CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
Recredenciado pela Portaria Ministerial nº 3.607, de 17/10/05, D.O.U. nº 202, de 20/10/2005
ASSOCIAÇÃO EDUCACIONAL LUTERANA DO BRASIL

Leomar Camargo de Souza

AVALIAÇÃO ANTROPOMÉTRICA ATRAVÉS DA DELIMITAÇÃO DA SILHUETA CORPORAL UTILIZANDO O KINECT

Palmas – TO

2015

Leomar Camargo de Souza

AVALIAÇÃO ANTROPOMÉTRICA ATRAVÉS DA DELIMITAÇÃO DA SILHUETA CORPORAL UTILIZANDO O KINECT

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Fabiano Fagundes

Palmas – TO

2015

Leomar Camargo de Souza

AVALIAÇÃO ANTROPOMÉTRICA ATRAVÉS DA DELIMITAÇÃO DA SILHUETA CORPORAL UTILIZANDO O KINECT

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Fabiano Fagundes

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. M.Sc. Fabiano Fagundes

Centro Universitário Luterano de Palmas - CEULP

_____

Prof. M.Sc. Fernando Luiz de Oliveira

Centro Universitário Luterano de Palmas - CEULP

_____

Prof. M.Sc. Pierre Soares Brandão

Centro Universitário Luterano de Palmas - CEULP

Palmas - TO

2015

Aos meus pais e minha irmã que, com muito apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

AGRADECIMENTOS

Aos meus pais, minha irmã e minha avó, apenas posso agradecer por tudo que me proporcionaram, pois nunca conseguirei compensar devidamente a dedicação que sempre manifestaram.

Ao meu orientador, Fabiano Fagundes, agradeço por tudo que me ensinou, pela paciência, pela insistência, pelas cobranças, pelos conselhos, por ter mostrado a minha real capacidade quando eu menos acreditava.

Aos professores da banca, Pierre Brandão e Fernando Luiz, agradeço pelas contribuições, pois cada dica foi de grande relevância para todos os trabalhos desenvolvidos ao longo do curso.

Aos professores Jackson, Madianita, Cristina, Parcilene e Edeilson, agradeço por todo conhecimento proporcionado e por mostrarem que os cursos de Ciência da Computação e Sistemas de Informação do Centro Universitário Luterano de Palmas não possuem apenas um grupo de professores e sim, uma família de professores extremamente dedicada.

A Rayssa Leão - que sem sombra de dúvidas é uma das pessoas mais incríveis que conheci - agradeço em especial pelas conversas, conselhos, zoeiras e apoio até aqui.

Também não posso deixar de mencionar outros nomes fundamentais nessa jornada. São eles: Jesiel Padilha, Jhonatan Mota, Ranyelson Neres, Marcus Germano, João Neto, João Elias, Jackeline Miranda e Thiago Mendes.

Enfim, agradeço a todos os amigos e colegas por cada momento que passamos juntos. Espero que continuem presentes nessa nova jornada que está por vir. Muito obrigado!

RESUMO

SOUZA, Leomar Camargo de. **Avaliação antropométrica através da delimitação da silhueta corporal utilizando o Kinect**. 2015. 85 f. TCC (Graduação) - Curso de Ciência da Computação, Centro Universitário Luterano de Palmas, Palmas, 2015.

A obtenção dos dados antropométricos é de extrema importância para a avaliação e adequação do ser humano em um determinado ambiente, uma vez que estes dados permitem obter diversas informações como, por exemplo, comparações entre o grau de adequação das pessoas a um ambiente de trabalho específico. O Kinect é uma das tecnologias que automatiza o processo de obtenção dos dados antropométricos da pessoa já que, com o uso dos sensores RGB e de profundidade presentes no dispositivo, é possível a visualização da representação do esqueleto do ser humano, bem como de suas articulações, em tempo real. Através da limiarização juntamente com os recursos oferecidos pelo sensor do Kinect, é possível a obtenção da representação do corpo humano, que pode ser utilizado para resolução de diversos problemas, como por exemplo, nos *softwares* que visam automatizar o processo de obtenção dos dados antropométricos.

PALAVRAS-CHAVE: Antropometria; Limiarização; Microsoft Kinect.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE FÓRMULAS E EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ADI - Análise Digital de Imagens

CEULP - Centro Universitário Luterano de Palmas

GEPETS - Grupo de Estudos e Pesquisa em Tecnologia, Saúde e Qualidade de Vida

ICC - *Intraclass Correlation Coefficient*

IMP - *Image Marking Procedure*

PA - Posição Anatômica

PADI - Processamento e Análise Digital de Imagens

PDI - Processamento Digital de Imagens

RGB - *Red, Green, Blue*

URL - *Uniform Resource Locator*

WPF - *Windows Presentation Foundation*

SUMÁRIO

1 INTRODUÇÃO

Está se tornando cada vez mais comum encontrar *softwares* voltados à área da saúde com o objetivo de propiciar um maior desempenho e facilidade aos profissionais da área na execução de suas tarefas. Desse modo, a saúde é uma das pioneiras na utilização de aplicações de visão computacional por meio de tecnologias de análise e processamento de imagens (BALLARD; BROWN, 1982). O Microsoft Kinect - pertencente ao console *Xbox* - é um dos acessórios que vem ganhando destaque nos últimos anos, devido aos recursos que o dispositivo oferece tanto para o desenvolvedor quanto para os usuários.

Diversos estudos, como por exemplo, os realizados por Souza *et al* (2015), Fiorini (2014) e Alves, Araújo e Madeiro (2012) demonstram que a fisioterapia é uma das áreas que vem se destacando nos últimos anos com a modernização das técnicas e avanço das pesquisas na utilização do Microsoft Kinect em prol da saúde, uma vez que o acessório pode ser usado na resolução de diversos problemas, como por exemplo, a reabilitação de pacientes com lesões neurológicas e musculares, além de ser útil na obtenção de alguns dados antropométricos do ser humano.

O Kinect oferece recursos que são capazes de fornecer certos dados antropométricos do ser humano. Porém, em alguns casos, esses recursos são limitados quando se deseja obter resultados com maior exatidão. Um exemplo disso é o mapeamento das articulações do corpo humano, na qual, os pontos mapeados pelo dispositivo divergem em relação aos pontos que são mapeados manualmente. Esse problema foi constatado por Chaves (2014) em seu estudo, onde se concluiu que o uso apenas dos pontos (*joints*) nativos oferecidos pelo Kinect se tornava inviável no cálculo do Índice de Percepção Corporal da Imagem Real, utilizando a técnica de Procedimento de Marcação do Esquema Corporal ou *Image Marking Procedure* (*IMP*).

Nesse mapeamento das articulações através método tradicional, o ser humano geralmente encontra-se na posição anatômica, a fim propiciar um estudo minucioso acerca das estruturas presentes no corpo humano. Para isso, o corpo deve encontrar-se em um dos planos de referências ou secção, de forma que ele seja “divido” em partes menores, com o objetivo de auxiliar a visualização das estruturas presentes em cada parte. Um dos planos de secção é plano sagital, que divide o corpo humano ao meio, de cima para baixo, originando assim duas metades simétricas, uma direita e outra esquerda (VERDERI, 2003).

Uma tecnologia que auxilia com maior precisão a obtenção dos pontos mapeados é a utilização um mecanismo computacional para delimitação de áreas sobre a imagem fornecida pelo sensor de profundidade do Kinect, a fim de se obter apenas a representação do corpo

humano. A segmentação é um dos mecanismos utilizados no contexto de obtenção de objetos/regiões em imagens computacionais. Essa técnica consiste na divisão em duas partes do histograma de uma imagem, onde os pixels com um tom de cinza maior ou igual a um valor de um limiar (T) são convertidos em branco e o restante em preto (FERRO NETO, 2012).

Assim sendo, o presente trabalho teve por objetivo testar a viabilidade do uso da limiarização juntamente com os recursos oferecidos pelo Kinect, de forma que fosse possível a obtenção da representação da silhueta do corpo humano para que posteriormente fossem calculados os seguintes atributos antropométricos: a estatura corporal, a largura dos ombros e a largura da cintura.

2 REFERENCIAL TEÓRICO

2.1 Anatomia humana

A anatomia (do grego *anatemnein*, onde *ana* significa partes, e *temnein* significa cortar) humana é a ciência capaz de descrever a fisionomia, o desenvolvimento, arquitetura e estrutura do corpo humano (SILVA, 2007). O estudo da anatomia vem sendo realizado há muito tempo e supõe-se que os egípcios foram os precursores da área, seguido pelos mesopotâmios até alcançar grandes estudiosos, como: Galeno, Leonardo da Vinci, Michelangelo e Andreas Versallius (PIAZZA, 2011).

Segundo Davis (2011), o estudo da anatomia é fundamental na compreensão de outras ciências médicas, como a fisioterapia, patologia, odontologia etc., uma vez que o estudo possibilita o entendimento dos detalhes estruturais do corpo humano. Sendo assim, faz-se necessário o estudo da fisiologia – representação da função do corpo humano – uma vez que "toda estrutura tende a refletir uma função" (GRAAFF, 2013, p.2). Drake *et al.* (2010) define duas abordagens para o estudo da anatomia: a primeira é a abordagem regional, que consiste no estudo do corpo humano por regiões e seus respectivos aspectos, e a segunda é a abordagem sistêmica, que consiste no estudo de cada sistema do corpo humano. De acordo com Limeira Junior (2009) atualmente a anatomia está subdividida em anatomia macroscópica e microscópica:

- Anatomia macroscópica – referente à estrutura do corpo humano visualizada a olho nu, com ou sem a utilização de recursos tecnológicos;
- Anatomia microscópica – referente à estrutura do corpo humano invisível a olho nu, fazendo com que haja necessidade do uso de alguma técnica de ampliação, como, por exemplo, lupas ou microscópio. Este grupo é fragmentado em citologia que é o estudo das células, e histologia que é o estudo dos tecidos e suas organizações.

Dangelo e Fattini (2002, p.2) definem onze sistemas que constituem o organismo humano, são eles: sistema tegumentar; sistema esquelético; sistema muscular; sistema nervoso; sistema circulatório; sistema respiratório; sistema digestivo; sistema urinário; sistema genital - feminino e masculino; sistema endócrino; sistema sensorial. O corpo humano está organizado desde o modo mais simples, que compreende a menor dimensão dos componentes do corpo, ao modo mais complexo, que compreende a maior dimensão dos componentes do corpo (BORBA, 2015). A Figura 1 exibe a organização do corpo humano.

Figura 1. Níveis de organização do corpo humano

Segundo Borba (2015) os níveis de organização do corpo humano apresentados na imagem acima podem ser sintetizados da seguinte maneira: as células formam os tecidos, que por sua vez compõe os órgãos que constituem os sistemas, formando assim o organismo.

2.1.1 Divisão do corpo humano

De acordo com Dangelo e Fattini (2002, p. 4) o corpo humano é divido em partes, sendo elas: cabeça, pescoço, tronco e membros - superior e inferior. Cada parte é subdivida, conforme o Quadro 1.

Quadro 1. Subdivisão do corpo humano

PARTE	SUBDIVISÃO
Cabeça	Crânio e face
Pescoço	Pescoço
Tronco	Tórax, abdômen e pelve
Membro Superior	Ombro, braço, antebraço e mão
Membro Inferior	Quadril, coxa, perna e pé

Fonte: Limeira Junior (2009, p. 63).

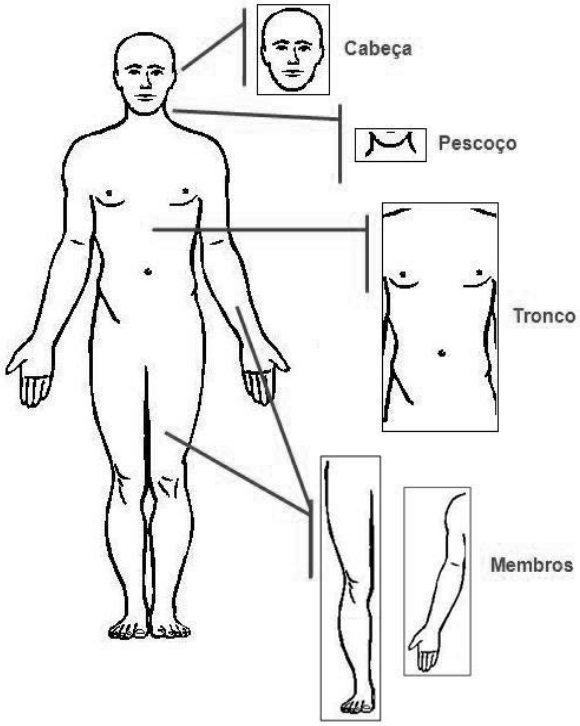

Figura 2. Divisão do corpo humano

Fonte: Ciências Morfológicas[1]

A Figura 2 ilustra o corpo humano com suas respectivas divisões, conforme especificado pelo Quadro 1.

2.1.2 Posição anatômica e plano anatômico

A Posição Anatômica (PA) do ponto de vista da anatomia humana é a posição que permite obter um nível mais detalhado de estudo do corpo humano, bem como, uma maior referência das estruturas presentes no mesmo (FRONZA, 2011). Este mesmo autor também explica que a PA em companhia dos Planos Anatômicos é capaz de fornecer uma referência da localização de um tecido, membro, órgão ou estrutura corporal, onde estas referências são fornecidas com ajudas de planos imaginários.

No plano anatômico o corpo encontra-se numa posição ereta, onde o rosto está voltado para frente com o olhar voltado ao horizonte, os "membros superiores estendidos paralelos ao tronco com as palmas das mãos voltadas para frente, membros inferiores estendidos e unidos com os pés voltados para frente" (CRUZ, 2014, online). A Figura 3 representa o corpo na posição anatômica.

[1] Disponível em: <http://goo.gl/oZzXpD>. Acesso em 05 de maio de 2015.

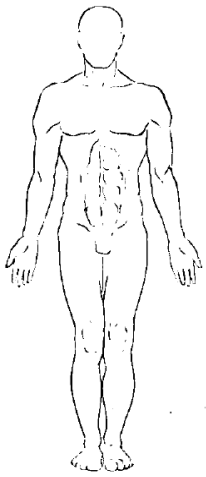

Figura 3. Corpo na posição na posição anatômica

Fonte: Dangelo e Fattini, 2002, p. 5

Por meio da observação de um conjunto de seres humanos é possível evidenciar as diferenças estruturais dos componentes que fazem parte do conjunto, que são denominadas de variações anatômicas, que podem se manifestar externamente ou em qualquer um dos sistemas do organismo, de forma que não afete a vida útil da pessoa (DANGELO; FATTINI, 2002, p. 1).

2.1.3 Planos de secção e termos de referências

Um plano de secção ou referência é uma espécie de "corte", onde o corpo é divido em partes menores, com o objetivo auxiliar a visualização das estruturas do mesmo (COSTA, 2008). Conforme Graaff (2003) existem três planos de secção no corpo humano: plano sagital, plano frontal e plano transversal. A Figura 4 ilustra esses planos.

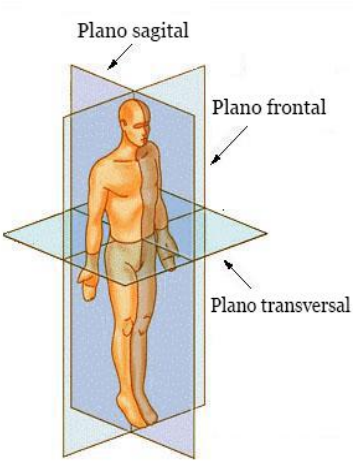

Figura 4. Representação dos planos de secção no corpo humano

Fonte: Enfermagem, Ciência e Arte no Cuidar[2]

[2] Disponível em: <http://goo.gl/jbMgno>. Acesso em 05 de maio de 2015.

Com a finalidade de classificar as estruturas corporais em relação às outras são utilizado alguns termos de direção (TORTORA, 2001), como os apresentado no Quadro 2, a seguir.

Quadro 2. Termos de direção do corpo humano

TERMO	DEFINIÇÃO	EXEMPLO
Superior (cefálico ou cranial)	Em direção à cabeça ou parte superior de uma estrutura	O coração é superior ao fígado
Inferior	Em direção à cabeça ou em direção à parte inferior de uma estrutura	O estômago é inferior aos pulmões
Anterior	Mais perto ou na frente do corpo	O coração é anterior à coluna vertebral
Posterior	Mais perto ou no dorso do corpo	O esôfago é posterior à traqueia
Medial	Mais próximo à linha mediana do copo ou de uma estrutura	A ulna está na porção medial do antebraço
Lateral	Distante da linha mediana do corpo ou de uma estrutura	O rádio está na porção lateral do antebraço
Intermédio	Entre duas estruturas	O dedo anular é intermédio entre o dedo médio e dedo mínimo
Proximal	Mais próximo à fixação de um membro ao tronco ou estrutura; mais próximo ao ponto de origem	O úmero é proximal ao rádio.
Distal	Mais distante da fixação de um membro ao tronco ou estrutura; mais distante do ponto de origem	As falanges são distais aos carpais
Superficial	Em direção à superfície do corpo	O esterno é superficial ao coração
Profundo	Distante da superfície do corpo	As costelas são profundas à pele do tórax

Fonte: TORTORA, 2001, p. 8

Como exibido no Quadro 2, ao total existem onze termos direção do corpo humano, que são utilizados com frequência pelos anatomistas.

2.1.3.1 Plano sagital

O plano sagital pode ser definido como um desenho imaginário, que divide o corpo humano ao meio, de cima para baixo, originando assim duas metades simétricas, uma direita e outra esquerda (VERDERI, 2003). A Figura 5 ilustra melhor o plano sagital.

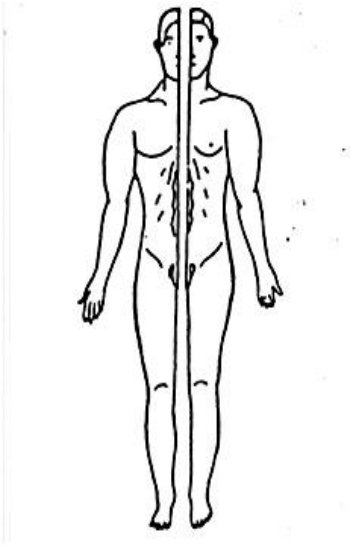

Figura 5. Visão do corpo humano no plano sagital

Fonte: Unama[3]

De acordo com Cunha (2011) no momento em que um elemento anatômico estiver próximo ao plano sagital mediano, este é denominado de medial ou interno e quando o mesmo se distância pode ser denominado lateral ou externo.

Uma vez que os conceitos de anatomia humana foram assimilados, faz-se necessário, para realização do presente trabalho a compreensão de conceitos de processamento digital de imagens, que serão apresentados nas próximas seções.

2.2 Processamento digital de imagens

O processamento digital de imagem é uma área que está em constante crescimento, onde existem diversos domínios de estudos, como a interpretação e extração de características, análise em multi-resolução e multi-frequência, codificação e transmissão de imagens, etc. (ALBUQUERQUE; ALBUQUERQUE; 2000). O processamento e análise de imagens digitais baseiam-se no uso de operações matemáticas capazes de modificar os valores dos pixels de uma imagem para facilitar a visualização de suas características e até mesmo as prepararem para que possam ser analisada por um computador (GOMES, 2001, p. 4).

Ao processar uma imagem não pode ser considerada a execução de apenas uma tarefa, uma vez que este processo exige a execução de várias tarefas, de modo que a tarefa inicial se dá no momento em que a imagem for capturada, cuja etapa conhecida como pré-processamento, que "envolve passos como a filtragem de ruídos introduzidos pelos sensores e a correção de distorções geométricas causadas pelo sensor" (QUEIROZ; GOMES, 2011).

[3] Disponível em: <http://goo.gl/GYKCN0>. Acesso em 06 de maio de 2015.

Paciornik (2001) elaborou uma sequência padrão de estudo, a fim de estruturar os passos do Processamento e Análise de Imagens Digitais (PADI) de forma que os procedimentos fossem separados em etapas. A Figura 6 apresenta o fluxograma com a sequência padrão de PADI.

