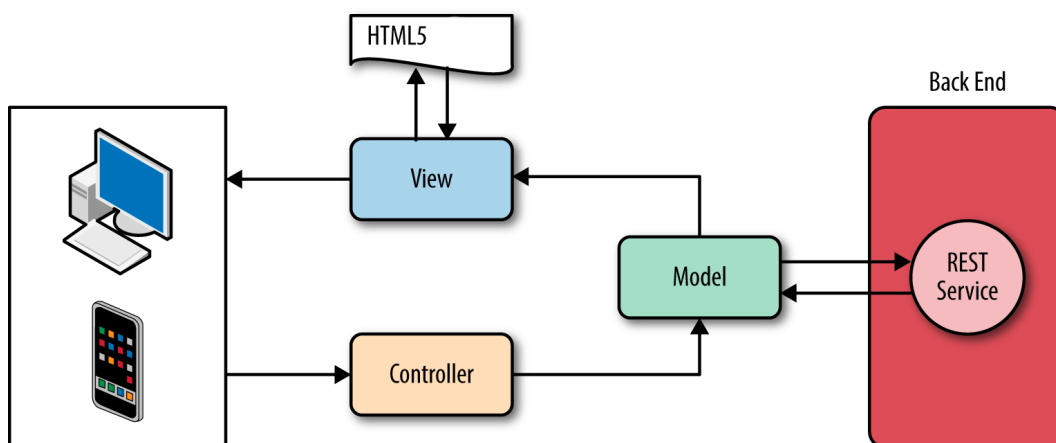


Figura 13 – Diagrama de uma aplicação *MVC* usando *AngularJS*.

Fonte: (WILLIAMSON, 2015).

Segundo Williamson (2015), a Figura 13 mostra o diagrama de uma aplicação *AngularJS* e os componentes *MVC*. Uma vez que a aplicação é lançada, os componentes *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 negócio. Outra característica bastante apreciada pelos

desenvolvedores que usam *AngularJS*, é a possibilidade de criar *single-page applications* (*SPA*). *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 14 mostra o *layout* inicial de diretórios que a ferramenta gera após ser clonada do *Github*.

```

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/                              --> the view1 view template and logic
  view1.html                       --> the partial template
  view1.js                         --> the controller logic
  view1_test.js                    --> tests of the controller
view2/                              --> the view2 view template and logic
  view2.html                       --> the partial template
  view2.js                         --> the controller logic
  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/                         --> end-to-end tests
  protractor-conf.js               --> Protractor config file
  scenarios.js                     --> end-to-end scenarios to be run by Protractor

```

Figura 14 – Layout original de um projeto baseado em *angular-seed*. Fonte: (GOOGLE, 2015b)

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).

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 design 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 15.

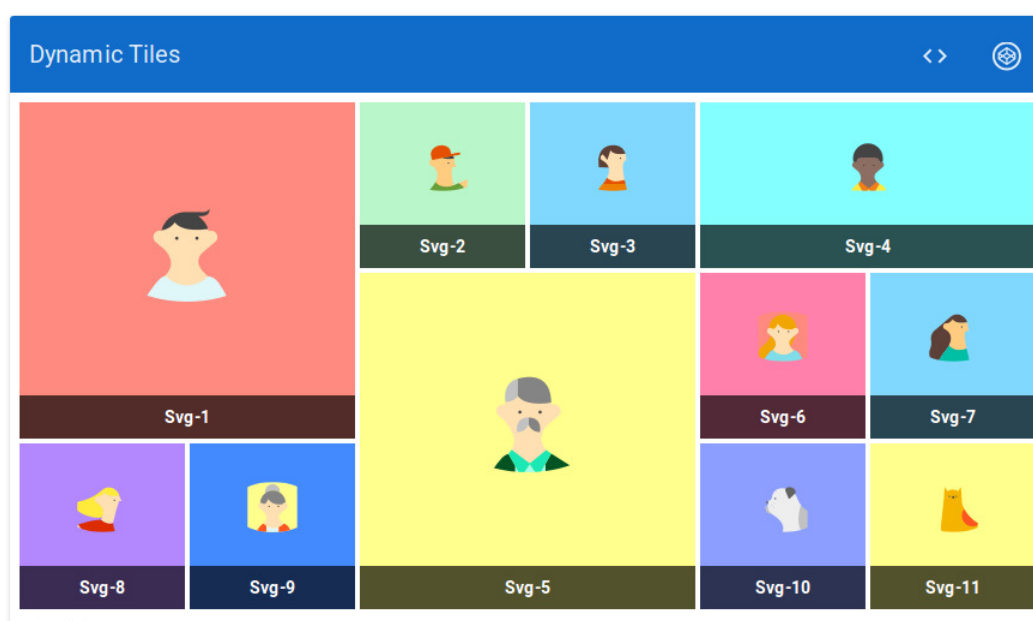


Figura 15 – Exemplo de aplicação que utiliza *angular-material*. Fonte (GOOGLE, 2015a).

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 16. Mais exemplos podem ser vistos no site *threejs.org*.

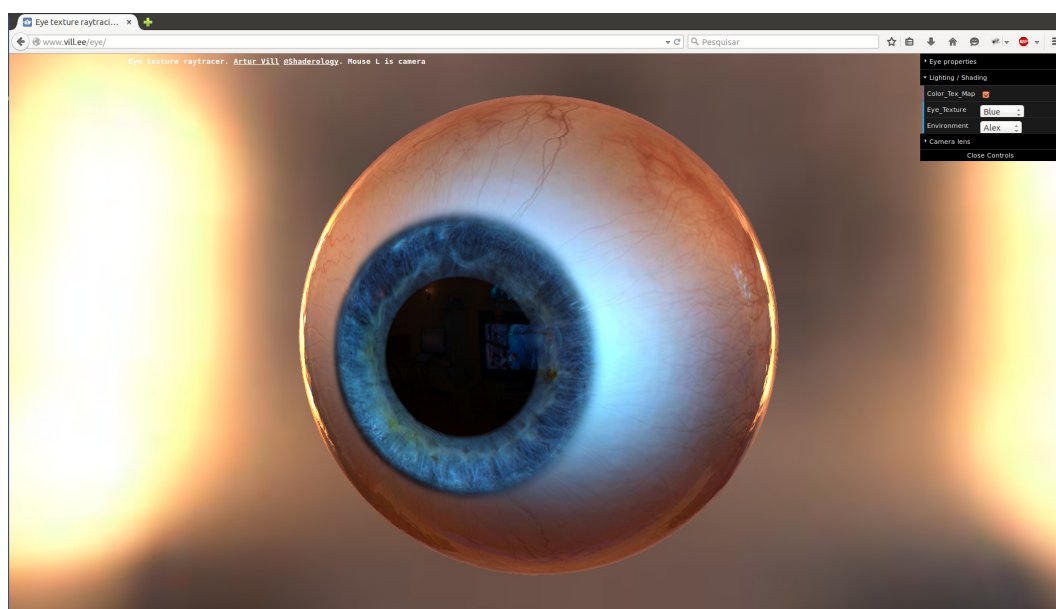


Figura 16 – Exemplo de aplicação que utiliza *ThreeJS*.

Fonte: <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 requer 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.

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 todas 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, as vezes pode parecer um contrassenso. Plugge, Membrey e Hows (2014) diz que o pináculo das filosofias do *MongoDB* é a noção de que um tamanho não se adequa 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 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 17 mostra como os dados são organizados nesta tecnologia.

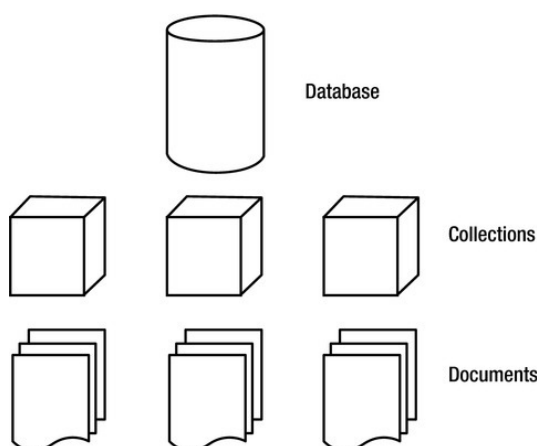


Figura 17 – Organização dos dados no *MongoDB*. Fonte (PLUGGE; MEMBREY; HOWS, 2014).

Segundo a Figura 17 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 *BSON*, um primo muito próximo do *JSON*. O *BSON* é muito parecido com o *JSON*, ele foi criado para ser mais otimizado nas leituras e escritas no banco.

A listagem abaixo retirada de Dayley (2014), mostra como um documento se parece.

```
{
  name: "New_Project",
  version: 1,
  languages: ["JavaScript", "HTML", "CSS"],
  admin: {name: "Brad", password: "****"}
}
```

```

paths: {temp: "/tmp", project: "/opt/project",
        html: "/opt/project/html"}
}

```

Para quem conhece *JSON*, provavelmente compreendeu todo o documento. Um documento fica entre `{` e `}` e pode conter outros documentos, tipos simples e listas. O documento possui atributos, no exemplo, *name*, *version*, *languages*, *admin*, *paths*. Os atributos *name* e *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 18 mostra como se pode fazer a referência entre documentos.

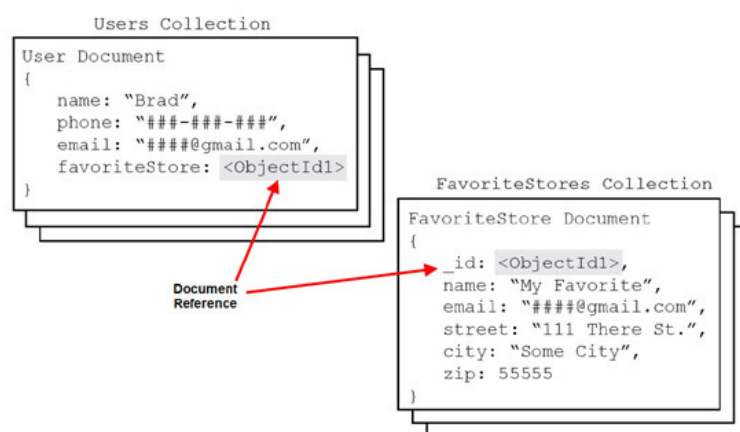


Figura 18 – Definindo documentos normalizados com *MongoDB*. Fonte (DAYLEY, 2014).

