Posture and Fall Detection System Using 3D Motion Sensors

Hamza Qassoud *, Miodrag Bolic *, Sreeraman Rajan†
*School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Ontario, Canada
*Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada
Email: {hqass076@uottawa.ca, mbolic@eecs.uottawa.ca, sreeramanr@sce.carleton.ca}

***Abstract*—This work presents a supervised learning approach for training a posture detection classifier, and implementing a fall detection system using the posture classification results as inputs with a Microsoft Kinect v2 sensor. The Kinect v2 skeleton tracking provides 3D depth coordinates for 25 body parts. We use these depth coordinates to extract seven features consisting of the height of the subject and six angles between certain body parts. These features are then fed into a fully connected neural network that outputs one of three considered postures for the subject: standing, sitting, or lying down. An average classification rate of over 99.30% for all three postures was achieved on test data consisting of multiple subjects where the subjects were not even facing the Kinect depth camera most of the time and were located in different locations. These results show the feasibility to classify human postures with the proposed setup independently of the location of the subject in the room and orientation to the 3D sensor.**

***Index Terms*—Fall detection systems; Posture detection systems; Supervised learning; Machine learning.**

I. Introduction

Two classes of devices can be used for posture classification: wearable (contact-based) and contactless. Wearable devices are attached to different parts of the human body. As an example, a strain sensor was used for measuring strain in the clothing of the subject, eventually leading to classify 27 different upper body postures [1]. In [2], various wearable sensors such as 3-axis accelerometers, 3D gyroscope and 3D magnetometer (3D compass) were placed on different body locations to compute a 3D orientation estimates with high accuracy and with no drift for both static and dynamic movements.

The contactless posture classification is mostly achieved using image and video processing principles. There are several devices such as radars, sonars, and cameras which can be used for posture classification. For instance, in [3] a background subtraction on the depth image of the human body to extract a silhouette contour of a human was presented. A horizontal projection of the silhouette contour was then employed to determine whether or not the subject is kneeling. When not kneeling, star skeleton technique was applied to the silhouette contour to obtain its feature points. These points were then used, alongside with the centre of gravity, to calculate the feature vectors and depth values of the body. These calculated values were then fed into a pre-trained LVQ (learning vector quantization) neural network to obtain one of the four possible postures: standing, sitting, stooping, or lying down. The proposed method in [4] used a head model to locate the human position. This work used image processing techniques such as edge extraction, template matching to achieve human detection. The HOG (histogram of oriented gradients) feature was then extracted from the depth images to get the characteristic vector of the original image. Finally, a generalized regression neural network was used to classify and identify human postures.

Our proposed work presents a method using only the Microsoft Kinect v2 sensor. We present a system to accurately classify the posture of the subject in any frame containing the human body in real-time. As training a neural network using supervised learning to perform classification can take a long time, a general purpose graphics processing units (GP-GPU) are leveraged to accelerate neural network training and runtime by executing in vector form on highly parallel and pipelined processing units rather than sequential processing with a Central Processing Unit (CPU) [5]. To accomplish speedup, TensorFlow or Theano is used to load and store data into and out of the GP-GPU memory and from the hard disk [6] [7]. An NVIDIA GeForce 940M GPU was used in this work.

In this paper, we present an efficient, and real-time posture classification system that uses 3D depth motion sensor inputs. We also implement a fall detection system by detecting whether the detected subject has been lying down on the floor for more than two seconds. We also provide the capability to different a subject from an inanimate object. We achieve our set objective of posture classification in three steps. In the first step, we extract features from the raw data that the motion sensor outputs which is presented in Section II. In the second step, we train a classifier using the features as input to determine the posture of the subject which is described in Section III). In the third step, using the posture classification output obtained second step, we detect falls as described in Section IV).

II. Features Extraction From Motion Sensor Raw Data

The Microsoft Kinect V2 sensor uses infrared (IR) camera to recognize up to six users in its field of view and provides skeletal tracking features to up to two of these users. The sensor switches between 15 and 30 FPS (frames per second) depending on lighting conditions. Using the Kinect libraries, one can extract the 3D depth coordinates of the desired body parts.

