Kaunas University of Technology

Faculty of Electrical and Electronics Engineering

Research of 3D Human Body Parts Measurement Precision Using Kinect Sensor

Master's Final Degree Project

Tony AlBitar

Project author

Lect. Kęstas Rimkus

Supervisor

Kaunas, 2021

Kaunas University of Technology

Faculty of Electrical and Electronics Engineering

Research of 3D Human Body Parts Measurement Precision Using Kinect Sensor

Master's Final Degree Project

Control Technologies (6211EX014)

Tony AlBitar

Project author

Lect. Kęstas Rimkus

Supervisor

Prof. Rimvydas Simutis

Reviewer

Kaunas, 2021

Kaunas University of Technology

Faculty of Electrical and Electronics Engineering

Tony AlBitar

Research of 3D Human Body Parts Measurement Precision Using Kinect Sensor

Declaration of Academic Integrity

I confirm the following:

1. I have prepared the final degree project independently and honestly without any violations of the copyrights or other rights of others, following the provisions of the Law on Copyrights and Related Rights of the Republic of Lithuania, the Regulations on the Management and Transfer of Intellectual Property of Kaunas University of Technology (hereinafter – University) and the ethical requirements stipulated by the Code of Academic Ethics of the University;
2. All the data and research results provided in the final degree project are correct and obtained legally; none of the parts of this project are plagiarised from any printed or electronic sources; all the quotations and references provided in the text of the final degree project are indicated in the list of references;
3. I have not paid anyone any monetary funds for the final degree project or the parts thereof unless required by the law;
4. I understand that in the case of any discovery of the fact of dishonesty or violation of any rights of others, the academic penalties will be imposed on me under the procedure applied at the University; I will be expelled from the University and my final degree project can be submitted to the Office of the Ombudsperson for Academic Ethics and Procedures in the examination of a possible violation of academic ethics.

Tony AlBitar

Confirmed electronically

AlBitar, Tony. Research of 3D Human Body Parts Measurement Precision Using Kinect Sensor. Master's Final Degree Project / supervisor / lect. dr. Kęstas Rimkus; Faculty of Electrical and Electronics Engineering, Kaunas University of Technology.

Study field and area (study field group): Electronics engineering (engineering science).

Keywords: Kinect sensor, augmented reality, human body measurement, virtual dressing application.

Kaunas, 2021. Number of pages 47 p.

Summary

Non-contact human body parts measurement plays a vital role in augmented reality, virtual fitting, and physical healthcare applications. Existing Kinect-based body measurement methods encounter many challenges handling humans wearing clothes or are time-consuming since their objective also includes 3D scanning and reconstruction. This research project introduces a viable and effective Kinect-based method to generate 3D human body measurements of human wearing clothes. The proposed approach involves capturing frames of a subject from a single view and constructing a denoised 3D point cloud, which allows the acquisition of ten distinct measurements in total. Extensive experimental results demonstrated that the developed method requires a short execution time of approximately 20 seconds while producing accurate measurements with comparable quality. Ultimately, this research offers a convenient approach for designing and improving a real-time 3D virtual dressing application.

AlBitar, Tony. Trimačio žmogaus kūno dalių matavimo Kinect jutikliu tikslumo tyrimas. Magistro baigiamasis projektas / vadovas lekt. dr. Kęstas Rimkus; Kauno technologijos universitetas, Elektros ir elektronikos inžinerijos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Elektronikos inžinerija (inžinerijos mokslai).

Reikšminiai žodžiai: Kinect jutikis, pridėtinė realybė, žmogaus kūno matavimas, virtualaus apsirengimo programėlės.

Kaunas, 2021. Puslapių sk. 47 p.

Santrauka

Bekontaktis žmogaus kūno dalių matavimas vaidina svarbų vaidmenį papildytoje realybėje, virtualiuose matavimuose ir fizinės sveikatos priežiūros programose. Esami kūno matavimo metodai paremti „Kinect“ tipo jutikiais susiduria su daugybe problemų susijusių su: žmonių dėvimų drabužių įvairove, arba užima daug laiko, nes jų matavimai apima 3D nuskaitymą bei rekonstrukciją. Šiame tiriamajame projekte pristatomas perspektyvus ir efektyvus „Kinect“ tipo jutikliams pritaikytas metodas trimačių žmogaus matmenų gavimui, nepaisant dėvimų rūbų. Siūlomame metode objektas fotografuojamas iš vienos pozicijos ir konstruoja išvalytą 3D taškų debesį, kuris leidžia iš viso gauti dešimt skirtingų matavimų. Išsamūs eksperimentiniai rezultatai parodė, kad sukurtam metodui reikalingas trumpas, maždaug 20 sekundžių vykdymo laikas, tuo pat metu gaunami tikslūs matavimai su nedideliu išsibarstymu. Galų gale, šis tyrimas siūlo metodą puikiai tinkantį kuriant ir tobulinant realaus laiko 3D virtualios aprangos programas.

Table of contents

List of algorithms

List of figures

List of tables

List of abbreviations and terms

Abbreviations:

Lect. – lecturer

RGB-D – red green blue-depth;

2D – two-dimensional;

3D – three-dimensional;

RGB – red green blue;

SCAPE – shape completion and animation of people;

DoF – degree of freedom;

LBS – linear blending skinning;

API – application programming interface;

SDK – software development kit;

ICP – iterative closest point;

PCL – point cloud library;

AEE – average euclidean error;

ToF – time-of-flight;

AAE – average absolute error;

Woc – without compensation;

Wc – with compensation;

Introduction

Recent theoretical developments have revealed that augmented reality and virtual dressing technologies have advanced tremendously and expanded to several areas such as health care, marketing, gaming, and others. Contactless human body measurement plays a significant role in clothing and virtual fitting applications [1] as online shopping has become a trend.

In this context, the use of low-cost RGB-D sensors such as Microsoft Kinect, introduced by Microsoft for Xbox games, in 3D human body measurement and scanning has led to a significant increase in its convenience. Meanwhile, existing laser-based [2] and structured light sensors are either sophisticated or expensive as they require expert knowledge for operation, even though they produce high-quality data. Due to its many capabilities, Microsoft Kinect has particular advantages over conventional 3D scanners. One of the primary benefits is its ability to capture color and depth data at a video rate with a minimum consideration of the texture and light condition. These advantages allowed the Kinect technology to advance in areas such as defense [3], sports [4], virtual dressing applications [5, 6, 7, 8], and health care [9, 10, 11, 40].

With the increasing importance of the human body measurement data in online shopping applications, researchers focused on finding new methods for obtaining the measurement without using the traditional measuring techniques that rely on the measuring tape. Therefore, based on Kinect's RGB-D information, there have been numerous studies on measuring human body parts to accomplish effectiveness and cost-saving. At the same time, a subject maintains a steady position and using fewer Kinect sensors [7, 12, 13]. Other researchers' attention has been devoted to achieving measurements using Kinect sensor for health reasons, such as in [14] that developed a method to estimate waist circumference to help diagnose abdominal obesity.

Additionally, several researchers endeavored to improve the measured human body parts' accuracy regardless of the large-scale motion during data capturing [15, 16]. Some authors have also conducted studies to develop software-based methods to obtain a significant number of measurements regardless of the human clothes' looseness [17]. Nevertheless, such approaches require multiple operators, a higher computational time, and an increasingly sophisticated algorithm.

This research project presents a fast, easy, and efficient method for obtaining 3D human body parts measurements by detecting the human body skeleton using a single Kinect v2 sensor based on the time-of-flight principle [18] for real-time 3D virtual dressing application. At first, the Kinect sensor was used to capture depth, color, and skeleton data of users maintaining a "T" pose. Afterward, based on the captured segmented depth data and the generated 3D point clouds, the system acquired personalized human body parameters such as height, arm length, and waist front end perimeter with the aid of the 25 joints points identified by Kinect v2. Ultimately, the resulting measurement errors were reduced by implementing an error compensation technique. Furthermore, this thesis provides several suggestions for future research to address and consider to overcome the raised challenges and improve the achieved results.

In brief, this thesis documents several key contributions presented as follows:

- Adopting a fast and easy data capturing approach to scan the human body that requires no second operator or any complicated setup. In other words, users can configure the frames' capturing scenario using their personal computer to start and finish the 3D human body parts measurement process by themselves, with embedded voice commands to assist them.
- Offering an efficient 3D body parts measurement method that generates up to ten different measurements from the captured data in less than ten seconds using any RGB-D sensor that provides at least 15 skeleton points.
- Applying an error compensation strategy to minimize the acquired body parts measurement errors. One of the key benefits of this algorithm is the ability to be implemented in public environments to achieve realistic 3D virtual fitting applications. To put it another way, modifying the reconstructed 3D avatars of a subject to match the generated body parts length.

1. Related work

This chapter discusses and compares several proposed 3D human body parts measurement methods with a single Kinect sensor to avoid the complicated system setup. In addition, this chapter provides a short comparison about RGB-D sensors' efficiency from the perspective of depth accuracy, captured image quality, device performance, and configuration to present useful information about the chosen RGB-D sensor in this research project.

1.1. Measurements with large scale motion method

Xu *et al.* [15] presented an efficient approach to obtaining accurate non-contact human body parts measurements under the influence of a large-scale motion using a single Microsoft Kinect sensor. Their method considers the motion effect on the human clothes, which may drive the attached clothes to the human body tightly or loosely. Therefore, they introduced a space-time evaluation to extract the information throughout different posture variations, allowing them to obtain accurate human body parameters and thus made their proposed system feasible to be applied in public environments.

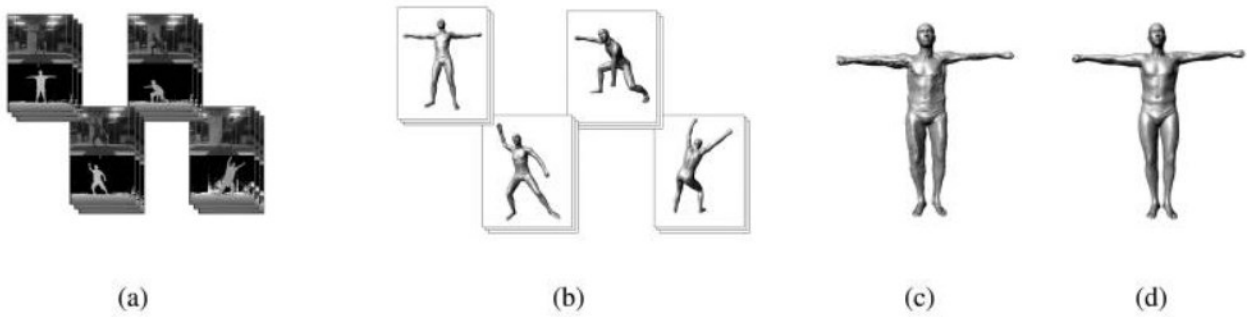


Fig. 1. The pipeline of the presented approach: (a) the input depth maps and RGB images from Kinect; (b) various poses for every frame; (c) the spatial-temporal average model of all frames; (d) final model after reduction of the clothes effect [15]

Fig. 1 displays a summary of the proposed method. Firstly, a video sequence of subjects behaving in various poses was acquired using a single Kinect sensor (**Fig. 1**(a)). Then, a multi-layer framework restored each frame's pose in the sequence (**Fig. 1**(b)), which consisted of a failure-detection and a pose tracking module. Initially, for pose detection, a SCAPE model was applied to generate the training database, including body shapes of 300 different individuals and around 50,000 different poses. In further steps, the authors employed the SCAPE model for modeling the sophisticated non-rigid deformations of human bodies produced by both the shape and pose variations. In other words, they approximated the human body geometry by 16 different rigid parts with 36 degrees of freedom to identify the subjects' different poses.

For pose tracking, the authors implemented the proposed method by Wei *et al.* [19] while considering the RGB images' silhouette constraint, which ensured a precise pose estimation for each frame. Accordingly, the 3D human models were estimated based on the depth maps for distinct poses, allowing to convert all the obtained poses into a standard pose to eventually reconstruct a spatial-temporal average model (**Fig. 1**(c)). Additionally, the application of a space-time evaluation allowed reducing the clothes' effect (**Fig. 1**(d)). **Fig. 2** displays a comparison between the generated average model's accuracy and the spatial-average model after mitigating the clothes effect.

Fig. 2. Results for the reduction of the clothes effect: (a) a person wearing tight clothes; (b) a person wearing loose clothes [15]

Moreover, to acquire the body parts measurements, each frame's SCAPE model was initially optimized using the depth map of the video sequence's first five frames. All the obtained models of different poses were then transformed into one standard T-pose 3D model using the inverse LBS model. Specifically, synthesizing the 2.5D information from several distinct views along the time axis allowed recovering the complete 3D model information. **Fig. 3** shows all the defined human body parameters in the developed system. The indices of two points for each body part were defined based on the body bone segment to determine the leg, arm, and neck to hip length. Also, the indices of multiple points surrounding the corresponding body part locations were primarily specified to measure the waist, chest, and hip circumference. Subsequently, the system reconstructed a circular convex structure from the predefined points (**Fig. 3**(b)), allowing them to generate measurements of the corresponding body parts parameters automatically during runtime.

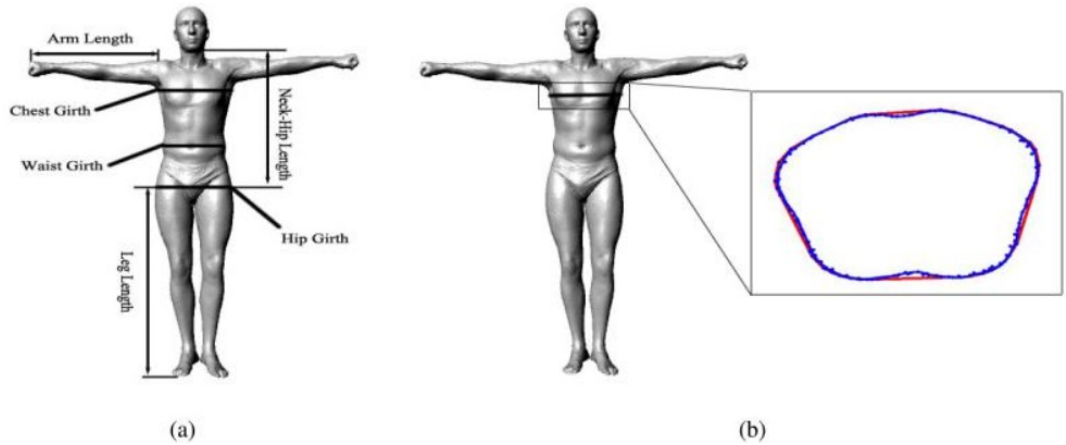


Fig. 3. Measurements: (a) the human body parameters defined in the presented system; (b) method for measuring the circumference parameters using the girth [15]

Based on the developed method to acquire the human body measurements, experimental data were collected and maintained using 55 video sequences belonging to 25 men and ten women while assessing several subjects more than once. The tested individuals were between 20 and 45 years old, ranging from 1.55m to 1.90m.

Fig. 4 presents the 3D model reconstruction results using the color and depth images obtained from Kinect. The 3D models were captured with the motion capture module with different poses and were following this transformed into T-pose. Experimental results confirmed that without using the failed pose tracking method for pose recovery, the pose estimation module's inaccuracy adversely affected the acquired body parts measurements to no small degree. Mainly failed poses during motion capture often increase the outliers, which leads to an increment in the mean error of the body measurements. Accordingly, the implementation of a failure detection module enabled the detection of the unsuccessful poses automatically. **Table 1** displays the average running time for the developed system achieved using a dual-core 2.33 GHz Intel processor.