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).

Na Figura 19 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^* .

A CMAC da Figura 19 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 s_1 de S aceita valores de -1 até 1 e se quer 5 valores possíveis, quantiza-se então os valores de -1 até 1, conforme Figura 20.

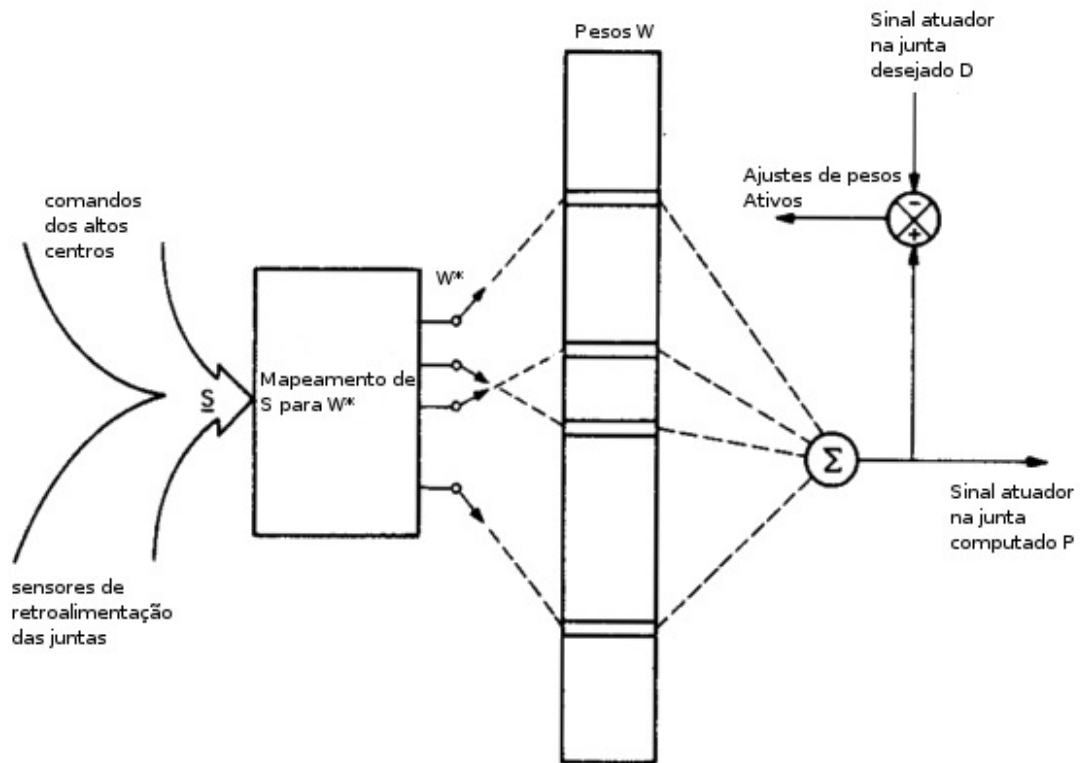
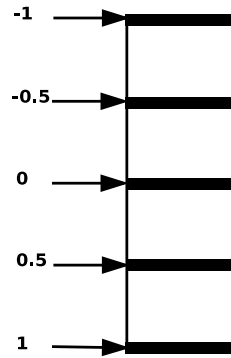


Figura 19 – CMAC para controle de uma junta. Adaptado de (ALBUS, 1975a).

Isto significa que quaisquer que sejam os valores de s_1 os mesmos devem ser convertidos para -1, -0,5, 0, 0,5 e 1. Por exemplo, se o valor de s_1 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

Figura 20 – Quantização de $s1$.

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 2 e na Tabela 3. 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 2 e Tabela 3 (ALBUS, 1975a).

Tabela 2 – 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 3 – 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 4.

O mapeamento da Tabela 4 é feito da seguinte forma (ALBUS, 1975a): Coloca-se cada valor quantizado de $s1$ nomeando cada uma das linhas da tabela. Coloca-se cada

Tabela 4 – Mapeamento para os pesos W

$s1 / s2$	1	2	3	4
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)

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 2 e na Tabela 3, 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 4, 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 .

2.5.2 LICENÇA MIT

Por meio de software livre, grandes projetos de software tornaram-se realidade. Existem muitas licenças de software livre, cada uma com suas vantagens e desvantagens. Como este trabalho trata da construção de um software, foi decidido que ele seria livre, e que inclusive poderia ser usado para fins comerciais. O tipo de licença escolhida foi a MIT, que é uma das mais permissivas. O texto na íntegra é mostrado a seguir (MIT, 2015):

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.

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 21.

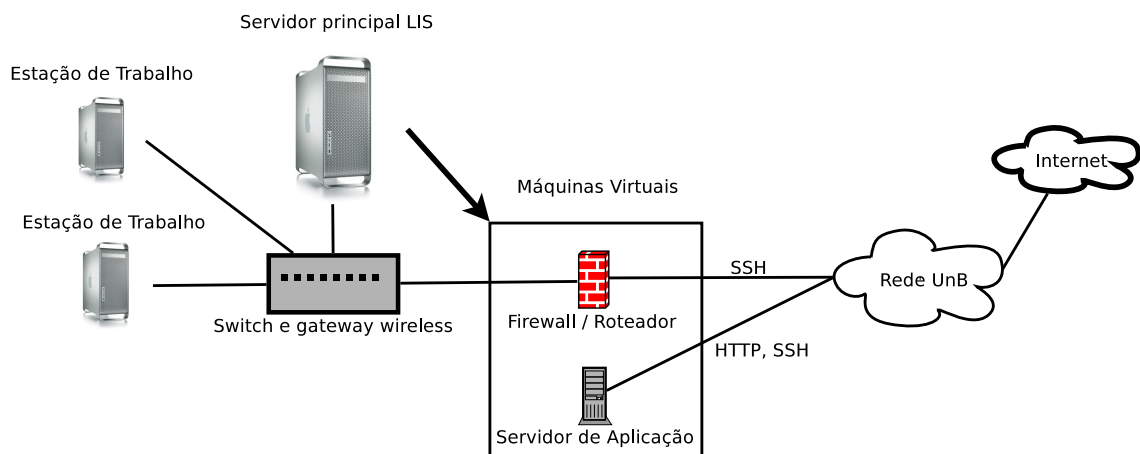


Figura 21 – Rede LIS.

O LPH/FCE foi utilizado para captura de dados de marcha humana. Este laboratório está equipado para coletar dados de plataformas de força, eletromiógrafos 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. Para este trabalho

foi utilizado o software *QTM 3.2* da *Qualisys*, ver Figura 4, que é responsável pela coleta de dados *MOCAP*.

O processo de coleta é demonstrado na Figura 22.

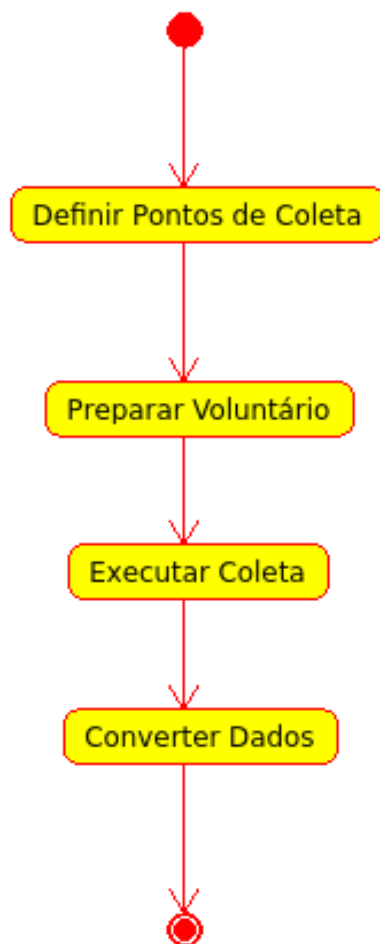


Figura 22 – Processo de coleta de dados.

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 23.



Figura 23 – Dados disponibilizados pelo QTM.

São retornados vários dados do QTM, mas os de interesse para o projeto são os que estão na Figura 3. 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.

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. Por exemplo, um desenvolvedor sênior no mercado de Brasília, não custaria menos de R\$ 100.000,00 por ano para uma empresa.

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 histórias de usuários, conforme 2.2.2. Na Figura 24 é apresentada a visão geral do processo.

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 25.