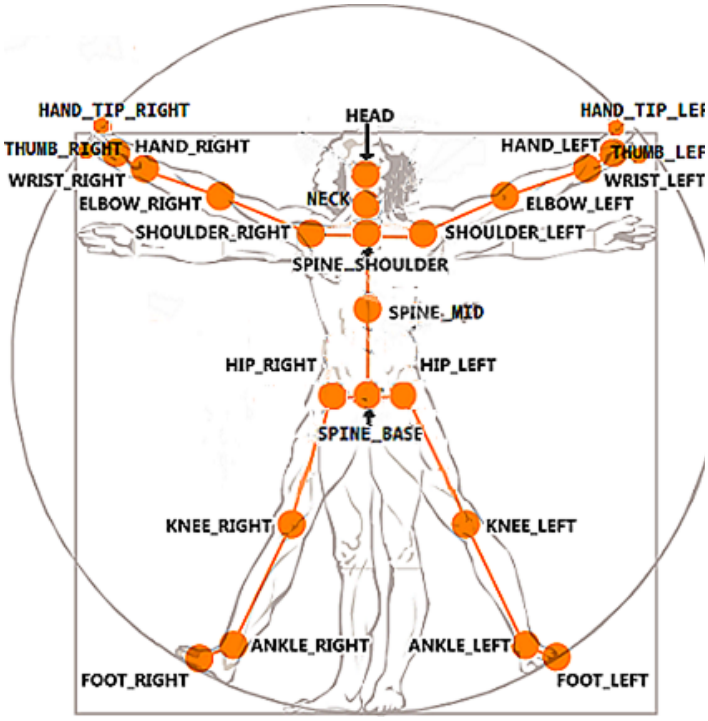

Fig. 1: The Microsoft Kinect v2 skeleton positions relative to the human body. Copied from [https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx]

Figure 1 illustrates the Kinect skeleton. The skeleton has 25 body joints that can be tracked. The X, Y, and Z position of these body joints relative to the camera can be extracted using the Kinect libraries in C++. Thus, we obtain 75 data points per frame per skeleton representing the 3D depth coordinates of each body part, or 2250 data points per second.

A simple approach to the posture detection would be to feed the 75 data points representing body joints locations into a neural network and classify the posture for each frame. However, this does not only slows down the classification process but also would require a very large amount of data, in hundreds of GB, since we would need our training data to be recorded from all possible angles and depths.

In order to circumvent the above issue, we extract location-independent features from the 75 data points representing the body joints depth coordinates. We extract seven features from these body joints. This will not only make the posture classification setting-independent, but it will also drastically reduce the amount of data collected since we now only extract 7 data points per body per frame instead of 75.

The Kinect skeleton provides the X, Y, and Z position of each one of the 25 body joints. As shown in Figure 2, we define the Y-axis to be the height (vertical location), the X-axis to be the width (horizontal location), and the Z-axis to denote the depth.

Figure 3 illustrates six of the seven extracted features from the raw data, named as follows: Left hip angle (1), right hip angle (2), left knee angle (3), right knee angle (4), chest angle (5), and chest-knee angle (6). Algorithm 1 shows us how the computation of the $LEFTHIP$ angle (one of the seven features) is done from the body joints depth coordinates using the cosine formula. The other five angles are computed in a very similar way. The seventh feature is the height of the subject which can be computed by taking the Y-position of the head and subtracting the lower Y-position value of the right and left foot from it, as shown in the Algorithm 2.

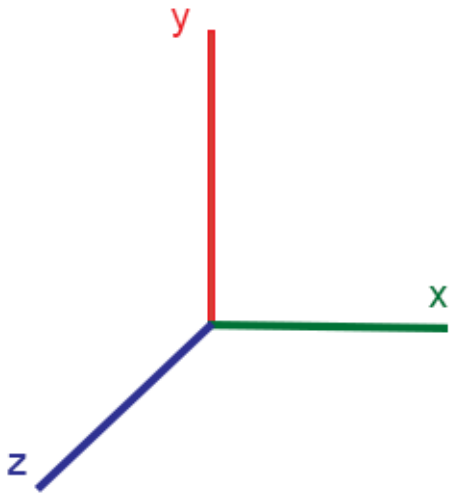

Fig. 2: 3D Cartesian plane.