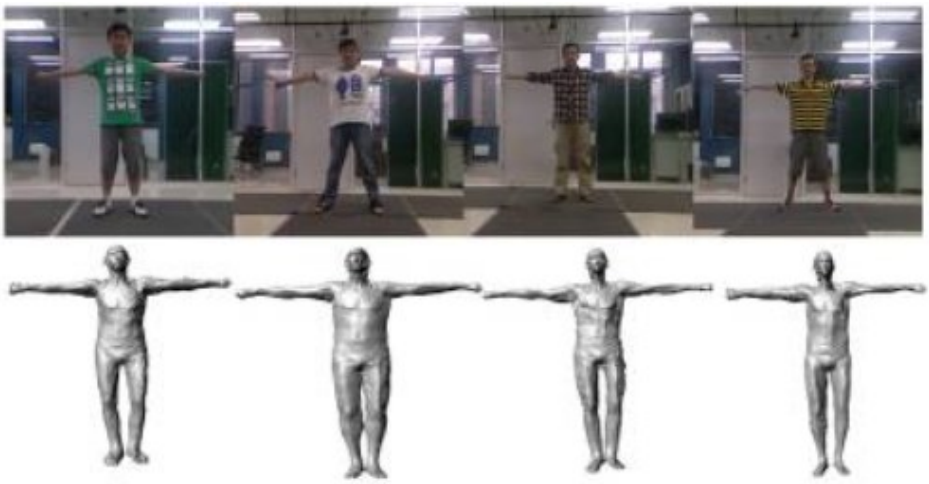


Fig. 4. Results of 3D model reconstructions [15]

Table 1. The average computational time for the proposed system [15]

Procedure	Time Consuming
Pose Recovery	6.32 s per frame
Shape Parameter Recovery	55.2 s
Weighed-Average Model Recovery	5.45 s per frame
Mitigation of the Effect of Clothes	7.94 s per frame

Furthermore, the authors' compared their developed system against the approach presented in [20]. Cui *et al.* [20] method can reconstruct all the human bodies' details while considering the subjects' small-scale motion while capturing approximately 40 frames from each view of the selected 12 views. **Fig. 5** displays a comparison between the calculated average absolute errors of measurements reported in [15] and those reported in [20]. The derived results demonstrated Xu et al. [15] system's effectiveness in measuring the human body parameters with large-scale motion. Meanwhile, the average error of neck-to-hip length was slightly higher than that in [20]. Such difference arises from the extensive human body motion in front of the Kinect sensor during capturing, while in [20], the experimental subjects experienced minimal movement.

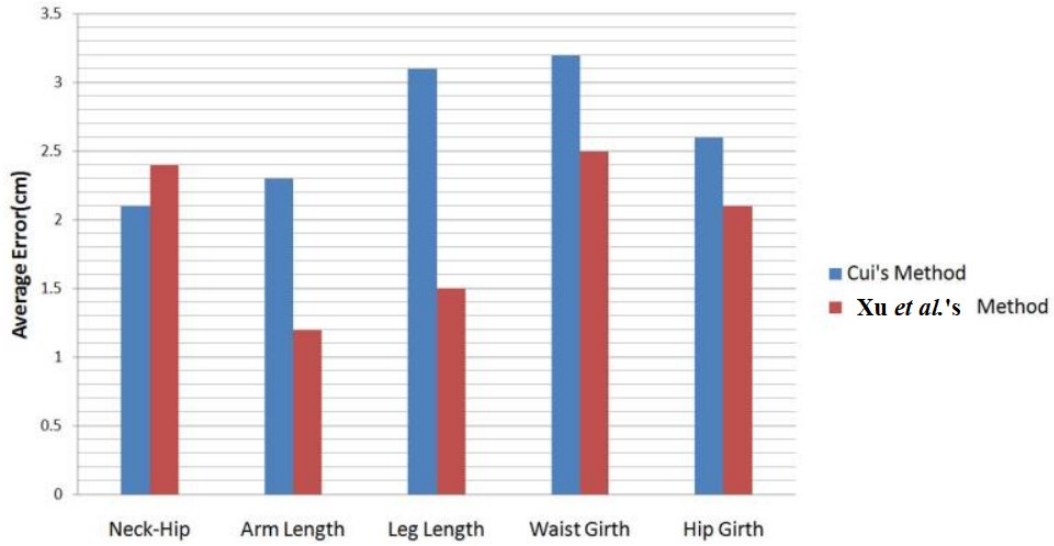


Fig. 5. Average absolute errors of measurements of people wearing clothes [15]

Additionally, the reported findings involved calculating the waist and chest circumference's relative errors to analyze the obtained results before and after mitigating the clothes effect. Accordingly, the use of tight clothes produced minor errors compared to those measured with higher looseness.

On this basis, the developed method by Xu *et al.* [15] can accurately acquire human body measurements of humans with large-scale motion. In other words, the conducted experiment demonstrates the applicability of such a system in several public applications, such as hospitals and shopping malls, with a relatively small price of approximately 150$. Ultimately, quantitative evaluations of the achieved results demonstrated that their approach produced more precise measurements than similar works.

1.2. Measurements with steady human position method

Adhikari *et al.* [12] presented a method to obtain contactless human body parts measurements dependent on a single Kinect v2 sensor for 3D virtual dressing applications. **Fig. 6** shows an outline of the proposed procedure. Initially, the Kinect v2 device captured the human body of the subject, maintaining a steady T-pose. Then, the authors determined the corresponding human body depth information based on the acquired frames.

Afterward, the implementation of several image processing techniques available in the Kinect sensor [21] allowed segmenting the moving objects and minimizing the background noise. The developed system could then acquire personalized human body parameters such as height, neck to hip, and leg length using the sensor's skeleton joints' positions. Similarly, the chest, waist, and stomach's front-end perimeters were estimated using the corresponding 3D pixels.

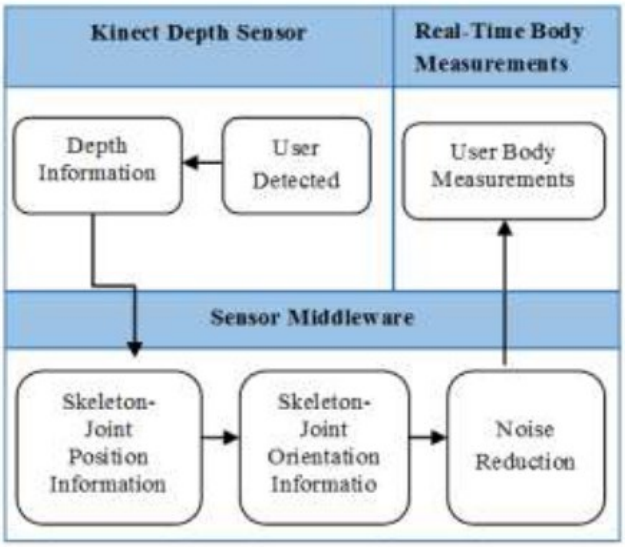


Fig. 6. The pipeline of the presented method [12]

Although the Kinect sensor can store skeleton points of six people simultaneously, the authors aimed initially to determine the following measurements of a single subject at once by employing Pythagoras theorem for 3D space to measure the distance between the defined skeleton points using a C# code.

- Height of the user: measured as the distance between head and ankle skeleton points.
- Shoulder length: distance from shoulder left to shoulder right points.
- Leg length: spine-base to ankle points.
- Neck to hip length: neck to spine-base points.
- Arm length: shoulder to wrist points.
- Chest, stomach, and waist circumference: calculated as the average front-end perimeter around each corresponding body part area accordingly. Note that the skeleton points have not been considered for these measurements as new depth Y coordinates were defined. The corresponding depth pixels were then converted into real-world camera space points in meters through Kinect SDK API, which allowed them to generate measurements in meters.

Moreover, the skeleton model and the necessary body parameters were registered for the 3D model as labels, as shown in **Fig. 7**. Following this, the developed system reconstructed the corresponding 3D human body model based on the determined measurements and implemented error calibration for all of them. In other words, the algorithm generated all the necessary body parameters for two users and then calibrated the obtained results based on the calculated average error percentage as follows:

$$Calculated\ value\ +\ (Calculated\ value\ \times\ Error\ percentage)$$

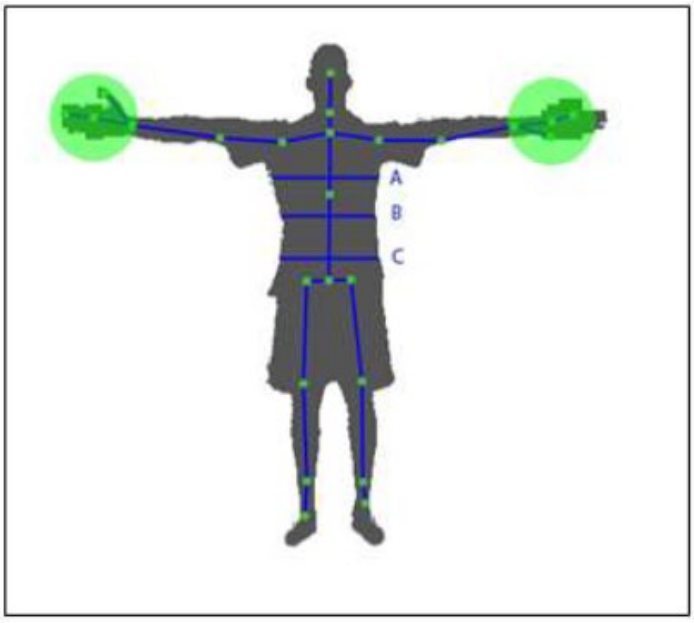


Fig. 7. Defined skeleton model and body parameters points [12]

Table 2 displays a comparison between the acquired measurements using the Kinect sensor and one test subject's manual measurements determined with a measuring tape's assistance. According to the experimental results, the sensor-based measured parameters were relatively accurate since the calculated errors were less than 5%. The results for the chest, stomach, and waist front-end perimeters incorporated a maximum error of 12%. Further steps consisted of performing satistical analyses by applying the average weighted sampling technique to minimize each parameter's errors. The achieved experimental results were adequate and provided an appropriate approach for designing a 3D virtual dressing application.

Ultimately, the authors utilized Microsoft visual gesture builder to introduce a gesture controlling and identification system. A user interface was also designed with three control buttons for recording, resetting, and saving images of the user captured by the Kinect sensor. Subsequently, the developed interface displayed the user's skeleton image with all the corresponding body parts measurements based on the collected human body frames.

Table 2. Experimental body parts measurements [12]

Measurement	Manually Measured Value (cm)	Sensor Measured Value (cm) Eq.(2)	Error $\left[\frac{\lvert\Delta x\rvert}{x} \times 100\%\right]$
Height of the user	185.9	186.0	0.05
Shoulder length	45.6	47.2	3.60
Neck to hip length	78.0	79.4	4.82
Hip to leg length	79.9	77.5	3.00
Arm length	55.9	53.2	4.80
Chest front end perimeter	51.0	56.9	11.60
Stomach front end perimeter	50.5	55.6	10.20
Waist front end perimeter	43.2	47.9	10.97

$|\Delta x|$ = Absolute value of [Sensor Measured - Manually Measured]

1.3. Measurements with multi-view alignment method

Mao *et al.* [22] proposed a method to obtain the human body parts measurement using the skeleton points provided by the Kinect v1 sensor. At first, they reconstructed a full 3D human body by capturing 18 frames from 6 views. The subsequent step consisted of applying an alignment algorithm to rapidly realize a globally aligned point cloud of the complete human body's captured frames. **Fig. 8** presents an outline of the reported approach, in which the processing time for each model was about 10 minutes on average.

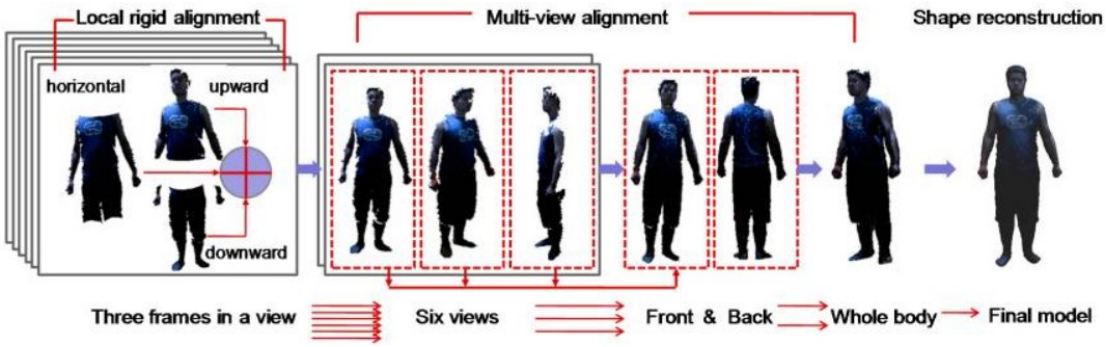


Fig. 8. Outline of the proposed method setup [22]

To scan the human body, the user had to maintain a steady position within 1m of the sensor. Meanwhile, the system's setup required adjusting the Kinect sensor to the subject's waist height. Then, the user had to rotate consistently at approximately 60 degrees to capture three data frames in six different views, presented as follows:

- one frame horizontal to the Microsoft Kinect.
- another one 20 degrees above the horizontal by rotating the motor controlling the Kinect's tilt-in base.
- last frame captured 20 degrees below the horizontal.

Compared to the previous methods, the main advantage was the reduced number of captured frames, which allowed to minimize and enhance the required computation speed for the denoising and alignment processes. Additionally, the developed algorithm consisted of a local rigid alignment procedure to merge every three frames of one view. This system then reconstructed two final views representing the human body's front and back sides from the obtained views before ultimately combining them to generate the corresponding full human body model.

The following step was the segmentation and denoising of the calibrated RGB-D images because the captured data frames contained both the human body and the enclosing environment in the scanning range. For the segmentation process, the authors defined the bottom of the human body by cutting a plane. They thus provided a threshold to the depth value for dividing the pixels as the scanned human body was close to the sensor. As for the lateral noise, the implementation of Barron and Malik's [23] proposed method allowed to smooth the noise resulting from the surface changes, lighting environment, and other factors during data capturing. A final illustration of the human body segmentation and denoising is shown in **Fig. 9**, with the segmented body's edges highlighted to demonstrate the noise reduction.

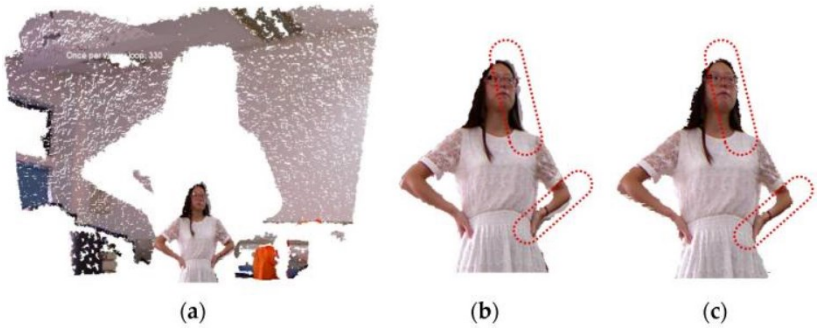


Fig. 9. Data frames segmentation and denoising: (a) raw data, (b) after segmentation and removing the background, (c) after denoising [22]

Following the segmentation and denoising procedure, the authors implemented a local rigid alignment technique combined with the ICP algorithm point-to-point search method to merge every three frames of a single view. In general, the ICP algorithm calculates the correspondence distance between the source and destination points. However, since the point cloud density increased after merging the frames, a down-sampling process was conducted using the PCL voxel grid. The following produced a point cloud divided into multiple grid elements, with each grid's total points approximated with their corresponding centroid.

After rigidly aligning all the captured frames, the researchers introduced a three-step multi-view alignment method to combine the obtained six views into two views consisting of:

- A pre-alignment of the frames implemented by finding their pairwise correlation on the human body silhouette. This process was repeated five times on both the front and back views considering that they have more data points.
- A rigid alignment of the previously aligned partial views in a group of two-point clouds in an adjacent position to acquire a more detailed point cloud, as illustrated in **Fig. 10**.

Fig. 10. Identified correspondence points between two frames in adjacent views [22]

- A non-rigid alignment of the multi-view frames to cope with the accumulation of errors, which originates from multiple rigid alignment iterations, by distributing them equally over all the consecutive frames. This approach was required because, during the scanning process, the human body can unavoidably experience movement. The following may cause deformations that can be similarly produced by the human clothes or during the Kinect sensor calibration. Ultimately, the presented method enabled the merging of the adjacent point clouds views progressively to form one final view of the human body.

Furthermore, the resulting globally aligned human body model required texture mapping and a watertight mesh surface. In terms of the watertight mesh surface, the developed algorithm applied the Poisson reconstruction method [24], which generates polygonal mesh from the dense point cloud. Regarding texture mapping, a mesh of the 3D human portrait was first developed by segmenting and projecting the 3D surface to the 2D domain. Afterward, the Kinect's sensor calibration of RGB images and depth images allowed to assign all the captured frames with colors before aligning them without down-sampling by the reported global and rigid alignment algorithms. It is important to note that obtaining the texture mapping of any point in the 3D mesh relied on the weighted mean of the generated point cloud data's color values. **Fig. 11** shows the 3D reconstruction results, which contained the 3D reconstructed human body avatar with mesh, color, and texture. The achieved results demonstrated the accuracy of the proposed method in reconstructing 3D avatars with realistic deformation, clothing wrinkles, and hairstyles.