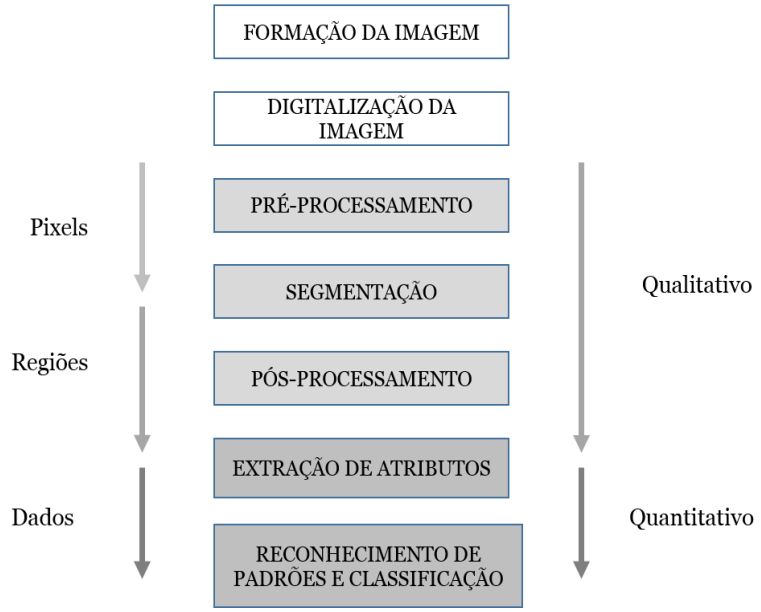

Figura 6. Sequência padrão de PADI (GOMES, 2001)

Fonte: Adaptado de IGNACIO, 2013, p.25

A sequência de PADI é dividida em três categorias básicas, que são, segundo Gomes (2001, p. 42, *apud* PACIORNIK, 2001): Aquisição, Processamento Digital de Imagens (PDI) e Análise Digital de Imagens (ADI).

- Aquisição - é o processo que engloba as etapas de formação da imagem e digitalização da imagem; a primeira etapa está associada à eficácia e à qualidade do equipamento no momento de captura da imagem, e a segunda etapa é referente ao momento em que a imagem é transformada em uma representação numérica que, a partir desta etapa, fica em condições para realização de operações nos seus pixels (IGNACIO, 2013);
- PDI - é o processo responsável por receber a imagem após a realização das etapas de Aquisição; seu principal objetivo é fornecer uma imagem final totalmente processada. Seu processo consiste apenas na etapa de pré-processamento, que tem por objetivo melhorar a imagem, realizando as correções ideais para que ela seja processada nas etapas seguintes (IGNACIO, 2013);
- ADI - é o processo que herda a imagem dos processos anteriores, que é composto pelas quatro etapas finais da sequência padrão de PADI (IGNACIO, 2013):

    - segmentação - é responsável pelo reconhecimento dos conteúdos de interesse presentes na imagem, tornando-se a etapa mais notável, já que qualquer procedimento mal executado tende a anular o processo;
    - pós-processamento - é responsável por coletar os conteúdos de interesse - obtidos na etapa anterior - da imagem e realizar melhorias, como suavização e correções de distorções. É de extrema importância da etapa, a produção de um artefato que não interfira no processo de análise da imagem;
    - extração de atributos - é responsável por transformar os conteúdos da imagem e em valores numéricos que serão aproveitados na pesquisa;
    - reconhecimento de padrões e classificação - é responsável por identificar e classificar os conteúdos extraídos da imagem.

Manzi (2007) classifica as técnicas de processamento de imagem em duas áreas: a análise, cujo objetivo é a extração de características presentes na imagem e a melhoria que tem por objetivo reparar as imperfeições de uma imagem de modo que sua análise torne-se mais simples. As técnicas abaixo são frequentemente utilizados no PADI, conforme Ignacio (2013):

- operações aritméticas
    - adição - consiste na redução de ruídos de uma imagem;
    - subtração - consiste na remoção de dados inertes ao fundo de uma imagem e identificação de desigualdades entre imagens;
    - multiplicação - consiste em calibrar o brilho de uma imagem;
    - divisão - consiste em regularizar o brilho da imagem;
- operadores lógicos - são aplicados em imagens binárias “com uso no mascaramento, detecção de características e análise de forma”;
- filtragem: são técnicas com o propósito de reparar, atenuar ou realçar os aspectos de uma imagem dentro de um domínio específico;
    - domínios de filtragem
        - espacial - atua diretamente nos pixels da imagem em seu formato original;
        - frequência - atua sobre uma determinada função a partir da imagem em seu formato original;
    - tipos de filtros
        - lineares - é responsável por suavizar, destacar e minimizar efeitos de uma imagem sem realizar modificações no nível médio de cinza da mesma;

- não-lineares - é responsável por aplicar os filtros sem se preocupar o nível médio de cinza de uma imagem;

O surgimento do PADI fez com que diversas áreas como a biologia, medicina, física, geografia, arqueologia entre outras a aderissem, tornando-se assim uma tecnologia de extrema importância (IGNÁCIO, 2013).

2.2.1 Imagem digital

De acordo com o Dicionário Aurélio (1975) uma imagem é a representação de uma pessoa ou objeto, e um pixel é um conjunto de pontos que forma uma imagem. No contexto computacional, uma imagem consiste em uma representação bidimensional "de um objeto como um conjunto finito de valores digitais inteiros, onde cada valor é chamado de *picture element*, ou *pixel*" (DIAS, 2008, p. 33). Segundo Bessani (2012 *apud* Mascarenhas; Velasco, 1989) uma imagem consiste em função matemática, denotada por F(x, y), onde x e y representam as coordenadas de cada pixel da imagem e F representa a intensidade de brilho no pixel (x, y).

Nesse contexto, a quantidade de pixels existentes em uma imagem determina a sua resolução. De acordo com Oliveira (2010) a forma mais comum de representar uma imagem é através do modo matricial, na qual os elementos da matriz representam os pixels e os índices dos elementos indicam os valores de intensidade presentes nos pixels. Dessa forma, por ser uma matriz o início da imagem se encontra no primeiro pixel do canto superior esquerdo, conforme ilustra a Figura 7.

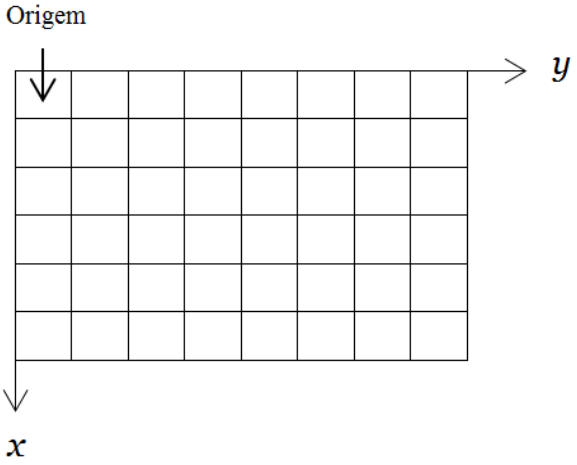

Figura 7. Representação de uma imagem no modo matricial

Fonte: Adaptado de OLIVEIRA, 2010, p.28

As características de uma imagem variam de acordo com o seu número de cores, que em certos casos são definidos pelo usuário no momento aquisição da imagem (SOUZA, 2000).

2.2.1.1 Imagem binária

Uma imagem binária normalmente é o resultado da aplicação de algum processamento - geralmente limiarização - na qual os pixels assumem valores de zero (preto) ou um (branco) (BESSANI, 2012). Segundo Souza (2000, p. 18), uma imagem binária

> "é normalmente utilizada para distinguir dois tipos de elementos, o fundo e os objetos de interesse. A maioria dos algoritmos de análise de imagem operam sobre este tipo, retirando informações relacionadas à geometria dos objetos, como área, perímetro, ou quantificando estes elementos".

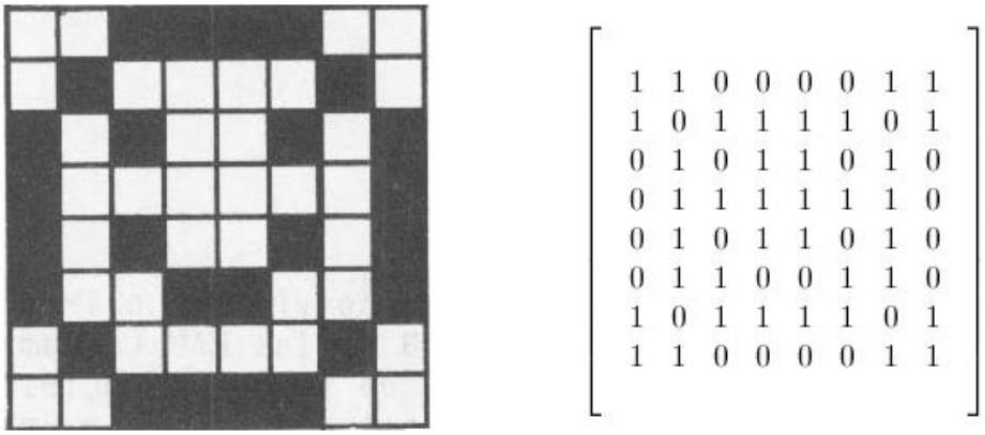

Figura 8. Imagem binária com sua representação numérica

Fonte: IGNACIO, 2013, p. 20

A Figura 8 ilustra uma imagem binária e a representação numérica - ao lado esquerdo - de seus respectivos pixels - ao lado direito.

2.2.1.2 Imagem em tons de cinza

De acordo com Souza (2000) uma imagem binária é composta por 8 *bits*, onde cada pixel equivale a um valor inteiro positivo entre 0 e 255. Desse modo, uma imagem em tons de cinza possui menos informações se comparada a uma imagem colorida, porém, isso não a torna menos eficiente na busca por resultados satisfatórios (PERROTTI, 2015).

2.2.1.3 Imagem colorida

Bessani (2012, p. 36) define uma imagem colorida como "uma composição de três cores primárias em uma única imagem, onde a cor de cada pixel será uma tripla ordenada de intensidade (R, G, B)". Sendo assim, a representação de uma imagem colorida é composta por 24 *bits* e cada cor primária é composta por 8 *bits* (SOUZA, 2000).

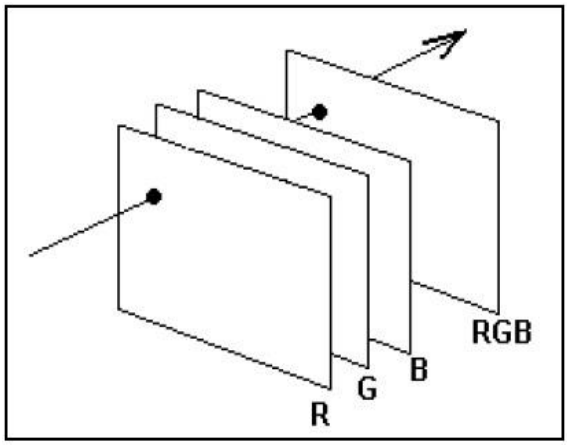

Figura 9. Formação de uma imagem colorida

Fonte: SOUZA, 2000, p. 20

A Figura 9 ilustra a representação da formação de uma imagem colorida com suas respectivas composições R (*red*), G (*green*), B (*blue*).

2.2.2 Segmentação

O processo de segmentação, de um modo geral, é o primeiro passo ao analisar uma imagem (QUEIROZ; GOMES, 2001). Esse processo consiste na identificação de regiões de objetos relevantes em uma imagem, sendo uma das etapas mais complexas de se programar (MARQUES FILHO; VIERA NETO, 1999). De acordo com Queiroz e Gomes (2001, p. 25), os algoritmos de segmentação consistem na distinção de um ou mais objetos "tanto entre si quanto entre si e o *background*". O sistema visual do ser humano tem a capacidade de executar essa função naturalmente, já não encontra dificuldades em localizar objetos ao observar uma determinada imagem (CONTI, 2010).

Segunda Vega (2011) "não existe um modelo padrão para a segmentação, o processo é essencialmente empírico e deverá se ajustar a diferentes tipos de imagem". O resultado obtido na extração de um objeto em uma imagem é inerente ao grau de processamento aplicado aos dados da imagem (SALDANHA; FREITAS, 2009). Existem duas peculiaridades em comum aos objetos que compõe uma imagem: a semelhança interna conforme a propriedade da imagem e o contraste em relação a sua vizinhança (SALDANHA; FREITAS, 2009).

Conforme Silva (2012) "uma imagem geralmente pode ser segmentada de acordo com duas propriedades dos seus níveis de cinza: descontinuidades ou similaridades, gerando, respectivamente, dois tipos básicos de segmentos: fronteiras ou regiões".

2.2.2.1 Descontinuidade

De acordo com Silva (2012), existem três formas de identificar uma descontinuidade em uma imagem. São elas:

- um ponto isolado - é o modo mais simples, que ocorre se um pixel conter um nível cinza com valor diferente de sua vizinhança (SILVA, 2012);
- linhas - realiza-se a verificação para descobrir se os pixels equivalentes estão em linha. O uso de mascaras pode auxiliar na descoberta (SILVA, 2012);
- borda de objeto - esse processo visa encontrar e ressaltar os pixels bordas, onde aumenta-se a diferença entre o *background* e a borda da imagem (SILVA, 2012);

2.2.2.2 Transformada de Hough

A Transformada de Hough (TH) é uma técnica desenvolvida por Paul Hough em 1962 - sendo posteriormente patenteada pela empresa IBM - para descoberta de padrões mais complexos em imagens, como círculos, linhas, elipses etc. (PIVETTA; MANTOVANI; ZOTTIS, 2010; DUDA; HART, 1972). Geralmente, esses padrões são "encontrados com descontinuidades e com inclusão de ruídos" (DUARTE, S. d, p. 1). A Figura 10 ilustra as etapas utilizadas pela TH.

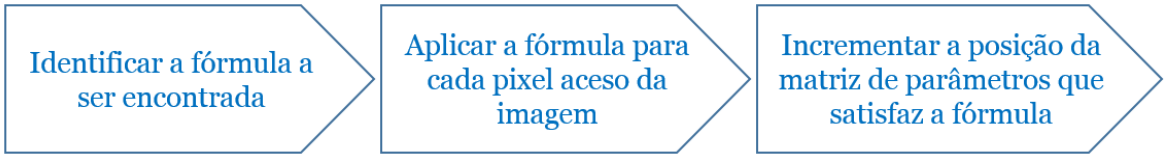

Figura 10. Etapas da Transformada de Hough para descoberta de uma forma geométrica

Fonte: Adaptado de MACEDO, 2005, p. 4

De acordo com Silva (2012, p. 38)

> o conceito principal da transformada está em definir um mapeamento entre o espaço de imagem e o espaço de parâmetros. Cada borda de uma imagem é transformada pelo mapeamento para determinar células no espaço de parâmetros, indicadas pelas primitivas definidas através do ponto analisado. Essas células são incrementadas e indicarão, no final do processo, através da máxima local do acumulador, quais os parâmetros correspondem à forma especificada.

2.2.2.3 Limiarização

A limiarização ou *threshold* é uma das técnicas mais simples na descoberta de objetos ou parte de objetos presentes em uma imagem (FERRAREZI, 2010). De acordo com Marques Filho e Vieira Neto (1999) a limiarização também é conhecida como binarização, já que uma imagem binária é obtida ao final do processamento. Essa técnica consiste na divisão em duas partes do histograma de uma imagem, onde os pixels com tom de cinza maior ou igual a um valor de um limiar (T) são convertidos em branco e o restante em preto (FERRO NETO, 2012).

No princípio matemático a limiarização consiste em um método de PDI na qual, uma imagem de entrada $f(x, y)$ de N níveis de cinza resulta em uma imagem $g(x, y)$ com níveis de

cinza menores que *N*. Desse modo, a imagem resultante apresenta dois níveis de cinza, sendo 0 para preto se *N* for maior ou igual a limiar (T) e 1 para branco se *N* for menor que a limiar (T) (MARQUES FILHO; VIEIRA NETO, 1999).

2.3 Recursos do Kinect para o processamento de imagens

O Kinect é um dispositivo desenvolvido pela Microsoft em 2010 para os consoles Xbox One e 360 que dispensa o uso de controles, fazendo com que o usuário utilize a mão para interagir com o dispositivo (MICROSOFT, 2015). Os seus componentes são: uma câmera RGB, uma câmera infravermelha, um conjunto de microfones e um motor. Com esses recursos, o dispositivo permite o processamento de cores em imagem, profundidade, além de identificar o esqueleto do corpo humano (PAULA, 2011; MICROSOFT, 2015). Em maio de 2011, a Microsoft lançou o Kinect SDK, um conjunto de ferramentas que possibilita o desenvolvimento de aplicações utilizando os recursos oferecidos pelo dispositivo (BOUCHARD, 2011). A Figura 11 exibe um resumo da arquitetura básica do funcionamento de uma aplicação que utiliza os recursos do sensor Kinect.

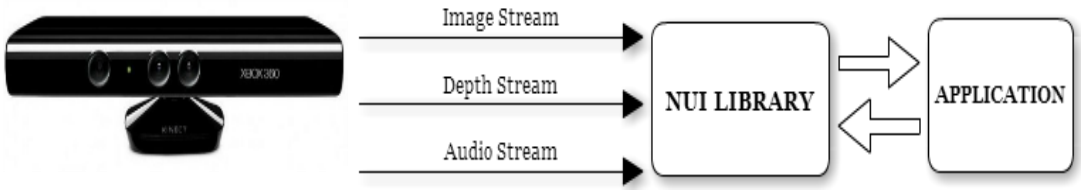

Figura 11. Resumo da arquitetura básica do Microsoft Kinect

Fonte: Adaptado de Microsoft (2015)

A câmera RGB do dispositivo fornece imagem com resolução de 640x480 em 30 frames por segundo ou 1280x1024 em no máximo 15 frames por segundo (MICROSOFT, 2014; PAULA, 2011). O sensor de profundidade presente na câmera infravermelha permite a digitalização em três dimensões do ambiente que está em volta da pessoa, permitindo a distinção entre o fundo e a pessoa (MICROSOFT, 2015). Desse modo, o sensor de profundidade associado junto à câmera RGB do Kinect permite o mapeamento de superfície e o reconhecimento de objetos (CHAVES, 2014).

Em seus estudos, Panger (2012) cita que o sensor de profundidade é muito eficiente no reconhecimento de objetos através do gesto dos usuários. Desse modo, as características que são obtidas pelo Kinect têm chamado à atenção dos pesquisadores da área de modelagem e mapeamento 3D (KHOSHELHAM; ELBERINK, 2012). A Figura 12 mostra um exemplo de uma aplicação que utiliza o sensor de profundidade para distinguir o ser humano de um determinado cenário.

Figura 12. Exemplo do uso do sensor de profundidade do Kinect

Fonte: Norrislabs[4]

Em suma, o sensor Microsoft Kinect fornece imagens de qualidade que possibilitam a distinção de objetos presentes no campo de visão do dispositivo. Além disso, o Kinect permite que seja calculada a distância dos objetos expostos no seu panorama (LÓPEZ, 2012).

[4] Disponível em: <http://goo.gl/WaAKTD>. Acesso em 11 de junho de 2015.

3 METODOLOGIA

3.1 Desenho de estudo

Para que o objetivo do presente trabalho, que foi verificar a viabilidade do uso de uma técnica computacional de delimitação de áreas, que utilizando os recursos oferecidos pelo sensor do Kinect, fosse capaz de delimitar a silhueta corporal de seres humanos em uma vista do plano sagital, fosse atingido, se utilizou de uma desenho de estudo de pesquisa aplicada através de experimento de desenvolvimento e testagem do comportamento e coleta de dados da ferramenta. A Figura 13 ilustra o design da metodologia do trabalho.

Figura 13. Resumo da metodologia

A definição da técnica mais adequada para o problema proposto se deu por meio de uma fundamentação teórica, adquirida através de pesquisas bibliográficas, artigos científicos, teses e livros da área de anatomia humana, computação gráfica, processamento digital de imagens e outras áreas. Em seguida, foi planejado e desenvolvido um *software* para verificar essa viabilidade. O desenvolvimento desse *software* se deu por meio de etapas, de forma que as alterações fossem efetuadas a cada etapa, sem afetar as demais. Após isso, realizou testes para verificar o comportamento da ferramenta, onde percebeu-se que haveria a necessidade de elaborar um protocolo para que os resultados fossem obtidos com maior precisão. As subseções seguintes exibem com mais detalhes todo o processo ilustrado pela Figura 13.

3.2 Local e período de realização do estudo

O trabalho foi desenvolvido no Laboratório de Tecnologia em Saúde com auxílio do Laboratório de Multimídia e Computação Gráfica, ambos localizados no Complexo de Informática do Centro Universitário Luterano de Palmas (CEULP).

3.3 Hardware

Em termos de *hardwares* foram utilizados um computador de uso pessoal, um Kinect acompanhando de um adaptador - responsável pela conexão entre Kinect e o computador - disponibilizado pelo Grupo de Estudos e Pesquisa em Tecnologia, Saúde e Qualidade de Vida (GEPETS) do CEULP.