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

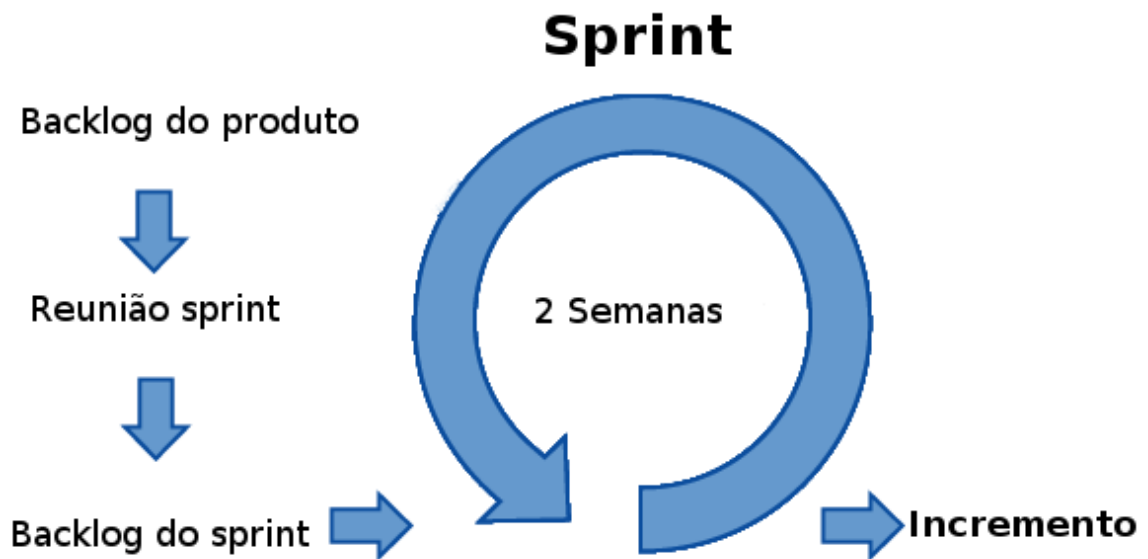


Figura 24 – Processo de desenvolvimento.

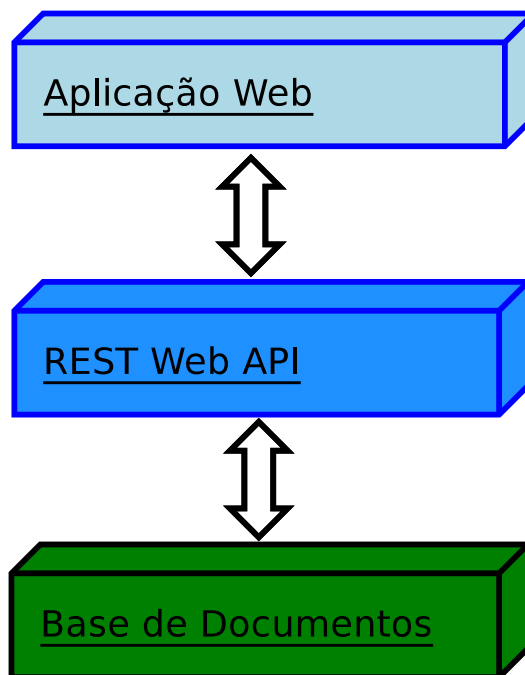


Figura 25 – Camadas arquiteturais.

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 26, 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 26 – 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 necessários 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), conforme o comando abaixo.

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

VISÃO ARQUITETURAL DA CAMADA WEB

A Figura 27 mostra o funcionamento e os padrões mínimos a serem seguidos para implementação da camada *web*.

Este esquema funciona da seguinte forma: O usuário usando seu *browser*, acessa o site do *Open Gait Analytics*. 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 a programação pesada acontece, ele é associado a um *template HTML*. Este template funciona de visão, e é o que o controlador manipula para mostrar informações ao usuário. Quando o controlador precisa executar lógica de negócio, 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

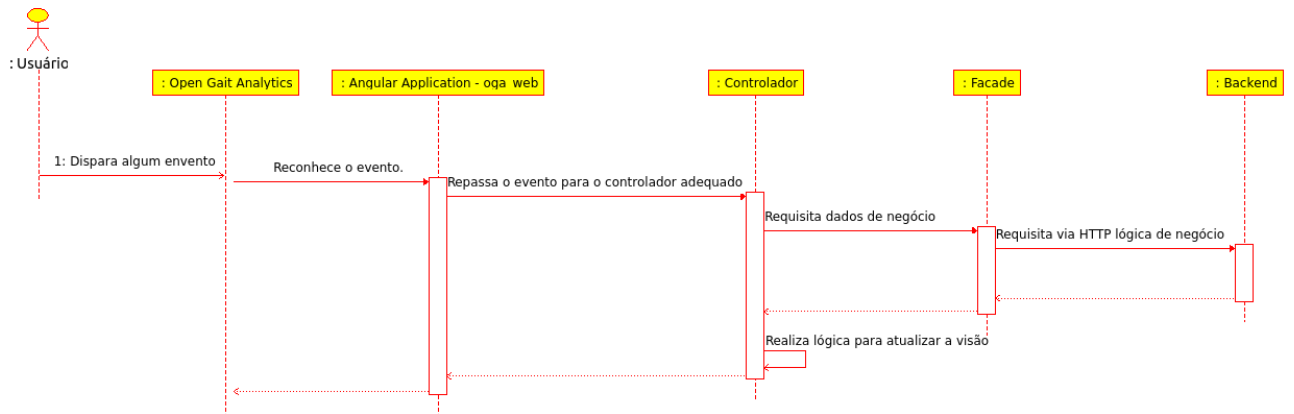


Figura 27 – Camada web

negócio. 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*.

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 disto 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 28 mostra a configuração básica dos arquivos e diretórios da aplicação.

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

Figura 28 – Camada web

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 e o mais importante, é a linguagem mais divertida que o autor já aprendeu.

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 29, mostra um *blueprint* de como funciona e como deve ser desenvolvida esta camada.

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

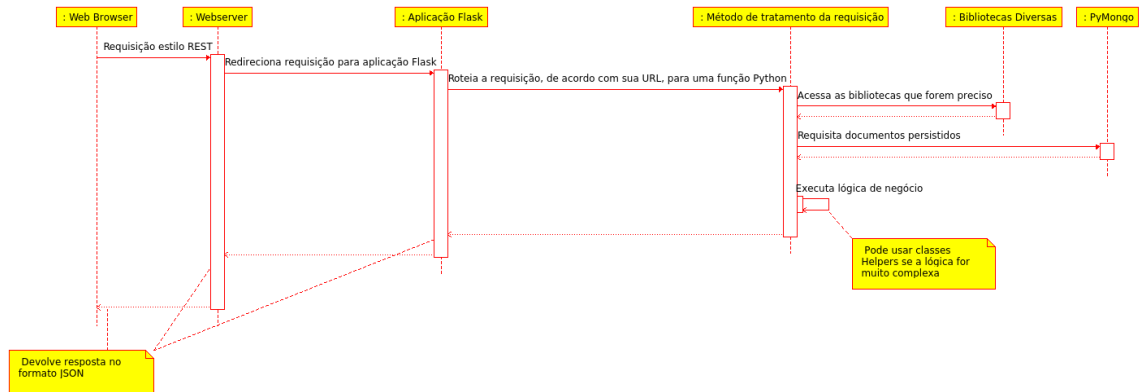


Figura 29 – Camada *REST WEB API*.

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*.

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 torna possível criar esta primeira versão do software em tão pouco tempo.

A estrutura do banco da aplicação é mostrada na Figura 30. O banco é composto por duas coleções: os dados dos pacientes na coleção *patients* e os dados recuperados do *QTM positionals_data*.

A listagem abaixo mostra um exemplo de documento da coleção *patients*.

```

{
  "_id" : ObjectId("5601ed9e54bac75828a154e4"),
  "city" : "Brasilia",
  "name" : "Beto_Jamais",
  "weight" : 75,
  "profession" : "Professor",
  "zipCode" : "70000000",

```

```

    "sex" : "male",
    "phone" : "61_5555-5555",
    "state" : "DF",
    "race" : "white",
    "birth" : "2000-01-01T02:00:00.000Z",
    "address" : "Rua_da_Serra",
    "maritalStatus" : "married",
    "height" : 1.8
}

```

A listagem abaixo mostra um exemplo de documento da coleção *positionals_data*.

```

{
  "\_id" : ObjectId("55a8f9874f9336e37084be92"),
  "patient\_id" : ObjectId("55a8f97654bac7124a76ec4d"),
  "gait\_sample\_index" : 0,
  "original\_filename" : "Walk1.mat",
  "angles" : [
    {
      "origin" : "0",
      "component\_b" : "2",
      "component\_a" : "1",
      "description" : "Angulo_Joelho_Direito"
    },
    {
      "origin" : 4,
      "component\_b" : 6,
      "component\_a" : 5,
      "description" : "teste"
    }
  ],
  "initial\_frame" : 300,
  "number\_markers" : 3,
  "markers" : [
    "Joelho_Direito",
    "Tibia_Direita",
    "Trocanter_Direito"
  ],
  "frame\_rate" : 315,
  "frames" : 1491,
}

```

```
"final\_frame" : 1300  
"trajectories":[ Aqui deveria aparecer uma lista com  
                 milhares de linhas numa matriz de 3 dimensoes]  
}
```

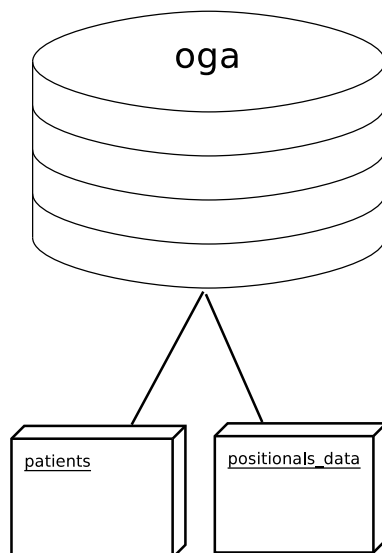


Figura 30 – Banco de documentos da aplicação.

4 RESULTADOS

O software construído por este projeto, o *Open Gait Analytics* versão 0.1, foi inteiramente construído pelo autor desta obra, e está liberado no site https://github.com/rob-nn/open_gait_analytics, sob a licença *MIT*. Para saber mais sobre esta licença veja 2.5.2. A intenção do autor é aumentar as chances de que futuras versões do software sejam construídas, não importando se serão comerciais ou gratuitas. Importante frizar também que o software continuará sendo desenvolvido pelo autor nos próximos meses.

O software em si é composto por dois módulos que são apresentados na Figura 31. 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.