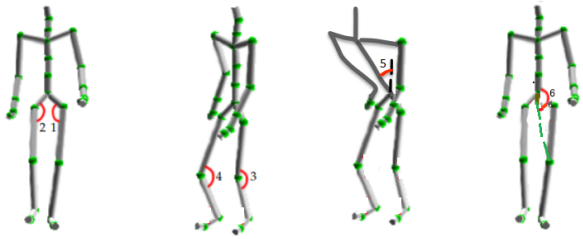

Fig. 3: Visualization of six of the seven features that will be used for posture classification in the Kinect Skeleton.

Although the Microsoft Kinect sensor estimates reasonably well the position of the body parts that it is unable to see, it may be good idea to check if the calculated height and angle features are not null before feeding them into the neural network.

III. Posture Detection

Six people of different heights and shapes were asked to participate in the data collection phase. The data collection phase consisted of four five-minute rounds, where in each round, the subject was allowed to either move or stay in a fixed posture: that is standing for five minutes, sitting for five minutes, and lying down for five minutes. Since the camera had a frame rate of 30 frames per second, 1800 training

Algorithm 1: The computation algorithm of the left hip feature from the raw body joints data.

Input: The X, Y, and Z position of the HIP_CENTER, the HIP_LEFT, and the $KNEE\_LEFT$ body joints.

Output: The left hip angle feature.

#Extract the body joints locations.
$spineBasePos = joints[JointType\_SpineBase].Position$;
$hipLeftPos = joints[JointType\_HipLeft].Position$;
$kneeLeftPos = joints[JointType\_KneeLeft].Position$;
#Calculate the distance between the body joints.
$hipKneeDistance = sqrt(pow(hipLeftPos.X - kneeLeftPos.X, 2) + pow(hipLeftPos.Y - kneeLeftPos.Y, 2) + pow(hipLeftPos.Z - kneeLeftPos.Z, 2))$;
$spineHipDistance = sqrt(pow(spineBasePos.X - hipLeftPos.X, 2) + pow(spineBasePos.Y - hipLeftPos.Y, 2) + pow(spineBasePos.Z - hipLeftPos.Z, 2))$;
$spineKneeDistance = sqrt(pow(spineBasePos.X - kneeLeftPos.X, 2) + pow(spineBasePos.Y - kneeLeftPos.Y, 2) + pow(spineBasePos.Z - kneeLeftPos.Z, 2))$;
#Use the cosine formula to determine the left hip angle feature.
$leftHipAngle = acos((pow(hipKneeDistance, 2) + pow(spineHipDistance, 2) - pow(spineKneeDistance, 2))/(2 * hipKneeDistance * spineHipDistance)) * 180/PI$;

Algorithm 2: The computation algorithm of the height feature from the raw body joints data.

Input: The X, Y, and Z position of the $HEAD$, the $FOOTLEFT$, and the $FOOTRIGHT$ body joints.

Output: The height feature.

$headPos = joints[JointType\_Head].Position$;
$footLeftPos = joints[JointType\_FootLeft].Position$;
$footRightPos = joints[JointType\_FootRight].Position$;
#$fmin(x, y)$ returns the smaller of its arguments: either x or y.
$height = headPos.Y - fmin(footLeftPos.Y, footRightPos.Y)$;

frames per minute, or 9000 frames for each 5-minute round per subject per posture were collected which led to about 54000 training frames per posture in total.

Features extraction was performed on the collected data. Three possible postures: standing, sitting, or lying down were considered as classes. For a fast posture classification, we first trained a fully connected 7-input 3-output and 0-(hidden) layer neural network in Tensorflow and trained it on the 54000 frames of each posture, or 378000 data points given that each training frame contains seven features. After the training phase, the posture classification accuracy of the neural network was tested on 20000 frames of labelled data collected on five different people who were not part of the training phase, and 96.08% classification accuracy was obtained.

A 1-layer neural network with 10 neurons in the hidden layer and 2-layer neural network with 10 neurons in each hidden layer were also implemented and trained as a goal to understand the trade-off between the classification accuracy and the processing time. We did not see a significant increase in the classification accuracy (accuracy was always less than 97%) with both the 1-layer and 2-layer networks. For a mere 1% increase in accuracy, significantly higher processing time was required for both neural networks. Given that the Kinect runs at 30 frames per second, and the motivation was to implement a real-time system, we decided to choose the 0-layer fully connected implementation for the posture classification classifier.