Fig. 11. Reconstruction results of full 3D human bodies in different scanning scenarios [22]

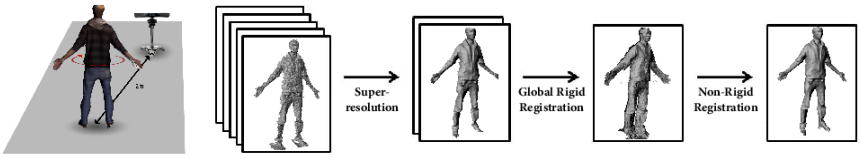
Based on the obtained 3D models from the multi-view alignment method, the researchers used skeleton points provided by the Kinect sensor to generate several 3D human body parts measurements for multiple subjects. **Table 3** shows the average error between the calculated and the actual human body measurements presented in centimeters while comparing them to previous work with similar results using two Kinects [25, 26] and others using one Kinect [20, 27]. Contrary to these previous works' findings, Mao *et al.*'s approach slightly improved the average error between measurements while generating more precise 3D human body models. Nevertheless, this method had particular limitations in terms of the captured data quality and modeling robustness. Accordingly, the authors suggested several recommendations for future research, such as utilizing higher resolution RGB-D sensors and more advanced denoising algorithms.

Table 3. The average error of measurements between the real and virtual human bodies [22]

	Height Length (cm)	Neck to Hip Length (cm)	Shoulder Length (cm)	Waist Girth (cm)	Hip Girth (cm)	Arm Length (cm)	Leg Length (cm)
In [26] with two Kinect	1.0	2.4	1.96	6.5	4.0	3.2	2.2
In [25] with two Kinect		2.5	1.5	6.2	3.8	3.0	2.1
In [20] with one Kinect		2.1	1.0	3.2	2.6	2.3	3.1
In [27] with one Kinect		3.7			3.3	1.4	
Mao *et al.*'s with one Kinect	0.5	2.0	1.2	3.1	2.2	1.4	1.4

1.4. Measurements with the registration of many depth images method

Cui *et al.* [20] proposed a full 3D human body measurements and reconstruction method that focuses on the rigid and non-rigid registration of many depth images and does not require a reference model or additional equipment. The following allowed them to reconstruct detailed human body features such as faces or clothing by applying a super-resolution algorithm that considers the color constraints. **Fig. 12** presents an outline of the reported approach, in which the processing time for each model was about 14 minutes on average.



(a) Working setup (b) Processing pipeline

Fig. 12. Outline of the proposed method setup [20]

To scan the human body, the user had to stand in a range of 2 meters from the Kinect's v1 sensor while maintaining a "T" pose and turning around 360 degrees for about 20 to 30 seconds to capture ten frames within 0.5 seconds intervals. Afterward, based on the smoothing super-resolution algorithm previously described in [28], super-resolved depth and color frames were produced with an enhanced resolution. **Fig. 13** shows the developed algorithm's accuracy in preserving the captured model frames' smooth surface after integrating the color constraints. Meanwhile, it is essential to highlight that the AEE, which is the average of error values over all the points in the mesh, is represented by the color-coded plots (**Fig. 13**(c) and **Fig. 13**(e)). Extensive analysis of the presented AEE results in **Fig. 13** demonstrates the added color constraints' effectiveness in decreasing the euclidean errors.

(a) Raw data (b) Off color (c) Off color (AEE = 4.84) (d) With color (e) With color (AEE = 2.98)

Fig. 13. Super-resolved mesh of a lion model [20]

Further steps consisted of a global rigid and non-rigid registration of the noisy Kinect data using a probabilistic global alignment algorithm, which combined the super-resolved scans into a final model by aligning all the adjacent frames. For this purpose, the researchers considered the human body as articulated with rigid structures connected by joints. Note that the articulated model was employed to describe the non-rigid motion of the human body. Subsequently, the presented non-rigid algorithm accomplished correct point cloud registration and detection of joint positions when tested against frames with waving arms.

Moreover, the researchers utilized the Poisson surface reconstruction algorithm [24] to generate 3D human body mesh with minimized noise. The final step included a texture depth map for each view of the human body created from the Kinect's raw color data. **Fig. 14** displays the reconstructed 3D human body models of 5 experimental subjects with accurate geometric surface features. Nonetheless, one concern about this paper's findings was that excessive motion in the user's arms or legs could distort the developed system's registration process. One of the reported suggestions to solve this problem involved investigating more complex noise and distorted models to handle extensive user movements.

Fig. 14. Full 3D human body scanning results of multiple subjects [20]

Based on the obtained 3D avatars, the authors used the Kinect sensor's skeleton points to generate several 3D human body parts measurements for all reconstructed models with an Intel Xeon 2.67 GHz CPU 12GB RAM. Lastly, the following allowed them to calculate the corresponding average error of the biometric measurements in centimeters between the reconstructed and real human bodies alongside the proposed algorithm's average runtime, as shown in Table 4.

Table 4. Runtime for each processing part and the average error of the biometric measurements [20]

Frames	**S.R.**	**Rigid**	**Non-Rigid**	**Poisson**	**All**
36	28 sec	110 sec	620 sec	68 sec	826 sec (13.8 min)
Neck-Hip	**Shoulder Width**	**Arm Length**	**Leg Length**	**Waist Girth**	**Hip Girth**
2.1 cm	1.0 cm	2.3 cm	3.1 cm	3.2 cm	2.6 cm

1.5. Kinect v1 versus Kinect v2

With the increasing popularity of RGB-D sensors in the computer vision community, several studies have compared Kinect v1 and Kinect v2 [29, 30, 31]. Generally, Kinect v1 is a structured light camera that captures a projected light form and identifies the distance separating the object and the camera by monitoring the light pattern's deformation [29]. Although previous research focused on Kinect v1 as one of the most common RGB-D sensors with several algorithms developed for it, a new Kinect sensor that depends on a different technology was released in 2014. In brief, Kinect v2 is a ToF camera that estimates the distance between the camera and the object by calculating the sensor's signal's travel time and emitted from the projector [29].

Meanwhile, Oliver *et al.* [32] provided a systematized comparison between Kinect sensors' versions by evaluating their captured depth images accuracy. Their experiment primarily consisted of acquiring approximately 100,000 depth images for about an hour of runtime while analyzing the sensor's temperature influence on every frame. Accordingly, the obtained experimental results suggested that for more reliable depth image quality, the Kinect v2, which has an internal fan, should be pre-heated for at least 25 minutes before capturing any image. By contrast, Kinect v1 required a smaller warm-up duration.

Moreover, the conducted research in [32] delivered useful information about the capturing distance's influence. The presented study findings initially indicated that Kinect v1 accuracy dropped while increasing the scanning distance. On the contrary, higher distance slightly affected Kinect v2 accuracy as it remained approximately constant, although the precision dropped substantially. Lastly, the authors concluded by recommending using Kinect v2 for both 3D reconstruction and human body measurements applications due to its higher accuracy and ability to identify up to 6 people. On this basis, the low precision of the acquired depth frames by Kinect v2 has to be considered by applying pre-processing algorithms on the depth images before using them.

1.6. Kinect versus other RGB-D sensors

In the past decade, the image processing field witnessed an unprecedented shift with the development of numerous RGB-D sensors integrating the characteristics of laser-based 3D scanners and optical sensors such as Microsoft Kinect, PrimeSense, and Intel RealSense cameras. Such devices can achieve 3D scanning and measurements of objects and are, for the most part, portable like optical sensors. This technology has allowed the following applications to become more convenient with affordable low-cost RGB-D sensors that can capture both depth and color images in real-time.

Table 5 displays a thorough comparison of the main characteristics of some of the most frequently used RGB-D sensors in 3D human body measurements and reconstruction applications. Nevertheless, the displayed RGB-D sensors in this section are not the only available sensors in the market. There are many other cameras such as Asus Xtion PRO Live, SwissRanger 4000, PMD CamCube 3.0, CubeEye, Intel RealSense Camera F200, Intel RealSense Depth Camera D415.

To put it differently, determining the most suitable RGB-D device for any application relies on several factors such as task requirements, budget, and the required sensor’s features.

Table 5. Main characteristics of different RGB-D sensors [30, 33, 34]

	Kinect 1.0	**Kinect 2.0**	**PrimeSense**	**RealSense D435**
Sensor type	Infrared Structured Light	Time of Flight (ToF)	Structured Light	Active Stereoscopic
RGB resolution (pixel) and frame rate (fps)	640×480 at 30 fps or 1280×1024 at 12 fps	1920×1080 at 30 fps	640×480 at 30 fps	1920×1080 at 30 fps
Depth resolution (pixel) and frame rate (fps)	640×480 at 30 fps	512×424 at 30 fps	640×480 at 30/60 fps	1280×720 at 90 fps
Depth distance range (m)	$0.8 - 4$ (default) $0.4 - 3.5$ (near)	$0.5 - 4.5$	$0.8 - 3.5$ or $0.35 - 1.4$	$0.11 - 10$
Field of view of depth image (Horizontal, Vertical)	$57° \times 43°$	$70° \times 60°$	$57.5° \times 45°$	$85.2° \times 58°$
Field of view of RGB image (Horizontal, Vertical)	$62° \times 48.6°$	$84.1° \times 53.8°$	$57.5° \times 45°$	$69.4° \times 42.5°$
Tilt motor	Yes	No	No	No
Skeleton joints defined	20	25	19	15
Maximum skeletal tracking	2	6	6	6
USB interface	2.0	3.0	2.0	3.0 Type C
Power Supply	External Adapter	External Adapter	USB	USB
Price ($)	80	200	295	179

2. Overview

This chapter gives an overview of the proposed system for obtaining 3D human body parts measurement of a subject using a single Kinect v2 sensor. Similarly, a comprehensive description of each part of the suggested algorithm is displayed.

The presented research project aims to devise and implement a fast and unsophisticated approach for real-time non-contact 3D human body measurement of a user. With a single RGB-D sensor such as Microsoft Kinect v2, a user can easily configure the scanning scenario with a personal computer within a limited workspace, whereas no second operator is needed. **Fig. 15** presents an outline of the proposed method. Initially, the system captures ten frames containing depth, color, and skeleton data of the human body front view using Kinect v2. Afterward, the acquired human body frames are segmented, and 3D point clouds of the human body are constructed and denoised.

The next step incorporates using the segmented depth image and the human body's skeleton data, which consists of 25 skeleton joints points, for defining the depth points' coordinates of the human body parts. Accordingly, the obtained 2D coordinates are mapped to the 3D point cloud data to estimate the measurement for each of the body parts. Ultimately, error compensation is applied to reduce the obtained measurement errors and generate more accurate measurements. For instance, the introduced algorithm can produce 13 different human body measurements in particular:

- Height,
- Arm left and right lengths,
- Hand left and right lengths,
- Leg left and right lengths,
- Neck to hip length,
- Shoulder length,
- Hip, waist, stomach, and chest front end perimeter,

Fig. 15. Pipeline of the proposed method

2.1. Data capturing

In brief, the data capturing process consists of applying Elise *et al.*'s [35] recommendations. The following study findings highlighted several suggestions to obtain better measurements from Kinect v2 and reduce the frames' noise, illustrated as follows:

- Considering a pre-heating time for the device of almost 30min,
- Collecting a tenth of consecutive depth maps from a single viewpoint,

Fig. 16 presents an outline of the proposed system setup for the data capturing process coupled with the corresponding algorithm. During the frames' acquisition, a user has to maintain a steady "T" pose in front of the sensor for approximately 10 seconds to capture ten frames while standing within the selected scanning distance (2.5m). Meanwhile, each subject has ten seconds to stand in front of the sensor before the capturing process starts. The main advantage of this delay is eliminating the necessity for a second operator. Additionally, the system verifies the human body skeleton's identification before saving the collected frames. In other words, an error occurs upon failure in detecting the required skeleton or exceeding the data capturing limit, which leads to the cancellation of the entire procedure.

On the contrary, in the case of a successful tracking and identification of the human skeleton, all the acquired frames metadata are saved into the project's database. The metadata generally consists of the joints' position coordinates, color image, and depth image data. Furthermore, the user can decide whether to capture more frames or save the collected data to the corresponding directory and end the capturing process. Ultimately, using the method presented in [36], Microsoft Cortana voice commands were added to the system to notify the user about the capture process's failure or success.

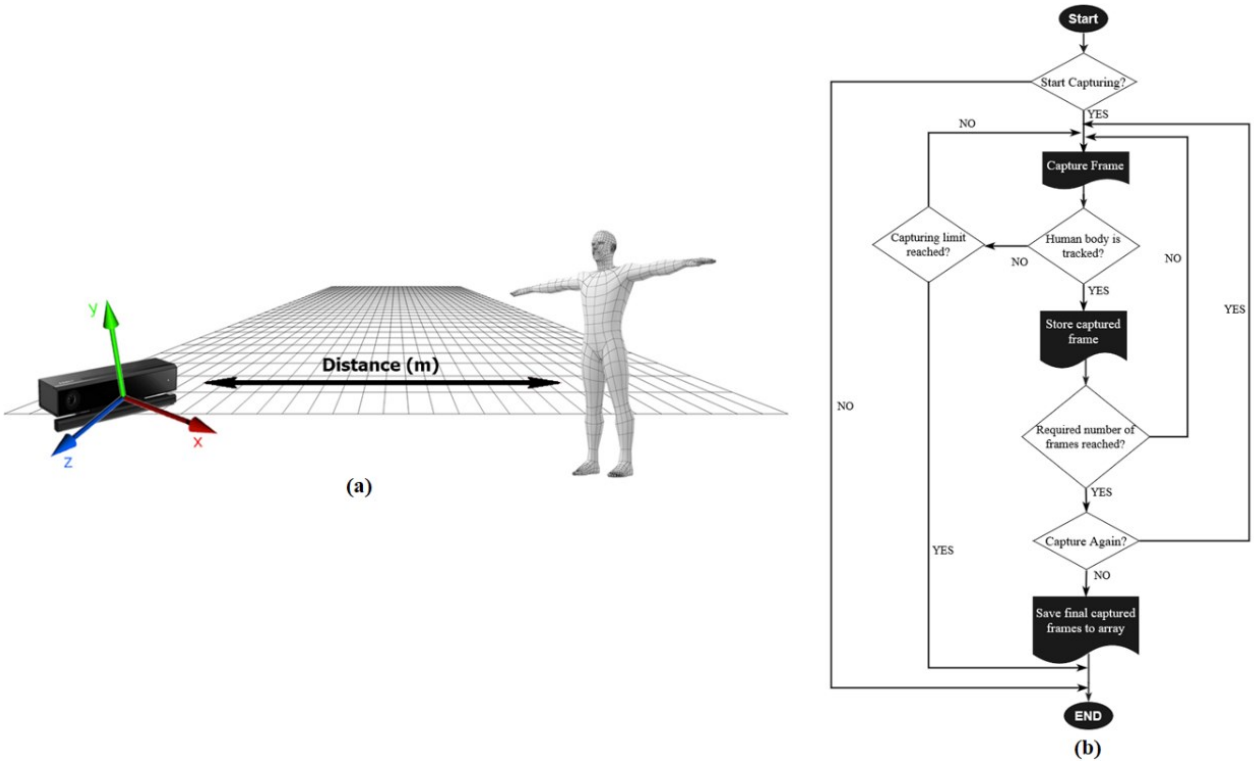


Fig. 16. Data capturing: (a) capturing scenario; (b) capturing algorithm flowchart

Besides, few limitations arise as it is necessary to consider several factors during the collection of the human body frames, such as:

- Human must be wearing regular clothes that are not so loose because loose clothes could affect several body measurements such as waist, chest, and stomach front end perimeter [15],
- Human body scanning must not be performed in an outdoor space as the sun rays may result in data distortion and increase the noise. Nevertheless, there exists a considerable body of literature recommending performing the scanning in an enclosed indoor environment such as in [35],
- A user must maintain a steady position during the capturing process because an increased motion may cause deformations and eventually affect the accuracy of the obtained results [22],

Further to this, to be noted that all the RGB-D datasets in the capturing experiments were acquired using a single Kinect v2 facilitated with an HP Pavilion notebook computer (CPU Intel i7-6700HQ 2.6 GHz, 16 GB RAM) and MATLAB R2018a (C programming language).

2.2. Segmentation, Denoising and 3D Point Cloud

Foreground segmentation and background removal are essential techniques in computer vision. As mentioned previously, a substantial part of the data capturing is the skeleton tracking executed by the sensor. The following enables characterizing the human body in each frame by several joint coordinates, represented as 25 joints points and displayed as a depth joints map in **Fig. 17**. Additionally, the generated joints points include both 2D and 3D coordinates, allowing Kinect to perform depth base segmentation and return the segmented body index frame.