3.4 Linguagem de programação e *softwares*

Abaixo, encontra-se a linguagem de programação e os *softwares* utilizados no desenvolvimento da aplicação que verificará a eficácia da técnica de delimitação:

- *C# (CSharp)* - é a linguagem de programação desenvolvida pela Microsoft para a plataforma .NET, sendo a principal linguagem da mesma. Possui tipagem dinâmica e estática, além de ser orientada a objetos;
- *Microsoft Windows 8.1* - atualmente é última versão do sistema operacional da Microsoft, usado em grande parte dos computadores pessoais;
- *Microsoft Kinect SDK 1.8* - é o kit de desenvolvimento da Microsoft para aplicações do Kinect;
- *Visual Studio 2013* - é a IDE oficial da Microsoft para desenvolvimento de *softwares* na plataforma .NET. Também é usada no desenvolvimento de aplicações na plataforma ASP.NET.

3.5 Desenvolvimento do *software*

A aplicação foi desenvolvida seguindo o padrão de projeto utilizado nos trabalhos do GEPETS. Para que desse início ao desenvolvimento, elaborou-se uma lógica para identificação dos atributos antropométricos na imagem obtida por meio do sensor de profundidade, ou seja, padronizou-se um conjunto de medidas, um protocolo de coleta e estrutura de apresentação das informações. Tal elaboração se fez necessário, uma vez, que ficou definido que o *software* não utilizaria nenhum outro recurso computacional para identificação da região dos atributos antropométrico na imagem mencionada. Sendo assim, o desenvolvimento da ferramenta se deu por etapas, de modo que correções e aperfeiçoamento das funcionalidades fossem realizados ainda em fase de desenvolvimento.

Na primeira etapa, foram desenvolvidas as classes básicas para o funcionamento dos componentes do Kinect. Essas classes são responsáveis em ativar e configurar o sensor de profundidade e a câmera RGB. Posteriormente, foi desenvolvido o *layout* do *software*, de acordo com o padrão utilizado pelo GEPETS.

Com as configurações básicas realizadas, a segunda etapa consistiu em implementar os métodos para delimitação do corpo humano em relação ao ambiente que ele se encontra. Para isso, foi utilizada a imagem gerada pelo sensor de profundidade do Kinect para obtenção da representação do corpo humano por meio da aplicação da técnica de limiarização. Também se implementou a lógica para identificação dos atributos antropométricos a serem calculados na terceira etapa.

Após a implementação das duas etapas anteriores, a terceira etapa teve como objetivo o desenvolvimento de uma técnica para verificação da eficiência da delimitação de áreas no corpo humano. Para esse propósito, foi desenvolvido um módulo para calcular alguns dos atributos antropométricos do ser humano, tais como: a estatura, a largura do ombro e a largura do quadril. Na obtenção do valor da mensuração do atributo antropométrico, primeiro utilizou-se como base a medida fornecida pelos *joints* através do cálculo de distância entre dois pontos - já que *joints* permitem a realização de tal recurso devido aos valores em metros das propriedades X, Y e Z. Com isso, a partir da referência para identificação dos atributos antropométricos e da representação do corpo humano desenvolvido na segunda etapa calculou-se a provável medida utilizando o cálculo da proporção.

Por fim, a quarta etapa teve como propósito a integração do *software* desenvolvido na plataforma FisioKinect. Nessa etapa foi modelado o banco de dados para que a aplicação pudesse salvar todas as informações das avaliações.

3.6 Panorama do software desenvolvido

As subseções seguintes informações sobre a arquitetura do *software* desenvolvido e a plataforma FisioKinect.

3.6.1 Arquitetura do *software*

Para compreender o funcionamento do *software* desenvolvido, logo abaixo, na Figura 14, é apresentada a arquitetura do mesmo.

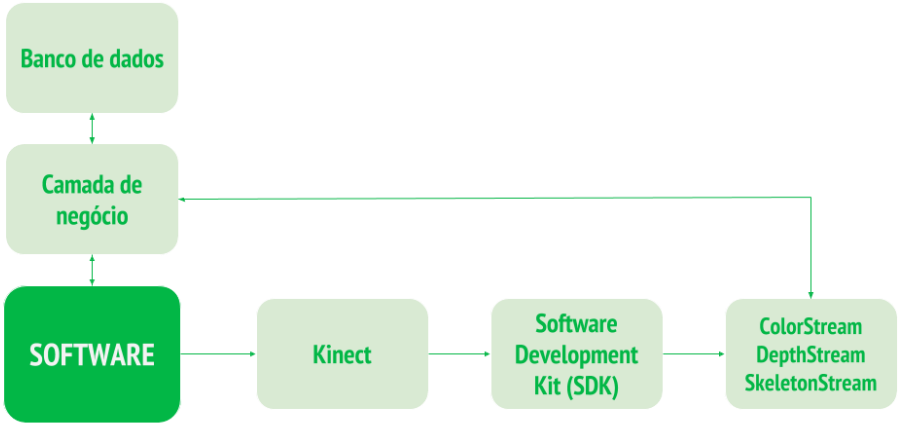

Figura 14. Arquitetura de desenvolvimento do software

O *software* funciona juntamente com as imagens fornecidas pelo Kinect. Para que isso aconteça, em seu desenvolvimento foi utilizado o SDK do Kinect. Para obter as imagens do Kinect, utilizaram-se três recursos presente no Kinect e o SDK. São eles:

- *ColorStream* - é o recurso responsável pela geração das imagens através do sensor RGB;

- *DepthStream* - é o recurso utilizado para geração das imagens do sensor de profundidade. São nessas imagens que a limiarização é aplicada;
- *SkeletonStream* - é o recurso responsável pelo mapeamento do esqueleto do corpo humano por meio dos *joints*. É partir dos *joints* específicos para cada atributo antropométrico que se obtém o valor do segmento M_b, cuja explicação será apresentada na subseção 4.2;

Os recursos apresentados são indispensáveis para funcionamento do *software*, pois eles interagem com sua camada de negócio. Essa camada é responsável por toda parte lógica da aplicação. Nela são mensurados os atributos antropométricos por meio da imagem obtida pelo sensor de profundidade do Kinect, além de ser responsável pela comunicação entre o banco de dados e a aplicação.

3.6.2 A plataforma FisioKinect

O FisioKinect é uma plataforma contendo *softwares* destinados a área da saúde, desenvolvidos pelos alunos dos cursos de Ciência da Computação e Sistemas de Informação do CEULP. O diferencial dessa plataforma é a integração entre os *softwares*, onde existe uma base de dados única para o gerenciamento dos pacientes e avaliações. Portanto, qualquer *software* que venha a ser integrado na plataforma não necessitará de um gerenciamento de pacientes, posto que, a plataforma oferece todos os recursos necessários para tal ação. A Figura 15 exibe a tela inicial da plataforma em questão.

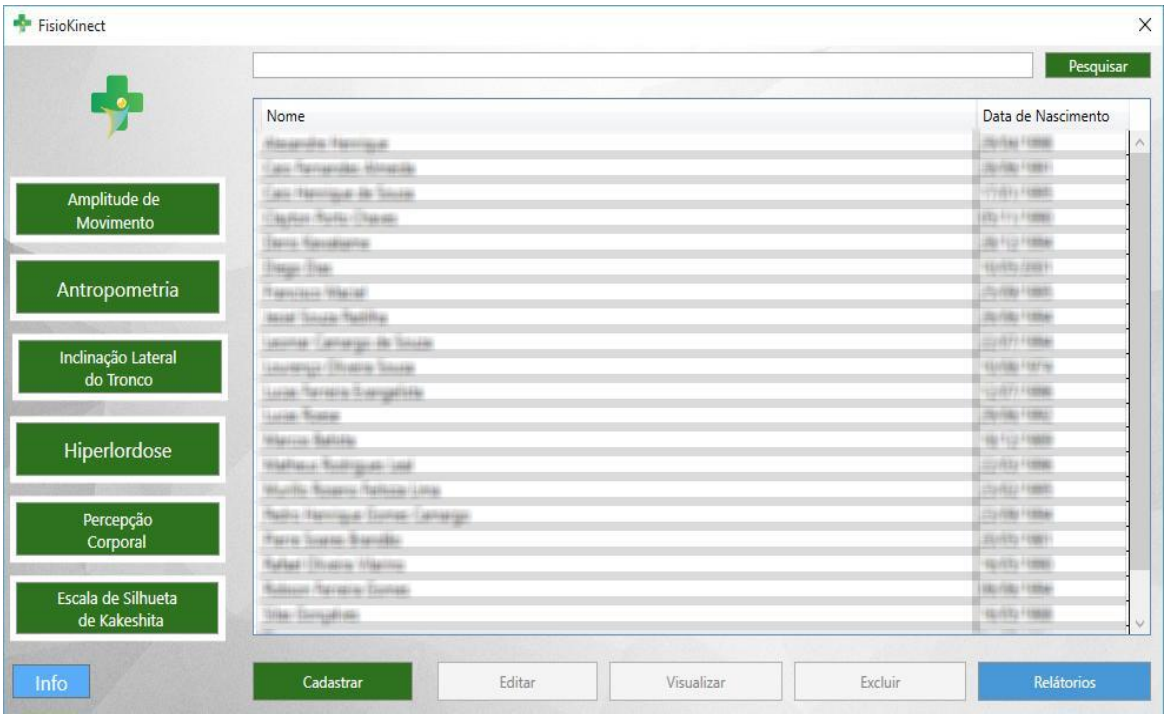

Figura 15. Tela inicial do FisioKinect

Como nota-se na figura acima, no lado esquerdo há todas as aplicações integradas na plataforma. No lado direito é exibido à lista contendo todos os pacientes cadastrados. Logo abaixo existem as opções para gerenciá-los. Para acessar o *software* desenvolvido, clica-se na opção Antropometria, que será exibido uma nova tela, após isso basta clicar em Antropometria - Silhueta Corporal.

A plataforma foi desenvolvida utilizando as linguagens C# utilizando a tecnologia *Windows Presentation Foundation* (WPF) e SQL Server. Dessa forma, todos os *softwares* que são integrados utilizam esse critério no seu desenvolvimento. Além disso, todos os *softwares* seguem o *layout* padrão utilizado pela plataforma, de modo que se tenha um único padrão de *design*.

3.7 Testes e elaboração do protocolo

Uma vez concluído o processo de desenvolvimento do *software*, foram realizados testes para verificar o desempenho das funcionalidades implementadas. Nesses testes realizou-se a coleta dos atributos antropométricos de algumas pessoas, a fim de verificar o funcionamento do *software*, de forma que qualquer anormalidade identificada em alguma funcionalidade fosse revista.

Além disso, foram constatados que alguns elementos afetavam o valor final do resultado devido a forma como o cálculo é realizado pelo *software*. Assim sendo, foi elaborado um protocolo para minimizar qualquer risco que venha ocasionar imprecisões no resultado final.

4 RESULTADOS E DISCUSSÕES

A presente seção expõe os desfechos obtidos no decorrer do desenvolvimento do presente trabalho. A princípio será apresentada uma visão geral da ferramenta desenvolvida, bem como a arquitetura e funcionamento da mesma. Posteriormente, será apresentado um protocolo para a utilização da aplicação, abrangendo o passo a passo para sua utilização. Por último, será detalhado o processo de codificação da aplicação acerca das principais funcionalidades.

4.1 O *software*

Para comprovar a eficiência da técnica computacional de delimitação de áreas, foi necessário o desenvolvimento de algumas funcionalidades que fossem capazes de mensurar certos atributos antropométricos do ser humano. Dessa forma, optou-se por mensurar três atributos: a altura, a largura dos ombros e a largura da cintura. Esses atributos satisfazem a avaliação da técnica computacional, uma vez que os mesmos têm como características a delimitação do corpo para ter um resultado preciso. A figura abaixo exibe a tela *software* desenvolvido no presente trabalho.

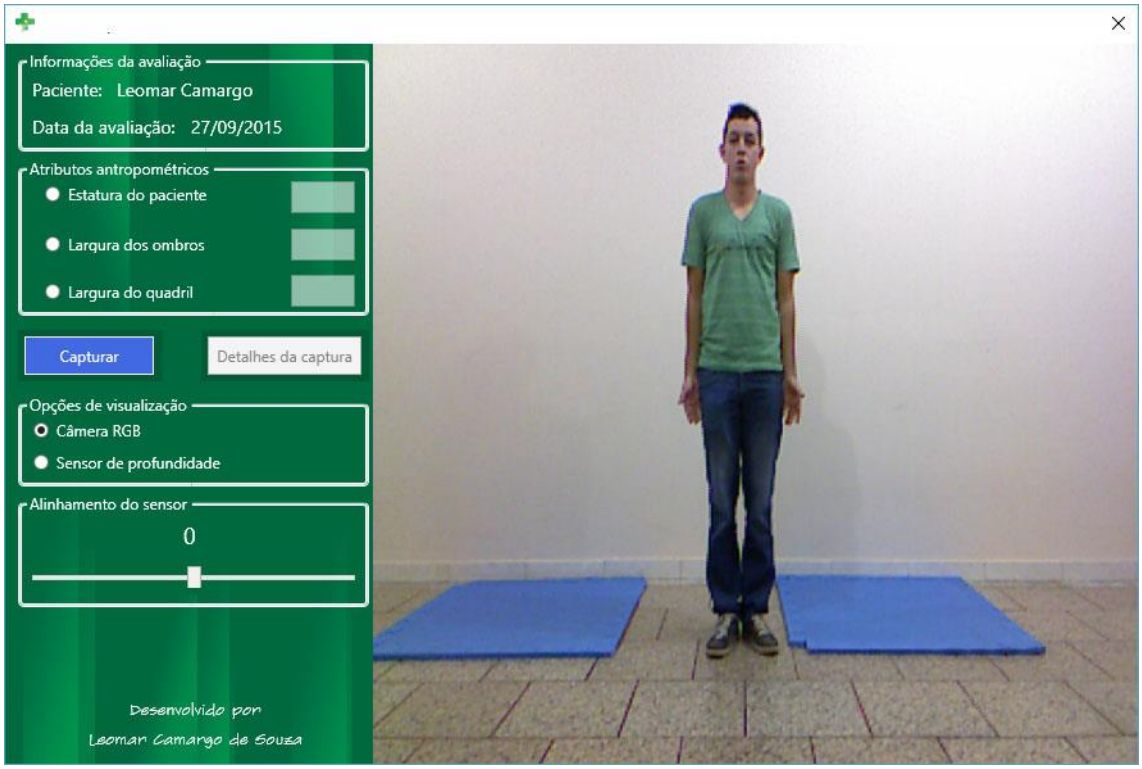

Figura 16. Tela do software desenvolvido utilizando a câmera RGB

Na parte superior esquerda, encontra-se o Box Informações para exibir o nome do paciente que está realizando a avaliação e a data em que a avaliação está sendo realizada. Logo abaixo, o Box Atributos Antropométricos exibe as opções dos atributos que o *software* realiza

a coleta. Para coletar uma informação, basta selecionar o atributo desejado e clicar no botão Capturar.

No Box Opções de visualização o usuário define se deseja visualizar as imagens fornecidas pela câmera RGB ou pelo sensor de profundidade. Nessa segunda opção, é realizado um tratamento na imagem fornecida pelo sensor de profundidade, de modo que a mesma consiga exibir uma representação do corpo humano com o máximo de detalhes possível, já que essa imagem é responsável pelo cálculo dos dados antropométricos.

É importante ressaltar que a opção Sensor profundidade deve estar selecionada para que o cálculo seja realizado, mesmo com as duas imagens - RGB e profundidade - estarem executando em paralelo. Isso se dá devido ao atributo `visibly` presente em cada `canvas` que reproduz as imagens e pelo fato do cálculo ser realizado sob a imagem do sensor de profundidade. Dessa forma, faz-se necessário que seja exibida a imagem do sensor de profundidade.

Para que o corpo seja exibido da melhor forma possível, é aplicada a técnica de limiarização na imagem emitida pelo sensor RGB em tempo real. Para isso, na exibição do corpo humano, optou-se por deixar a silhueta do mesmo com a cor preta e o background com a cor branca. Abaixo, a Figura 17 apresenta o *software* com a opção Sensor profundidade ativada.

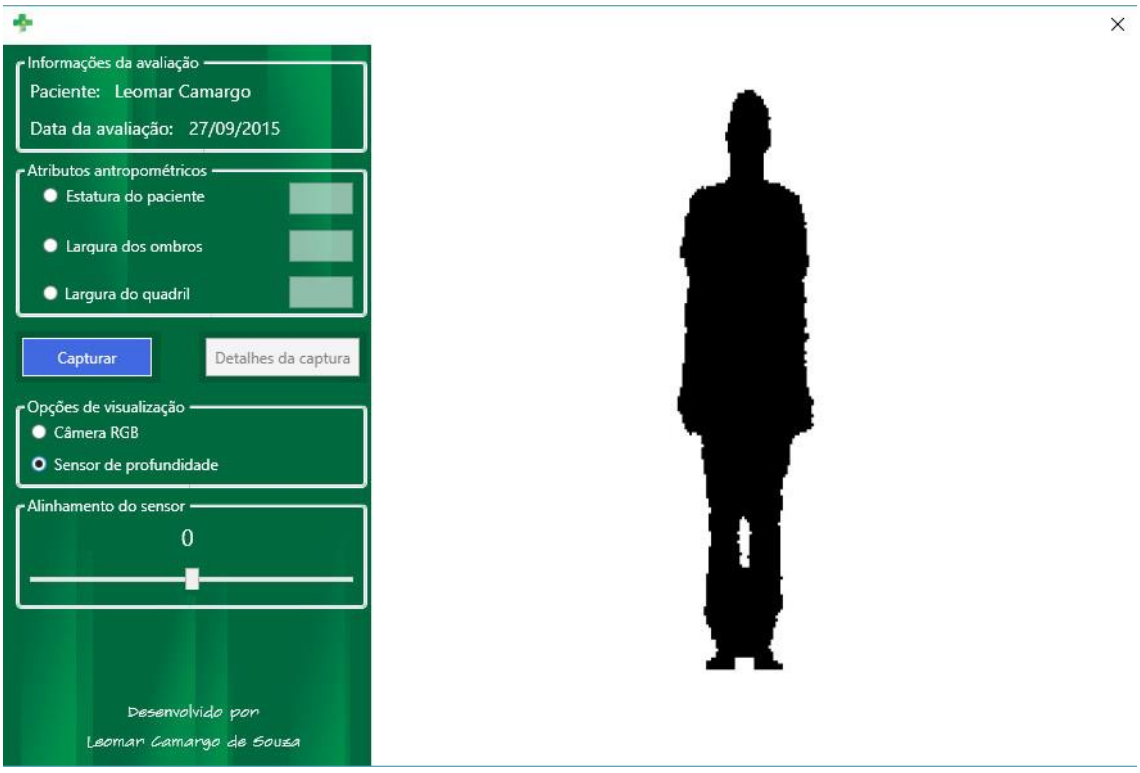

Figura 17. Software com a opção do sensor de profundidade ativo

A imagem obtida através do sensor de profundidade aliada à técnica de limiarização consegue realizar uma representação do corpo humano quase que ideal. Porém, ocorre uma pequena imprecisão por conta da borda da silhueta do corpo humano, visto que a representação em questão sofre com algumas deformações no seu redor. Entretanto, é possível afirmar que a técnica de limiarização é válida para o objetivo proposto no presente trabalho, já que a mesma consegue delimitar todo o corpo humano, fazendo com que seja possível calcular os dados antropométricos como proposto no presente trabalho.

4.2 Lógica utilizada para o cálculo dos atributos antropométricos

Para que seja possível realizar o cálculo dos atributos antropométricos, faz-se necessário o uso dos *joints* oferecidos pelo próprio sensor Kinect, pois, a partir deles, é possível obter uma determinada medida, denotada por M_b - medida base. O valor de M_b é obtido utilizando a fórmula para cálculo de distância entre dois pontos, já que cada *joint* tem como propriedade os valores em metros das coordenadas X, Y e Z.

$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$	(1)

Esse valor de M_b é utilizado para que se obtenha o possível valor do atributo antropométrico desejado, denotado por M_p, cujo significado é medida provável. A Figura 18 exibe a ideia de como é realizado o cálculo da largura do ombro. Essa mesma ideia é utilizada para os demais atributos, ocorrendo apenas uma adaptação para o contexto dos demais atributos.