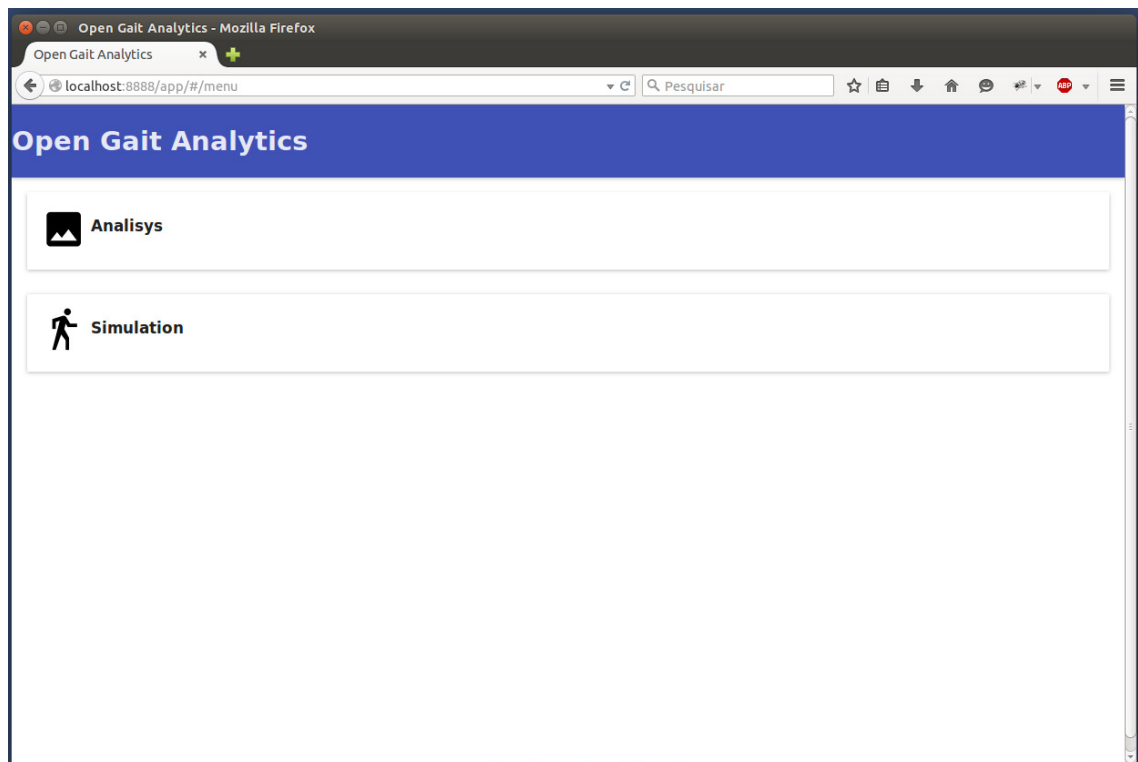


Figura 31 – Tela de seleção dos módulos.

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 32. Esta tela apresenta a listagem de pacientes, cadastrados no sistema. O botão abaixo a direita, é a função para adicionar novos pacientes.

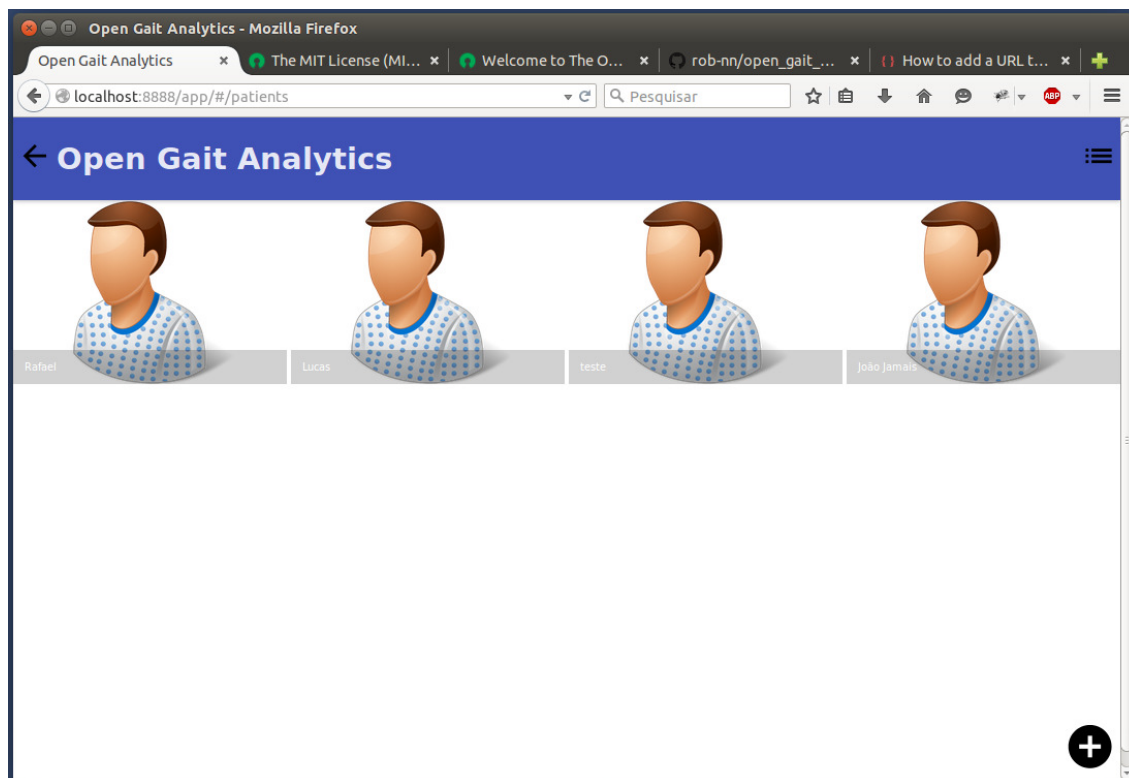


Figura 32 – Tela com a listagem de pacientes.

A Figura 33 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).

Ao se selecionar um paciente da tela mostrada na Figura 32, a tela da Figura 34 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 35.

Depois de salvar as informações o sistema pede para que o usuário selecione o arquivo proveniente do *QTM*, conforme a Figura 36. Depois de selecionado o arquivo com os dados da marcha, seus dados são mostrados para o usuário, conforme a Figura 37.

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 38. Esta animação foi construída usando a tecnologia *ThreeJS*, resumida na seção 2.4.4.

The screenshot shows a web browser window titled "Open Gait Analytics - Mozilla Firefox". The address bar displays "localhost:8888/app/#/patient_new". The page has a blue header with the text "Open Gait Analytics". Below the header is a form for entering patient information. The form fields and their values are as follows:

Field	Value
Name	Roberto Aguiar Lima
Sex	<input type="radio"/> Female <input checked="" type="radio"/> Male
Address	Av. Central, Bl. 191 N. 10, Núcleo Bandeirante
City	Brasília
State	DF
ZIP Code	71720005
Phone	61 81543841
Birth	1977-11-19
Race	<input type="radio"/> Asian <input type="radio"/> Black <input checked="" type="radio"/> White
Profession	Programador
Marital Status	<input type="radio"/> Divorced <input type="radio"/> Married <input checked="" type="radio"/> Single <input type="radio"/> Widowed
Weight	87
height	1.75

At the bottom right of the form, there is a black floppy disk icon representing a save button.

Figura 33 – Informações do paciente.

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.

A tela da animação também possui controle de perspectivas, ver Figura 39, controle de *zoom*, ver Figura 40, e controle *pan*, ver Figura 40. Também foram implementados, até o momento, botões de *play*, *pause*, fechar e um contador de *frames*, ver Figura *animacao4a*.

É de fundamental importância que o usuário configure os parâmetros *Initial Con-*

tact e *Terminal Swing* mostrados na Figura 37. 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 37, é a opção *Markers*, esta opção permite nomear os marcadores e visualizar sua progressão espacial. A Figura 43 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 44. O domínio é o percentual do ciclo de marcha, já a imagem são dados espaciais brutos oriundos do *QTM*.

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 45, 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*), ver a Figura ?? . Agora para o sistema o marcador 30 é sempre o joelho esquerdo, veja a Figura 46.

Outra funcionalidade importante é o gerador de ângulos. Esta opção está disponível da Figura 37. E após selecionada é mostrada na Figura 47. Para se gerar um ângulo o usuário necessita selecionar a opção de inclusão de ângulo e preencher os dados da Figura 48. 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 tíbia. Estes pontos poderiam representar o ângulo de um joelho por exemplo.

Depois dos ângulos criados é possível ver seus valores durante o ciclo de marcha ou suas velocidades angulares, veja a Figura 49 e a Figura 50.

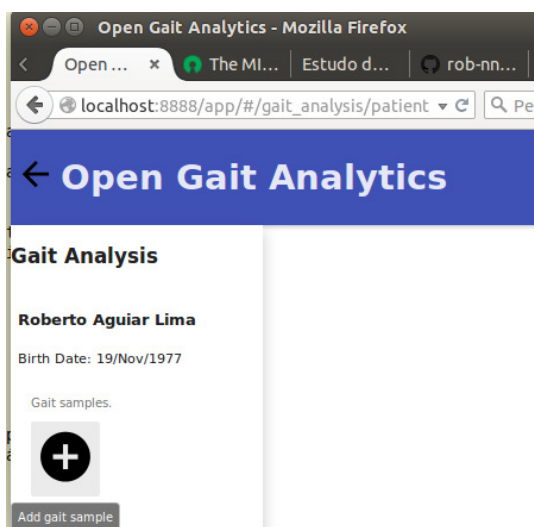


Figura 34 – Tela inicial da dados coletados do paciente.