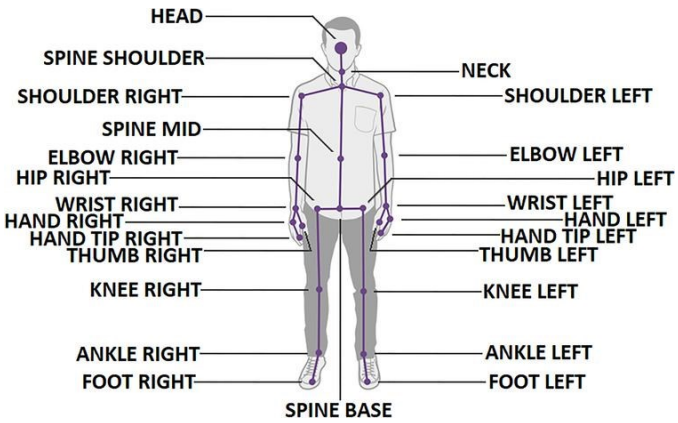


Fig. 17. Depth joints map [41]

Accordingly, the segmentation of the 3D point cloud of the human body consisted of the following steps (see **Fig. 18**):

- Acquiring the depth inverse of the binary image from the sensor's segmented depth body index frame.
- Transforming the binary image inverse to the same type of the original depth image and multiplying them together to obtain the main segmented depth image.
- Aligning the previously segmented depth image with the original color image to generate the corresponding segmented 3D point cloud.

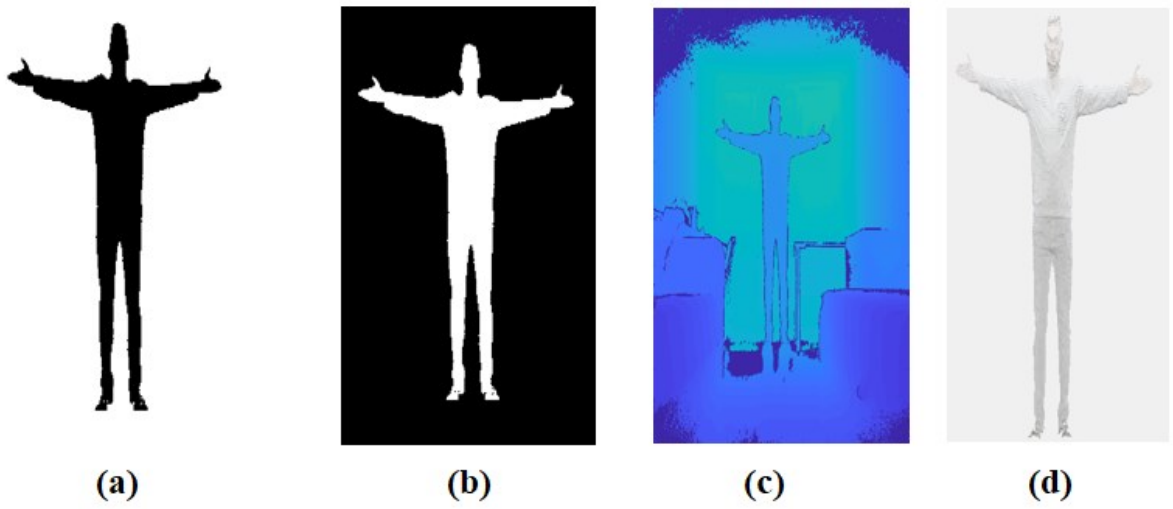


Fig. 18. 3D Point cloud segmentation: (a) depth body index frame; (b) depth inverse of binary image; (c) original full depth image; (d) segmented 3D point cloud

Moreover, numerous studies investigated the captured data's lateral noise and focused on removing it in 3D only, such as in [22]. However, unlike previous works, this research project adopts filtering the noise from the captured 2D depth images and the segmented 3D point cloud. The following is important when defining the corresponding human body joints' coordinates in 2D before mapping them to 3D.

Furthermore, the developed algorithm implements the 2D averaging filter to remove the noise from the segmented depth image in 2D to generate a new denoised 3D segmented point cloud data. Generally, the averaging filter performs image smoothing by decreasing the intensity differences between the adjacent pixels and substituting each pixel with the mean of all the neighboring pixels using a convolution mask over each pixel. Meanwhile, the chosen size of the neighborhood mainly controls the amount of filtering. Therefore, a relatively small neighborhood size (2) was selected when using the averaging filter because a larger neighborhood proportion may result in a loss of image details, although it produces higher noise reduction [38].

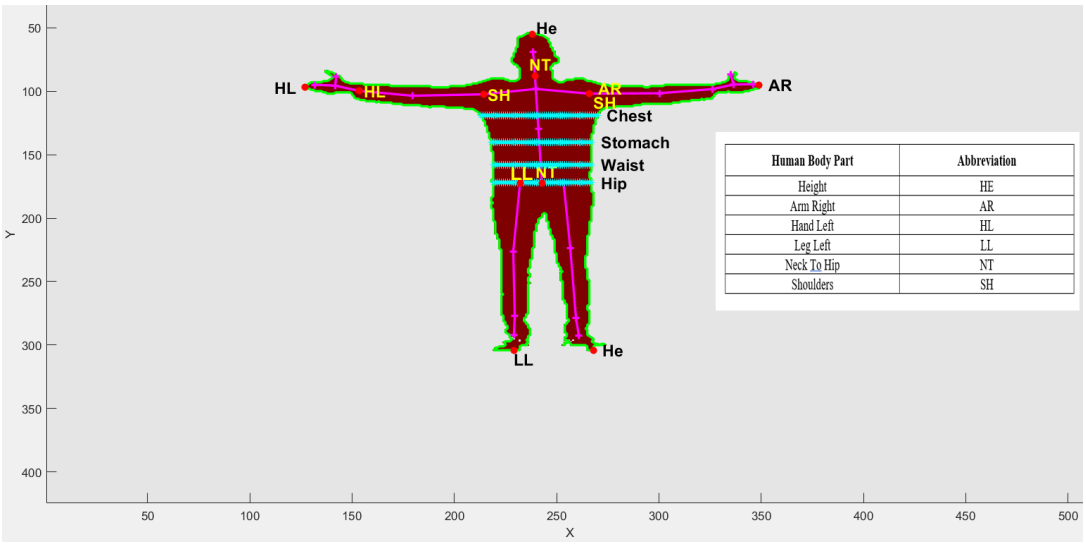
Algorithm 1 presents an example of utilizing the 2D averaging filter to smooth both the segmented 2D depth image and the segmented 3D point cloud data. At first, the point cloud location property, which consists of the point cloud points in the form of a 3-dimensional matrix, was divided. Afterward, the averaging filter was applied to denoise the main segmented depth image. Subsequently, the system generates the corresponding X, Y, and Z coordinates of the new denoised 3D point cloud before ultimately obtaining a new denoised 2D depth image.

Algorithm 1. Denoising with an averaging filter algorithm

1: Denote: ptCloud as the segmented 3D point cloud of the human body
2: Denote: N = 2 as the neighboorhood size for the average filter
3: ptCloud.Location = [X,Y,Z] coordinates of the point cloud points in the form of a 3-column matrix
4: General equation of averaging filter in which all zeros must be replaced by ones = ones(n(1),n(2))/(n(1)×n(2))
5: X = ptCloud.Location(:,:,1)
6: Y = ptCloud.Location(:,:,2)
7: Main segmented depth image: Z = ptCloud.Location(:,:,3)
8: Applying averaging filter: Z_1 = filter(('average',N),Z)
9: tmp_1 = zeros(size(X))
10: tmp_1(find(X)) = 1
11: tmp_2 = zeros(size(Z_1))
12: tmp_2(find(Z_1)) = 1
13: obtain new denoised X,Y,Z coordinates matrix of the point cloud
14: X_1 = tmp_1×tmp_2×X
15: Y_1 = tmp_1×tmp_2×Y
16: Z_2 = tmp_1×tmp_2×Z_1
17: New denoised 3D point cloud = ptCloudOut(X_1,Y_1,Z_2)
18: New denoised 2D segmented depth image = binary(Z_2)

2.3. Defining human body parts points

In general, several RGB-D sensors, which provide human body tracking capabilities, define the skeleton points as joints' centers described with several properties, including the corresponding 2D and 3D coordinates. However, utilizing these points directly to obtain human body parts measurements can decrease the achieved results' accuracy. Accordingly, in contrast to previous works that used the predefined skeleton points directly to acquire measurements [22], this research's essential contribution was to estimate new points located at the human body's edges. The main advantage of this is that it can be implemented with minimal skeleton points to reduce measurement errors. In other words, the following allows using any available RGB-D sensor with human body tracking functionality and at least 15 defined human body skeleton points. **Fig. 19** shows all human body parameters defined by the presented system. First, determining several measurements such as the height, leg, and arm length required identifying two points' indices for each of them. On the other hand, to measure the waist, chest, stomach, and hip circumference, the indices of multiple points surrounding the corresponding body parts' locations were specified.



Human Body Part	Abbreviation
Height	HE
Arm Right	AR
Hand Left	HL
Leg Left	LL
Neck To Hip	NT
Shoulders	SH

Fig. 19. Defined human body parts points

Moreover, the developed method consisted of multiple steps before defining new body parts points. The first step was acquiring the inverse of the binary image of the segmented depth body index frame. Afterward, the coordinates of all the contour points of the generated binary image were determined using a function implemented based upon [39], which uses the Moore-Neighbor contour tracing algorithm adjusted by Jacob's stopping criteria. In brief, to determine the edges in an image, the Moor neighborhood method identifies the indirect neighbors of a pixel, which are a set of 8 pixels sharing an edge pixel or a vertex with that pixel.

Based on the obtained depth coordinates of the human body contour, the corresponding points of each body part were estimated as follows:

5. Height Points:
   - Upper point depth coordinates (X, Y) were estimated by searching in the contour coordinates for the closest point to the skeleton head point, namely the point with identical X coordinate and minimal Y coordinate.
   - Lower point depth coordinates were obtained by searching for the contour point with the maximal Y coordinate.
6. Arm Left Points:
   - The first point has the same depth coordinates as the skeleton's shoulder left point.
   - Second point's depth coordinates were obtained by searching for the contour point with the minimal X coordinates.
7. Arm Right Points:
   - The first point has the same depth coordinates as the skeleton's shoulder right point.
   - Second point's depth coordinates were acquired by searching for the contour point with the maximal X coordinates.
8. Hand Left Points:
   - The first point has the same depth coordinates as the skeleton's wrist left point.
   - Second point's depth coordinates were obtained by searching for the contour point with the minimal X coordinates.
9. Hand Right Points:
   - The first point has the same depth coordinates as the skeleton's wrist right point.
   - Second point's depth coordinates were acquired by searching for the contour point with the maximal X coordinates.
10. Neck to Hip Points:
    - The first point has similar depth coordinates as the skeleton's neck point.
    - The second point has the same depth coordinates as the skeleton's spine base point.

11. Shoulder Points:

- The first point has similar depth coordinates as the skeleton's shoulder left point.
- The second point has the same depth coordinates as the skeleton's shoulder right point.

12. Leg Left Points:

- The first point has the same depth coordinates as the skeleton's hip left point
- Second point's depth coordinates were obtained by searching in the contour coordinates for the closest point to the skeleton foot left point, namely the point with identical X coordinate and maximal Y coordinate.

13. Leg Right Points:

- The first point has similar depth coordinates as the skeleton's hip right point.
- Second point's depth coordinates were acquired by searching in the contour coordinates for the closest point to the skeleton foot right point, namely the point with identical X coordinate and maximal Y coordinate.

14. Hip Circumference Points:

- In a real-life scenario, hips are not measured as the distance between 2 points but as a curvature. Therefore, in this research, the hip front end perimeter's points were determined. At first, the algorithm specified one contour point with the same Y coordinate as the hip left or hip right skeleton points. Afterward, the depth binary image coordinates were searched for all the points with the same Y-axis coordinate as the previously defined hip contour point. The final obtained points allow measuring the hip circumference.

15. Waist Circumference Points:

- In general, waists are also not measured as the distance between 2 points but as curvature. Therefore, in this research, the waist front end perimeter's points were determined. At first, the system specified the contour's Y coordinate for the waist by searching for the point positioned one-third the distance between spine-mid and spine base skeleton points. Then, the depth binary image coordinates were searched for all the points with the same Y-axis coordinate as the previously defined waist contour point. The final generated points correspond to the waist circumference points.

16. Stomach Circumference Points:

- Similarly, the stomach circumference points were determined by finding a contour Y coordinate for the stomach by searching for the point positioned three-fourths the distance between spine-mid and spine base skeleton points. Accordingly, the depth binary image coordinates were searched for all the points with the same Y-axis coordinate as the previously defined stomach contour point. The final obtained points define the stomach circumference points.

17. Chest Circumference Points:

- Likewise, chest circumference points were determined by first finding a contour Y coordinate for the chest by searching for the contour point positioned one-third the distance between spine-mid and spine shoulder skeleton points. Following this, the depth binary image coordinates were searched for all the points with the same Y-axis coordinate as the previously defined chest contour point. The final collected points allow measuring the chest circumference.

2.4. Measuring human body parts

After defining all the required human body parts points and their corresponding depth coordinates (x,y), 2D coordinates had to be mapped into 3D coordinates (x,y,z) by entering the depth coordinates as parameters in the point cloud location property to generate the corresponding 3D coordinates. Afterward, based on the acquired 3D coordinates of all the previously defined human body parts points, all the respective body parts measurements were estimated using the Pythagoras theorem of 3D space. **Algorithm 2** presents an illustration of how to determine the human body height length in meters.

Furthermore, a similar method was applied to obtain the measurements of all the remaining body parts. However, the main difference was that for several body measurements such as hip, waist, stomach, and chest, the calculation of the length depended on defining the sum of lengths between every 2 points of the front-end perimeter points.

Algorithm 2. Measuring the human height length

1: Denote: (X,Y) coordinate of depth Height point
2: Denote: ptCloud as the 3D point cloud of the human
3: 2D to 3D Mapping: [xyzArray] = ptCloud.Location(Y,X,:)
4: 3D coordinate of first height point: $P_1 = [x_1\ y_1\ z_1]$
5: 3D coordinate of second height point: $P_2 = [x_2\ y_2\ z_2]$
6: Height length in meters using Pythagoras theorem of 3D space = $[(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2]^{1/2}$
7: Height length in centimeters = Height length in meters × 100;

2.5. Selecting the Capturing Distance

Generally, when selecting the most suitable capturing distance, many factors must be considered, such as the sensor's position, the sensor's field of view, and the captured frame resolution. Additionally, with greater distances, the quality of the captured depth images is degraded by the low resolution of the depth measurements and the noise. **Fig. 20** shows the Kinect's field of view in which the sensor was placed 1.5 meters above the ground within a capturing distance of 2 meters. The specified field of view then enables covering the full human body with a maximal vertical point of 2.2m and a minimal vertical point of -0.1m. However, the sensor must be placed with a relatively small inclination (approximately 13°) because positioning the sensor horizontally to the ground can lead to losing part of the captured human body legs.