Further improvements were applied to the posture detection algorithm to handle real world scenario. One of the improvements was to make the height factor more significant than the other six features. For instance, the angles for both the standing and the lying down position are the same; hence, it is only the height factor that will provide discrimination between these postures and avoid classification errors. A threshold of 45 cm was chosen to differentiate the two postures: That is, if the subject's height is less than this threshold, then the subject cannot be in the standing position and if it is greater, then the subject cannot be lying down. Defining the height as the the Y-position of the head less than the Y-position of the foot as discussed in Algorithm 2 helps us avoid classification errors in many scenarios such as if the subject lays down but raises his hands above the 45cm threshold. In such as scenario, the system will still detect a height closer to 0 because the Y-position of the head and the foot will be close to each other. The improvements to the posture detection algorithm are presented in Algorithm 3.

We were able to achieve an improvement in the posture classification result of 99.30%, from 96.08%, on the same test data after incorporating the improvements as shown in Algorithm 3.

IV. Fall Detection

Falls can be detected if two conditions are met: (i) The subject has been lying down for a more than a specified amount of time; (ii) it is confirmed that it is not an object (such as bed, couch) that has been identified as on the floor. Figure 4 represents a summary of the whole fall detection system and Algorithm 4 implements the fall detection system based on the posture detection, and will be executed every time new posture data is made available to the system.

Algorithm 3: Improving the posture classification result based on the height factor.

Input: The features extracted from the raw body joints depth data which will give us a certain posture classification result from the neural network. The height feature is given in position 0.

Output: The improved posture detection result. Position 0 of the posture array is standing, position 1 is sitting and position 2 is lying down.

Extract the seven features for the current frame.
$features = extractFeatures()$;
$posture = getPostureClassification(features)$;
$height = features[0]$;
if $height < 0.45$ **then**
    The subject cannot be standing in this case.
    posture[0] = 0;
else
    The subject cannot be lying down in this case.
    posture[2] = 0;
end

Algorithm 4: Fall detection system based on the posture detection results.

Input: Posture detection results over a window of two seconds.

Output: Fall detection result

Retrieve two seconds of posture data (i.e. 60 frames at 30 FPS).
$postures = getPostureClassification(features)$;
$lyingDownCount = 0$;
$fallDetected = False$;
for ***each*** $posture$ *in* $postures$ **do**
    if $posture == "LYINGDOWN"$ **then**
        $lyingDownCount = lyingDownCount + 1$;
    end
end
A fall will be detected if and only if the subject has been lying down for more than 90% of the time (i.e. 90%*60frames = 54frames).
if $lyingDownCount > 54$ **then**
    $fallDetected = True$;
end

As can be observed from Figure 4, a fall detection system requires to detect if a subject is lying down on the floor. In order to reduce possible false alarms, the locations of all the objects currently present in the room should be *a priori* known. If subject's current location is either overlapping or inside the boundaries of each object, then the subject can be declared as not lying down on the floor and hence no fall will be detected in this condition.

The method suggested above for detecting whether the subject is lying down on the floor or not requires the knowledge

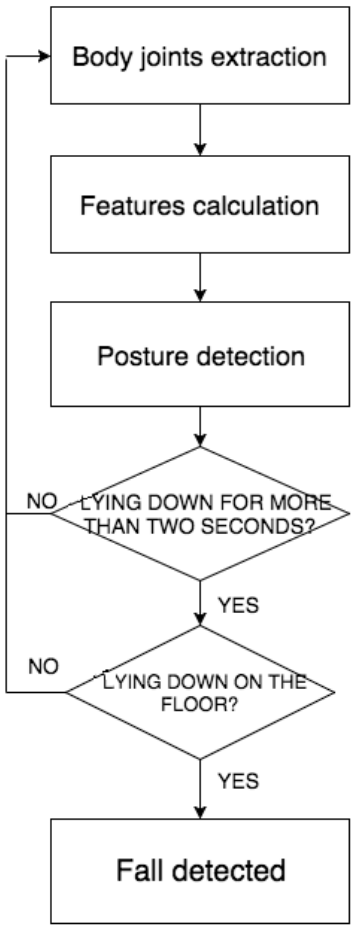

Fig. 4: Steps to implement the fall detection system.

of the floor plan of the room. Alternatively, an object detection algorithm may be used to detect all the objects present in the room and their X and Z boundaries may be obtained from the location estimates. Then a decision can be made about fall by checking the subject's location and comparing with the boundaries of the object. This may not be very efficient real time implementation since every object needs to be detected and located.