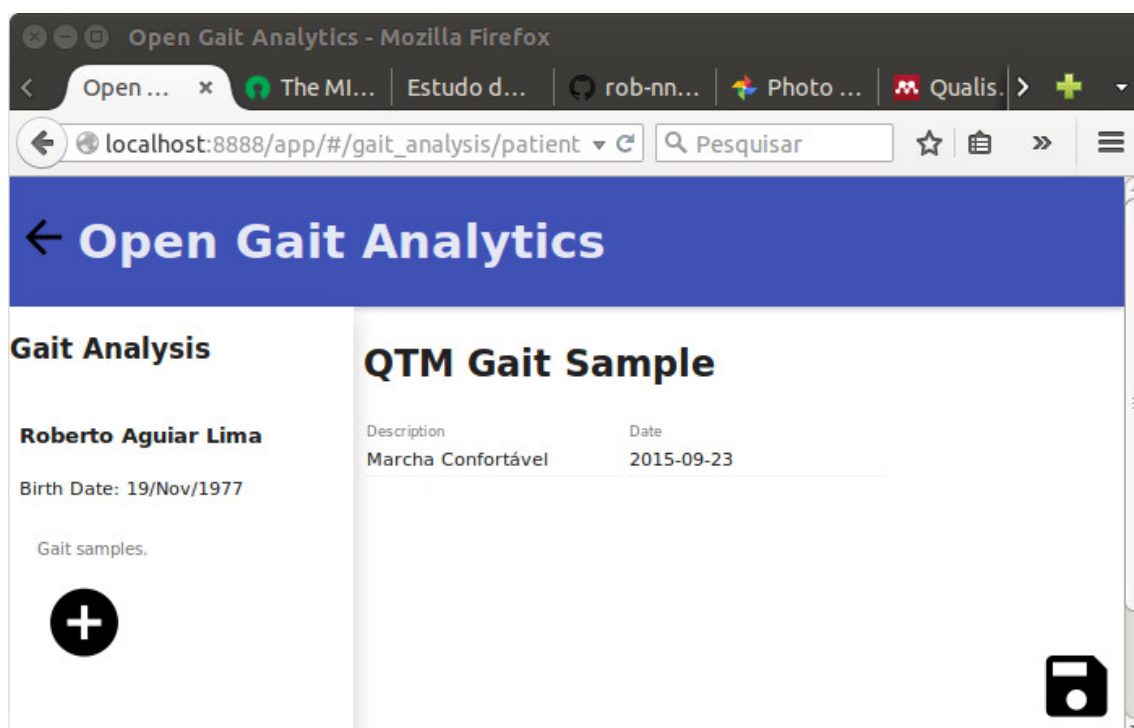


Figura 35 – Inclusão de amostra de marcha

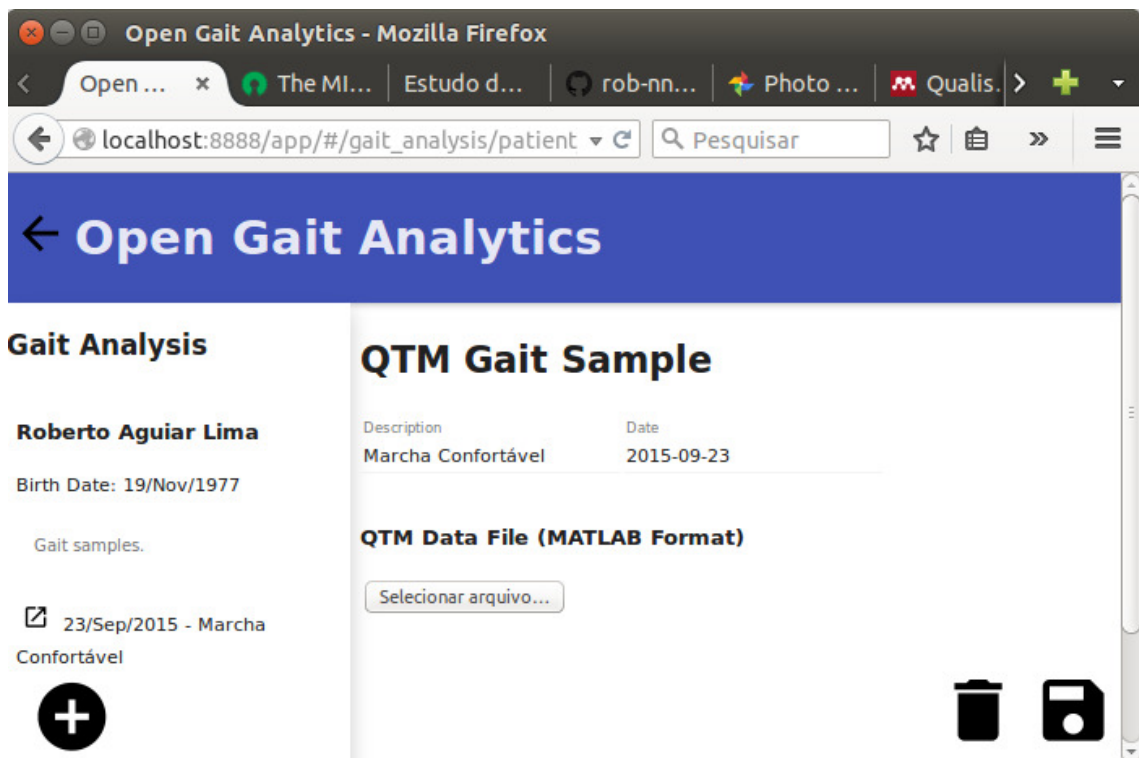


Figura 36 – Seleção do arquivo no formato *MATLAB* proveniente do *QTM*.

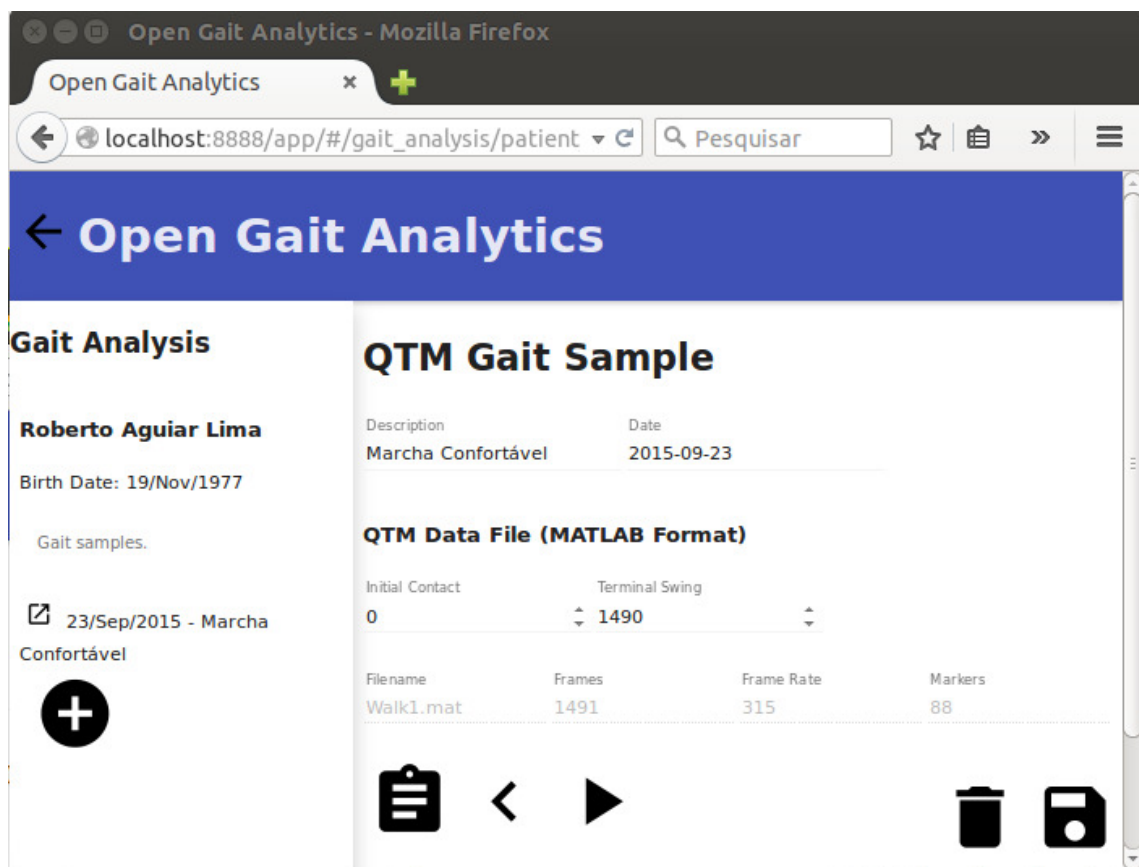


Figura 37 – Dados do arquivo provenientes do *QTM*

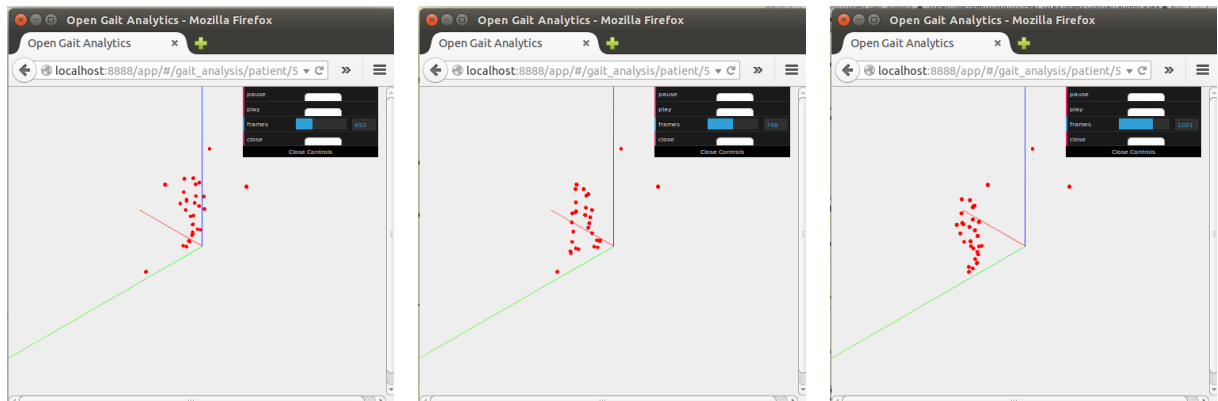


Figura 38 – Animação dos marcadores em 3D.