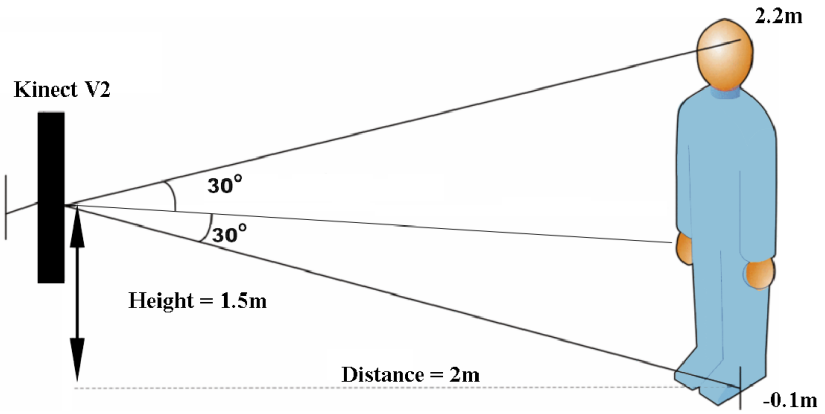


Fig. 20. Kinect v2 sensor field of view

In brief, to select the most suitable capturing distance for the proposed method, the conducted experiment consisted of collecting and maintaining a dataset of 300 frames. Each 100 corresponded to one of the following distances: 2m, 2.5m, and 3m. Note that the captured frames belonged to one subject wearing the same clothes, in the same indoor environment and lighting conditions. Ultimately, the system generates the corresponding 3D human body parts parameters for each frame in addition to the resulting absolute errors of measurements (see **Fig. 21**).

$$Absolute\ error\ in\ cm = |measured\ body\ part\ length\ with\ sensor - real\ body\ part\ length|$$

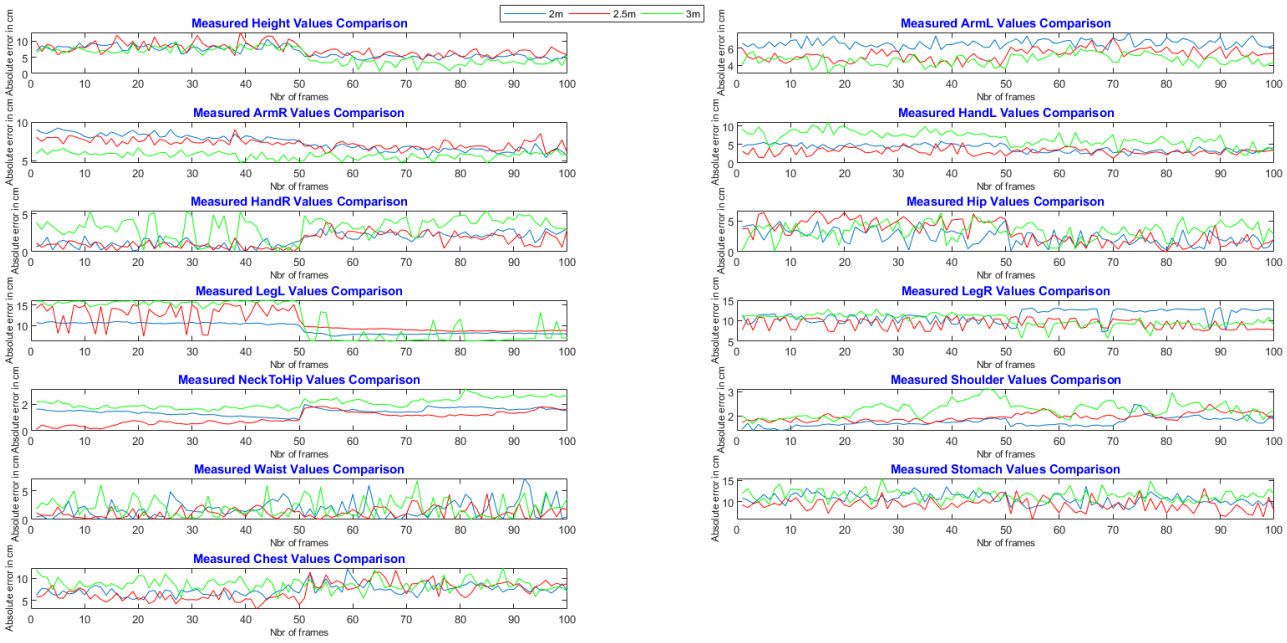


Fig. 21. Comparison of the obtained human body parts measurements from 3 distances (2m, 2.5m, 3m)

Moreover, the evaluation of the data presented in this experiment (**Fig. 21**) proves the developed approach's capability in generating human body parts measurements from all the selected distances with a variation of the absolute error from one distance to another. Nevertheless, the achieved results were insufficient to determine the most suitable capturing distance. Accordingly, the following step consisted of calculating the AAE of the obtained results, as shown in **Table 6**.

Table 6. Average absolute errors (AAE) (in cm) of the obtained human body parts measurements from 3 distances (2m, 2.5m, 3m)

Measurement	**From 2m**	**From 2.5m**	**From 3m**
Height Length	6.6	7.4	**5.4**
Arm Left Length	6.6	5.2	**4.7**
Arm Right Length	7.3	7.2	**5.7**
Hand Left Length	3.9	**3.2**	6.7
Hand Right Length	1.6	**1.4**	3.2
Hip Circumference	**2.4**	3.2	3.4
Leg Left Length	**9.3**	10.9	11.5
Leg Right Length	11.2	**9.2**	10.2
Neck to Hip Length	1.4	**0.9**	2.1
Shoulder Length	**1.7**	1.9	2.3
Waist Circumference	1.8	**1.2**	2.0
Stomach Circumference	10.6	**9.2**	11.4
Chest Circumference	7.4	**7.2**	8.6
Average of AAE	5.5	**5.2**	5.9

$$Average\ absolute\ error\ in\ cm = average(|array\ of\ measured\ body\ part\ length\ with\ sensor - real\ body\ part\ length|)$$

Table 6 displays the AAE of the human body parts measurements of the captured data from 3 different distances. According to the acquired data, the AAE of seven body parts measurements was the smallest for the collected results from 2.5m compared to that from 2m and 3m, where the AAE of only three measurements was the smallest. Similarly, a comparison of the final average of all AAE shows that 2.5m had the smallest outcome (5.2cm), which is 0.3cm lower than that 2m and 0.7cm smaller than that obtained from 3m.

2.6. Developing Error Compensation

In general, many factors may affect the accuracy of the obtained human body parts measurements, such as lighting, clothing, and noisy depth data. For this purpose, this research attempted to integrate an error compensation strategy to minimize the reconstructed human body parts measurement errors. Meanwhile, numerous previous studies, such as in [12], implemented error compensation by calculating the average error of measurements using real human body parts measurements to improve the final values as follows:

$$Calculated\ value + (Calculated\ value \times Error\ percentage)$$

However, one limitation of such an approach is that it might not be accurate because, in virtual dressing applications, the main goal is to provide individuals with their final human body parts measurements without knowing their real measurements.

Accordingly, this research project investigated a different strategy for implementing error compensation. At first, this experiment's dataset incorporated 1000 captured frames of one subject wearing the same clothes, in the same indoor environment and lighting conditions. Afterward, the 3D human body parts measurements for each captured frame were estimated and stored in a separate array corresponding to each body part. The following allowed to approximate a compensation value for every human body parameter by the expression:

$$Error\ Compensation = average\ (measured\ body\ part\ array - real\ body\ part\ length)$$

Furthermore, the obtained error compensation values were positive and negative:

- With positive error compensation:

$$new\ body\ part\ measured\ length = measured\ length\ with\ sensor - error\ compensation$$

- With negative error compensation:

$$new\ body\ part\ measured\ length = measured\ length\ with\ sensor + error\ compensation$$

Ultimately, based on the conducted experiment, the final obtained error compensation values for the corresponding human body parts measurements are presented in **Table 7**.

Table 7. Estimated error compensation values

Human Body Parameter	**Error Compensation Value (cm)**
Height	−6.1
Arm Left	−5.2
Arm Right	−7.1
Hand Left	3.8
Hand Right	1.9
Hip	−5.9
Leg Left	−9.4
Leg Right	−9.4
Neck to Hip	2.2
Shoulder	−1.8
Waist	−0.9
Stomach	8.3
Chest	5.6

3. Results and discussion

The following chapter presents the achieved 3D human body parts measurement results from this research project's proposed method.

3.1. Experimental Results

Based on the previously reported error compensation values, the developed system was initially tested on three subjects. At first, this required collecting the real body parts measurements of all of them manually using a measurement. Afterward, ten frames were captured for each user separately while maintaining a "T" pose within a distance of 2.5m of the Kinect sensor. Ultimately, the obtained 3D human body part measurement results were recorded before and after implementing the error compensation (see **Table 8**, **Table 9**, and **Table 10**). Note that the proposed algorithm consists of estimating the final obtained measurements with the sensor as the average measurements of each captured ten frames.

$$Body\ part\ measured\ length = round(average(array\ of\ measured\ body\ part\ length\ of\ 10\ frames\ ))$$

Table 8. Final body parts measurements of subject 1 (S1)

Measurement	**Manually measured values (cm)**	**Sensor values before compensation (cm)**	**Sensor values after compensation (cm)**
Height Length	182	176	**182**
Arm Left Length	81	75	**80**
Arm Right Length	81	72	**79**
Hand Left Length	20	27	**23**
Hand Right Length	20	25	**23**
Hip Circumference	44	39	**45**
Leg Left Length	103	95	**104**
Leg Right Length	103	95	**104**
Neck to Hip Length	60	**58**	56
Shoulder Length	40	35	**37**
Waist Circumference	43	**43**	44
Stomach Circumference	42.5	**43**	35
Chest Circumference	46	**45**	40

Table 9. Final body parts measurements of subject 2 (S2)

Measurement	**Manually measured values (cm)**	**Sensor values before compensation (cm)**	**Sensor values after compensation (cm)**
Height Length	178	172	**178**
Arm Left Length	74	70	**75**
Arm Right Length	74	69	**76**
Hand Left Length	19	23	**20**
Hand Right Length	19	23	**21**
Hip Circumference	43	38	**44**
Leg Left Length	96	87	**96**
Leg Right Length	96	87	**97**
Neck to Hip Length	60	61	**59**
Shoulder Length	40	35	**37**
Waist Circumference	42	38	**39**
Stomach Circumference	40	**42**	34
Chest Circumference	47	**45**	39

Table 10. Final body parts measurements of subject 3 (S3)

Measurement	**Manually measured values (cm)**	**Sensor values before compensation (cm)**	**Sensor values after compensation (cm)**
Height Length	167	161	**167**
Arm Left Length	73	71	**76**
Arm Right Length	73	66	**74**
Hand Left Length	18	22	**18**
Hand Right Length	18	19	**17**
Hip Circumference	58	51	**57**
Leg Left Length	93	80	**89**
Leg Right Length	93	80	**89**
Neck to Hip Length	55	58	**56**
Shoulder Length	41	39	**41**
Waist Circumference	56	52	**53**
Stomach Circumference	52	**54**	46
Chest Circumference	57	**59**	53

Table 8, **Table 9**, and **Table 10** display the acquired 3D human body parts measurements of 3 subjects in a user-friendly and efficient approach based on the reported algorithm. The presented results incorporated 13 different measurements. Subsequently, the data's evaluation demonstrates the introduced method's ability to achieve desirable results and reduce measurement errors. In other words, the obtained measurements after error compensation for the majority of the collected data were approximately closer to the actual human body values. Compared to the current methods based on infrared scanning or laser scanning, the Kinect scanning and measuring methods are hence utterly flexible and capable of conveniently obtaining the 3D measurement of users at low cost.

To further investigate the experimental results, this section evaluates the average absolute error of the human body parts measurements between the real and the final measured human body parts data with the Kinect sensor, with and without error compensation. **Table 11** shows the AAE of each captured ten frames corresponding to each subject separately in addition to the total average of AAE of all the individuals. The displayed results show that the developed compensation was performing adequately because the calculated average absolute errors for most of the measurements and the average of AAE of all the generated measurements decreased significantly after implementing the error compensation.

Meanwhile, the average absolute error of several measurements such as neck to hip, stomach, and chest increased after implementing the error compensation for the following reasons:

- Neck to hip measurement primarily depends on the human's skeleton points generated by the sensor, which are the neck and spine base points. Therefore, this measurement can be influenced by the inclination or the position of the user's head while maintaining a steady "T" pose. The following may result in an unpredictable measurement error. Nevertheless, error compensation for this body parameter might not be accurate.
- Stomach and chest circumference are influenced by the tightness or looseness of the individual clothes during the data capturing process, resulting in an unpredictable measurement error. Thus, error compensation for these measurements might not be accurate.

Table 11. Average absolute error (in cm) of measurements between the measured and real human body parts with and without error compensation

Measurement	AAE of S1		AAE of S2		AAE of S3		Average of AAE of all subjects	
	Woc	Wc	Woc	Wc	Woc	Wc	Woc	Wc
Height Length	6	**0**	6	**0**	6	**0**	6	**0**
Arm Left Length	6	**1**	4	**1**	**2**	3	4	**1.7**
Arm Right Length	9	**2**	5	**2**	7	**1**	7	**1.7**
Hand Left Length	7	**3**	4	**1**	4	**0**	5	**1.3**
Hand Right Length	5	**3**	4	**2**	1	1	3.3	**2**
Hip Circumference	5	**1**	5	**1**	7	**1**	5.7	**1**
Leg Left Length	8	**1**	9	**0**	13	**4**	10	**1.7**
Leg Right Length	8	**1**	9	**1**	13	**4**	10	**2**
Neck to Hip Length	**2**	4	**1**	2	3	**1**	**1.7**	2
Shoulder Length	5	**3**	5	**3**	2	**0**	4	**2**
Waist Circumference	0	**1**	4	**3**	4	**3**	2.7	**2.3**
Stomach Circumference	**0.5**	7.5	**2**	6	**2**	6	**1.5**	6.5
Chest Circumference	**1**	6	**2**	8	**2**	4	**1.7**	6
Average of AAE of all measurements	4.8	**2.6**	4.6	**2.3**	5.1	**2.1**	4.8	**2.3**

3.2. Performance Comparison

Before assessing the presented method's effectiveness and performance, the first step consisted of recomputing all experimental individuals' final biometric measurements after executing the suggested error compensation. However, as previously explained, error compensation was not applied for the neck to hip, stomach, and chest measurements. For this purpose, the following measurements' final values remained equal to the values obtained by the sensor before compensation.

Moreover, in further steps, additional data collection of more participants was performed. Hence, there are six people tested in the conducted experiments. **Table 12** shows the average absolute errors of human body parts measurement in centimeters between the real and the measured results of the human bodies with the Kinect sensor. Accordingly, this table compares all computed AAE results against the traditional method's reported results in [12, 15, 20, 22] using one Kinect sensor. These findings go beyond previous reports, showing that the algorithm highlighted in this thesis provided ten different body parts measurements, while no other method achieved more than eight. Additionally, the evaluation of the produced data demonstrates a significant improvement in the AAE of several human body parts, including height, hip, and waist, as slightly superior outcomes were achieved.

Meanwhile, it must be pointed out that although the compared studies displayed relatively better results for few measurements such as arm, leg, and shoulder, several arguments can make their findings not considered accurate and are summarized as follows:

- Some of the previous works presented in **Table 12**, such as [20, 22], are 3D modeling applications that focused on reconstructing 3D avatars of human bodies. In other words, the authors did not introduce any method that justifies the process used to acquire the human body measurements because they used the skeleton points defined by the RGB-D sensor directly to obtain them. Consequently, the following process may decrease the accuracy of the generated measurements because RGB-D sensors define skeleton points as joint centers and not points at the human body's edge.
- Although the final AAE in this research corresponded to the average of six subjects only, the results of the majority of the existing methods, including in [12], were for an unknown number of subjects or one subject only. Nevertheless, this makes it challenging to analyze the accuracy of the provided experimental results.
- Numerous previous works, such as [15], measured the arm's length of subjects from shoulder to wrist with a closed hand, while this thesis algorithm measured the entire arm length from shoulder to the edge of the opened hand.

Table 12. AAE of human body parts measurement (cm) between the real and the measured results of the human bodies with the Kinect sensor

	Height Length (cm)	Arm Length (cm)	Hand Length (cm)	Hip Girth (cm)	Leg Length (cm)	Neck to Hip Length (cm)	Shoulder Length (cm)	Waist Girth (cm)	Stomach Girth (cm)	Chest Girth (cm)
In [12] with one Kinect	0.1	2.7			2.4	1.4	1.6	4.7	5.1	5.9
In [15] with one Kinect		**1.2**		2.1	1.5	2.4		2.5		2.2
In [22] with one Kinect	0.5	1.4		2.2	**1.4**	2.0	1.2	3.1		
In [20] with one Kinect		2.3		2.6	3.1	2.1	**1.0**	3.2		
This thesis with one Kinect	**0.0**	1.5	**1.4**	**0.8**	1.7	**1.3**	1.7	**2.2**	**1.9**	**1.5**

Furthermore, the presented results in **Table 12** were statistically compared using the corresponding standard deviations displayed in **Fig. 22**. The calculated standard deviations of the average absolute error of measurements show a statistically considerable improvement achieved by this research. In other words, this thesis accomplished the lowest standard deviation (0.6) as opposed to that of all the presented previous works (≥ 1).

To further investigate the experimental results, **Fig. 23** displays the accuracy of the acquired human body parts measurements of all the tested subjects (6 participants) calculated as follows:

$$Accuracy = 100 - \left(\frac{measured\ body\ part\ length\ \ with\ sensor - real\ body\ part\ length}{real\ body\ part\ length}\right) * 100$$

The findings show that this research achieved measurement accuracy between 92% and 100% on average. Note that the low accuracy of hands' measurements (92%) is caused by the human's hand's relative motion during the capturing process.