We propose a faster and reliable solution to determine whether the subject is lying down on the floor or not. The Algorithm in 5 first determines the floor level Y-coordinate relative to the camera by having an initialization step before the system starts. This initialization step will analyze the first three second of skeleton data and take the lowest recorded Y-coordinate of the left and right foot of the subject then store it. For the subject to be lying down on the floor, the Y-coordinate of his or her body joints cannot be much greater than this lowest Y-point. Using a safety threshold (say 25 centimetres), a decision that a subject is lying down on the floor is made if and only if the Y-coordinate of subject's foot is less than the floor level Y-coordinate of the room plus the threshold.

In other words, the subject is lying down on the floor if and only if the Y-coordinates of subject's both feet are less than 25 centimetres away from the floor. This algorithm will be executed if the posture is classified as lying down.

Algorithm 5: Floor level detection and determining whether the subject is lying down on the floor.

Input: The Y-coordinate of the right and left foot of the subject over a window of three seconds. Next detected posture and the Y-coordinates of the right and left foot.

Output: Boolean representing whether the subject is lying down on the floor.

```
Initialization step. min() function returns the smallest number in an array.
rightLeftFootYcoordinates = readSkeleton();
floorLevelY = min(rightLeftFootYcoordinates);
posture, footLeftY, footRightY = getNextPosture();
if posture == "LYING DOWN" then
    Is the subject's' feet currently 0.25m away from the floor?
    if footLeftY < (floorLevelY + 0.25) & footRightY < (floorLevelY + 0.25) then
        return True;
    end
    return False;
end
```

To measure the accuracy of Algorithm 5, two people were asked to lie down on the floor, on a bed, and on a couch for three minutes in three different rooms. The system was able to recognize 100% of the time that the subject was not lying on the floor when the subject was lying on the bed or couch, and also detected 100% of the time that the subject was lying on the floor when the subject was indeed lying on the floor.

V. Conclusion

This work presents a real time fast and accurate implementation of a posture classification and fall detection system using the 3D body joints depth coordinates extracted from the Microsoft Kinect skeleton v2. These depth coordinates are used to compute seven features of the subject: height, left hip angle, right hip angle, left knee angle, right knee angle, chest angle, and the chest-knee angle. These seven features are used as inputs to a pre-trained neural network which then outputs three possible postures: standing, sitting, and lying down. Furthermore, a fall detection system is also implemented using the outputs of the posture classification system as well as the output of Algorithm 5 that determines whether the subject is lying down on the floor or on an object.

Limitations of this work include that only three posture have been tested (standing, sitting, and lying down). Furthermore, the system was not tested in the case where the subject is covered by a blanket, which may affect the accuracy of the Kinect skeleton.

Future work planned for the system is to implement the object detection algorithm discussed in Section IV and compare its performance and accuracy results with the proposed implementation in Algorithm 5 of determining whether a subject is lying down on the floor. In addition, an additional $transition$ class will be added to the posture classifier to allow us to determine when the user switches posture. Lastly, more thorough testing of the system in realistic conditions will be done to potentially discover new limitations of the system.

References

[1] C. Mattmann, O. Amft, H. Harms, G. Troster, and F. Clemens, "Recognizing upper body postures using textile strain sensors," in *Proceedings of the 2007 11th IEEE International Symposium on Wearable Computers*, ser. ISWC '07.

[2] H. Gjoreski and M. Gams, "Activity/posture recognition using wearable sensors placed on different body locations," p. 716724, 08 2011.

[3] W.-J. Wang, J.-W. Chang, S.-F. Haung, and R.-J. Wang, "Human posture recognition based on images captured by the kinect sensor," *International Journal of Advanced Robotic Systems*, vol. 13, no. 2, pp. 54 – 69, 2016. [Online]. Available: https://doi.org/10.5772/62163

[4] X. Li, M. Sun, and X. Fang, "An approach for detecting human posture by using depth image," vol. 133, 01 2016.

[5] K.-S. Oh and K. Jung, "Gpu implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, 2004.

[6] M. Abadi and A. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[7] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 3 – 10.