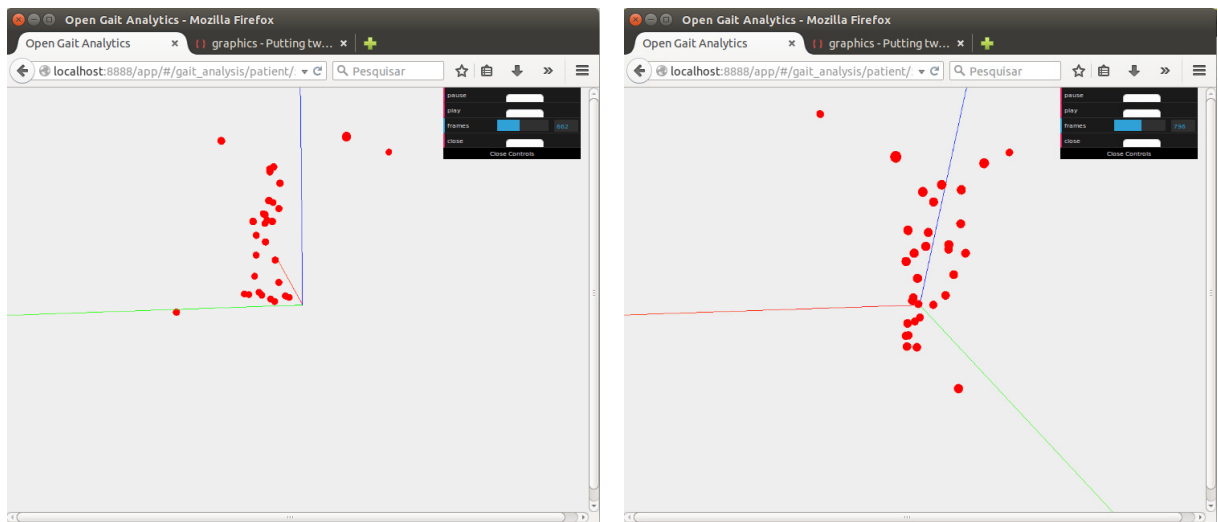


Figura 39 – Controle de perspectivas.

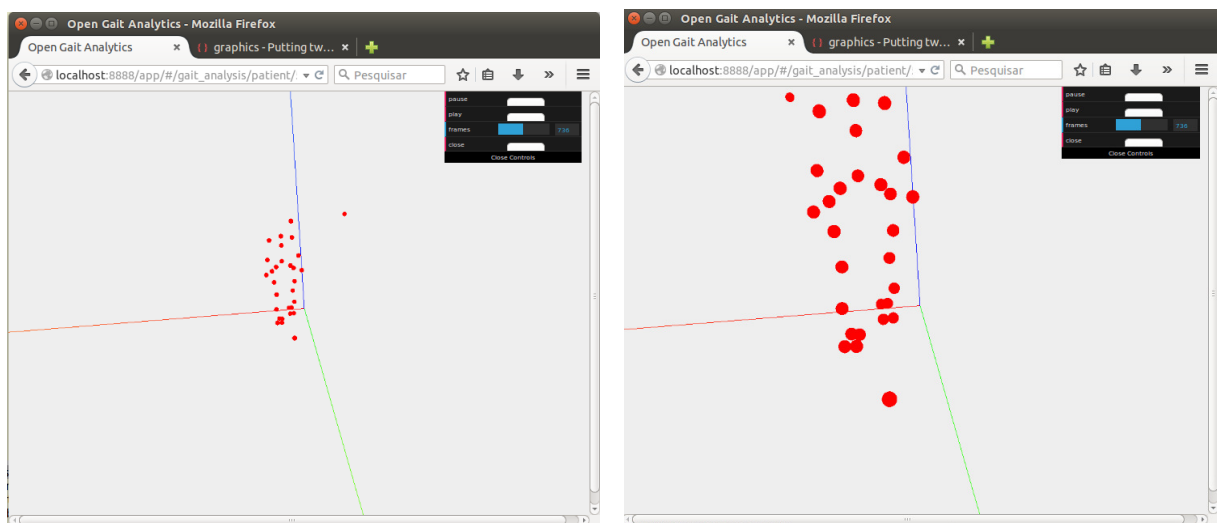


Figura 40 – Controle de zoom.

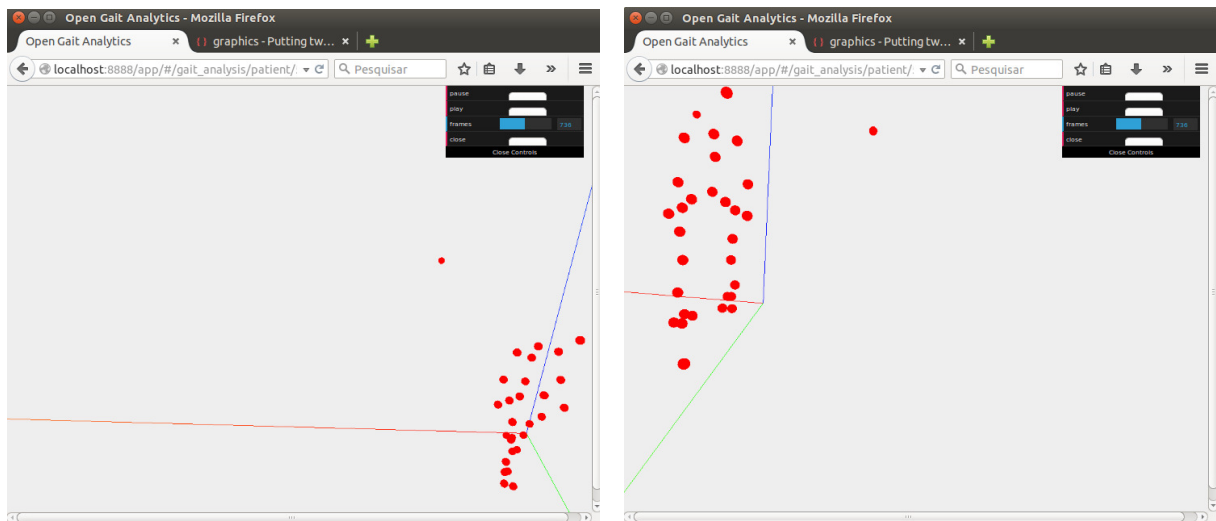


Figura 41 – Controle *pan*.

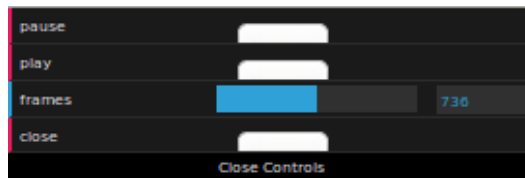


Figura 42 – Controles da animação.

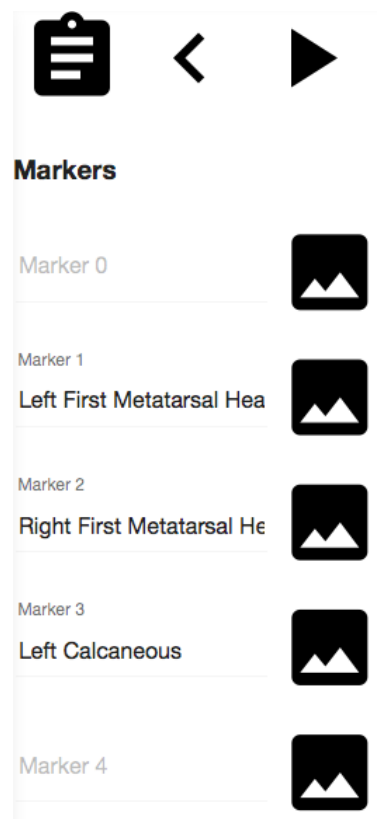


Figura 43 – Opção *markers*.

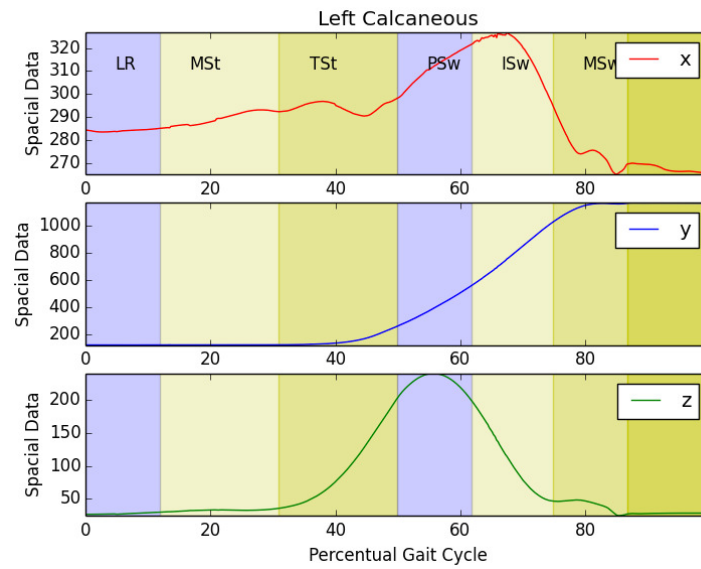


Figura 44 – Progreção espacial de um marcador.