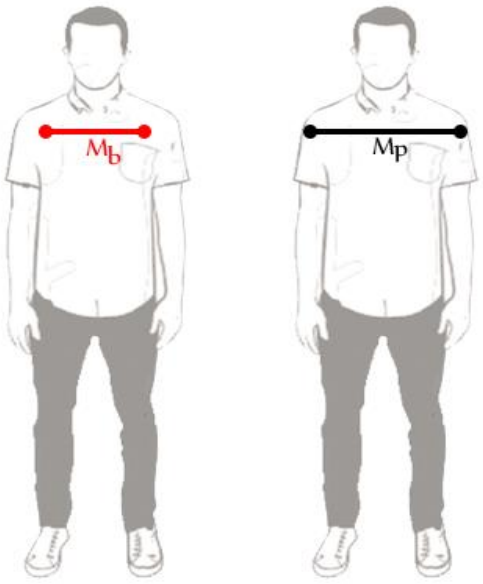

Figura 18. Visão do cálculo da largura do ombro

Na figura acima, notam-se dois segmentos de reta: M_b e M_p. No segmento M_b é possível obter a distância do mesmo, por meio dos *joints* - informação oferecida pelo Kinect - e do

cálculo de distância entre dois pontos, bem como, a quantidade de pixels presentes no segmento, já que ele utiliza como base os *joints shoulder right* e *shoulder left*. No segmento M_b também é possível notar que os *joints* não chegam à extremidade do corpo, de forma, que o valor da distância desse segmento não é um valor válido nesse contexto de obter a mensuração real dos ombros.

Sendo assim, para obter o valor do comprimento do segmento M_p calcula-se a quantidade total de pixels pretos - que representa o corpo humano, conforme explicado na subseção anterior - a partir das coordenadas de M_b referentes à linha de onde se encontra o segmento. Por fim, com a quantidade de pixels do segmento M_p e a quantidade de pixels juntamente com distância - em metros - do segmento M_b, realiza-se o cálculo de proporção para obter a provável distância de M_p.

$$\frac{distância\ de\ M_b\ \times\ total\ de\ pixels\ de\ M_p}{total\ de\ pixels\ de\ M_b} \quad (2)$$

4.2.1 *Joints* utilizados pelo segmento M_b

Como apresentado na subseção acima, a distância do segmento M_b é obtida por meio do cálculo da distância entre dois *joints* referentes ao atributo antropométrico desejado. O Quadro 3 exibe os *joints* referentes a cada atributo antropométrico calculado pelo *software* desenvolvido no presente trabalho.

Quadro 3. *Joints* de referência para os atributos antropométricos

ATRIBUTO ANTROPOMÉTRICO	*JOINTS* DE REFERÊNCIA
Estatura	*Head*, *shoulder center* e *foot right*
Largura do ombro	*Shoulder right* e *shoulder left*
Largura do quadril	*Rip right* e *rip left*

Os *joints* apresentados no quadro acima são indispensáveis para a realização do cálculo do atributo antropométrico desejado, conforme a lógica apresentada na subseção anterior. O uso de três *joints* para estatura será explicado posteriormente.

4.3 Processo de localização dos segmentos M_b e M_p e cálculo da proporção

A seguir, a Figura 19 ilustra as etapas do processo utilizado para localizar os segmentos M_b e M_p, e obtenção do valor final.

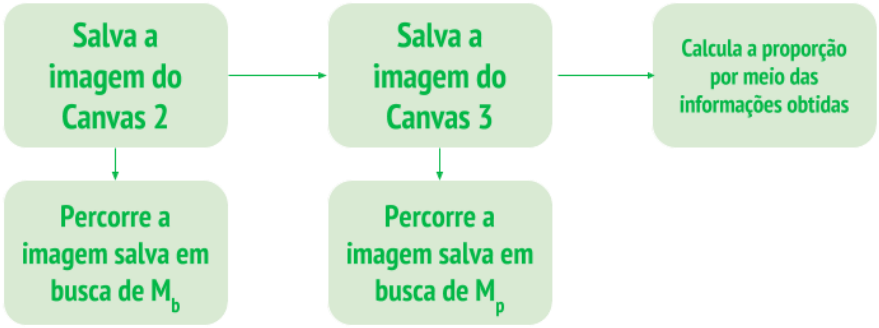

Figura 19. Etapas para localização dos segmentos e cálculo da proporção

O componente `canvas` - presente na tecnologia WPF - é responsável por exibir as imagens da câmera RGB e o sensor profundidade ao usuário. Para realizar o processo de captura, existem três `canvas` executando em paralelo, onde a imagem de dois deles são capturadas e salvas na pasta temporária do sistema operacional. Essas imagens salvas são utilizadas pelo *software* para localizar os segmentos M_b e M_p. A Figura 20 exibe um exemplo das imagens de cada `canvas`.

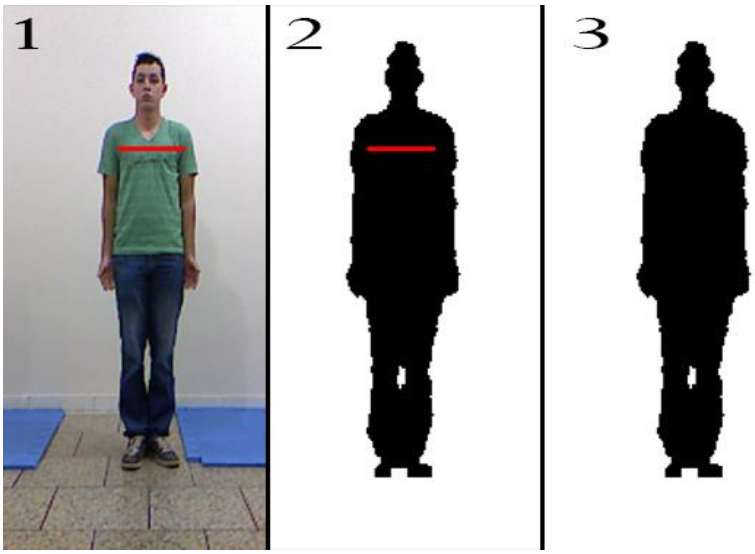

Figura 20. Imagens obtidas pelos três canvas presente no software

A primeira imagem é obtida a partir do sensor do RGB do Kinect, sendo apenas para visualização do campo de visão do Kinect. As outras duas imagens são obtidas a partir da aplicação da limiarização na imagem do sensor de profundidade. Contudo, a segunda imagem contém um segmento - M_b - em vermelho, que é uma forma de referenciar os *joints* usados como base em cada atributo antropométrico. Tal segmento faz-se necessário para que o *software* saiba onde localizar o atributo antropométrico na imagem capturada, uma vez que não se utiliza nenhuma outra tecnologia para realizar tal ação. A referência de cada atributo antropométrico é desenhada no `canvas` por meio dos métodos desenvolvidos no trabalho de Chaves (2014).

Dessa forma, o *software* trabalha em cima da segunda e terceira imagem. A segunda imagem é percorrida por um método a fim de localizar as coordenadas dos pixels que fazem parte do segmento M_b. Sem a execução dessa etapa, se torna impossível realizar as etapas seguintes.

Após encontrar as coordenadas de M_b, é executado o método que percorre a terceira imagem a partir da coordenada referente a linha em que se encontra o segmento M_b. Nesses dois métodos entra a caracterização da delimitação do corpo humano, onde os pixels na cor branca representam o *background* e os pixels na cor preta representam o corpo humano. Após localizar ambos os segmentos, executa-se o método que calcula a provável medida do atributo desejado. Os códigos-fonte dos métodos mencionados nessa subseção serão apresentados nas subseções seguintes.

4.3.1 Orientação dos segmentos

O método para contagem dos pixels que fazem partes dos segmentos M_b e M_p sofre uma alteração dependendo do atributo antropométrico a ser calculado já que, em certos casos, os segmentos se tornam horizontais - largura do ombro e do quadril - ou verticais - estatura. A Figura 21 apresenta a orientação do segmento de acordo com o atributo antropométrico.

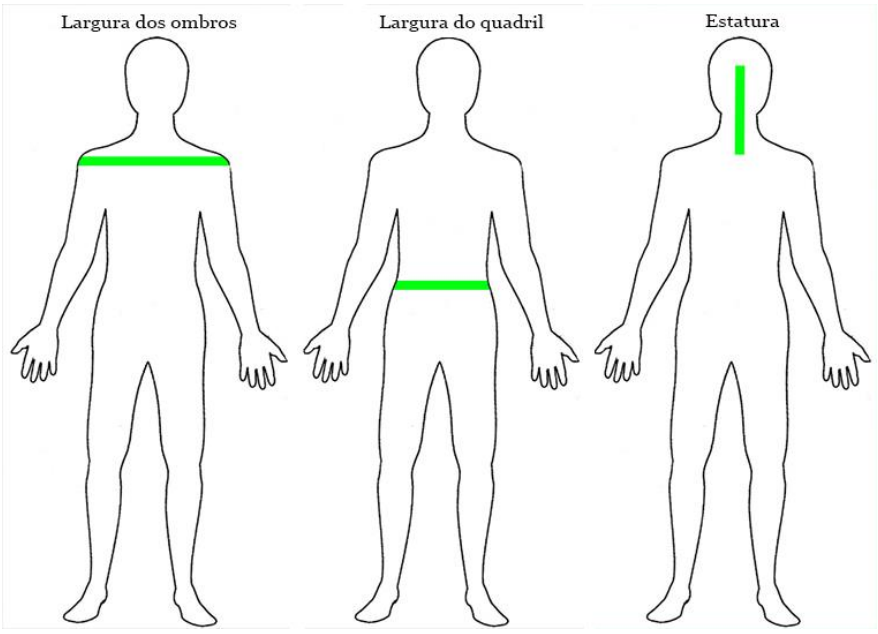

Figura 21. Orientação dos segmentos

No caso da largura do ombro e quadril, são utilizados os mesmos métodos de localização dos segmentos para obter o valor da mensuração final. O mesmo não ocorre com a estatura, de forma que esse atributo antropométrico exigiu a implementação de novos métodos para se realizar a mensuração.

4.4 Detalhe da captura

Nas Figuras 16 e 17, nota-se o botão Detalhes da captura. Esse botão é habilitado após o término da captura do atributo antropométrico desejado. A Figura 22 exibe a tela apresentada após o usuário clicar no botão em questão.

Figura 22. Tela de detalhes da captura

Na figura acima, notam-se as informações básicas de cada segmento - M_b e M_p - de acordo com a explicação apresentada na subseção 4.2. Essas informações são obtidas a partir da execução dos métodos mencionados na subseção anterior. As informações contidas nessa tela têm como objetivo apresentar as variáveis resultantes do cálculo do tamanho do segmento M_p.

4.5 Desenvolvimento do *software*

As subseções a seguir explicam todo o processo de desenvolvimento da ferramenta, desde as configurações básicas para funcionamento do Kinect até os métodos para cálculo dos atributos antropométricos.

4.5.1 Arquitetura de desenvolvimento

Foi adotado o padrão de desenvolvimento em três camadas, o mesmo padrão utilizado na plataforma FisioKinect, conforme apresentado na subseção 3.6.2. A primeira camada - exibição - consiste na apresentação das informações da aplicação ao usuário. A próxima camada é denominada regra de negócio, sendo responsável pelo armazenamento de toda lógica de negócio da aplicação. A terceira, e última camada, denominada camada de dados, tem como responsabilidade a comunicação entre o *software* e o banco de dados.

4.5.2 Declaração de variáveis

A Figura 23 exibe as declarações iniciais de variáveis de classe utilizadas no desenvolvimento do *software*.

```
18          private bool _imagemRgbHabilitado;
19          private bool _imagemSensorProfundidadeHabilitado;
20          private double _distancia;
21          private double _alturaPeloKinect;
22          private readonly DetalhesCaptura _detalhes;
23          private KinectSensor _sensorKinect;
24          private WriteableBitmap _imagemProfundidadeBruta;
25          private Int32Rect _imagemProfundidadeBrutaRect;
26          private short[] _dadosPixelProfundidadeBruta;
27          private int _imagemProfundidadePassadaBruta;
28          private WriteableBitmap _imagemProfundidadeTratada;
29          private Int32Rect _imagemProfundidadeTratadaRect;
30          private int _imagemProfundidadePassadaTratada;
```

Figura 23. Declaração inicial das variáveis

As duas primeiras variáveis apresentadas nas linhas 18 e 19 são variáveis do tipo `bool`, usadas para o controle da exibição dos `canvas` referente à imagem da câmera RGB e a imagem do sensor de profundidade. A variável na linha 20 é usada para armazenamento da distância entre os *joints* - de acordo com o que foi apresentado na subseção 4.2. Na linha 21, a variável é utilizada para obter a estatura da pessoa por meio dos *joints head* e *foot right*. Essa altura é utilizada quando a opção quando o usuário seleciona a opção da estatura no *software*. Na linha 22, é apresentada a variável `_detalhes`, que é responsável por exibir as informações na tela de detalhes da captura - conforme explicado na subseção 4.4.

Na linha 23, a variável `_sensorKinect` - do tipo `KinectSensor` proveniente do próprio SDK do Kinect - é responsável pelos métodos e variáveis utilizadas pelo Kinect. As variáveis das linhas 24 a 30 são utilizadas para aplicação da limiarização na imagem do sensor de profundidade. A subseção a seguir apresenta outra variável, responsável pela inicialização e configuração do Kinect, bem como, a configuração da técnica de limiarização na imagem obtida a partir do sensor de profundidade.

4.5.2.1 Inicialização e configuração do Kinect e da limiarização

Para inicialização e configuração do sensor Kinect, declarou-se a variável `SensorKinect` do tipo `KinectSensor`. A Figura 24 exibe o código com os passos de inicialização e configuração do Kinect e da limiarização.

```
32  public KinectSensor SensorKinect
33  {
34      get { return _sensorKinect; }
35      set
36      {
37          if (_sensorKinect != value)
38          {
39              if (_sensorKinect != null)
40              {
41                  _sensorKinect.Stop();
42                  _sensorKinect.DepthFrameReady -= SensorKinectDepthFrameReady;
43                  _sensorKinect.DepthStream.Disable();
44                  _sensorKinect.SkeletonStream.Disable();
45                  ImagemSensorProfundidadeComSegmento.Children.Clear();
46                  ImagemSensorProfundidadeSemSegmento.Children.Clear();
47              }
48              _sensorKinect = value;
49
50              if (_sensorKinect != null)
51              {
52                  if (_sensorKinect.Status == KinectStatus.Connected)
53                  {
54                      _sensorKinect.SkeletonStream.Enable();
55                      _sensorKinect.DepthStream.Enable();
56
57                      var fluxoProfundidade = _sensorKinect.DepthStream;
58                      _imagemProfundidadeBruta = new WriteableBitmap(fluxoProfundidade.FrameWidth,
59                          fluxoProfundidade.FrameHeight, 96, 96, PixelFormats.Gray16, null);
60                      _imagemProfundidadeBrutaRect = new Int32Rect(0, 0, (int)Math.Ceiling(_imagemProfundidadeBruta.Width),
61                          (int)Math.Ceiling(_imagemProfundidadeBruta.Height));
62                      _imagemProfundidadePassadaBruta = fluxoProfundidade.FrameWidth * fluxoProfundidade.FrameBytesPerPixel;
63                      _dadosPixelProfundidadeBruta = new short[fluxoProfundidade.FramePixelDataLength];
64
65                      _imagemProfundidadeTratada = new WriteableBitmap(fluxoProfundidade.FrameWidth,
66                          fluxoProfundidade.FrameHeight, 96, 96, PixelFormats.Bgr32, null);
67                      _imagemProfundidadeTratadaRect = new Int32Rect(0, 0, (int)Math.Ceiling(_imagemProfundidadeTratada.Width),
68                          (int)Math.Ceiling(_imagemProfundidadeTratada.Height));
69                      _imagemProfundidadePassadaTratada = fluxoProfundidade.FrameWidth * 4;
70
71                      ImagemSensorProfundidadeComSegmento.Background = new ImageBrush(_imagemProfundidadeTratada);
72                      ImagemSensorProfundidadeSemSegmento.Background = new ImageBrush(_imagemProfundidadeTratada);
73
74                      _sensorKinect.DepthFrameReady += SensorKinectDepthFrameReady;
75                      _sensorKinect.ColorStream.Enable();
76                      _sensorKinect.AllFramesReady += SensorKinectSensorOnAllFramesReady;
77                      _sensorKinect.Start();
78                  }
79              }
80          }
81      }
82  }
```

Figura 24. Variável SensorKinect

Na linha 34, a propriedade retorna a variável `_sensorKinect`, declarada na linha 23 da Figura 23. A partir da linha 35, por meio da definição `set`, são realizadas as configurações básicas do Kinect, além de trabalhar com a imagem do sensor de profundidade. Na linha 39 é feita a verificação do conteúdo da variável `_sensorKinect`, de forma que se ele for diferente de `null`, todas as configurações realizadas para inicialização do Kinect, limiarização e exibição de imagens serão encerradas. É importante notar o código das variáveis presentes nas linhas 45 e 46, pois essas variáveis são responsáveis pela exibição das imagens do sensor de profundidade após a aplicação da técnica de limiarização, por meio do componente `canvas` - de acordo com a explicação presente na subseção 4.3. Dessa forma, o que acontece nesse trecho é apenas a limpeza de `canvas`.

A verificação se o Kinect está conectado ao computador ocorre na linha 52. O trecho de código presente nas linhas 54 e 55 habilita os quadros do esqueleto - `SkeletonStream` - e o sensor de profundidade - `DepthStream`. A variável `fluxoProfundidade` da linha 57

recebe os dados da imagem resultante do sensor de profundidade do Kinect. Com isso, o trecho de código entre as linhas 58 e 69 realiza o tratamento para aplicação da limiarização. Vale destacar que as variáveis utilizadas nesse processo foram instanciadas anteriormente, como mostrou a Figura 23. O Quadro 4, a seguir, apresenta comentários a respeito desse trecho de código.

Quadro 4. Comentários sobre a configuração da limiarização

LINHA(S)	COMENTÁRIO
58 e 59	É criado uma nova imagem do tipo WriteableBitmap com a mesma dimensão da imagem original. Essa nova imagem possui tonalidade cinza
60 e 61	Cria uma espécie de retângulo a partir da imagem resultante do trecho de código das duas linhas anteriores
62 e 63	Calcula e codifica em *bytes* a imagem resultante do sensor de profundidade, de acordo com a linha 57
65 e 66	Gera uma nova imagem, sendo esse um processo parecido com o apresentado nas linhas 58 e 59. A única diferença é o formato de pixels, agora, no padrão RGB
67 a 68	Cria uma espécie de retângulo através da imagem resultante no trecho de código das linhas 65 e 66. Variáveis desse tipo são utilizadas para escrever dados em um Bitmap
69	Obtém o quadro de pixels multiplicado pelo número de bytes por pixel

Nas linhas 71 e 72, as variáveis dos `canvas` dois e três recebem a imagem com a representação da silhueta do corpo humano gerada pelo trecho de código das linhas 65 e 66. Quase todas as variáveis presentes no trecho de código entre as linhas 58 e 69 são utilizadas no evento `DepthFrameRead` - Figura 25 - habilitado na linha 74.

```
296  private void SensorKinectDepthFrameReady(object sender, DepthImageFrameReadyEventArgs e)
297  {
298      try
299      {
300          using (var frame = e.OpenDepthImageFrame())
301          {
302              if (frame != null)
303              {
304                  frame.CopyPixelDataTo(_dadosPixelProfundidadeBruta);
305                  _imagemProfundidadeBruta.WritePixels(_imagemProfundidadeBrutaRect,
306                      _dadosPixelProfundidadeBruta, _imagemProfundidadePassadaBruta, 0);
307                  GerarImagemProfundidade(frame, _dadosPixelProfundidadeBruta);
308              }
309          }
310      }
311      catch (Exception ex)
312      {
313          MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK, MessageBoxImage.Error);
314      }
315  }
```

Figura 25. Código do evento DepthFrameReady

Na linha 300 a variável frame recebe os quadros obtidos pelo sensor de profundidade. Essa variável é do tipo `DepthImageFrame`, pertencente ao próprio SDK do Kinect. Na linha 304, é chamado o método `CopyPixelDataTo()` que recebe os bytes obtidos pela variável `_dadosPixelProfundidadeBruta`. Posteriormente, na linha seguinte a variável

`_imagemProfundidadeBruta` atualiza os seus pixels por meio dos valores de algumas das variáveis presentes na Figura 23. Logo abaixo, o método `GerarImagemProfundidade()` é invocado, cujos parâmetros são as variáveis `frame` e `_dadosPixelProfundidadeBruta`. A Figura 26 exibe o código desse método.

```
316  private void GerarImagemProfundidade(ImageFrame frame, IList<short> pixels)
317  {
318      try
319      {
320          const int byteProfundidadePorPixel = 4;
321          var dadosPixels = new byte[frame.Width * frame.Height * byteProfundidadePorPixel];
322          for (int i = 0, j = 0; i < pixels.Count; i++, j += byteProfundidadePorPixel)
323          {
324              var playerIndex = pixels[i] & DepthImageFrame.PlayerIndexBitmask;
325              if (playerIndex == 0)
326              {
327                  dadosPixels[j] = 0xFF;
328                  dadosPixels[j + 1] = 0xFF;
329                  dadosPixels[j + 2] = 0xFF;
330              }
331              else
332              {
333                  dadosPixels[j] = 0x00;
334                  dadosPixels[j + 1] = 0x00;
335                  dadosPixels[j + 2] = 0x00;
336              }
337          }
338          _imagemProfundidadeTratada.WritePixels(_imagemProfundidadeTratadaRect, dadosPixels, _imagemProfundidadePassadaTratada, 0);
339      }
340      catch (Exception ex)
341      {
342          throw new Exception("Não foi possível renderizar a imagem do sensor de profundidade!", ex);
343      }
344  }
```

Figura 26. Método para geração da imagem do sensor de profundidade

O método acima é responsável por tratar a imagem obtida pelo frame do sensor de profundidade. Esse método é executado em tempo real. Dessa forma, ele realiza a separação da silhueta do corpo humano do seu *background*. A variável da linha 324 recebe os bytes da representação da representação do corpo humano através do frame do sensor de profundidade que foi recebido como parâmetro. A linha seguinte verifica se na região do byte percorrido existe a presença de uma pessoa. Se não existir, o background dessa região receberá a cor branca. Caso contrário, o byte ficará na cor preta - representando assim a presença de pessoa no frame. Com isso, conclui-se o processo de limiarização utilizado para obter a representação do ser humano por meio da imagem do sensor de profundidade do Kinect.