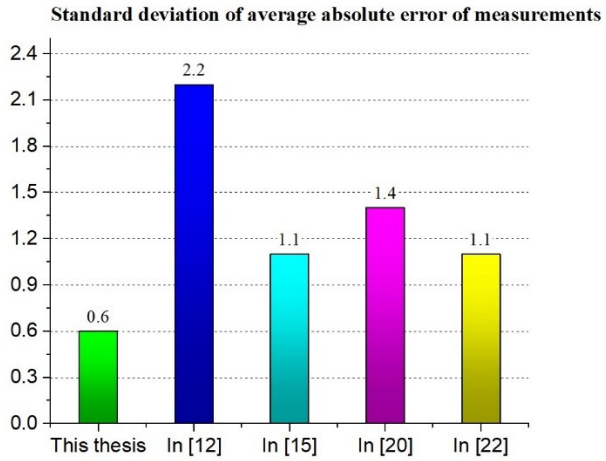


Fig. 22. Standard deviations of the average absolute error of measurements between the real and the measured results of the human bodies with the Kinect sensor

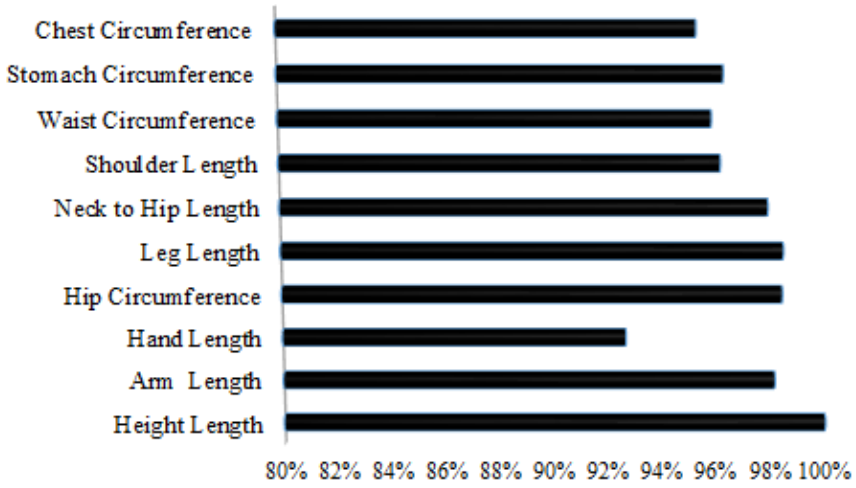


Fig. 23. Accuracy of the acquired human body parts measurements based on the real and the measured results of the human bodies with the Kinect sensor

3.3. Performance Analysis

In general, this research aimed to develop an overarching approach while reducing all implementations' runtime to avoid unnecessary computations and save processing power. Therefore, to further demonstrate the proposed method's performance, this section evaluates the thorough body parts measurement algorithm's execution time. **Table 13** shows the average running time, and the computation cost during each stage of the presented algorithm. A total of 20 seconds on average only was required to generate the entire human body parts measurements of a subject. Overall, this thesis provided a concise data capture strategy, an efficient body part measurement algorithm within a short time, and a simple hardware configuration.

Table 13. Average running time spent during each step in the thorough body parts measurement procedure

With CPU Intel i7-6700HQ 2.6 GHz, 16 GB RAM	
Step	**Time (seconds)**
Data capturing	10
Segmentation and 3D point cloud	8
Denoising and obtaining final human body measurements	2
Total (seconds)	20

3.4. Advantages of the proposed method

Generally, in 3D human body scanning and 3D human body parts measurement applications, researchers focus on reconstructing 3D avatars before generating the corresponding body parts parameters. In essence, such method's algorithms consist of the following steps:

- Data capturing,
- Segmentation and Denoising,
- Rigid and non-rigid alignment,
- Surface reconstruction,
- Obtaining human body parts measurements of reconstructed 3D models,

On the contrary, this thesis investigated measuring the human body parts of a subject without reconstructing any 3D mesh or using preprocessing algorithms. The main advantage of this is providing future researchers with the ability to improve virtual dressing applications. Otherwise stated, based on the acquired body parts measurements, it is possible to modify the reconstructed 3D avatars' mesh and the 3D surface to match the subject's actual body parts length. Accordingly, virtual dressing applications would become more accurate and realistic by developing algorithms consisting of the following steps:

- Data capturing,
- Segmentation and Denoising,
- Acquiring human body parts measurements of reconstructed 3D point clouds,
- Rigid and non-rigid alignment,
- Surface reconstruction,

Furthermore, the additional advantages of the proposed method in this research included minimizing the runtime required to generate the measurements (20 seconds) while implementing an error compensation strategy that allowed to acquire measurements with an accuracy between 92% and 100% on average.

Results and Conclusions

1. This thesis evaluted the performance of Microsoft Kinect v2 by measuring 10 different human body parameters. On average, an accuracy of 92% to 100% was obtained among various parameters.
2. The developed error compensation algorithm showed promising results of increased accuracy. In case of the height, the error was reduced from 6cm on average to 0cm while that of the hand from 4.15cm on average to 1.4cm.
3. This research's main contribution is the solution it provides to generate measurements with an average runtime of approximately 20 seconds, fewer captured frames (up to 10), and without the need for a second operator.
4. Quantitavely, the standard deviation of average absolute error of measurements in this research ($\sigma = 0.6$) outperformed that of the presented previous works ($\sigma >= 1$).
5. Future research should consider the potential effects of loose clothes more carefully, for example, by implementing an offset to minimize the resulting errors. Also, future studies should be devoted to increasing the robustness of the captured data when there is substantial motion and enhancing the scanned frames' quality. The former may require the use of higher resolution RGB-D sensors and more sophisticated denoising algorithms.

List of references

1. T. Liu, L. Li and X. Zhang, "Real-time 3D virtual dressing based on users' skeletons", in *4th International Conference on Systems and Informatics (ICSAI)*, Hangzhou, 2017, pp. 1378-1382, DOI: 10.1109/ICSAI.2017.8248501.
2. T. Jiang, "Three-Dimensional Data Registration in Laser Based 3D Scanning Reconstruction", in *8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, Hangzhou, 2016, pp. 457-460.
3. "Army to test Microsoft's Kinect in helicopter cockpits", April 2012, [online] Available from: http://www.livescience.com/19991-army-kinect-helicopter-cockpits.html.
4. M. Vij and S. Dawn, "Corrective Self Defence Training Unit using sensing of Kinect Maps", in *International Journal of Computer Applications*, 2014, 89(10):8-11, DOI: 10.5120/15665-3692.
5. A. Masri and M. Al-Jabi, "Virtual Dressing Room Application," in *IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Amman, Jordan, 2019, pp. 694-698.
6. S.B. Adikari, N.C. Ganegoda, R.G. Meegama, and I.L. Wanniarachchi. "Applicability of a Single Depth Sensor in Real-Time 3D Clothes Simulation: Augmented Reality Virtual Dressing Room Using Kinect Sensor", in *Advances in Human-Computer Interaction*, 2020, DOI: 10.1155/2020/1314598.
7. N. Yoshino, S. Karungaru, and K. Terada, "Body Physical Measurement Using Kinect for Vitual Dressing Room", in *6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, Hamamatsu, 2017, pp. 847-852, DOI: 10.1109/IIAI-AAI.2017.207.
8. R. Nakamura, M. Izutsu, and S. Hatakeyama, "Estimation Method of Clothes Size for Virtual Fitting Room with Kinect Sensor", in *IEEE International Conference on Systems, Man, and Cybernetics*, Manchester, 2013, pp. 3733-3738, DOI: 10.1109/SMC.2013.636.
9. Y.C. Du, C.B. Shih, S.C. Fan, H.T. Lin, and P.J. Chen, "An IMU-compensated skeletal tracking system using Kinect for the upper limb", in *Microsystem Technologies*, 2018, vol. 24, pp. 4317–4327.
10. O. Postolache, "Remote sensing technologies for physiotherapy assessment", in *10th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, Bucharest, 2017, pp. 305-312, DOI: 10.1109/ATEE.2017.7905141.
11. E. Silverstein and M. Snyder. "Comparative analysis of respiratory motion tracking using Microsoft Kinect v2 sensor", in *Journal of Applied Clinical Medical Physics*, 2018, vol. 19, pp. 193–204. DOI: 10.1002/acm2.12318.
12. A.M.T.S.B. Adikari, N.G.C. Ganegoda, and W.K.I.L. Wanniarachchi, "Non-Contact Human Body Parameter Measurement Based on Kinect Sensor", in *IOSR Journal of Computer Engineering*, 2017, 19(3), pp. 80-85, DOI: 10.9790/0661-1903028085.
13. I. Samejima, K. Maki, S. Kagami, M. Kouchi, and H. Mizoguchi, "A body dimensions estimation method of subject from a few measurement items using KINECT", in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Seoul, 2012, pp. 3384-3389, DOI: 10.1109/ICSMC.2012.6378315.
14. D. Seo, E. Kang, Y.M. Kim, S.Y. Kim, I.S. Oh, and M.G. Kim, "SVM-based waist circumference estimation using Kinect", in *Comput Methods Programs Biomed*, 2020, Jul;191:105418. DOI: 10.1016/j.cmpb.2020.105418. Epub 2020 Feb 24. PMID: 32126448.

15. H. Xu, Y. Yu, Y. Zhou, Y. Li, and S. Du, "Measuring accurate body parameters of dressed humans with large-scale motion using a Kinect sensor", in *Sensors*, 2013, 13(9), pp. 11362–11384, DOI: 10.3390/s130911362.
16. S. Tim, G. Eric, G. Christoph, W. Jurij, and L. Alexande, "Accuracy evaluation of two markerless motion capture systems for measurement of upper extremities: Kinect V2 and Captiv", in *Human Factors and Ergonomics in Manufacturing*, 2020, 30(4), pp. 291-302, DOI: 10.1002/hfm.20840.
17. D. Camila and B. Hurwitz, "Automatic Body Part Measurement of Dressed Humans Using Single RGB-D Camera", in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2016, pp. 3042-3048, DOI: 10.1145/2851581.2892337.
18. M. Grzegorzek, C. Theobalt, R. Koch, and A. Kolb, "Time-of-Flight and Depth Imaging. Sensors, Algorithms and Applications", in *Springer*, Berlin/Heidelberg, Germany, 2013, vol. 8200.
19. X. Wei, P. Zhang and J. Chai, "Accurate realtime full-body motion capture using a single depth camera", in *ACM Transactions on Graphics*, 2012, 31(6).
20. Y. Cui, W. Chang, T. Nöll, and D. Stricker, "Kinectavatar: fully automatic body capture using a single kinect", in *ACCV Proceedings of the 11th international conference on Computer Vision*, 2012, pp. 133–147.
21. L. Yang, L. Zhang, H. Dong, A. Alelaiwi, and A. E. Saddik, "Evaluating and Improving the Depth Accuracy of Kinect for Windows v2", in *IEEE Sensors Journal*, 2015, 15(8), pp. 4275-4285, DOI: 10.1109/JSEN.2015.2416651.
22. A. Mao, H. Zhang, Y. Liu, Y. Zheng, G. Li, and G. Han, "Easy and fast reconstruction of a 3D avatar with an RGB-D sensor", in *Sensors*, 2017, 17(5), DOI: 10.3390/s17051113.
23. J.T. Barron and J. Malik, "Shape, Illumination, and Reflectance from Shading", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1 August 2015, 37(8), pp. 1670-1687.
24. M. Kazhdan and H. Hugues, "Screened poisson surface reconstruction", in *ACM Transactions on Graphics*, 2013, vol. 32, DOI: 10.1145/2487228.2487237.
25. J. Tong, J. Zhou, L. Liu, Z. Pan, and H. Yan, "Scanning 3D Full Human Bodies Using Kinects", in *IEEE Transactions on Visualization and Computer Graphics*, April 2012, 18(4), pp. 643-650.
26. Y. Chen, G. Dang, Z.Q. Cheng, and K. Xu, "Fast capture of personalized avatar using two Kinects", in *Journal of Manufacturing Systems*, 2014, 33(1), pp. 233-240.
27. H. Zhu, Y. Yu, Y. Zhou, and S. Du, "Dynamic human body modeling using a single RGB camera", in *Sensors*, March 2016, 16(3), DOI: 10.3390/s16030402.
28. Y. Cui, S. Schuon, S. Thrun, D. Stricker, and C. Theobalt, "Algorithms for 3D Shape Scanning with a Depth Camera," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, May 2013, 35(5), pp. 1039-1050.
29. S. Hamed, L. Damien, and K. Andreas, "Kinect range sensing: structured-light versus time-of-flight Kinect", in *Computer Vision and Image Understanding*, 2015, vol. 139, pp. 1–20.
30. P. Diana and P. Livio, "Calibration of Kinect for Xbox One and Comparison between the Two Generations of Microsoft Sensors", in *Sensors*, 2015, 15(1), DOI: 27569-27589. 10.3390/s151127569.
31. S. Zennaro *et al*., "Performance evaluation of the 1st and 2nd generation Kinect for multimedia applications", in *IEEE International Conference on Multimedia and Expo (ICME)*, Turin, 2015, pp. 1-6, DOI: 10.1109/ICME.2015.7177380.

32. W. Olivier and S. Didier, "Comparison of Kinect V1 and V2 Depth Images in Terms of Accuracy and Precision", in *Computer Vision - ACCV International Workshops Part II*, 2016, pp. 34-45, DOI: 10.1007/978-3-319-54427-4_3.
33. C. Jing, J. Potgieter, F. Noble, and R. Wang, "A comparison and analysis of RGB-D cameras' depth performance for robotics application", in *24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, Auckland, 2017, pp. 1-6.
34. C. Chiu, T. Michael, S. Terry, C. Simon, H. John, and W. Jon, "Comparison of depth cameras for three-dimensional reconstruction in medicine", in *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, 2019, vol. 233, DOI: 095441191985992. 10.1177/0954411919859922.
35. L. Elise, M. Hélène, L. Tania, and G. Pierre, "Assessment and Calibration of a RGB-D Camera (Kinect v2 Sensor) Towards a Potential Use for Close-Range 3D Modeling", in *Remote Sensing*, 2015, vol. 7, pp. 13070-13097, DOI: 10.3390/rs71013070.
36. Siyi Deng (2021), text-to-speech, MATLAB Central File Exchange, [online] Available from: (https://www.mathworks.com/matlabcentral/fileexchange/18091-text-to-speech).
37. R. Sanne et al., "Human Action Recognition Using Hierarchic Body Related Occupancy Maps", in *Integrated Computer-Aided Engineering*, 2019, 26(3), pp. 223 – 241, DOI: 10.3233/ICA-190599.
38. A. Nath, "Image Denoising Algorithms: A Comparative Study of Different Filtration Approaches Used in Image Restoration", in *International Conference on Communication Systems and Network Technologies*, Gwalior, 2013, pp. 157-163, DOI: 10.1109/CSNT.2013.43.
39. P.Rajashekar Reddy, V.Amarnadh, and Mekala Bhaskar, "Evaluation of Stopping Criterion in Contour Tracing Algorithms", in *International Journal of Computer Science and Information Technologies*, 2012, vol. 3, pp. 3888-3894.
40. A. Fuster-Guilló, J. Azorín-López, M. Saval-Calvo, J.M. Castillo-Zaragoza, N. Garcia-D'Urso, and R.B. Fisher, "RGB-D-Based Framework to Acquire, Visualize and Measure the Human Body for Dietetic Treatments", in *Sensors*, 2020, 20(13), DOI: 10.3390/s20133690.
41. S. Roegiers, G. Allebosch, P. Veelaert, and W. Philips, "Human action recognition using hierarchic body related occupancy maps", in *Integrated Computer-Aided Engineering*, 2019, vol. 26, no. 3, pp. 223-241, DOI: 10.3233/ICA-190599.