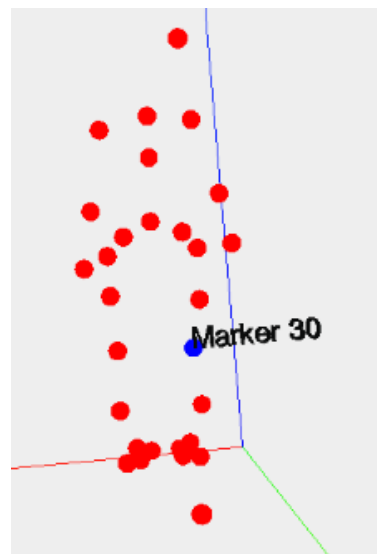


Figura 45 – Seleção de um marcador pelo *mouse*.

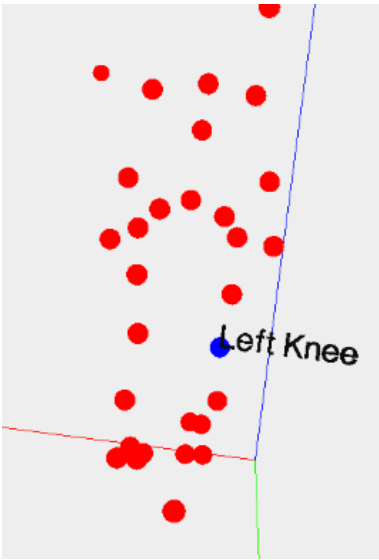











Figura 46 – Animação mostrando o marcador renomeado.



Angles

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




Figura 47 – Opção de visualização e criação de ângulos.

New Angle

Angle Description

Origin ▼

Component A ▼

Component B ▼

Required





Figura 48 – Inclusão de um novo ângulo.

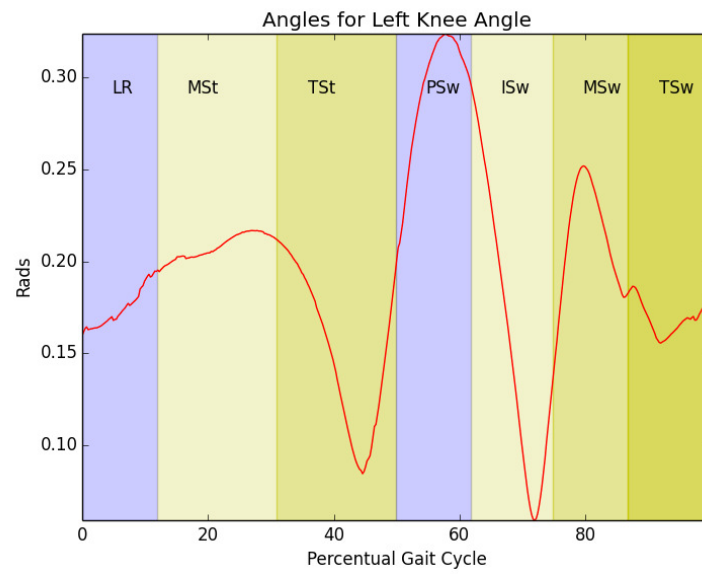


Figura 49 – Angulo de um joelho durante o ciclo de marcha.

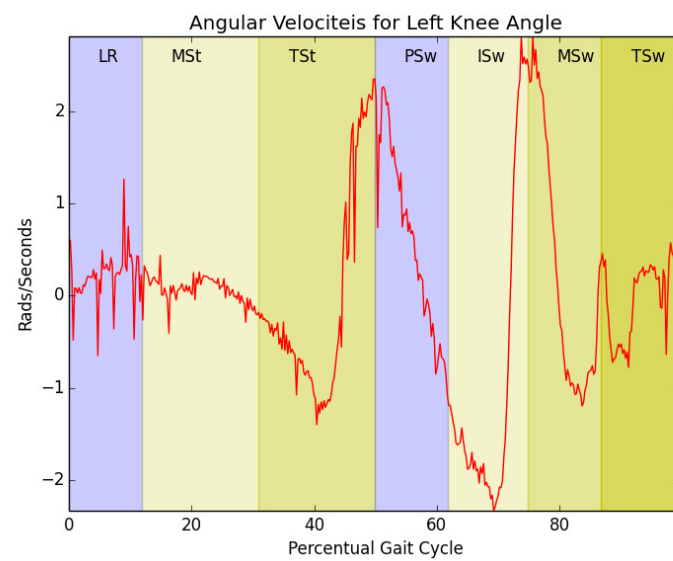


Figura 50 – Velocidades angulares de um joelho durante o ciclo de marcha.

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 mostra 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 da parafernália técnica exigida para fazer simulações num ambiente como o do *MATLAB*.

O primeira funcionalidade de simulação que foi implentada foi a simulação pela *CMAC*, esta RNA foi descrita na seção 2.5. Ao se selecionar o módulo simulação a tela da Figura 51 é apresentada. A opção *CMAC*, já aparece selecionada. Para efeitos da demonstração das funcionalidades deste módulo será realizada uma simulação das velocidades angulares de um joelho, a partir de outros sinais disponíveis.

Após ser selecionado o nome de um paciente, deve-se selecionar uma amostra de ciclo de marcha, Figura 52. 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 53 e 54 mostram uma fração dos sinais possíveis para seleção. Ao se selicionar um sinal de entrada é necessário também informar sua quantização.

A Figura 55 mostra uma configuração que gera a saída da figura 56. Infelizmente até o momento de escrita desta obra, não foi possível implementar a saída na aplicação. O gráfico mostrado é a saída de um protótipo desktop da *CMAC* que usa os mesmos parâmetros de entradas mostrados na aplicação *web*. A figura 57 mostra o erro quadrado médio da execução da simulação ao longo das iterações.

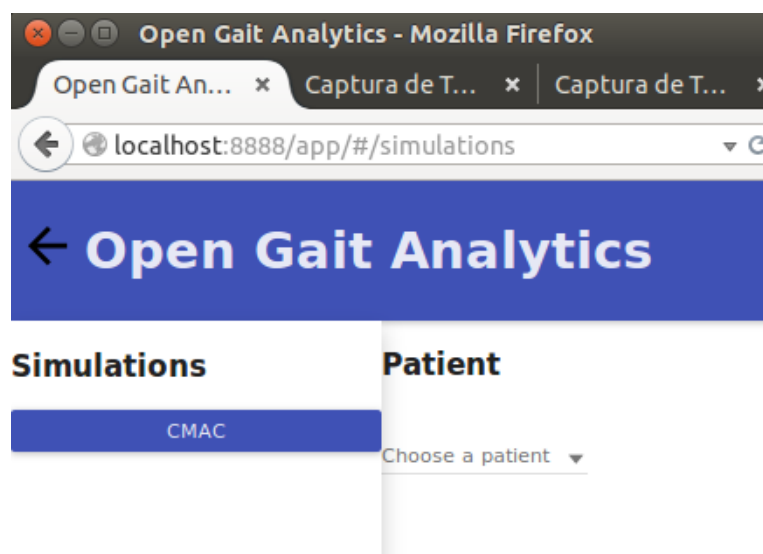


Figura 51 – Módulo de simulação.

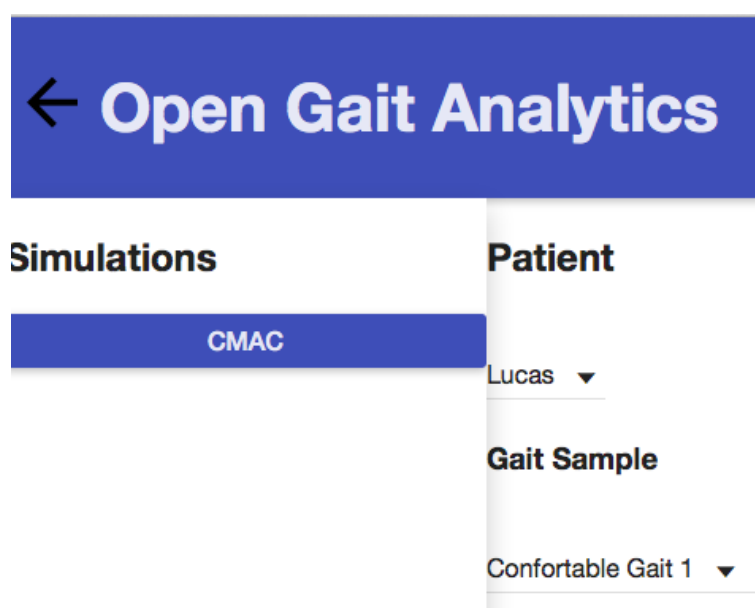


Figura 52 – Seleção do ciclo de marcha.

Input Signals

Select marker positions

Left First Metatarsal Head x ☐

y ☐

z ☐

Right First Metatarsal Head x ☐

y ☐

z ☐

Figura 53 – Sinais de entrada para a *CMAC*. No caso posições num plano 3D de marcadores.

Left Kenee x ☐

y ☐

z ☐

Select angles variables

Right Knee Angle Angles ☐

Angular Velocities ☐

Left Knee Angle Angles ☐

Angular Velocities ☐

Figura 54 – Sinais de entrada para a *CMAC*. Ângulos e velocidades angulares.

z ☐

Left Knee x ☒ Quantization 200

y ☒ Quantization 200

z ☒ Quantization 200

Select angles variables

Right Knee Angle Angles ☐

Angular Velocities ☐

Left Knee Angle Angles ☐

Angular Velocities ☐

CMAC Parameters

Activations Number 30 Iterations Number 50

Output Signal angle - Right Knee Angle - angular velocities ▼




Figura 55 – Exemplo de configuração para uma simulação usando *CMAC*.