4.5.3 Mensuração dos atributos antropométricos

Quando o usuário seleciona o atributo desejado para realização do cálculo, é executado um método de acordo com a orientação dos segmentos - horizontal ou vertical - para que o cálculo seja realizado. A Figura 27 apresenta o código do processo realizado para o cálculo dos atributos cujos segmentos são horizontais.

```
386 private double CapturaSegmentosHorizontais()
387 {
388     double distancia = 0;
389     var nomeImagemSegmentoMb = ArquivosManager.Criptografar(DateTime.Now.ToString(CultureInfo.InvariantCulture));
390     var urlImagemSegmentoMb = ProcessamentoImagem.GerarBitmap(ImagemSensorProfundidadeComSegmento,
391         this, nomeImagemSegmentoMb);
392     var coordenadasMb = ProcessamentoImagem.LocalizarSegmentoMb(urlImagemSegmentoMb);
393     var maiorSegmentoPixel = ProcessamentoImagem.ProcurarMaiorSegmentoPixelsMb(coordenadasMb);
394     var totalPixelsMb = maiorSegmentoPixel[1][1] - maiorSegmentoPixel[0][1];
395     if (coordenadasMb != null && coordenadasMb.Count > 0)
396     {
397         ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Hidden;
398         ImagemSensorProfundidadeSemSegmento.Visibility = Visibility.Visible;
399         Thread.Sleep(1000);
400         var nomeImagemSegmentoMp = ArquivosManager.Criptografar(DateTime.Now.ToString(CultureInfo.InvariantCulture));
401         var urlImagemSegmentoMp = ProcessamentoImagem.GerarBitmap(ImagemSensorProfundidadeSemSegmento,
402             this, nomeImagemSegmentoMp);
403         var linha = maiorSegmentoPixel.First()[0];
404         var coordenadasMp = ProcessamentoImagem.LocalizarSegmentoHorizontalMp(urlImagemSegmentoMp, linha);
405         distancia = KinectMath.Proporcao(totalPixelsMb, coordenadasMp.Count, _distancia);
406         PreencherDetalhes(totalPixelsMb, coordenadasMp.Count, _distancia, maiorSegmentoPixel[0],
407             maiorSegmentoPixel[1], coordenadasMp.First(), coordenadasMp.Last());
408         ArquivosManager.Excluir(urlImagemSegmentoMp);
409     }
410     ArquivosManager.Excluir(urlImagemSegmentoMb);
411     ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Visible;
412     ImagemSensorProfundidadeSemSegmento.Visibility = Visibility.Hidden;
413     return distancia;
414 }
```

Figura 27. Método para captura dos atributos cujos segmentos são horizontais

Primeiramente, declarou-se a variável `distancia` que recebe ao final do processo o valor da mensuração do atributo selecionado. Como explicado na subseção 4.3, o *software* salva as imagens exibidas pelos `canvas` dois e três. Para isso, na linha 389, a variável recebe um nome criptografado a partir da data e hora em que o processo de captura foi iniciado. Dessa forma, não existe problema em sobrescrever uma imagem já existente na pasta temporária do sistema operacional.

O método `GerarBitmap` da classe `ProcessamentoImagem` é responsável por capturar e salvar a imagem do `canvas` dois. Dessa forma, ele recebe como parâmetros o objeto do `canvas` dois, uma instância da tela em execução e o nome da imagem a ser salva na pasta temporária. Com isso, após concluir a execução, o método retorna a localização completa de onde a imagem ficou salva no sistema operacional. Com a primeira imagem salva, a próxima etapa consiste em localizar o segmento M_b. Para isso, na linha 392 executa-se o método responsável pela localização, passando como parâmetro a URL (*Uniform Resource Locator*) da imagem salva. A Figura 28 apresenta o código do método em questão.

```
15  public static Dictionary<int, List<int>> LocalizarSegmentoMb(string url)
16  {
17      try
18      {
19          var coordenadasSegmento = new Dictionary<int, List<int>>();
20          var valoresY = new List<int>();
21          var imagem = new Bitmap(url);
22          for (var x = 0; x < imagem.Height; x++)
23          {
24              for (var y = 0; y < imagem.Width; y++)
25              {
26                  if (imagem.GetPixel(y, x).R > 0 && imagem.GetPixel(y, x).G == 0 && imagem.GetPixel(y, x).B == 0)
27                  {
28                      valoresY.Add(y);
29                  }
30              }
31              if (valoresY.Count > 0)
32              {
33                  coordenadasSegmento.Add(x, new List<int>(valoresY));
34                  valoresY.Clear();
35              }
36          }
37          return coordenadasSegmento;
38      }
39      catch (Exception ex)
40      {
41          throw new Exception("Ocorreu um erro ao tentar localizar o SEGMENTO Mb na imagem!", ex);
42      }
43  }
```

Figura 28. Método para localização do segmento M_b

Na linha 19, instancia-se a variável `coordenadasSegmento` do tipo `Dictionary` tendo como chave um inteiro, que representa a linha do segmento, além de uma lista - instanciada na linha 20 - de inteiros, que representa cada coluna que se localizou o segmento desejado. Em seguida, o método abre a imagem salva pelo `canvas` dois para iniciar o processo de localização.

O conceito para percorrer as coordenadas de uma matriz equivale ao conceito de coordenadas da imagem. Logo, para localizar o segmento M_b criaram-se os laços de repetição para percorrer os pixels da imagem. O primeiro laço equivale à linha e o segundo, a coluna. A linha 26 compara se o pixel em que se está percorrendo possui tonalidade vermelha. Caso possua, adiciona-se na variável `valoresY` o valor da coluna do pixel que satisfez o condicional. Na linha 31 é verificado se existe algum item na variável `valoresY`, de forma que se existe, indica que foi localizado pelo menos uma coordenada do segmento M_b. Sendo assim, salvam-se as coordenadas encontradas na variável `coordenadasSegmento`, cujos valores são retornados na linha 37.

Após finalizar a execução do método da Figura 28, continua-se a execução do método da Figura 27. O processo do método `ProcurarMaiorSegmentoPixelsMb()` da linha 393 será explicado na subseção seguinte. A priori, esse método retorna a primeira e última coordenada do segmento M_b, possibilitando, assim, que seja calculada a quantidade de pixels encontrados.

O próximo passo consiste em verificar se foi localizado algum pixel correspondente à M_b. Caso o condicional seja positivo, desabilita-se o `canvas` dois e habilita o `canvas` três para que a sua imagem seja salva na pasta temporária do sistema operacional. A linha 399 é necessária para que processo de alternação do `canvas` no *software* seja totalmente concluído, de forma que a imagem exibida no `canvas` três possa ser salva sem que haja alguma avaria. Em seguida, obtém-se o nome da imagem de M_p e salva a imagem exibida pelo `canvas` três. A variável da linha 403 recebe o valor da linha correspondente ao segmento M_b já localizado anteriormente. Sendo assim, o próximo passo consiste em localizar a coordenada M_p, cujo método é apresentado na Figura 29.

```
46  public static List<int[]> LocalizarSegmentoHorizontalMp(string url, int? valorInicialX)
47  {
48      try
49      {
50          var imagem = new Bitmap(url);
51          var coordenadasSegmento = new List<int[]>();
52          if (valorInicialX == null) return coordenadasSegmento;
53          for (var y = 0; y < imagem.Width; y++)
54          {
55              if (imagem.GetPixel(y, valorInicialX.Value).R == 0 && imagem.GetPixel(y, valorInicialX.Value).G == 0
56                  && imagem.GetPixel(y, valorInicialX.Value).B == 0)
57              {
58                  coordenadasSegmento.Add(new[]
59                  {
60                      valorInicialX.Value, y
61                  });
62              }
63          }
64          return coordenadasSegmento;
65      }
66      catch (Exception ex)
67      {
68          throw new Exception("Ocorreu um erro ao tentar localizar o SEGMENTO HORIZONTAL Mp na imagem!", ex);
69      }
70  }
```

Figura 29. Método para localização do segmento M_p com orientação horizontal

O método acima consiste em percorrer a imagem do `canvas` três a partir da linha em que se localizou o segmento M_b, já que percorrer toda imagem implicaria um gasto desnecessário de processamento. Dessa forma, a cada pixel preto encontrado, significa que ele faz parte do segmento M_p, então, salvam-se as respectivas coordenadas.

Após localizar os dois segmentos, executa-se o método para o cálculo da proporção, passando como parâmetro o total de pixels encontrados nos dois segmentos, além da distância entre os *joints* referentes ao segmento a ser calculado. A figura abaixo exibe o método da proporção, bem como o método do cálculo de distância entre dois pontos - esse, conforme já explicado, é usado para calcular a distância de M_b.

```
14  public static double DistanciaEntreDoisPontos(double xa, double ya, double xb, double yb)
15  {
16      return Math.Sqrt(Math.Pow((xb - xa), 2) + Math.Pow((yb - ya), 2));
17  }
18
19  public static double Proporcao(int totalMb, int totalMp, double distanciaMb)
20  {
21      return Math.Round(distanciaMb * totalMp / totalMb, 3);
22  }
```

Figura 30. Métodos para cálculo da distância entre dois pontos e cálculo da proporção

É importante lembrar que os valores de cada ponto - no método da distância entre dois pontos - são obtidos pelas propriedades X e Y dos dois *joints* correspondentes ao atributo calculado. Além disso, vale ressaltar que essa distância é calculada no instante em que o usuário seleciona o atributo antropométrico desejado.

Após a localização dos segmentos e cálculo da provável mensuração dos atributos antropométricos, é chamado o método `PreencherDetalhes()` que consiste em atribuir os valores para a variável da linha 22 apresentado na Figura 23, cujas informações são exibidas na tela de detalhes apresentada na subseção 4.4. Com isso, removem-se as imagens que foram salvas na pasta temporária do sistema operacional, além de alternar a exibição novamente para o `canvas` dois. Concluído todo o processo, é retornado o valor do possível atributo antropométrico.

4.5.3.1 Método de localização do maior segmento M_b

Na fase de desenvolvimento dos métodos para a realização do cálculo dos atributos, encontrou-se um problema na localização do segmento M_b. Esse problema deu-se devido a cor e a quantidade dos pixels presentes no segmento. A Figura 31 a seguir apresenta a região dos pixels do segmento M_b de forma aumentada, sendo possível visualizar cada pixel que faz parte do segmento.

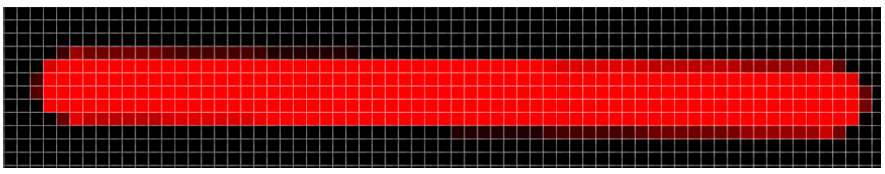

Figura 31. Visualização do segmento M_b pixel a pixel

Na figura é possível notar que em certas linhas da imagem os segmentos não são contínuos, bem como nota-se a existência de alguns pixels com tonalidade vermelha inferior ao valor real da cor vermelha. Além disso, por existir mais de uma linha seria totalmente inviável desconsiderar as outras e afirmar que somente a primeira linha faz parte de M_b, já que isso causaria uma imprecisão no resultado final. Essa afirmação pode ser constada olhando atentamente a Figura 31, pois nota-se que a quantidade de pixels da primeira linha - que contém

tonalidade vermelha - é inferior a mesma quantidade de pixels das demais linhas. Assim, houve a necessidade de implementar um método para obter o maior segmento através das coordenadas encontradas pelo método de localização de M_b. A Figura 32 expõe esse método.

```
154 public static List<int[]> ProcurarMaiorSegmentoPixelsMb(Dictionary<int, List<int>> coordenadas)
155 {
156     int[] maiorValor = { int.MinValue };
157     var x = 0;
158     var listaMaior = new List<int>();
159     foreach (var valor in coordenadas.Where(valor => valor.Value.Count > maiorValor[0]))
160     {
161         maiorValor[0] = valor.Value.Count;
162         x = valor.Key;
163         listaMaior = valor.Value;
164     }
165     return new List<int[]>
166     {
167         new[] {x, listaMaior.Min()},
168         new[] {x, listaMaior.Max()}
169     };
170 }
```

Figura 32. Método para localização do maior segmento M_b

O método acima consiste em procurar em qual índice da variável `coordenadas` - do tipo `Dictionary` - encontra-se a maior quantidade de valores, equivalendo assim ao maior segmento nas coordenadas localizadas. Ao final, retorna-se uma lista de vetores que representam as coordenadas de M_b. O primeiro vetor contém a coordenada inicial e o segundo vetor a coordenada final.

4.5.3.2 Localização do segmento M_p para a estatura

Na Figura 21, é possível notar que o segmento M_b referente a estatura não chega a ser traçado no corpo inteiro, se comparado com os segmentos dos outros atributos. Isso se dá devido ao processo adotado para obter o valor desse atributo antropométrico. A diferença desse procedimento em relação aos demais que foram desenvolvidos é o tamanho do segmento M_b, pois a quantidade de pixels desse segmento é obtida entre os *joints head e shoulder center*, bem como o seu tamanho. Dessa forma, não há a necessidade de percorrer toda a imagem a fim de localizar quais coordenadas fazem parte de M_b. Outra diferença está no segmento M_p, cuja localização tem como condição de parada a linha pertencente a primeira coordenada de M_b. A seguir, a Figura 33 apresenta o código para localização dos segmentos referentes à estatura da pessoa.

```
416 private double CapturaSegmentosVerticais()
417 {
418     double distancia = 0;
419     var nomeImagemSegmentoMb = ArquivosManager.Criptografar(DateTime.Now.ToString(CultureInfo.InvariantCulture));
420     var urlImagemSegmentoMb = ProcessamentoImagem.GerarBitmap(ImagemSensorProfundidadeComSegmento, this, nomeImagemSegmentoMb);
421     var coordenadasMb = ProcessamentoImagem.LocalizarSegmentoMb(urlImagemSegmentoMb);
422     var tamanhoMb = coordenadasMb.Last().Key - coordenadasMb.First().Key;
423     if (coordenadasMb != null && coordenadasMb.Count > 0)
424     {
425         ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Hidden;
426         ImagemSensorProfundidadeSemSegmento.Visibility = Visibility.Visible;
427         Thread.Sleep(1000);
428         var nomeImagemSegmentoMp = ArquivosManager.Criptografar(DateTime.Now.ToString(CultureInfo.InvariantCulture));
429         var urlImagemSegmentoMp = ProcessamentoImagem.GerarBitmap(ImagemSensorProfundidadeSemSegmento, this, nomeImagemSegmentoMp);
430         var linha = coordenadasMb.First().Key;
431         var coluna = coordenadasMb.First().Value[0];
432         var coordenadasMp = ProcessamentoImagem.LocalizarSegmentoVerticalMp(urlImagemSegmentoMp, coluna, linha);
433         distancia = KinectMath.Proporcao(tamanhoMb, coordenadasMp.Count, _distancia);
434         PreencherDetalhes(tamanhoMb, coordenadasMp.Count, _distancia, new[] { coordenadasMb.First().Key,
435             coordenadasMb.First().Value[0] }, new[] { coordenadasMb.Last().Key, coordenadasMb.Last().Value[0] },
436             new[] { coordenadasMp.First()[1], coordenadasMp.First()[0] }, new[] { coordenadasMp.First()[1], coordenadasMp.First()[0] });
437         ArquivosManager.Excluir(urlImagemSegmentoMp);
438     }
439     ArquivosManager.Excluir(urlImagemSegmentoMb);
440     ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Visible;
441     ImagemSensorProfundidadeSemSegmento.Visibility = Visibility.Hidden;
442     return distancia;
443 }
```

Figura 33. Método para captura dos atributos cujos segmentos são verticais

O código acima é semelhante com o código apresentado na Figura 27. Difere-se apenas em alguns métodos e variáveis, o que não altera a lógica apresentada na subseção 4.2. Nas linhas 430 e 431 as variáveis recebem os valores referentes a linha e a coluna da primeira coordenada localizada no segmento M_b. Em seguida, chama-se o método localização do segmento M_p - cuja orientação nesse atributo antropométrico é vertical - passando como parâmetro as duas variáveis - `linha` e `coluna` - e a URL da imagem salva - referentes ao `canvas` três. A Figura 34 mostra o código desse método.

```
72  public static List<int[]> LocalizarSegmentoVerticalMp(string url, int? coluna, int? linha)
73  {
74      try
75      {
76          var imagem = new Bitmap(url);
77          var coordenadasSegmento = new List<int[]>();
78          if (coluna == null || linha == null) return coordenadasSegmento;
79          for (var x = 0; x < imagem.Width; x++)
80          {
81              if (x > linha.Value)
82              {
83                  break;
84              }
85              if (imagem.GetPixel(coluna.Value, x).R == 0 && imagem.GetPixel(coluna.Value, x).G == 0
86                  && imagem.GetPixel(coluna.Value, x).B == 0)
87              {
88                  coordenadasSegmento.Add(new[]
89                  {
90                     coluna.Value, x
91                  });
92              }
93          }
94          return coordenadasSegmento;
95      }
96      catch (Exception ex)
97      {
98          throw new Exception("Ocorreu um erro ao tentar localizar o SEGMENTO VERTICAL Mp na imagem!", ex);
99      }
100 }
```

Figura 34. Método para localização do segmento M_p com orientação vertical

O método acima segue o princípio básico dos métodos de localização apresentados anteriormente. A diferença encontra-se na ordem em que a imagem é percorrida e na condição de parada referente à linha inicial da primeira coordenada do segmento M_b. É importante destacar que, para a estatura, o segmento M_p inicia-se no ápice da cabeça do paciente, encerrando-se no início do segmento M_b. Além disso, a localização é iniciada pela linha, percorrendo as respectivas colunas. Na linha 81, é verificado se a linha do pixel atual é maior que a linha do pixel recebido como parâmetro. Caso seja, encerra-se o laço de repetição e retorna as coordenadas que foram encontradas. A Figura 35 ajuda na compreensão da lógica adotada. Nas linhas 85 e 86 é verificado se o pixel percorrido faz parte do segmento M_p.

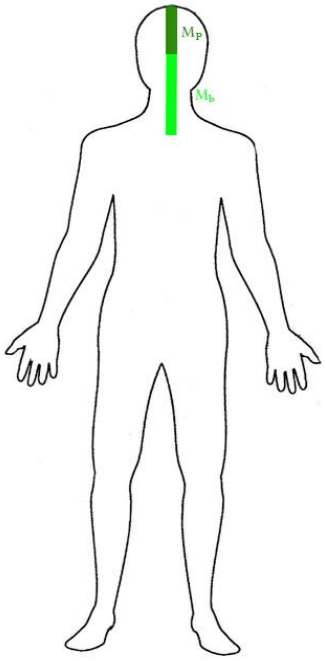

Figura 35. Segmentos M_p e M_b referentes a estatura

Após a execução das etapas descritas nessa subseção, é importante destacar que no instante em que o usuário selecionou a opção de cálculo da estatura, foi obtida a distância entre os *joints head* e *foot right*. Essa distância encontra-se na variável `_alturaPeloKinect` apresentada na Figura 23. Assim, o resultado final desse atributo antropométrico é dado pela soma do valor do tamanho do segmento M_p com o valor da variável `_alturaPeloKinect`.