Appendices

Appendix 1. Matlab code for frames capturing

```
%%                          %%%%%%%%%%%%%%Frames Capturing Main Code%%%%%%%%%%%

clc;clear;close all;

tic

%%                          %%%%%%%%%%%%%%Variables Declaration Section%%%%%%%%%%%
% create video object for color and depth  cameras
vid = videoinput('kinect',1,'BGR_1920x1080');
vid2 = videoinput('kinect',2,'Depth_512x424');

%-specific properties of the depth camera
srcDepth = getselectedsource(vid2);
srcDepth.EnableBodyTracking = 'on';

% number of frames to capture per  trigger
vid.FramesPerTrigger = 1;
vid2.FramesPerTrigger = 1;


%in order to acquire 201 frames from both the color sensor and the depth sensor.
%trigger can be called triggerRepeat + 1 time
vid.TriggerRepeat = 110;
vid2.TriggerRepeat = 110;

proceed = false;

framesArr = [];
index = 0;

%maximum allowed triggers per loop
maxTriggers = 110;
numTriggers = 1;

%maximum allowed frames per rotation view
maxAllowedFramesPerView = 10;

%directory
directory = 'images/data_set_subject/';
saveFileNameAs = 'all_frames_array_1';

triggerconfig([vid vid2],'manual');

start([vid vid2]);

%%                          %%%%%%%%%%%%%%Frames Capturing Section%%%%%%%%%%%
while ~proceed

    prompt = 'Start capturing frames? Y/N [Y]: ';
    %get user input and trim spaces
    userInput = strtrim(input(prompt,'s'));

    counter = 1;

    if (strlength(userInput)==1) && (userInput == 'y' || userInput == 'Y')
        % enable capturing
```

```
        pause(11);

        % allow capturing 2 frames only
        while (counter-1)~=maxAllowedFramesPerView

            numTriggers = numTriggers + 1;

            if numTriggers > maxTriggers
                % too many triggers with no tracking
                disp('No body was tracked');
                tts('Body Tracking Error','Microsoft Eva Mobile - English (United
States)',0)
                userInput = '';
                break;
            end
           % pause(2);
            trigger([vid vid2])

            % Get the acquired frames and metadata.
            [imgDepth, ts_depth, metaData_Depth] = getdata(vid2);
            [imgColor, ts_color, metaData_Color] = getdata(vid);

            %check if any body is tracked
            anyBodiesTracked = any(metaData_Depth.IsBodyTracked ~= 0);
            trackedBodies = find(metaData_Depth.IsBodyTracked);
            nBodies = length(trackedBodies);
            % A body was tracked
            if nBodies~=0
                counter = counter +1
                index = index+1;

                colorData = {imgColor, ts_color, metaData_Color};
                depthData = {imgDepth, ts_depth, metaData_Depth};
                framesArr{index,1}=colorData;
                framesArr{index,2}=depthData;

                %keep track of how many frames were captured
                capturedFrames = length(framesArr);

                if (counter-1) == maxAllowedFramesPerView
                    % 2 frames per view were captured
                    disp([num2str(maxAllowedFramesPerView),' frames per view were
captured']);
                    % make sound when all frames are captured
                    %load('splat')
                    %sound(y,Fs)
                    tts('Capturing Success','Microsoft Eva Mobile - English (United
States)',0)
                    userInput = '';
                    break;
                end
            end
        end
    elseif (strlength(userInput)==1) && (userInput == 'N' || userInput == 'n')
        % cancel capturing

        proceed = true;
        % imaqhwinfo
        stop([vid vid2]);
        break;
    end
end
```

```
%%                          %%%%%%%%%%%%%%Saving frames to Folder Section%%%%%%%%%%
%once all frames were captured
%check if any frames were captured
if ~isempty(framesArr)
    %check if directory folder exists and create it if not
    if not(isfolder(directory))
        mkdir(directory)
    end

    save(strcat(directory,saveFileNameAs,'.mat'),'framesArr');
else
    disp("No frames were captured");
    tts('No frames were captured','Microsoft Eva Mobile - English (United States)',0)
end

%%
toc;
```

Appendix 2. Matlab code for segmentation and 3D point cloud

```
clc;clear;close all;

tic

dirName = 'data_set_subject/';

framesDir = strcat('images/',dirName);
framesList=dir(fullfile(framesDir,'**/*.mat')); %%find all frames meta data
framesCounter=size(framesList,1);

for i=1:framesCounter
    framesDataSet{i,1}= load(strcat(framesDir,framesList(i).name));
end

segmentedDirectory = strcat('segmented_point_cloud/',dirName);

colorDevice = imaq.VideoDevice('kinect',1)
depthDevice = imaq.VideoDevice('kinect',2) %Create a System object for the depth device.

%Initialize the camera
step(colorDevice);
step(depthDevice);

for j=1:framesCounter

    framesArr = framesDataSet{j,1}.framesArr;

    for i=1:length(framesArr)

        imgColor = framesArr{i,1}{1};
        imgDepth = framesArr{i,2}{1};

        metaData_Depth = framesArr{i,2}{3};
        imD = metaData_Depth.BodyIndexFrame;
        %imB = ~(imbinarize(imD)); % inverse of binary image
        % inverse of binary image (substract a value bigger than max value of
        % tracked human bodies which is 6 as imb works only with index 0 and 1
        imB = ~(imbinarize(imD-10));
        imgDepthSegmented=imgDepth.*uint16(imB);
```

```
        ptCloud = pcfromkinect(depthDevice, imgDepthSegmented, imgColor,'depthCentric');

        allSegmentedFramesArr{i,j} = ptCloud;
    end

end

release(colorDevice);
release(depthDevice);

% creatr directory if it doesnt exist
if not(isfolder(segmentedDirectory))
        mkdir(segmentedDirectory)
end

[rows,cols] = size(allSegmentedFramesArr);

for i=1:cols
    % read all columns of row i and make arr as rows arr
    segmentedFramesArr = allSegmentedFramesArr(:,i);

save(strcat(segmentedDirectory,'all_segmented_frames_array_',num2str(i),'.mat'),'segmente
dFramesArr');
end

toc
```