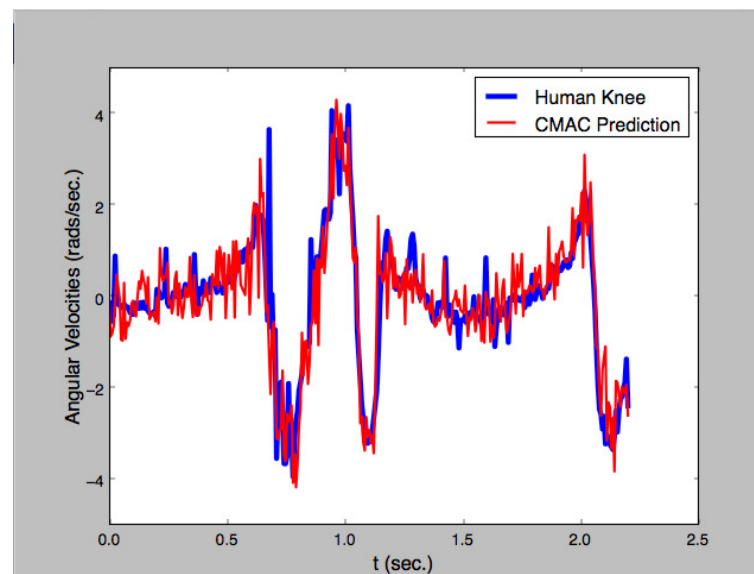


Figura 56 – Resultado da simulação.

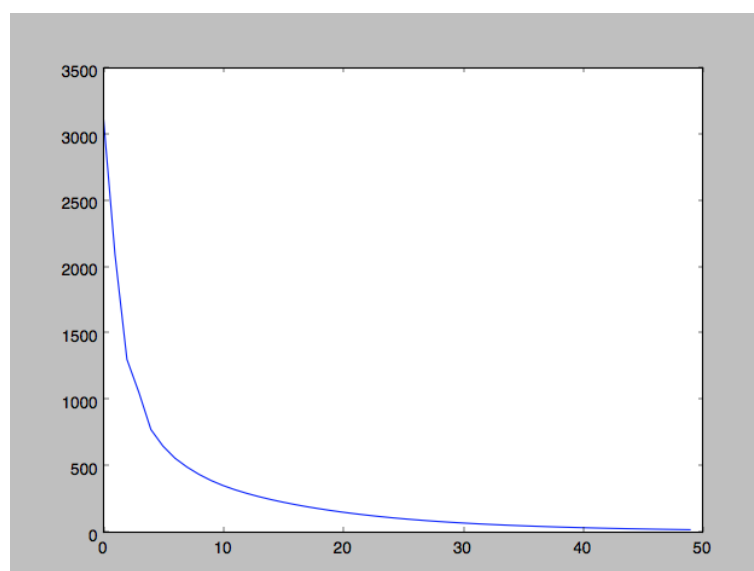


Figura 57 – Erro quadrado médio em cada iteração da simulação

5 DISCUSSÃO E CONCLUSÃO

Há vários softwares de análise de marcha no mercado, mas a idéia de criar um totalmente na *web* tem suas vantagens. A mais óbvia é disponibilidade, já que a aplicação fica num servidor na internet, o usuário só precisa acessar um site com um browser. Outra não tão óbvia e que diz respeito principalmente ao módulo de simulação, é 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 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 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 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.

Rassalta-se novamente, que o software que está sendo entregue não está pronto para produção, o prazo disponível para desenvolvê-lo, inclusive adquirindo conhecimentos sobre muitas das tecnologias adotadas, foi de aproximadamente cinco meses. Isto ocorreu por que 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. Esse estresse arquitetural 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 grande um *Data Center* poder demorar dias. A solução para este problema é desenvolver um método de disparar a simulação assincronamente, e 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 como mencionado acima. 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 *crowd funding* 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.

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.

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, fazendo que toda a arquitetura foi expressada.

6 TRABALHOS FUTUROS

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, 35, 36, 37
- 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>. 35
- 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>. 35
- 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>. 35
- ALBUS, S. A Theory of Cerebellar Function. v. 10, p. 25–61, 1971. 35
- 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. 17
- BECK, K. *Extreme Programming Explained: Embrace Change*. 2. ed. [S.l.]: Addison-Wesley Professional, 2004. ISBN 9780321278654. 24
- BECK, K. et al. *Manifesto para Desenvolvimento Ágil de Software*. 2001. Disponível em: <http://www.agilemanifesto.org/iso/ptbr/>. 21, 22
- 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>. 16
- BRANAS, R. *AngularJS Essentials*. Birmingham: Packt Publishing Ltd., 2014. ISBN 978-1-78398-008-6. 29
- CHODOROW, K. *MongoDB: The Definitive Guide*. 2. ed. [S.l.]: O'Reilly Media, Inc., 2013. ISBN 9781449344689. 33
- CIPPITELLI, E. et al. Kinect as a Tool for Gait Analysis: Validation of a Real-Time Joint Extraction Algorithm Working 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/>. 16
- COHN, M. *Users Stories Applied*. Crawfordsville: Addison Wesley, 2004. ISBN 978-0-321-20568-1. 11, 27, 28
- DAYLEY, B. *Node.js, MongoDB, and AngularJS Web Development*. [S.l.]: Addison-Wesley Professional, 2014. 11, 33, 34, 35

- DIRKSEN, J. *Learning Three.js - the JavaScript 3D Library for WebGL*. 2. ed. [S.l.]: Packt Publishing, 2015. ISBN 9781784392215. 32
- DUHAMEL, a. et al. Statistical tools for clinical gait analysis. *Gait and Posture*, v. 20, n. 2, p. 204–212, 2004. ISSN 09666362. 16
- 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. 16
- 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>. 32
- FOX, A.; PATTERSON, D. *Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing*. [S.l.]: Strawberry Canyon LLC, 2012. ISBN 0984881212. 28
- FREEMAN, A. *Pro AngularJS*. [S.l.]: Apress, 2014. ISBN 9781430264484. 29
- 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. 16
- GOOGLE. *Angular Material Demo Grid List*. 2015. Disponível em: <https://material.angularjs.org/latest/demo/material.components.gridList>. 11, 31
- GOOGLE. *Angular-Seed*. 2015. Disponível em: <https://github.com/angular/angular-seed>. 11, 30
- GOOGLE. *Material Design*. 2015. Disponível em: www.google.com.br/design/spec/material-design/introduction.html. 31
- GREENE, J.; STELLMAN, A. *Learn Agile*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2014. ISBN 9781449331924. 11, 21, 23
- GRINBERG, M. *Flask Web Development*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2014. ISBN 9781449372620. 32
- 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. 19
- 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. 27
- HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. 2. ed. [S.l.]: Prentice Hall, 1998. ISBN 978-0132733502. 38
- 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-anthropomet
11, 20, 22

LAYTON, M. C. *Agile Project Management For Dummies*. 1. ed. [S.l.]: For Dummies, 2012. ISBN 9781118235850. 22

LEITE, W. V. et al. Avaliação cinemática comparativa da marcha humana por meio de unidade inercial e sistema de video. In: *XXIV Congresso Brasileiro de Engenharia Biomédica-CBEB 2014*. Uberlândia: CBEB, 2014. p. 1–4. 11, 19, 21, 22

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>. 35

MAIA, I. *Building Web Applications with Flask*. 1. ed. [S.l.]: Packt Publishing, 2015. ISBN 9781784396152. 33

MALAS, B. *Book Review - Gait Analysis: Normal and Pathological Function, 2nd Edition*. 2010. Disponível em: <http://www.oandp.org/reading/gaitfunction.asp>. 15

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>. 52

MATSUDA, K.; LEA, R. *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL (OpenGL)*. [S.l.]: Addison-Wesley Professional, 2013. ISBN 978-0321902924. 32

MIT. *The MIT License (MIT)*. 2015. Disponível em: <https://opensource.org/licenses/MIT>. 39

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. 16

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. 16

MUYBRIDGE, E. *The Human Figure In Motion*. London: Chapman & Hall, 1885. 11, 17

NG, A. *Machine Learning by Stanford University*. 2015. Disponível em: <https://www.coursera.org/learn/machine-learning/home/welcome>. 39

PATTON, J. *User Story Mapping: Discover the Whole Story, Build the Right Product*. 1. ed. [S.l.]: O'Reilly Media, 2014. ISBN 978-1491904909. 26

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, 15, 18, 19

- PLUGGE, E.; MEMBREY, P.; HOWS, D. *MongoDB Basics MongoDB Basics*. 1. ed. [S.l.]: Apress, 2014. ISBN 9781484208953. 11, 33, 34
- PRESSMAN, R.; MAXIM, B. *Software Engineering: A Practitioner's Approach*. 8. ed. [S.l.]: McGraw-Hill Education, 2014. ISBN 978-0078022128. 11, 24
- 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, 20
- QUALISYS. *Gait analysis & Rehabilitation*. 2013. Disponível em: <http://www.qualisys.com/applications/biomechanics/gait-analysis-and-rehabilitation/>. 11, 21
- QUALISYS. *Oqus MRI*. 2013. Disponível em: <http://www.qualisys.com/products/hardware/oqus-mri/>. 11, 20
- RUBIN, K. S. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. [S.l.]: Addison-Wesley Professional, 2012. ISBN 9780137043293. 25
- RUNYAN, K.; ASHMORE, S. *Introduction to Agile Methods*. [S.l.]: Addison-Wesley Professional, 2014. 24
- 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>. 35
- 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>. 39
- SCHWABER, K. *Agile Project Management with Scrum*. [S.l.]: Microsoft Press, 2004. ISBN 9780735619937. 11, 25, 26
- SCHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. [S.l.]: Pearson, 2001. ISBN 0130676349. 25, 26
- SYED, B. A. *Beginning Node.js*. [S.l.]: Apress, 2014. ISBN 9781484201879. 30
- VIEIRA, A. et al. Software for human gait analysis and classification. In: *4th Portuguese BioEngineering Meeting*. Porto: [s.n.], 2015. 16
- WILLIAMSON, K. *Learning AngularJS*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 9781491916759. 29