4.6 Testando o *software*

Com a conclusão de desenvolvimento da ferramenta, realizou-se alguns testes para verificar o seu funcionamento. No decorrer dos testes com o *software*, percebeu-se que os resultados variavam dependendo da distância em que a pessoa encontrava-se do Kinect. Esse problema ocorre devido a variação de tamanho da silhueta corporal com relação a essa distância. Em

outras palavras: quanto mais próximo do Kinect a pessoa estiver, maior será sua silhueta, bem como, quanto mais longe a pessoa estiver, menor será sua silhueta. Esse problema ocorre devido ao modo como o cálculo é realizado, já que é levado em consideração a quantidade de pixels dos segmentos M_b e M_p na obtenção do resultado final.

Também se observou outro fator que pode gerar imprecisão no resultado. Trata-se do tipo de roupa usada e o tipo de corte do cabelo, pois o algoritmo de limiarização não faz distinção de roupas, cabelo e couro cabeludo em relação a silhueta do corpo humano. O algoritmo de limiarização consiste em tentar representar da melhor forma possível a silhueta corporal, dessa forma, se a pessoa estiver com uma roupa folgada, o resultado da largura da cintura e/ou largura do quadril será impreciso, já esse detalhe pode aumentar a quantidade de pixels do segmento M_p. O mesmo problema ocorre com o cálculo da estatura, sendo que esse tem como fator de imprecisão o tipo de penteado da pessoa. Por exemplo, uma pessoa cujo penteado for um topete terá imprecisão no resultado se comparado a uma pessoa que não possui tal penteado, já que o topete também aumentará a quantidade de pixels de M_p.

Assim sendo, elaborou-se um protocolo para minimizar qualquer fator que viesse a intervir na coleta dos dados e no resultado, desde a inicialização do *software* até a posição em que o avaliado deve se encontrar no momento da avaliação.

4.6.1 Protocolo de avaliação

Para evitar os problemas supracitados na subseção anterior, o avaliado deve estar utilizando roupas justas. Dependendo do penteado, deve-se utilizar uma toca de natação, pois essa reduz qualquer chance de imprecisão quanto à estatura. Além disso, o avaliador deve certificar-se que Kinect encontre em condições do usuário situar-se totalmente em seu campo de visão. Nos testes realizados o dispositivo encontrou-se a 1,15m de altura. O Xbox (2015) recomenda que o dispositivo esteja entre 0,6 e 1,8m de altura. Em seguida, o avaliador posiciona a pessoa avaliada a 2,7m de distância do sensor.

As subseções a seguir explicam os procedimentos a serem feitos para iniciar a coleta dos atributos antropométricos desenvolvidos no *software*.

4.6.1.1 Estatura e largura do ombro

Para calcular a estatura e a largura do ombro, o avaliado deve realizar uma apneia respiratória, onde o avaliador deve estar atento ao instante em que esse momento acontecer, já que cabe a ele clicar no botão Capturar. Além disso, os braços do avaliado devem estar rentes ao corpo. A Figura 36 mostra aposição em correta em que o avaliado deve se encontrar no momento da avaliação.

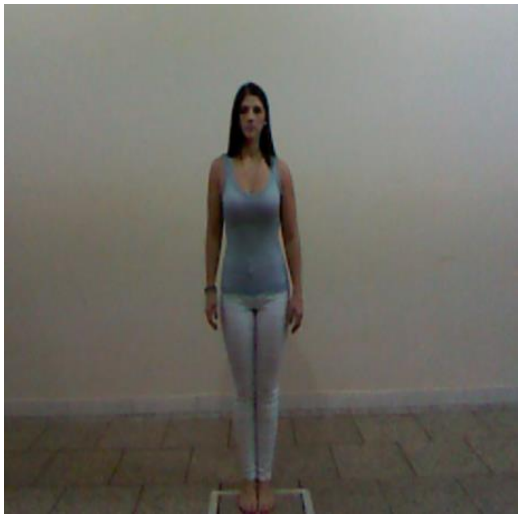

Figura 36. Captura da estatura e largura do quadril

Após o avaliador clicar no botão Capturar, o paciente deve permanecer imóvel até que o *software* exiba o resultado do atributo a ser calculado.

4.6.1.2 Largura do quadril

Para a largura do quadril, o avaliado deve realizar uma abdução dos braços em 90º. Caso o avaliado esteja com os braços rentes ao corpo, conforme a Figura 36 haverá uma imprecisão muito grande, já que a posição dos braços aumentará a quantidade de pixels do segmento M_p. A Figura 37 mostra o avaliado na posição ideal para cálculo da largura do quadril.

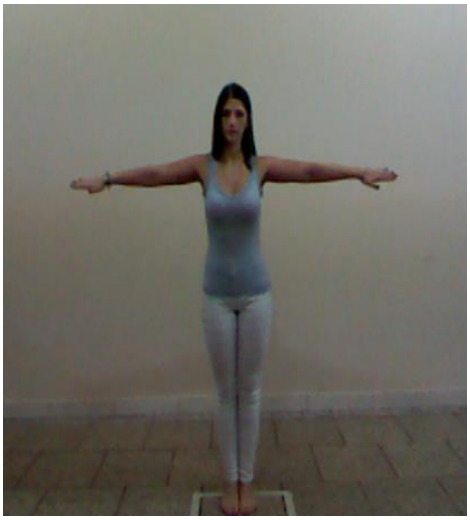

Figura 37. Captura da largura do quadril

Da mesma forma citada na subseção anterior, durante a avaliação o paciente deve encontrar-se totalmente imóvel, até que o resultado seja exibido na tela. Desta forma, com o uso desse protocolo, esperam-se resultados que demonstram a viabilidade de utilização do sensor Kinect para automatizar os procedimentos manuais que são utilizados para a obter a

mensuração dos atributos antropométricos, garantindo assim, um resultado em poucos cliques, posto que, o *software* desenvolvido é de fácil utilização.

4.6.2 Comparativo entre a mensuração através do *software* e do método tradicional

Em seu estudo, Lima (2015) utilizou o *software* desenvolvido para realizar a coleta de dados de 142 de alunos do CEULP (n=59M; n=83F), com idades entre 18 e 50 anos. Cada atributo antropométrico foi coletado três vezes através de métodos tradicionais e três vezes através do *software*. Na coletar os dados pelo método tradicional utilizou-se o estadiômetro para mensurar a estatura e o paquímetro para mensurar as larguras do ombro e do quadril.

Para análise dos dados a autora utilizou estatística descritiva (média, desvio padrão, *Intraclass Correlation Coefficient* - ICC e Alfa de Cronbach, todos com intervalo de confiança 95%), com análise da normalidade dos dados por meio do teste de Shapiro-Wilk e utilização do teste t pareado bicaudal para comparações do perfil antropométrico da avaliação tradicional com a obtida pelo sistema com sensor Kinect. Todas as análises estatísticas foram realizadas através do *software Statistical Package for Social Sciences* (SPSS, Inc., v. 20.0; IBM Corporation, Somers, NY, USA.) com um nível de significância de $p \leq 0{,}05$.

A Tabela 1 abaixo exibe a média e o desvio padrão dos dados coletados tanto pelo *software* quanto pelo método tradicional utilizando o paquímetro e estadiômetro.

Tabela 1. Média e desvio padrão dos dados coletados por Lima (2015)

ATRIBUTO	MÉDIA±DP DA AVALIAÇÃO PELO SOFTWARE	MÉDIA±DP DA AVALIAÇÃO TRADICIONAL
Estatura	1,690± 0,108	1,686± 0,097
Largura do ombro	0,499± 0,062	0,441± 0,048
Largura do quadril	0,314± 0,049	0,344± 0,026

Fonte: Lima (2015)

Como se nota, as diferenças entre as médias das avaliações tradicionais e as médias das avaliações realizadas pelo software foram de -0.004 para estatura, -0,058 para a largura do ombro; e de 0,03 para a largura do quadril, sendo que nenhuma das diferenças foi estatisticamente significativa.

Os dados coletados para a estatura apresentaram escores altos tanto para o ICC, quanto para o Alfa de Cronbach, indicando que os resultados apresentam excelentes correlação intraclasse e consistência, com valores de 0,962 e de 0,981, respectivamente.

Já para a largura do ombro, o Alfa de Cronbach foi de 0,908, mostrando excelente consistência, porém o ICC foi igual a 0,622, denotando uma correlação intraclasse moderada. O mesmo foi notado na largura do quadril, onde o Alfa de Cronbach teve valor igual a 0,926 e o ICC valor é

igual a 0,675. Sobre isto, a autora argumenta que, talvez, este resultado esteja associado tanto ao tamanho da área avaliada, como a qualidade da câmera da primeira versão do *Kinect*.

Quanto ao tamanho/comprimento da área avaliada, a autora argumenta que em áreas maiores, como a estatura, as diferenças em pixels sejam pouco relevantes enquanto que, em áreas menores, as diferenças tenham impacto muito mais significativo.

Além disto, a imprecisão da silhueta gerada pela primeira versão do *Kinect* possa também influenciar tal resultado, todavia, ela ainda propõe o método seja adaptado para a versão 2.0 do *Kinect* em trabalhos futuros, dada a possibilidade de que esta nova versão gere uma silhueta mais precisa pela qualidade de sua câmera.

5 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo o desenvolvimento de um *software* para delimitação da silhueta corporal utilizando o Kinect e recursos de computação gráfica. O estudo dos oferecidos pelo Kinect juntamente com as técnicas da computação gráfica foram fundamentais para que se chegasse aos resultados apresentados. Para verificar a viabilidade do uso de recursos do Kinect aliados às técnicas de computação gráfica, optou-se pela implementação de módulos para realizar mensuração de alguns atributos antropométricos, já que, o conceito do cálculo desses atributos envolve a representação do corpo humano.

A técnica de limiarização aplicada à imagem obtida pela câmera de profundidade do Kinect ajuda a obter uma representação do corpo humano quase que ideal. O único fator que deixou a desejar foi a imprecisão encontrada na borda da silhueta do corpo humano, pois a mesma nem sempre é obtido de forma precisa. Tal fator não torna a técnica de delimitação de áreas inviável, posto que é quase impossível obter uma representação perfeita com a qualidade que a câmera de profundidade fornece a imagem para aplicação da técnica.

Uma das etapas mais complexas de ser realizada nesse trabalho foi a elaboração da lógica para realização do cálculo, pois, a princípio, decidiu-se que não se utilizaria nenhum outro recurso computacional para identificar a região referente aos atributos antropométricos na imagem após a aplicação da técnica de limiarização. Assim sendo, surgiu a ideia dos segmentos M_b e M_p a partir de estudos realizados em outros trabalhos, de forma que essa ideia se tornou indispensável para realização do cálculo de cada atributo antropométrico.

Após concluir a fase de implementação do *software*, os testes realizados foram essenciais para a elaboração do protocolo de avaliação, pois, sem a realização dos mesmos não existiria uma melhor compreensão de quais fatores afetavam o resultado final do cálculo, já que o algoritmo de limiarização não faz distinção dos elementos de imprecisão mencionados no trabalho.

Trabalhos futuros que darão continuidade a este poderão aplicar outros recursos computacionais gráficos a fim de melhorar ainda mais a silhueta, de forma que se tenha uma representação com o mínimo de imprecisão possível. Também, podem ser implementados módulos para mensurar novos atributos antropométricos. Do mesmo modo, novos trabalhos podem ser desenvolvidos utilizando o Kinect v2, visto que essa versão do dispositivo é superior a versão utilizada no presente trabalho, além de fornecer novos recursos que podem melhorar ainda mais a precisão do cálculo realizado pelo *software*.

Dessa forma, conclui-se que com o uso dos recursos do Kinect junto com recursos de computação gráfica, é possível que se obtenha uma representação da silhueta do corpo humano e que se tenha uma provável medida de determinados atributos antropométricos.

REFERÊNCIAS

ALBUQUERQUE, Márcio Portes de; ALBUQUERQUE, Marcelo Portes de. Processamento de imagens: métodos e análises. **Centro Brasileiro de Pesquisas Físicas MCT**, 2000.

ALVES, Rodrigo de Sales; ARAUJO, Jefferson Oliveira Alves de; MADEIRO, Francisco. AlfabetoKinect: Um aplicativo para auxiliar na alfabetização de crianças com o uso do Kinect. In: **Anais do Simpósio Brasileiro de Informática na Educação**. 2012.

BALLARD, Dana H.; BROWN Christopher M. **Computer Vision**. New Jersey: Prentice-Hall, Inc., 1982.

BESSANI, Michel. **Viabilidade de um sistema para auxílio no diagnóstico de cáries através de imagem de fluorescência e processamento digital de imagens.** 2012. 79 f. TCC (Graduação) - Curso de Engenharia Elétrica com ênfase em Eletrônica, Universidade de São Paulo, São Carlos, 2012.

BORBA, Anderson. **Níveis de Organização do Corpo Humano.** Disponível em: <http://cienciasmorfologicas.webnode.pt/fisiologia sistêmica/niveis de organização do corpo humano/>. Acesso em: 10 jun. 2015.

BOUCHARD, Samuel. **Using the Kinect for robotic manipulation.** 2011. Disponível em: <http://blog.robotiq.com/bid/40428/Using-The-Kinect-For-Robotic-Manipulation>. Acesso em: 10 jun. 2015.

CHAVES, Carlos Diego Carvalho. **Automatização do Image Marking Procedure para análise do esquema corporal utilizando o Kinect**. 2014. 67 f. TCC (Graduação) - Curso de Sistemas de Informação, Centro Universitário Luterano de Palmas, Palmas - To, 2014.

CONTI, Caroline. **Rastreamento de movimentos da mão humana usando Visão Computacional**. 2010. 112 f. TCC (Graduação) - Curso de Engenharia de Computação Com ênfase em Eletrônica, Universidade de São Paulo, São Carlos, 2010. Disponível em: <http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-30082010-150415/?&lang=br>. Acesso em: 19 maio 2015.

COSTA, Valéria Catelli Infantozzi. **Anatomia geral humana**. 2008. Disponível em: <http://neurociencia.tripod.com/labs/lela/textos/APOSTILA_ANATOMIA_HUMANA.pdf>. Acesso em: 10 mar. 2015.

CRUZ, Roberta Kelly Cavalcante. **Posição Anatômica, Planos e Movimentos**. 2014. Disponível em: <http://www.portaleducacao.com.br/fisioterapia/artigos/57590/posicao-anatomica-planos-e-movimentos>. Acesso em: 11 mar. 2015.

CUNHA, Gladis Franck da. **Aspectos gerais e históricos da anatomia humana**. 2011.

DÂNGELO, José Geraldo; FATTINI, Carlo Américo. **Anatomia Humana Básica**. São Paulo: Atheneu, 2002. 184 p.

DAVIS, Steve. **Divisions of human anatomy.** 2011. Disponível em: <http://bodyhuman.blog.com/2011/03/02/divisions-of-human-anatomy/>. Acesso em: 02 jun. 2015.

DIAS, Felipe da Cruz. **Uso do software Image J para análise quantitativa de imagens de microestruturas de materiais**. 2008. 148 f. Dissertação (Mestrado) - Curso de Engenharia e Tecnologia Espaciais/Ciência e Tecnologia de Materiais e Sensores, Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2008. Disponível em: <http://www.teliga.net/2011/02/aspectos-gerais-e-historicos-da.html>. Acesso em: 02 mar. 2015.

DRAKE, Richard *et al*. **Gray's - Anatomia para Estudantes**. S.l: Elsevier, 2010. 1136 p.

DUARTE, Glaucius Décio. **Uso da Transformada de Hough na Detecção de Círculos em Imagens Digitais**. Pelotas, S. d, 14 p.

DUARTE, Hamilton Emídio. **Anatomia Humana**. Florianópolis, 2009. 174 p.

DUDA, Richard O.; HART, Peter E. **Use of the Hough transformation to detect lines and curves in pictures**. Communications of the ACM., vol. 15, pp. 11-15, 1972.

FACON, J. **Processamento e Análise de Imagens**. Pontifícia Universidade Católica do Paraná.

FERRAREZI, Gabriel Paschoal. **Sistema de aquisição de imagem do olho humano para avaliação da resposta da pupila submetida a estímulos luminosos.** 2010. 63 f. TCC (Graduação) - Curso de Curso de Engenharia Elétrica Com ênfase em Eletrônica, Universidade de São Paulo, São Carlos, 2010. Disponível em: <http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-23082010-113542/?&lang=br>. Acesso em: 28 maio 2015.

FERRO NETO, Jarbas da Silva. **Expansão de técnica de detecção de agrupamentos de microcalcificações em mamografias digitais.** 2012. 76 f. TCC (Graduação) - Curso de Engenharia Elétrica Com ênfase em Eletrônica, Universidade de São Paulo, São Carlos, 2012. Disponível em: <http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-06022013-144403/?&lang=br>. Acesso em: 28 maio 2015.

FIORIN, Magliani Reis et al. **Motion Rehab: um jogo sério para idosos com sequelas de Acidente Vascular Encefálico**. 2014. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wim/2014/009.pdf>. Acesso em: 14 de maio 2015.

FRONZA, Thiago. **Anatomia Humana - Posição e Planos Anatômicos**. 2011. Disponível em: <http://focobiomedico.blogspot.com.br/2011/03/anatomia-humana-posicao-anatomica.html>. Acesso em: 10 mar. 2015.

GOMES, Otávio da Fonseca Martins. **Processamento e análise de imagens aplicados à caracterização automática de materiais**. 2001. 141 f. Curso de Ciências da Engenharia Metalúrgica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2001.

GRAAFF, van de. **Anatomia Humana**. 6. ed. [S.l]: Manole, 2003. 900 p.

IGNACIO, Juliano da Silva. **Processamento e análise de imagens da cinética de recristalização das ligas Al-Mg-X.** 2013. 106 f. Dissertação (Mestrado) - Curso de Ciências, Instituto de Pesquisas Energéticas e Nucleares, São Paulo, 2013. Disponível em: <http://pelicano.ipen.br/PosG30/TextoCompleto/Juliano da Silva Ignacio_M.pdf>. Acesso em: 24 mar. 2015.

KHOSHELHAM, Kourosh; ELBERINK, Sander Oude. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. **Sensors,** [s.l.], v. 12, n. 12, p.1437-1454, 1 fev. 2012. MDPI AG. DOI: 10.3390/s120201437.

LIMA, Stephanie Tamarozzi. **A confiabilidade e consistência de medidas antropométricas através do sensor Microsoft Kinect**. 2015. 49 f. Trabalho de Conclusão de Curso (Graduação) – Curso de Educação Física (Bacharelado), Centro Universitário Luterano de Palmas, Palmas/TO, 2015.

LIMEIRA JUNIOR, Francisco de Assis. **Anatomia Humana**. João Pessoa: Universitária, 2009.

LÓPEZ, José Juan Hernández et al. Detecting objects using color and depth segmentation with Kinect sensor. **Procedia Technology,** [s.l.], v. 3, p.196-204, 2012. Elsevier BV. DOI: 10.1016/j.protcy.2012.03.021. Disponível em: <http://api.elsevier.com/content/article/PII:S2212017312002502?httpAccept=text/xml>. Acesso em: 13 jun. 2015.

MACEDO, Maysa Malfiza Garcia de. **Uso da Transformada de Hough na Vetorização de Moldes e Outras Aplicações.** 2005. 117 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal Fluminense, Niterói, 2005. Disponível em: <http://www2.ic.uff.br/PosGraduacao/Dissertacoes/253.pdf>. Acesso em: 27 maio 2015.

MANZI, Filipe Augusto. **Aplicação de visão computacional para extração de características em imagens do olho humano**. 2007. 47 f. TCC (Graduação) - Curso de Engenharia de Computação Com ênfase em Sistemas Embarcados, Universidade de São Paulo, São Carlos, 2007.

MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento Digital de Imagens**. Rio de Janeiro: Brasport, 1999.

MASCARENHAS, N.D.A.; VELASCO, F.R.D. (1989). **Processamento Digital de Imagens**. 2 ed. Buenos Aires: KAPELUSZ.

MICROSOFT. **Microsoft Kinect: Kinect for Windows**. 2015. Disponível em: <http://www.microsoft.com/en-us/kinectforwindows/>. Acesso em: 20 de maio de 2015.

OLIVEIRA, Leonardo Augusto de. **Localização e reconhecimento de caracteres em placas de automóveis. 2010**. 88 f. TCC (Graduação) - Curso de Engenharia Elétrica com ênfase em Eletrônica, Universidade de São Paulo, São Carlos, 2010. Disponível em: <http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-16112011-113239/?&lang=br>. Acesso em: 22 maio 2015.

PACIORNIK, Sidnei. **Processamento Digital de Imagens**. 2001. Disponível em: <http://www.dcmm.puc-rio.br/cursos/ipdi/index_files/frame.html>.

PANGER, Galen. **Kinect in the Kitchen: Testing Depth Camera Interactions in Practical Home Environments.** Berkeley: University Of California, 2012.