Appendix 3. Matlab code for generating the final human body parts parameters

```
clc;clear;close all;
tic

%%                                      Global variables

framesDataSet = [];
ptCloudFramesDataSet= [];

edgesCoordinatesArr = [];

theta = 180;

measuredHeightArr = [];
heightMeasurementDetails = [];

measuredArmLArr = [];
armLMeasurementDetails = [];

measuredArmRArr = [];
armRMeasurementDetails = [];

measuredHandLArr = [];
handLMeasurementDetails = [];
handLPoints = ([7, 22]);   % wristL, handtipL

measuredHandRArr = [];
handRMeasurementDetails = [];
handRPoints = ([11, 24]);   % wristR, handtipR

measuredHipArr = [];
hipMeasurementDetails = [];
```

```
measuredNeckToHipArr = [];
neckToHipMeasurementDetails = [];
%neckToHipPoints = ([3, 1]);   % Neck, SpineBase
neckToHipPoints =  [ [3, 21];   % Neck, SpineShoulder
                     [21, 2];   % SpineShoulder, SpineMid
                     [2, 1];    % SpineMid, SpineBase
                   ];

measuredShoulderArr = [];
shoulderMeasurementDetails = [];
shoulderPoints =  [ [5, 21];   % ShoulderLeft, SpineShoulder
                    [21, 9];   % SpineShoulder, ShoulderRight
                  ];

measuredLegLArr = [];
legLMeasurementDetails = [];

measuredLegRArr = [];
legRMeasurementDetails = [];

measuredWaistArr = [];
waistMeasurementDetails = [];

measuredStomachArr = [];
stomachMeasurementDetails = [];

measuredChestArr = [];
chestMeasurementDetails = [];

% (S1)
actualHeightLength = 182; %cm
actualArmLength = 81;
actualHandLength = 20;
actualHipLength = 44;
actualLegLength = 103;
actualNeckToHipLength = 60;
actualShoulderLength = 40;
actualWaistLength = 43;
actualStomachLength = 42.5;
actualChestLength = 46;
actualBodyMeasurementArr = [182,81,81,20,20,44,103,103,60,40,43,42.5,46];

tPose = 1; % flag for human pose

% plotting variables
dataSetIdx = 1;
plotIdx = 1;
plotRows = 7;
plotCols = 2;
plot_xLabel = "Nbr of frames";
plot_yLabel = " Absolute error in cm ";


measuredValuesArrBeforeCompensation = [];

% error compensation variables
finalBodyMeasurementValuesArr = [];
if tPose==1
    namesOfHumanBodyParts=
{'Height','ArmL','ArmR','HandL','HandR','Hip','LegL','LegR','NeckToHip','Shoulder','Waist
','Stomach','Chest'};
else
```

```
    namesOfHumanBodyParts=
{'Height','ArmL','ArmR','Hip','LegL','LegR','NeckToHip','Shoulder','Waist','Stomach','Che
st'};
end
finalSubjectsMeasurementsValuesDir=
'research_data_set_subjects_measurements/denoised_results/';

%%%%%%%%%%%%%%%%%%%%%%%%% compenstion values %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Height
heightCompensationErr = -6.1130;

% Arm
armLCompensationErr = -5.2324;
armRCompensationErr = -7.0760;

% Hand
handLCompensationErr = 3.8556;
handRCompensationErr = 1.8896;

% Hip
hipCompensationErr = -5.8623;

% Leg
legLCompensationErr = -9.4289;
legRCompensationErr = -9.4106;

% NeckToHip
neckToHipCompensationErr = 2.2251;

% Shoulder
shoulderCompensationErr = -1.8283;

% Waist
waistCompensationErr = -0.8839;

% Stomach
stomachCompensationErr = 8.3025;

% Chest
chestCompensationErr = 5.6477;

finalSubjectsMeasurementsValuesDirArrName = 'final_human_body_measured_data_subject_1';

%%                                      Load frames and point clouds
dirName = 'data_set_subject_1/';
framesDir = strcat('images/',dirName);
framesList=dir(fullfile(framesDir,'**/*.mat')); %%find all frames meta data
framesCounter=size(framesList,1);

for i=1:framesCounter
    framesDataSet{i,1}= load(strcat(framesDir,framesList(i).name));
end

ptCloudFramesDir = strcat('segmented_point_cloud/',dirName);
ptCloudFramesList=dir(fullfile(ptCloudFramesDir,'**/*.mat')); %%find all frames meta data
framesCounter=size(ptCloudFramesList,1);

for i=1:framesCounter
    ptCloudFramesDataSet{i,1}= load(strcat(ptCloudFramesDir,ptCloudFramesList(i).name));
end
```

```
%%                                  Loop through frames of first data set only

if length(framesDataSet)~=length(ptCloudFramesDataSet)
    disp('Error! Number of loaded frames and point cloud files do not match')
    return; % break execution
else

    framesArr = framesDataSet{1,1}.framesArr;
    segmentedFramesArr = ptCloudFramesDataSet{1,1}.segmentedFramesArr;

    if length(framesArr)~=length(segmentedFramesArr)
         disp('Error! Number of frames and segmented frames do not match')
         return; % break execution
    else
       for i=1:length(framesArr)
            % read depth data
            metaData_Depth = framesArr{i,2}{3};

            % flipped segmented depth frame to become x,y
            imD = metaData_Depth.BodyIndexFrame;
            % inverse of binary image
            %imB = ~(imbinarize(imD-10));

            ptCloud = segmentedFramesArr{i,1};

            % flip ptcloud to make it have same as depth
            ptCloud             =             pointCloud(fliplr(ptCloud.Location),'Color',
fliplr(ptCloud.Color));

            %%%%%% denoising using 2d average filter%%%%%%%%%%%%%%%%%%%%
            xLocation = ptCloud.Location(:,:,1);
            yLocation = ptCloud.Location(:,:,2);
            zLocation = ptCloud.Location(:,:,3);

            % filter depth image which is z
            filteredZ = filter2(fspecial('average',2),zLocation);

            tmp1 = zeros(size(xLocation));
            tmp1(find(xLocation)) = 1;
            tmp2 = zeros(size(filteredZ));
            tmp2(find(filteredZ)) = 1;

            % new filtered and denoised pointcloud
            newXLocation = tmp1.*tmp2.*xLocation;
            newYLocation = tmp1.*tmp2.*yLocation;
            newZLocation = tmp1.*tmp2.*filteredZ; % new denoised depth image

            ptCloudOut                                                                  =
pointCloud(cat(3,newXLocation,newYLocation,newZLocation),'Color',ptCloud.Color);

            ptCloud = ptCloudOut;

            % segmented and denoised binary image
            imB = imbinarize(newZLocation);

            % get full depth image coordinates
            [fullDepthYCol, fullDepthXCol] = find(imB);

            % get contour and its x and y coordinates
            contour = bwboundaries(imB,'noholes');
            edgesCoordinatesArr{i,1} = contour{1};
```

```
            if isempty(edgesCoordinatesArr{i,1})
               disp(['Error! Contour coordinates of frame ',num2str(i), ' were not
found!']);
               return;
            end

            depthXEdgeCol = edgesCoordinatesArr{i,1}(:,2);
            depthYEdgeCol = edgesCoordinatesArr{i,1}(:,1);


            % 3d joint positions and 2d depth joints coordinates
            jointPositions                                                    =
metaData_Depth.JointPositions(:,:,metaData_Depth.IsBodyTracked);
            %rotate 180 degree related to z
            jointPositions = ([cosd(theta),      -sind(theta),    0;...
                               sind(theta),       cosd(theta),    0;...
                                 0,                 0,            1]*jointPositions')';

            headJointPoints = jointPositions(4,:,:);
            footLJointPoints = jointPositions(16,:,:);
            footRJointPoints = jointPositions(20,:,:);

            depthJoints                                                       =
metaData_Depth.DepthJointIndices(:,:,metaData_Depth.IsBodyTracked);
            depthHipLPoints = depthJoints(13,:);
            depthHipRPoints = depthJoints(17,:);
            depthFootLPoints = depthJoints(16,:);
            depthFootRPoints = depthJoints(20,:);
            depthShoulderLPoints = depthJoints(5,:);
            depthHeadPoints = depthJoints(4,:);
            depthSpineBasePoints = depthJoints(1,:);
            depthSpineMidPoints = depthJoints(2,:);
            depthSpineShoulderPoints = depthJoints(21,:);

            %%                          Find Height Points
            % find height highest closest edge point to the skeleton head point
            heightHeadPointIdx                                                =
find(depthXEdgeCol==fix(depthHeadPoints(1))&depthYEdgeCol<depthHeadPoints(2),1);
            heightHeadPointArr{i,1}                                           =
ptCloud.Location(depthYEdgeCol(heightHeadPointIdx),depthXEdgeCol(heightHeadPointIdx),:);

            % height lowest leg point
            heightLegPointIdx = find(depthYEdgeCol==max(depthYEdgeCol),1);
            heightLegPointArr{i,1}                                            =
ptCloud.Location(depthYEdgeCol(heightLegPointIdx),depthXEdgeCol(heightLegPointIdx),:);


            %%                          Find Arm Points

           % Left Points
           armLPoint1 = jointPositions(5,:,:); % shoulder left point
           armLPointArr{i,1} = [armLPoint1(1),armLPoint1(2),armLPoint1(3)];
           armLPoint2 = find(depthXEdgeCol==min(depthXEdgeCol),1);
           armLPointArr{i,2}                                                  =
ptCloud.Location(depthYEdgeCol(armLPoint2),depthXEdgeCol(armLPoint2),:);

           % Right Points
           armRPoint1 = jointPositions(9,:,:); % shoulder right point
           armRPointArr{i,1} = [armRPoint1(1),armRPoint1(2),armRPoint1(3)];
           armRPoint2 = find(depthXEdgeCol==max(depthXEdgeCol),1);
           armRPointArr{i,2}                                                  =
ptCloud.Location(depthYEdgeCol(armRPoint2),depthXEdgeCol(armRPoint2),:);
```

```
            %%                          Find Hand Points

            % if tPose is enabled then allow obtaining hand points of human
            if tPose==1
                % right points
                handRPoint1 = jointPositions(11,:,:); % wrist right point
                handRPoint2 = jointPositions(24,:,:); % hand tip right point
                handRPointArr{i,1} = [handRPoint1(1),handRPoint1(2),handRPoint1(3)];
                handRPointArr{i,2} = [handRPoint2(1),handRPoint2(2),handRPoint2(3)];

                % left points
                handLPoint1 = jointPositions(7,:,:); % wrist left point
                handLPoint2 = jointPositions(22,:,:); % hand tip left point
                handLPointArr{i,1} = [handLPoint1(1),handLPoint1(2),handLPoint1(3)];
                handLPointArr{i,2} = [handLPoint2(1),handLPoint2(2),handLPoint2(3)];
            end

            %%                              Find Hip Points

            %hip edge L
            hipLPointIdx = find(depthYEdgeCol==fix(depthHipLPoints(2)),1);

            hipAllDepthPointsIdx = find(fullDepthYCol==depthYEdgeCol(hipLPointIdx));
            for k=1:length(hipAllDepthPointsIdx)
                 hipPointsArr{k,i}                                                                  =
ptCloud.Location(fullDepthYCol(hipAllDepthPointsIdx(k)),fullDepthXCol(hipAllDepthPointsId
x(k)),:);
            end

            %%                              Find Leg Points

            % right points
            legRPoint1 = jointPositions(17,:,:); % hip right point
            legRPointArr{i,1} = [legRPoint1(1),legRPoint1(2),legRPoint1(3)];
            LegRPoint2Idx                                                                           =
find(depthXEdgeCol==fix(depthFootRPoints(1))&depthYEdgeCol>depthFootRPoints(2),1);
            legRPointArr{i,2}                                                                       =
ptCloud.Location(depthYEdgeCol(LegRPoint2Idx),depthXEdgeCol(LegRPoint2Idx),:);

            % left points
            legLPoint1 = jointPositions(13,:,:); % hip left point
            legLPointArr{i,1} = [legLPoint1(1),legLPoint1(2),legLPoint1(3)];
            LegLPoint2Idx                                                                           =
find(depthXEdgeCol==fix(depthFootLPoints(1))&depthYEdgeCol>depthFootLPoints(2),1);
            legLPointArr{i,2}                                                                       =
ptCloud.Location(depthYEdgeCol(LegLPoint2Idx),depthXEdgeCol(LegLPoint2Idx),:);


            %%                              Find Neck to Hip Points

            measuredNeckToHipArr                                                                    =
[measuredNeckToHipArr,distanceSum(jointPositions,neckToHipPoints).*100];

            %%                              Find Shoulder Points

            measuredShoulderArr                                                                     =
[measuredShoulderArr,distanceSum(jointPositions,shoulderPoints).*100];

            %%                              Find Waist Points

            % waist edge L
```

```
            waistPointsIdx                                                                  =
find(depthYEdgeCol<depthSpineBasePoints(2)&depthYEdgeCol>depthSpineMidPoints(2));
            depthSpineMidBasePoints                                                         =
[depthSpineMidPoints(1),depthSpineMidPoints(2);depthSpineBasePoints(1),depthSpineBasePoin
ts(2)];
            spineMidBaseDistance  =  pdist(depthSpineMidBasePoints,'euclidean');  %euclidean
distance between points

            calculatedWaistDistances = [];

            for k=1:length(waistPointsIdx)
                 points                                                                     =
[depthXEdgeCol(hipLPointIdx),depthYEdgeCol(hipLPointIdx);depthXEdgeCol(waistPointsIdx(k))
,depthYEdgeCol(waistPointsIdx(k))];
                 calculatedWaistDistances                                                   =
[calculatedWaistDistances,pdist(points,'euclidean')];
            end

            [minDistance,      waistPointLIdx]      =      min(abs(calculatedWaistDistances-
(spineMidBaseDistance.*1/3)));

            waistAllDepthPointsIdx                                                          =
find(fullDepthYCol==depthYEdgeCol(waistPointsIdx(waistPointLIdx)));
            for k=1:length(waistAllDepthPointsIdx)
                 waistPointsArr{k,i}                                                        =
ptCloud.Location(fullDepthYCol(waistAllDepthPointsIdx(k)),fullDepthXCol(waistAllDepthPoin
tsIdx(k)),:);
            end

            %%                          Find Stomach Points

            % stomach edge L
            stomachPointsIdx = waistPointsIdx;
            calculatedStomachDistances = [];

            for k=1:length(stomachPointsIdx)
                 points                                                                     =
[depthXEdgeCol(hipLPointIdx),depthYEdgeCol(hipLPointIdx);depthXEdgeCol(stomachPointsIdx(k
)),depthYEdgeCol(stomachPointsIdx(k))];
                 calculatedStomachDistances                                                 =
[calculatedStomachDistances,pdist(points,'euclidean')];
            end

            [minDistance,    stomachPointLIdx]    =    min(abs(calculatedStomachDistances-
(spineMidBaseDistance.*3/4)));

            stomachAllDepthPointsIdx                                                        =
find(fullDepthYCol==depthYEdgeCol(stomachPointsIdx(stomachPointLIdx)));
            for k=1:length(stomachAllDepthPointsIdx)
                 stomachPointsArr{k,i}                                                      =
ptCloud.Location(fullDepthYCol(stomachAllDepthPointsIdx(k)),fullDepthXCol(stomachAllDepth
PointsIdx(k)),:);
            end
            %%                           Find Chest Points

            %chest edge L
            % find spine mid edges points (1 right and 1 left)
            spineMidPointsIdx = find(depthYEdgeCol==fix(depthSpineMidPoints(2)));
            chestPointsIdx                                                                  =
find(depthYEdgeCol<depthSpineMidPoints(2)&depthYEdgeCol>depthSpineShoulderPoints(2));
```

```
            spineMidShoulderPoints                                                  =
[depthSpineMidPoints(1),depthSpineMidPoints(2);depthSpineShoulderPoints(1),depthSpineShou
lderPoints(2)];
            spineMidShoulderDistance      =      pdist(spineMidShoulderPoints,'euclidean');
%euclidean distance between points

            calculatedChestDistances = [];

            for k=1:length(chestPointsIdx)

                points                                                              =
[depthXEdgeCol(spineMidPointsIdx(2)),depthYEdgeCol(spineMidPointsIdx(2));depthXEdgeCol(ch
estPointsIdx(k)),depthYEdgeCol(chestPointsIdx(k))];
                calculatedChestDistances                                            =
[calculatedChestDistances,pdist(points,'euclidean')];
            end

            [minDistance,      ChestPointLIdx]      =      min(abs(calculatedChestDistances-
(spineMidShoulderDistance.*1/3)));
            chestPoint1Arr{i,1}                                                     =
ptCloud.Location(depthYEdgeCol(chestPointsIdx(ChestPointLIdx)),depthXEdgeCol(chestPointsI
dx(ChestPointLIdx)),:);

            chestAllDepthPointsIdx                                                  =
find(fullDepthYCol==depthYEdgeCol(chestPointsIdx(ChestPointLIdx)));
            for k=1:length(chestAllDepthPointsIdx)
                chestPointsArr{k,i}                                                 =
ptCloud.Location(fullDepthYCol(chestAllDepthPointsIdx(k)),fullDepthXCol(chestAllDepthPoin
tsIdx(k)),:);
            end

        end

        %%                                  Calculate Measurements

         % measure height from obtained points
         if length(heightHeadPointArr)~=length(heightLegPointArr)
             disp('Error! Unable to measure height length as number of top and bottom points
do not match');
             return; % break execution
         else
             for i=1:length(heightHeadPointArr)
                 if ~isnan(heightHeadPointArr{i,1}) & ~isnan(heightLegPointArr{i,1})
                     measuredHeightArr                                              =
[measuredHeightArr,distance(heightHeadPointArr{i,1},heightLegPointArr{i,1})];
                 else
                    disp(['Error! Height points of frame ',num2str(i),' are nan']);
                    return; % break execution
                 end
             end
             measuredHeightArr = measuredHeightArr.*100; % m to cm
         end

         % measure arm from obtained points
         if length(armLPointArr)~=length(armRPointArr)
             disp('Error! Unable to measure arm length as number of left and right points
do not match');
             return; % break execution
         else
             for i=1:length(armRPointArr)
                 if     ~isnan(armLPointArr{i,1})     &     ~isnan(armLPointArr{i,2})     &
~isnan(armRPointArr{i,1}) & ~isnan(armRPointArr{i,2})
```

```
                armLengthL = distance(armLPointArr{i,1},armLPointArr{i,2});
                armLengthR = distance(armRPointArr{i,1},armRPointArr{i,2});

                measuredArmLArr = [measuredArmLArr, armLengthL];
                measuredArmRArr = [measuredArmRArr, armLengthR];
            else
               disp(['Error! Arm points of frame ',num2str(i),' are nan']);
               return; % break execution
            end
        end
        measuredArmLArr = measuredArmLArr.*100; % m to cm
        measuredArmRArr = measuredArmRArr.*100; % m to cm
    end

    % measure hand from obtained points
    % if tPose is enabled then allow measuring hand length of human
    if tPose==1
       if length(handRPointArr)~=length(handLPointArr)
         disp('Error! Unable to measure hand length as number of left and right points
do not match');
         return; % break execution
       else
            for i=1:length(handRPointArr)
                if ~isnan(handRPointArr{i,1}) & ~isnan(handLPointArr{i,1})
                    handLengthR = distance(handRPointArr{i,1},armRPointArr{i,2});
                    handLengthL = distance(handLPointArr{i,1},armLPointArr{i,2});

                    measuredHandLArr = [measuredHandLArr,handLengthL];
                    measuredHandRArr = [measuredHandRArr,handLengthR];
                else
                    disp(['Error! Hand points of frame ',num2str(i),' are nan']);
                    return; % break execution
                end
            end
        end
        measuredHandLArr = measuredHandLArr.*100; % m to cm
        measuredHandRArr = measuredHandRArr.*100; % m to cm
    end
    % measure hip from obtained points
    [numRowsHipPoints,numColsHipPoints] = size(hipPointsArr);
    hipLength = 0;
    if numColsHipPoints~=length(framesArr)
       disp('Error! Measurements of the hip of one of the frames is missing!');
       return; % break execution
    else
       for i=1:numColsHipPoints
            for k=1:(numRowsHipPoints-1)
                p1 = hipPointsArr{k,i};
                p2 = hipPointsArr{k+1,i};
                if isempty(p1) || isempty(p2)
                    continue;
                elseif isnan(p1) | isnan(p2)
                    disp(['Error! Hip points of frame ',num2str(i),' are nan']);
                    return; % break execution
                else
                    hipLength = hipLength + distance(p1,p2);
                end
            end

            measuredHipArr = [measuredHipArr,hipLength];
            hipLength = 0;
       end
```

```
            measuredHipArr = measuredHipArr.*100; % m to cm
        end
        hipPointsArr = []; % reset array

        % measure legs from obtained points
        if length(legRPointArr)~=length(legLPointArr)
            disp('Error! Unable to measure leg length as number of left and right points
do not match');
            return; % break execution
        else
            for i=1:length(legRPointArr)
                if     ~isnan(legLPointArr{i,1})     &     ~isnan(legLPointArr{i,2})     &
~isnan(legRPointArr{i,1}) & ~isnan(legRPointArr{i,2})
                    legLengthR = distance(legRPointArr{i,1},legRPointArr{i,2});
                    legLengthL = distance(legLPointArr{i,1},legLPointArr{i,2});

                    measuredLegRArr = [measuredLegRArr,legLengthR];
                    measuredLegLArr = [measuredLegLArr,legLengthL];
                else
                    disp(['Error! Leg points of frame ',num2str(i),' are nan']);
                    return; % break execution
                end
            end
            measuredLegRArr = measuredLegRArr.*100; % m to cm
            measuredLegLArr = measuredLegLArr.*100; % m to cm
        end
        % measure waist from obtained points
        [numRowsWaistPoints,numColsWaistPoints] = size(waistPointsArr);
        waistLength = 0;
        if numColsWaistPoints~=length(framesArr)
           disp('Error! Measurements of the waist of one of the frames is missing!');
           return; % break execution
        else
           for i=1:numColsWaistPoints
                for k=1:(numRowsWaistPoints-1)
                    p1 = waistPointsArr{k,i};
                    p2 = waistPointsArr{k+1,i};
                    if isempty(p1) || isempty(p2)
                        continue;
                    elseif isnan(p1) | isnan(p2)
                        disp(['Error! Waist points of frame ',num2str(i),' are nan']);
                        return; % break execution
                    else
                        waistLength = waistLength+ distance(p1,p2);
                    end
                end

                measuredWaistArr = [measuredWaistArr,waistLength];
                waistLength = 0;
           end
           measuredWaistArr = measuredWaistArr.*100; % m to cm
        end
        waistPointsArr = []; % reset array

        % measure stomach from obtained points
        [numRowsStomachPoints,numColsStomachPoints] = size(stomachPointsArr);
        stomachLength = 0;
        if numColsStomachPoints~=length(framesArr)
           disp('Error! Measurements of the stomach of one of the frames is missing!');
           return; % break execution
        else
           for i=1:numColsStomachPoints
```

```
                for k=1:(numRowsStomachPoints-1)
                    p1 = stomachPointsArr{k,i};
                    p2 = stomachPointsArr{k+1,i};
                    if isempty(p1) || isempty(p2)
                        continue;
                    elseif isnan(p1) | isnan(p2)
                        disp(['Error! Stomach points of frame ',num2str(i),' are nan']);
                        return; % break execution
                    else
                        stomachLength = stomachLength+ distance(p1,p2);
                    end
                end

                measuredStomachArr = [measuredStomachArr,stomachLength];
                stomachLength = 0;
            end
            measuredStomachArr = measuredStomachArr.*100; % m to cm
        end
        stomachPointsArr = []; % reset array

        % measure chest from obtained points
        [numRowsChestPoints,numColsChestPoints] = size(chestPointsArr);
        chestLength = 0;
        if numColsChestPoints~=length(framesArr)
            disp('Error! Measurements of the chest of one of the frames is missing!');
            return; % break execution
        else
            for i=1:numColsChestPoints
                for k=1:(numRowsChestPoints-1)
                    p1 = chestPointsArr{k,i};
                    p2 = chestPointsArr{k+1,i};
                    if isempty(p1) || isempty(p2)
                        continue;
                    elseif isnan(p1) | isnan(p2)
                        disp(['Error! Chest points of frame ',num2str(i),' are nan']);
                        return; % break execution
                    else
                        chestLength = chestLength+ distance(p1,p2);
                    end
                end

                measuredChestArr = [measuredChestArr,chestLength];
                chestLength = 0;
            end
            measuredChestArr = measuredChestArr.*100; % m to cm
        end
        chestPointsArr = []; % reset array

    end

    %%                  obtain new measurements values after compensation for other data
set

    % Height (arr with the measuredHeight values before and after compensation)
    heightMeasurementDetails      =      {measuredHeightArr,      measuredHeightArr      -
heightCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(heightMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(heightMeasurementDetails{1}))];

    % Arm
```

```
    armLMeasurementDetails = {measuredArmLArr, measuredArmLArr - armLCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(armLMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(armLMeasurementDetails{1}))];

    armRMeasurementDetails = {measuredArmRArr, measuredArmRArr - armRCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(armRMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(armRMeasurementDetails{1}))];

    % Hand
    if tPose==1
        handLMeasurementDetails     =     {measuredHandLArr,     measuredHandLArr     -
handLCompensationErr};
        finalBodyMeasurementValuesArr          =          [finalBodyMeasurementValuesArr,
round(mean(handLMeasurementDetails{2}))];
        measuredValuesArrBeforeCompensation    =    [measuredValuesArrBeforeCompensation,
round(mean(handLMeasurementDetails{1}))];

        handRMeasurementDetails     =     {measuredHandRArr,     measuredHandRArr     -
handRCompensationErr};
        finalBodyMeasurementValuesArr          =          [finalBodyMeasurementValuesArr,
round(mean(handRMeasurementDetails{2}))];
        measuredValuesArrBeforeCompensation    =    [measuredValuesArrBeforeCompensation,
round(mean(handRMeasurementDetails{1}))];
    end

    % Hip
    hipMeasurementDetails = {measuredHipArr, measuredHipArr - hipCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(hipMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(hipMeasurementDetails{1}))];


    % Leg
    legLMeasurementDetails = {measuredLegLArr, measuredLegLArr - legLCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(legLMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(legLMeasurementDetails{1}))];

    legRMeasurementDetails = {measuredLegRArr, measuredLegRArr - legRCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(legRMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(legRMeasurementDetails{1}))];


    % NeckToHip
    neckToHipMeasurementDetails    =    {measuredNeckToHipArr,    measuredNeckToHipArr    -
neckToHipCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(neckToHipMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(neckToHipMeasurementDetails{1}))];


    % Shoulder
```

```
    shoulderMeasurementDetails    =    {measuredShoulderArr,    measuredShoulderArr    -
shoulderCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(shoulderMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(shoulderMeasurementDetails{1}))];

    % Waist
    waistMeasurementDetails = {measuredWaistArr, measuredWaistArr - waistCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(waistMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(waistMeasurementDetails{1}))];

    % Stomach
    stomachMeasurementDetails    =    {measuredStomachArr,    measuredStomachArr    -
stomachCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(stomachMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(stomachMeasurementDetails{1}))];

    % Chest
    chestMeasurementDetails = {measuredChestArr, measuredChestArr - chestCompensationErr};
    finalBodyMeasurementValuesArr            =            [finalBodyMeasurementValuesArr,
round(mean(chestMeasurementDetails{2}))];
    measuredValuesArrBeforeCompensation      =      [measuredValuesArrBeforeCompensation,
round(mean(chestMeasurementDetails{1}))];



    %%                                Save compensation errors into one final array

    % merge array of compensations with the name of body parts
    finalHumanBodyMeasuredDataArr                                                    =
[namesOfHumanBodyParts;num2cell(finalBodyMeasurementValuesArr);namesOfHumanBodyParts;...

num2cell(measuredValuesArrBeforeCompensation);namesOfHumanBodyParts;num2cell(actualBodyMe
asurementArr)];

    % creatr directory if it doesnt exist
    if not(isfolder(finalSubjectsMeasurementsValuesDir))
            mkdir(finalSubjectsMeasurementsValuesDir)
    end


save(strcat(finalSubjectsMeasurementsValuesDir,finalSubjectsMeasurementsValuesDirArrName,
'.mat'),'finalHumanBodyMeasuredDataArr');

    toc

end


%%                                functions

% get distance between two points
function length = distance(P1, P2)
   length = sqrt((P1(1) - P2(1))^2 + (P1(2) - P2(2))^2 + (P1(3) - P2(3))^2);
end
```

```
% get distance in meters between group of points
function result = distanceSum(jointPositions,jointPoints)
    sum = 0;

    [row col] = size(jointPoints);

    if row==1
        P1 = jointPositions(jointPoints(1,1),:,:);
        P2 = jointPositions(jointPoints(1,2),:,:);
        sum = sum + distance(P1,P2);
    elseif row>1
        for i=1:length(jointPoints)
            P1 = jointPositions(jointPoints(i,1),:,:);
            P2 = jointPositions(jointPoints(i,2),:,:);
            sum = sum + distance(P1,P2);
        end
    end

    result = sum;
end
```