PAULA, Bruno Campagnolo de. **Adaptando e desenvolvendo jogos para uso com o Microsoft Kinect.** Paraná: Sbgames, 2011. 13 p. Disponível em: <http://www.sbgames.org/sbgames2011/proceedings/sbgames/papers/tut/1-kinect_FAAST_Final_MesmoComColunas.pdf>. Acesso em: 12 jun. 2015.

PERROTTI, Francesco Artur. I**magens em níveis de cinza**. Disponível em: <https://fperrotti.wikispaces.com/Imagens+em+níveis+de+cinza>. Acesso em: 13 maio 2015.

PIAZZA, Bruno Luis. **O ensino de anatomia humana nos cursos de Educação Física da região metropolitana de Porto Alegre**. Ciência em Movimento**,** Porto Alegre, v. 26, n. 13, p.99-109, fev. 2011.

PIVETTA, Cleber; MANTOVANI, Gustavo; ZOTTIS, Felipe. **Transformada de Hough.** Cascavel: Aula de PDI, 2010. Color. Disponível em: <http://www.inf.unioeste.br/~adair/PID/Notas Aula/Transformada de Hough.pdf>. Acesso em: 27 ago. 2015.

QUEIROZ, José Eustáquio Rangel de; GOMES, Herman Martins. **Introdução ao Processamento Digital de Imagens.** 2012. Disponível em: <http://dsc.ufcg.edu.br/~hmg/disciplinas/graduacao/vc-2011.2/Rita-Tutorial-PDI.pdf>. Acesso em: 03 mar. 2015.

SALDANHA Marcus F. S.; FREITAS, Corina da Cost. Segmentação de Imagens Digitais: Uma Revisão. In: **IX Workshop do Curso de Computação Aplicada - CAP Auditório Fernando de Mendonça**, LIT/INPE, outubro de 2009.

SILVA, Andre Grzybowski Albano. **Rastreamento de objetos em tempo real para aplicações em jogos esportivos.** 2012. 90 f. TCC (Graduação) - Curso de Engenharia de Computação Com ênfase em Eletrônica, Universidade de São Paulo, São Carlos, 2012. Disponível em: <http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-14112012-160937/?&lang=br>. Acesso em: 26 maio 2015.

SILVA, Rubéns. **Noções Básicas Em Anatomia Humana I**. 2007. Disponível em: <http://ucbweb2.castelobranco.br/webcaf/arquivos/12881/81/ANATOMIA_I_NOCOES_BASICAS.pdf>. Acesso em: 11 mar. 2015.

SOUZA, Fernando Peixoto Coelho de. **Localização e leitura automática de caracteres alfanuméricos – uma aplicação na identificação de veículos.** 2000. 96 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Departamento de Engenharia Elétrica da Ufrgs, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2000. Disponível em: <http://www.lume.ufrgs.br/handle/10183/1833>. Acesso em: 21 maio 2015.

SOUZA, L. C. de *et al*. Avaliação da Amplitude de Movimento (AMD) em plano coronal anterior utilizando o sensor Kinect: articulações do ombro e do quadril. **Encoinfo**, Brasil, out. 2015. Disponível em: <http://ulbra-to.br/encoinfo/index.php/encoinfo/encoinfo2015/paper/view/326/486>. Data de acesso: 10 Nov. 2015.

TORTORA, Gerard J.. **Corpo Humano - Fundamentos de Anatomia e Fisiologia.** 4. ed. [s. L.]: Artmed, 2001. 630 p.

VEGA, Maria Luiza Monteiro Dela. **Métodos e análise para seg-mentação de imagens ultrasso-nográficas da mama**. 2011. 83 f. TCC (Graduação) - Curso de Curso de Engenharia Elétrica Com ênfase em E-letrônica, Departamento de Engenharia Elétrica, Universidade de São Paulo, São Carlos, 2011. Disponível em: <http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-02042012-084929/?&lang=br>. Acesso em: 22 maio 2015.

VERDERI, Érica. **A importância da avaliação postural.** 2003. Disponível em: <http://www.efdeportes.com/efd57/postura.htm>. Acesso em: 02 mar. 2015.

XBOX, Support. **Mais sobre posicionamento do sensor Kinect.** 2015. Microsoft. Disponível em: <http://support.xbox.com/pt-BR/xbox-360/kinect/sensor-placement>. Acesso em: 22 out. 2015.

APÊNDICES

APÊNDICE A – Código-fonte da tela principal

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Threading;
using System.Windows;
using System.Windows.Controls.Primitives;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using KinectDepth.Controller;
using KinectDepth.Models;
using Microsoft.Kinect;

namespace KinectDepth
{
    public partial class MainWindowDepth : Window
    {
        private bool _imagemRgbHabilitado;
        private bool _imagemSensorProfundidadeHabilitado;
        private double _distancia;
        private double _alturaPeloKinect;
        private readonly DetalhesCaptura _detalhes;
        private KinectSensor _sensorKinect;
        private WriteableBitmap _imagemProfundidadeBruta;
        private Int32Rect _imagemProfundidadeBrutaRect;
        private short[] _dadosPixelProfundidadeBruta;
        private int _imagemProfundidadePassadaBruta;
        private WriteableBitmap _imagemProfundidadeTratada;
        private Int32Rect _imagemProfundidadeTratadaRect;
        private int _imagemProfundidadePassadaTratada;
        private Paciente _paciente;

        public KinectSensor SensorKinect
        {
            get { return _sensorKinect; }
            set
            {
                if (_sensorKinect != value)
                {
                    if (_sensorKinect != null)
                    {
                        _sensorKinect.Stop();
                        _sensorKinect.DepthFrameReady -= SensorKinectDepthFrameReady;
                        _sensorKinect.DepthStream.Disable();
                        _sensorKinect.SkeletonStream.Disable();
                        ImagemSensorProfundidadeComSegmento.Children.Clear();
                        ImagemSensorProfundidadeSemSegmento.Children.Clear();
                    }
                    _sensorKinect = value;

                    if (_sensorKinect != null)
                    {
                        if (_sensorKinect.Status == KinectStatus.Connected)
                        {
                            _sensorKinect.SkeletonStream.Enable();
                            _sensorKinect.DepthStream.Enable();

                            var fluxoProfundidade = _sensorKinect.DepthStream;
                            _imagemProfundidadeBruta = new
WriteableBitmap(fluxoProfundidade.FrameWidth,
```

```
                                fluxoProfundidade.FrameHeight, 96, 96,
PixelFormats.Gray16, null);
                            _imagemProfundidadeBrutaRect = new Int32Rect(0, 0,
(int)Math.Ceiling(_imagemProfundidadeBruta.Width),
                                (int)Math.Ceiling(_imagemProfundidadeBruta.Height));
                            _imagemProfundidadePassadaBruta =
fluxoProfundidade.FrameWidth * fluxoProfundidade.FrameBytesPerPixel;
                            _dadosPixelProfundidadeBruta = new
short[fluxoProfundidade.FramePixelDataLength];

                            _imagemProfundidadeTratada = new
WriteableBitmap(fluxoProfundidade.FrameWidth,
                                fluxoProfundidade.FrameHeight, 96, 96,
PixelFormats.Bgr32, null);
                            _imagemProfundidadeTratadaRect = new Int32Rect(0, 0,
(int)Math.Ceiling(_imagemProfundidadeTratada.Width),
                                (int)Math.Ceiling(_imagemProfundidadeTratada.Height));
                            _imagemProfundidadePassadaTratada =
fluxoProfundidade.FrameWidth * 4;

                            ImagemSensorProfundidadeComSegmento.Background = new
ImageBrush(_imagemProfundidadeTratada);
                            ImagemSensorProfundidadeSemSegmento.Background = new
ImageBrush(_imagemProfundidadeTratada);

                            _sensorKinect.DepthFrameReady +=
SensorKinectDepthFrameReady;
                            _sensorKinect.ColorStream.Enable();
                            _sensorKinect.AllFramesReady +=
SensorKinectSensorOnAllFramesReady;
                            _sensorKinect.Start();
                        }
                    }
                }
            }
        }

        public MainWindowDepth(Paciente paciente)
        {
            try
            {
                InitializeComponent();
                _paciente = paciente;
                _alturaPeloKinect = 0;
                _detalhes = new DetalhesCaptura();
                NomePaciente.Content = paciente.Nome;
                DataAvaliacao.Content = DateTime.Now.ToShortDateString();
                KinectSensor.KinectSensors.StatusChanged +=
KinectSensors_StatusChanged;
                SensorKinect = KinectSensor.KinectSensors.FirstOrDefault(k => k.Status
== KinectStatus.Connected);
                BtnDetalhesCaptura.IsEnabled = false;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void SensorKinectSensorOnAllFramesReady(object sender,
AllFramesReadyEventArgs e)
```

```
        {
            try
            {
                var imagem = CameraImagemRgb(e.OpenColorImageFrame());
                if (imagem != null)
                {
                    ImagemCameraRgb.Background = new ImageBrush(BitmapSource.Create(
                        _sensorKinect.ColorStream.FrameWidth,
_sensorKinect.ColorStream.FrameHeight,
                        96, 96, PixelFormats.Bgr32, null, imagem,
                        _sensorKinect.ColorStream.FrameWidth *
_sensorKinect.ColorStream.FrameBytesPerPixel));
                }
                ImagemCameraRgb.Children.Clear();
                ImagemSensorProfundidadeComSegmento.Children.Clear();
                ImagemSensorProfundidadeSemSegmento.Children.Clear();
                if (CameraRgbRadioButton.IsChecked == true)
                {
                    HabilitaImagemCameraRgb();
                }
                if (SensorProfundidadeRadioButton.IsChecked == true)
                {
                    HabilitaImagemSensorProfundidade();
                }
                FuncoesParaCaptura(e.OpenSkeletonFrame());
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private static byte[] CameraImagemRgb(ColorImageFrame quadro)
        {
            try
            {
                if (quadro == null) return null;
                using (quadro)
                {
                    var bytesImagem = new byte[quadro.PixelDataLength];
                    quadro.CopyPixelDataTo(bytesImagem);
                    return bytesImagem;
                }
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }
        }

        public void HabilitaImagemCameraRgb()
        {
            if (!_imagemRgbHabilitado)
            {
                ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Hidden;
                ImagemCameraRgb.Visibility = Visibility.Visible;
                _imagemRgbHabilitado = true;
                _imagemSensorProfundidadeHabilitado = false;
            }
        }
```

```
        public void HabilitaImagemSensorProfundidade()
        {
            if (!_imagemSensorProfundidadeHabilitado)
            {
                ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Visible;
                ImagemCameraRgb.Visibility = Visibility.Hidden;
                _imagemRgbHabilitado = false;
                _imagemSensorProfundidadeHabilitado = true;
            }
        }

        public void FuncoesParaCaptura(SkeletonFrame quadro)
        {
            try
            {
                if (quadro == null)
                {
                    return;
                }
                using (quadro)
                {
                    var esqueletos = new Skeleton[quadro.SkeletonArrayLength];
                    quadro.CopySkeletonDataTo(esqueletos);
                    var esqueletosRastreados = esqueletos.Where(e => e.TrackingState
== SkeletonTrackingState.Tracked);
                    var rastreados = esqueletosRastreados as Skeleton[] ??
esqueletosRastreados.ToArray();
                    if (rastreados.Any())
                    {
                        var esqueleto = rastreados.First();
                        var funcoesEsqueleto = new EsqueletoUsuario(_sensorKinect);
                        if (SelecionaEstaturaPaciente.IsChecked == true)
                        {
                            DesenharEstatura(funcoesEsqueleto, esqueleto);
                        }
                        else if (SelecionaLarguraQuadril.IsChecked == true)
                        {
                            DesenharLarguraQuadril(funcoesEsqueleto, esqueleto);
                        }
                        else if (SelecionaLarguraOmbros.IsChecked == true)
                        {
                            DesenharLarguraOmbro(funcoesEsqueleto, esqueleto);
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }
        }

        public void DesenharLarguraOmbro(EsqueletoUsuario funcoesEsqueleto, Skeleton
esqueleto)
        {
            try
            {
                var ombroDireito = esqueleto.Joints[JointType.ShoulderRight];
                var ombroEsquerdo = esqueleto.Joints[JointType.ShoulderLeft];
                funcoesEsqueleto.DesenharArticulacao(ombroEsquerdo,
ImagemSensorProfundidadeComSegmento, Brushes.Red);
```

```
                funcoesEsqueleto.DesenharArticulacao(ombroDireito,
ImagemSensorProfundidadeComSegmento, Brushes.Red);
                funcoesEsqueleto.DesenharArticulacao(ombroEsquerdo, ImagemCameraRgb,
Brushes.Red);
                funcoesEsqueleto.DesenharArticulacao(ombroDireito, ImagemCameraRgb,
Brushes.Red);
                funcoesEsqueleto.DesenharOsso(ombroDireito, ombroEsquerdo,
ImagemSensorProfundidadeComSegmento);
                funcoesEsqueleto.DesenharOsso(ombroDireito, ombroEsquerdo,
ImagemCameraRgb);
                _distancia =
KinectMath.DistanciaEntreDoisPontos(ombroEsquerdo.Position.X,
ombroEsquerdo.Position.Y,
                    ombroDireito.Position.X, ombroDireito.Position.Y);
            }
            catch (Exception ex)
            {
                throw new Exception("Não foi possível desenhar o segmento de Mb para a
largura do ombro!", ex);
            }
        }

        public void DesenharEstatura(EsqueletoUsuario funcoesEsqueleto, Skeleton
esqueleto)
        {
            try
            {
                var cabeca = esqueleto.Joints[JointType.Head];
                var ombro = esqueleto.Joints[JointType.ShoulderCenter];
                funcoesEsqueleto.DesenharArticulacao(cabeca,
ImagemSensorProfundidadeComSegmento, Brushes.Red);
                funcoesEsqueleto.DesenharArticulacao(ombro,
ImagemSensorProfundidadeComSegmento, Brushes.Red);
                funcoesEsqueleto.DesenharArticulacao(cabeca, ImagemCameraRgb,
Brushes.Red);
                funcoesEsqueleto.DesenharArticulacao(ombro, ImagemCameraRgb,
Brushes.Red);
                funcoesEsqueleto.DesenharOsso(ombro, cabeca,
ImagemSensorProfundidadeComSegmento);
                funcoesEsqueleto.DesenharOsso(ombro, cabeca, ImagemCameraRgb);
                _distancia = KinectMath.DistanciaEntreDoisPontos(cabeca.Position.X,
cabeca.Position.Y,
                    ombro.Position.X, ombro.Position.Y);
                var peDireito = esqueleto.Joints[JointType.FootRight];
                _alturaPeloKinect =
KinectMath.DistanciaEntreDoisPontos(cabeca.Position.X, cabeca.Position.Y,
                    peDireito.Position.X, peDireito.Position.Y);
            }
            catch (Exception ex)
            {
                throw new Exception("Não foi possível desenhar o segmento de Mb para a
estatura!", ex);
            }
        }

        public void DesenharLarguraQuadril(EsqueletoUsuario funcoesEsqueleto, Skeleton
esqueleto)
        {
            try
            {
                var quadrilDireito = esqueleto.Joints[JointType.HipRight];
                var quadrilEsquerdo = esqueleto.Joints[JointType.HipLeft];
```

```
                funcoesEsqueleto.DesenharArticulacao(quadrilDireito,
ImagemSensorProfundidadeComSegmento, Brushes.Red);
                funcoesEsqueleto.DesenharArticulacao(quadrilEsquerdo,
ImagemSensorProfundidadeComSegmento, Brushes.Red);
                funcoesEsqueleto.DesenharOsso(quadrilEsquerdo, quadrilDireito,
ImagemSensorProfundidadeComSegmento);
                funcoesEsqueleto.DesenharArticulacao(quadrilDireito, ImagemCameraRgb,
Brushes.Red);
                funcoesEsqueleto.DesenharArticulacao(quadrilEsquerdo, ImagemCameraRgb,
Brushes.Red);
                funcoesEsqueleto.DesenharOsso(quadrilEsquerdo, quadrilDireito,
ImagemCameraRgb);
                _distancia =
KinectMath.DistanciaEntreDoisPontos(quadrilEsquerdo.Position.X,
quadrilEsquerdo.Position.Y,
                    quadrilDireito.Position.X, quadrilDireito.Position.Y);
            }
            catch (Exception ex)
            {
                throw new Exception("Não foi possível desenhar o segmento de Mb para a
largura da cintura!", ex);
            }
        }

        private void KinectSensors_StatusChanged(object sender, StatusChangedEventArgs
e)
        {
            try
            {
                switch (e.Status)
                {
                    case KinectStatus.Initializing:
                    case KinectStatus.Connected:
                    case KinectStatus.NotPowered:
                    case KinectStatus.NotReady:
                    case KinectStatus.DeviceNotGenuine:
                        SensorKinect = e.Sensor;
                        break;
                    case KinectStatus.Disconnected:
                        SensorKinect = null;
                        break;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void SensorKinectDepthFrameReady(object sender,
DepthImageFrameReadyEventArgs e)
        {
            try
            {
                using (var frame = e.OpenDepthImageFrame())
                {
                    if (frame != null)
                    {
                        frame.CopyPixelDataTo(_dadosPixelProfundidadeBruta);

_imagemProfundidadeBruta.WritePixels(_imagemProfundidadeBrutaRect,
```

```
_dadosPixelProfundidadeBruta, _imagemProfundidadePassadaBruta, 0);
//////////////////////////////////////////////////////////
                        GerarImagemProfundidade(frame, _dadosPixelProfundidadeBruta);
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void GerarImagemProfundidade(ImageFrame frame, IList<short> pixels)
        {
            try
            {
                const int byteProfundidadePorPixel = 4;
                var dadosPixels = new byte[frame.Width * frame.Height *
byteProfundidadePorPixel];
                for (int i = 0, j = 0; i < pixels.Count; i++, j +=
byteProfundidadePorPixel)
                {
                    var playerIndex = pixels[i] & DepthImageFrame.PlayerIndexBitmask;
                    if (playerIndex == 0)
                    {
                        dadosPixels[j] = 0xFF;
                        dadosPixels[j + 1] = 0xFF;
                        dadosPixels[j + 2] = 0xFF;
                    }
                    else
                    {
                        dadosPixels[j] = 0x00;
                        dadosPixels[j + 1] = 0x00;
                        dadosPixels[j + 2] = 0x00;
                    }
                }
                _imagemProfundidadeTratada.WritePixels(_imagemProfundidadeTratadaRect,
dadosPixels, _imagemProfundidadePassadaTratada, 0);
            }
            catch (Exception ex)
            {
                throw new Exception("Não foi possível renderizar a imagem do sensor de
profundidade!", ex);
            }
        }

        private void Slider_OnDragCompleted(object sender, DragCompletedEventArgs e)
        {
            try
            {
                _sensorKinect.ElevationAngle = Convert.ToInt32(Slider.Value);
                AnguloSensor.Content = _sensorKinect.ElevationAngle;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void ConfiguraLarguraOmbro()
```

```
        {

        }

        private double CapturaSegmentosHorizontais()
        {
            double distancia = 0;
            var nomeImagemSegmentoMb =
ArquivosManager.Criptografar(DateTime.Now.ToString(CultureInfo.InvariantCulture));
            var urlImagemSegmentoMb =
ProcessamentoImagem.GerarBitmap(ImagemSensorProfundidadeComSegmento,
                this, nomeImagemSegmentoMb);
            var coordenadasMb =
ProcessamentoImagem.LocalizarSegmentoMb(urlImagemSegmentoMb);
            var maiorSegmentoPixel =
ProcessamentoImagem.ProcurarMaiorSegmentoPixelsMb(coordenadasMb);
            var totalPixelsMb = maiorSegmentoPixel[1][1] - maiorSegmentoPixel[0][1];
            if (coordenadasMb != null && coordenadasMb.Count > 0)
            {
                ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Hidden;
                ImagemSensorProfundidadeSemSegmento.Visibility = Visibility.Visible;
                Thread.Sleep(1000);
                var nomeImagemSegmentoMp =
ArquivosManager.Criptografar(DateTime.Now.ToString(CultureInfo.InvariantCulture));
                var urlImagemSegmentoMp =
ProcessamentoImagem.GerarBitmap(ImagemSensorProfundidadeSemSegmento,
                    this, nomeImagemSegmentoMp);
                var teste = (KinectMath.AlturaTriangulo(totalPixelsMb)/2);
                MessageBox.Show(teste.ToString(CultureInfo.InvariantCulture));
                var linha = SelecionaLarguraOmbros.IsChecked == true ?
maiorSegmentoPixel.First()[0] - teste : maiorSegmentoPixel.First()[0];
                MessageBox.Show(linha.ToString(CultureInfo.InvariantCulture));
                var coordenadasMp =
ProcessamentoImagem.LocalizarSegmentoHorizontalMp(urlImagemSegmentoMp, linha);
                distancia = KinectMath.Proporcao(totalPixelsMb, coordenadasMp.Count,
_distancia);
                PreencherDetalhes(totalPixelsMb, coordenadasMp.Count, _distancia,
maiorSegmentoPixel[0],
                    maiorSegmentoPixel[1], coordenadasMp.First(),
coordenadasMp.Last());
                ArquivosManager.Excluir(urlImagemSegmentoMp);
            }
            ArquivosManager.Excluir(urlImagemSegmentoMb);
            ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Visible;
            ImagemSensorProfundidadeSemSegmento.Visibility = Visibility.Hidden;
            return distancia;
        }

        private double CapturaSegmentosVerticais()
        {
            double distancia = 0;
            var nomeImagemSegmentoMb =
ArquivosManager.Criptografar(DateTime.Now.ToString(CultureInfo.InvariantCulture));
            var urlImagemSegmentoMb =
ProcessamentoImagem.GerarBitmap(ImagemSensorProfundidadeComSegmento, this,
nomeImagemSegmentoMb);
            var coordenadasMb =
ProcessamentoImagem.LocalizarSegmentoMb(urlImagemSegmentoMb);
            var tamanhoMb = coordenadasMb.Last().Key - coordenadasMb.First().Key;
            if (coordenadasMb != null && coordenadasMb.Count > 0)
            {
                ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Hidden;
```

```
                ImagemSensorProfundidadeSemSegmento.Visibility = Visibility.Visible;
                Thread.Sleep(1000);
                var nomeImagemSegmentoMp =
ArquivosManager.Criptografar(DateTime.Now.ToString(CultureInfo.InvariantCulture));
                var urlImagemSegmentoMp =
ProcessamentoImagem.GerarBitmap(ImagemSensorProfundidadeSemSegmento, this,
nomeImagemSegmentoMp);
                var linha = coordenadasMb.First().Key;
                var coluna = coordenadasMb.First().Value[0];
                var coordenadasMp =
ProcessamentoImagem.LocalizarSegmentoVerticalMp(urlImagemSegmentoMp, coluna, linha);
                distancia = KinectMath.Proporcao(tamanhoMb, coordenadasMp.Count,
_distancia);
                PreencherDetalhes(tamanhoMb, coordenadasMp.Count, _distancia, new[] {
coordenadasMb.First().Key,
                    coordenadasMb.First().Value[0] }, new[] {
coordenadasMb.Last().Key, coordenadasMb.Last().Value[0] },
                    new[] { coordenadasMp.First()[1], coordenadasMp.First()[0] },
new[] { coordenadasMp.First()[1], coordenadasMp.First()[0] });
                ArquivosManager.Excluir(urlImagemSegmentoMp);
            }
            ArquivosManager.Excluir(urlImagemSegmentoMb);
            ImagemSensorProfundidadeComSegmento.Visibility = Visibility.Visible;
            ImagemSensorProfundidadeSemSegmento.Visibility = Visibility.Hidden;
            return distancia;
        }

        private void PreencherDetalhes(int totalMb, int totalMp, double tamanhoMb,
int[] coordenadaInicialMb, int[] coordenadaFinalMb, int[] coordenadaInicialMp, int[]
coordenadaFinalMp)
        {
            _detalhes.TamanhoMb = tamanhoMb;
            _detalhes.QuantidadePixelsMb = totalMb;
            _detalhes.QuantidadePixelsMp = totalMp;
            _detalhes.CoordenadaInicialMb = coordenadaInicialMb;
            _detalhes.CoordenadaFinalMb = coordenadaFinalMb;
            _detalhes.CoordenadaInicialMp = coordenadaInicialMp;
            _detalhes.CoordenadaFinalMp = coordenadaFinalMp;
            _detalhes.AlturaKinect = Math.Round(_alturaPeloKinect, 4);
        }

        private void BtnCapturar_OnClick(object sender, RoutedEventArgs e)
        {
            try
            {
                if (SelecionaEstaturaPaciente.IsChecked == false &&
SelecionaLarguraQuadril.IsChecked == false
                    && SelecionaLarguraOmbros.IsChecked == false)
                {
                    MessageBox.Show("É necessário selecionar um atributo
antropométrico para realizar o cálculo!",
                        @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Warning);
                }
                else
                {
                    if (SensorProfundidadeRadioButton.IsChecked == true)
                    {
                        if (SelecionaLarguraQuadril.IsChecked == true)
                        {
                            ResultadoLarguraQuadril.Text =
CapturaSegmentosHorizontais().ToString(CultureInfo.InvariantCulture);
```

```
                            BtnDetalhesCaptura.IsEnabled = true;
                        }
                        else if (SelecionaLarguraOmbros.IsChecked == true)
                        {
                            ResultadoLarguraOmbros.Text =
CapturaSegmentosHorizontais().ToString(CultureInfo.InvariantCulture);
                            BtnDetalhesCaptura.IsEnabled = true;
                        }
                        else
                        {
                            var alturaTotal = Math.Round(CapturaSegmentosVerticais() +
_alturaPeloKinect, 4);
                            ResultadoEstaturaPaciente.Text =
alturaTotal.ToString(CultureInfo.InvariantCulture);
                            BtnDetalhesCaptura.IsEnabled = true;
                        }
                    }
                    else
                    {
                        MessageBox.Show(@"Selecione a opção 'Sensor de profundidade'
para continuar!", @"Kinect Depth",
                            MessageBoxButton.OK, MessageBoxImage.Information);
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BtnDetalhesCaptura_OnClick(object sender, RoutedEventArgs e)
        {
            var detalhes = new WindowDetalhesCaptura(_detalhes);
            detalhes.ShowDialog();
        }

        private void SelecionaEstaturaPaciente_OnChecked(object sender,
RoutedEventArgs e)
        {
            BtnDetalhesCaptura.IsEnabled = ResultadoEstaturaPaciente.Text == null;
        }

        private void SelecionaLarguraOmbros_OnChecked(object sender, RoutedEventArgs
e)
        {
            BtnDetalhesCaptura.IsEnabled = ResultadoLarguraOmbros.Text == null;
        }

        private void SelecionaLarguraCintura_OnChecked(object sender, RoutedEventArgs
e)
        {
            BtnDetalhesCaptura.IsEnabled = ResultadoLarguraQuadril.Text == null;
        }

        private void BtnSalvarAvaliacao_OnClick(object sender, RoutedEventArgs e)
        {
            try
            {
                if (ResultadoEstaturaPaciente.Text != null &&
ResultadoLarguraOmbros.Text != null && ResultadoLarguraQuadril.Text != null)
```

```
                {
                    ManagerAntropometriaSilhueta.Cadastrar(new Antropometria_Silhueta
                    {
                        IdPaciente = _paciente.Id,
                        Estatura = double.Parse(ResultadoEstaturaPaciente.Text),
                        LarguraQuadril = double.Parse(ResultadoLarguraQuadril.Text),
                        LarguraOmbros = double.Parse(ResultadoLarguraOmbros.Text)
                    });
                    MessageBox.Show("Avaliação antropométrica salva com sucesso!",
@"Kinect Depth", MessageBoxButton.OK, MessageBoxImage.Information);
                }

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }
    }
}
```

APÊNDICE B – Código-fonte da tela de detalhes da avaliação

```
using System;
using System.Windows;
using KinectDepth.Controller;
using KinectDepth.Models;

namespace KinectDepth
{
    public partial class WindowDetalhesCaptura : Window
    {
        public WindowDetalhesCaptura(DetalhesCaptura detalhes)
        {
            InitializeComponent();
            CarregarDetalhes(detalhes);
        }

        public void CarregarDetalhes(DetalhesCaptura detalhes)
        {
            try
            {
                TotalPixelsMb.Content = detalhes.QuantidadePixelsMb;
                TamanhoSegmentoMb.Content = Math.Round(detalhes.TamanhoMb, 3) + "m";
                CoordenadaInicialMb.Content = "[" + detalhes.CoordenadaInicialMb[0] +
"," + detalhes.CoordenadaInicialMb[1] + "]";
                CoordenadaFinalMb.Content = "[" + detalhes.CoordenadaFinalMb[0] + ","
+ detalhes.CoordenadaFinalMb[1] + "]";

                TotalPixelsMp.Content = detalhes.QuantidadePixelsMp;
                TamanhoSegmentoMp.Content =
KinectMath.Proporcao(detalhes.QuantidadePixelsMb, detalhes.QuantidadePixelsMp,
detalhes.TamanhoMb) + "m";
                CoordenadaInicialMp.Content = "[" + detalhes.CoordenadaInicialMp[0] +
"," + detalhes.CoordenadaInicialMp[1] + "]";
                CoordenadaFinalMp.Content = "[" + detalhes.CoordenadaFinalMp[0] + ","
+ detalhes.CoordenadaFinalMp[1] + "]";

                if (detalhes.AlturaKinect > 0)
                {
                    TextoAlturaKinect.Visibility = Visibility.Visible;
                    AlturaKinect.Visibility = Visibility.Visible;
                    AlturaKinect.Content = detalhes.AlturaKinect;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, @"Kinect Depth", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }
    }
}
```

APÊNDICE C – Código-fonte da classe `ArquivoManager`

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

namespace KinectDepth.Controller
{
    public class ArquivosManager
    {
        public static string Criptografar(string texto)
        {
            try
            {
                var md5 = MD5.Create();
                var inputBytes = Encoding.ASCII.GetBytes(texto);
                var hash = md5.ComputeHash(inputBytes);
                var sb = new StringBuilder();
                foreach (var t in hash)
                {
                    sb.Append(t.ToString("X2"));
                }
                return sb.ToString().ToLower();
            }
            catch (Exception ex)
            {
                throw new Exception("Ocorreu um erro ao tentar criptograr o texto!",
ex);
            }
        }

        public static string PastaTemporaria(string nomeArquivo)
        {
            try
            {
                return Path.Combine(Path.GetTempPath(), nomeArquivo + ".png");
            }
            catch (Exception ex)
            {
                throw new Exception("Ocorreu um erro ao retornar a url da pasta
temporária!", ex);
            }
        }

        public static void Excluir(string arquivo)
        {
            try
            {
                if (File.Exists(arquivo))
                {
                    File.Delete(arquivo);
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Ocorreu um erro ao tentar excluir a imagem
temporária!", ex);
            }
        }
    }
}
```

APÊNDICE D – Código-fonte da classe `EsqueletoUsuario`

```
using Microsoft.Kinect;
using System;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Shapes;

namespace KinectDepth.Controller
{
    public class EsqueletoUsuario
    {
        static private KinectSensor _kinect;

        public EsqueletoUsuario(KinectSensor kinect)
        {
            _kinect = kinect;
        }

        public static ColorImagePoint ConverterCoordenadasArticulacao(Joint
articulacao, double larguraCanvas, double alturaCanvas)
        {
            var posicaoArticulacao =
_kinect.CoordinateMapper.MapSkeletonPointToColorPoint(articulacao.Position,
_kinect.ColorStream.Format);
            posicaoArticulacao.X = (int)(posicaoArticulacao.X * larguraCanvas) /
_kinect.ColorStream.FrameWidth;
            posicaoArticulacao.Y = (int)(posicaoArticulacao.Y * alturaCanvas) /
_kinect.ColorStream.FrameHeight;

            return posicaoArticulacao;
        }

        private Ellipse CriarComponenteVisualArticulacao(int diametroArticulacao, int
larguraDesenho, Brush corDesenho)
        {
            return new Ellipse
            {
                Height = diametroArticulacao,
                Width = diametroArticulacao,
                StrokeThickness = larguraDesenho,
                Stroke = corDesenho
            };
        }

        public void DesenharArticulacao(Joint articulacao, Canvas canvas, Brush
corDesenho)
        {
            const int larguraDesenho = 4;
            var objetoArticulacao = CriarComponenteVisualArticulacao(5,
larguraDesenho, corDesenho);
            var posicaoArticulacao = ConverterCoordenadasArticulacao(articulacao,
canvas.ActualWidth, canvas.ActualHeight);
            var deslocamentoHorizontal = posicaoArticulacao.X -
objetoArticulacao.Width / 2;
            var deslocamentoVertical = (posicaoArticulacao.Y -
objetoArticulacao.Height / 2);
            if (deslocamentoVertical >= 0 && deslocamentoVertical <
canvas.ActualHeight && deslocamentoHorizontal >= 0 && deslocamentoHorizontal <
canvas.ActualWidth)
            {
                Canvas.SetLeft(objetoArticulacao, deslocamentoHorizontal);
```

```
                Canvas.SetTop(objetoArticulacao, deslocamentoVertical);
                Panel.SetZIndex(objetoArticulacao, 100);
                canvas.Children.Add(objetoArticulacao);
            }
        }


        public void DesenharOsso(Joint articulacaoOrigem, Joint articulacaoDestino,
Canvas canvasParaDesenhar)
        {
            const int larguraDesenho = 4;
            var corDesenho = Brushes.Red;
            var posicaoArticulacaoOrigem =
ConverterCoordenadasArticulacao(articulacaoOrigem, canvasParaDesenhar.ActualWidth,
canvasParaDesenhar.ActualHeight);
            var posicaoArticulacaoDestino =
ConverterCoordenadasArticulacao(articulacaoDestino, canvasParaDesenhar.ActualWidth,
canvasParaDesenhar.ActualHeight);
            var objetoOsso = CriarComponenteVisualOsso(larguraDesenho, corDesenho,
posicaoArticulacaoOrigem.X, posicaoArticulacaoOrigem.Y, posicaoArticulacaoDestino.X,
posicaoArticulacaoDestino.Y);
            if (Math.Max(objetoOsso.X1, objetoOsso.X2) <
canvasParaDesenhar.ActualWidth && Math.Min(objetoOsso.X1, objetoOsso.X2) > 0 &&
Math.Max(objetoOsso.Y1, objetoOsso.Y2) < canvasParaDesenhar.ActualHeight &&
Math.Min(objetoOsso.Y1, objetoOsso.Y2) > 0)
            {
                canvasParaDesenhar.Children.Add(objetoOsso);
            }
        }

        public static Line CriarComponenteVisualOsso(int larguraDesenho, Brush
corDesenho, double origemX, double origemY, double destinoX, double destinoY)
        {
            return new Line
            {
                StrokeThickness = 5,
                Stroke = corDesenho,
                X1 = origemX,
                X2 = destinoX,
                Y1 = origemY,
                Y2 = destinoY
            };
        }
    }
}
```

APÊNDICE E – Código-fonte da classe `KinectMath`

```
using System;

namespace KinectDepth.Controller
{
    public class KinectMath
    {
        public static double DistanciaEntrePontos3D(double ax, double ay, double az,
double bx, double by, double bz)
        {
            return (Math.Sqrt(Math.Pow((bx - ax), 2) + Math.Pow((by - ay), 2) +
Math.Pow((bz - az), 2)));
        }

        public static double DistanciaEntreDoisPontos(double xa, double ya, double xb,
double yb)
        {
            return Math.Sqrt(Math.Pow((xb - xa), 2) + Math.Pow((yb - ya), 2));
        }

        public static double Proporcao(int totalMb, int totalMp, double distanciaMb)
        {
            return Math.Round(distanciaMb * totalMp / totalMb, 3);
        }
    }
}
```

APÊNDICE F – Código-fonte da classe `ProcessamentoImagem`

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media.Imaging;

namespace KinectDepth.Controller
{
    public class ProcessamentoImagem
    {
        public static Dictionary<int, List<int>> LocalizarSegmentoMb(string url)
        {
            try
            {
                var coordenadasSegmento = new Dictionary<int, List<int>>();
                var valoresY = new List<int>();
                var imagem = new Bitmap(url);
                for (var x = 0; x < imagem.Height; x++)
                {
                    for (var y = 0; y < imagem.Width; y++)
                    {
                        if (imagem.GetPixel(y, x).R > 0 && imagem.GetPixel(y, x).G ==
0 && imagem.GetPixel(y, x).B == 0)
                        {
                            valoresY.Add(y);
                        }
                    }
                    if (valoresY.Count > 0)
                    {
                        coordenadasSegmento.Add(x, new List<int>(valoresY));
                        valoresY.Clear();
                    }
                }
                return coordenadasSegmento;
            }
            catch (Exception ex)
            {
                throw new Exception("Ocorreu um erro ao tentar localizar o SEGMENTO Mb
na imagem!", ex);
            }
        }

        public static List<int[]> LocalizarSegmentoHorizontalMp(string url, int?
valorInicialX)
        {
            try
            {
                var imagem = new Bitmap(url);
                var coordenadasSegmento = new List<int[]>();
                if (valorInicialX == null) return coordenadasSegmento;
                for (var y = 0; y < imagem.Width; y++)
                {
                    if (imagem.GetPixel(y, valorInicialX.Value).R == 0 &&
imagem.GetPixel(y, valorInicialX.Value).G == 0
                        && imagem.GetPixel(y, valorInicialX.Value).B == 0)
                    {
                        coordenadasSegmento.Add(new[]
```

```
                    {
                        valorInicialX.Value, y
                    });
                }
            }
            return coordenadasSegmento;
        }
        catch (Exception ex)
        {
            throw new Exception("Ocorreu um erro ao tentar localizar o SEGMENTO
HORIZONTAL Mp na imagem!", ex);
        }
    }

    public static List<int[]> LocalizarSegmentoVerticalMp(string url, int? coluna,
int? linha)
    {
        try
        {
            var imagem = new Bitmap(url);
            var coordenadasSegmento = new List<int[]>();
            if (coluna == null || linha == null) return coordenadasSegmento;
            for (var x = 0; x < imagem.Width; x++)
            {
                if (x > linha.Value)
                {
                    break;
                }
                if (imagem.GetPixel(coluna.Value, x).R == 0 &&
imagem.GetPixel(coluna.Value, x).G == 0
                    && imagem.GetPixel(coluna.Value, x).B == 0)
                {
                    coordenadasSegmento.Add(new[]
                    {
                       coluna.Value, x
                    });
                }
            }
            return coordenadasSegmento;
        }
        catch (Exception ex)
        {
            throw new Exception("Ocorreu um erro ao tentar localizar o SEGMENTO
VERTICAL Mp na imagem!", ex);
        }
    }

    public static int ContarPixels(string url)
    {
        try
        {
            var imagem = new Bitmap(url);
            var contador = 0;
            for (var x = 0; x < imagem.Height; x++)
            {
                for (var y = 0; y < imagem.Width; y++)
                {
                    if (imagem.GetPixel(y, x) == BrushPadraoSegmentoMb())
                    {
                        contador++;
                    }
                }
```

```
                    if (contador > 0)
                    {
                        break;
                    }
                }
                return contador;
            }
            catch (Exception ex)
            {
                throw new Exception("Ocorreu um erro ao tentar contar os píxels da
imagem!", ex);
            }
        }

        public static Color BrushPadraoSegmentoMb()
        {
            try
            {
                return new Pen(Brushes.Red).Color;
            }
            catch (Exception ex)
            {
                throw new Exception("Ocorreu um erro ao converter um Brush para um
Color!", ex);
            }
        }

        public static Color BrushPadraoSegmentoMp()
        {
            try
            {
                return new Pen(Brushes.Black).Color;
            }
            catch (Exception ex)
            {
                throw new Exception("Ocorreu um erro ao converter um Brush para um
Color!", ex);
            }
        }

        public static List<int[]> ProcurarMaiorSegmentoPixelsMb(Dictionary<int,
List<int>> coordenadas)
        {
            int[] maiorValor = { int.MinValue };
            var x = 0;
            var listaMaior = new List<int>();
            foreach (var valor in coordenadas.Where(valor => valor.Value.Count >
maiorValor[0]))
            {
                maiorValor[0] = valor.Value.Count;
                x = valor.Key;
                listaMaior = valor.Value;
            }
            return new List<int[]>
            {
                new[] {x, listaMaior.Min()},
                new[] {x, listaMaior.Max()}
            };
        }

        public static string GerarBitmap(Canvas canvas, Window window, string
nomeArquivo)
```

```
        {
            try
            {
                canvas.Measure(new System.Windows.Size(window.Width, window.Height));
                var renderizar = new RenderTargetBitmap(597, 561, 96, 96,
System.Windows.Media.PixelFormats.Pbgra32);
                renderizar.Render(canvas);
                var urlImagem = ArquivosManager.PastaTemporaria(nomeArquivo);
                var codificar = new PngBitmapEncoder();
                codificar.Frames.Add(BitmapFrame.Create(renderizar));
                using (var arquivo = File.Create(urlImagem))
                {
                    codificar.Save(arquivo);
                    arquivo.Close();
                }
                return urlImagem;
            }
            catch (Exception ex)
            {
                throw new Exception("Ocorreu um erro salvar a imagem para realização
do cálculo!", ex);
            }
        }
    }
}
```