UNIVERSITY OF NAIROBI

MASTERS IN COMPUTER SCIENCE

Msc PROJECT DOCUMENTATION

HUMAN POSTURE RECOGNITION AND GOOD POSTURE RECOMMENDATION

Submitted in Partial fulfillment

Of the Requirements for the Masters' Degree of Science in Computer Science

BY: SAMSON KAYEMBE

REG NO: P58/62440/2010

SUPERVISOR: Mr. EVANS MIRITI

DECLARATION

This project as presented in this report is my original work and has not been presented for any other institutional award.

Sign: _______________________ **Date: _____________________**

Kayembe Samson Mudimbi

P58/62440/2010

This project has been submitted in partial fulfillment of the requirements for the Masters of Science in Computer Science of the University of Nairobi with my approval as the university supervisor.

Sign: _______________________ **Date: _____________________**

Mr. Evans Miriti

School of Computing and Informatics

University of Nairobi

Abstract

This study presents a model human posture recognition method using skeleton tracking.
This is achieved using the power of Kinect, a motion sensing input device by Microsoft for the Xbox 360 video game console and a combination of posture recognition algorithm.

The project focuses on development of a prototype that analyses posture of a human object in a scene, recognizes the posture (standing or sitting) and provide recommendation on a good posture.

This has been achieved using a 2-stage head detection process to locate the people in a particular scene. This first explores the boundary information embedded in the depth map of Kinect to locate the candidate regions that may indicate the appearance of people. The algorithm implemented here scans across the whole image and gives the possible regions that may contain people. The regions are examined each using a 3D head model, which utilizes the relational depth information of the array for verification.
Then parameters of the head are extracted from the depth array and use them to build a 3D head model. Later, a region growing algorithm is also developed to find the entire body of the person and extract his/her whole body contour.

This project has been able to highlight how computing can be used to solve problems related to human behavior analysis. Particularly in scenarios where human behavior needs to be monitored in places such as Correctional facilities, Interogations, Classrooms, Healthcare, surveillance among others.

On good posture recommendation, this method points out that, it is possible to utilize the power of computer vision in promoting correct human posture by providing a tool that tracks human posture, asses the time consumed in that posture and alerts the person in order to reduce muscle strain.

ACKNOWLEDGEMENTS

First, my appreciations go to the Lord Almighty, for enabling me to get this far with the quest for knowledge.

Secondly, my appreciations go to my parents who have sacrificed a lot and worked tirelessly to get me where I am today, and to them I dedicate this work.

Thirdly, I am very grateful to my supervisor, Mr. Evans Miriti for his insights and guidance.

And finally, to other supervisors at the school of computing, thank you for your advice and encouragement.

List of abbreviations

- SDK	Software Development Kit
- RGB:	Red Green Blue (Primary colors)
- RAD:	Rapid Application Development
- 3D:	3 dimension (al)
- 2D:	2 dimension (al)
- CCD:	Charge Coupled Device
- HMM:	Hidden Markov Model
- ICP:	Iterative Closest Point
- MRI:	Magnetic Resonance Imaging
- SDLC:	Software Development Life Cycle
- JAD:	Joint Application Development
- WPF:	Windows Presentation Foundation
- TP:	True Positives
- FP:	False Positives
- FN:	False Negatives
- TN:	True Negatives

Table of Contents

Table of figures

1. Introduction

1.1 Background

Human posture and activity recognition has attracted a lot of interest in recent years because of its potential important applications in surveillance, human-computer interaction and computer animation. Posture recognition is also one of the most challenging problems in computer vision, because of articulated motion of human bodies and large appearance varieties of clothing.

Many traditional human posture recognition systems are based on still cameras and background subtraction; the silhouettes of characters are then used in posture recognition. The difficulty with this scheme is that background subtraction is not robust and not always available, and the method cannot distinguish postures when body parts are covered by silhouettes.

Detecting human in images or videos is a challenging problem due to variations in pose, clothing, lighting conditions and complexity of the backgrounds. There has been much research in the past few years in human detection and various methods are proposed

Estimating the 3D motion of humans or animals is a fundamental problem in many applications, including realistic character animation for games and movies, or motion analysis for medical diagnostics and sport science. Ideally, one expects to both estimate an articulated rigid-body skeleton that explains the overall motion of the character, as well as the potentially non-rigid deformation of the surface, e.g. caused by tissue or garment

Some potential applications of posture recognition are such as assessing the concentration time of students in a classroom, obtain statistics on human behavior in a particular environment

1.2 Definitions of terms:

- **Depth map:**
  An image or image channel that contains information relating to the distance of the surfaces of scene objects from a viewpoint.

- **Color mapping**
  This is a function that maps (transforms) the colors of one (source) image to the colors of another (target) image. A color mapping may be referred to as the algorithm that results in the

mapping function or the algorithm that transforms the image colors. This is also sometimes called color transfer.

- **Computer stereo vision:**
  The extraction of 3D information from digital images, such as obtained by a CCD camera. By comparing information about a scene from two vantage points, 3D information can be extracted by examination of the relative positions of objects in the two panels.

- **Hidden Markov model (HMM):**
  Is a situation in which a system the state is only partially observable.
  In other words, observations are related to the state of the system, but they are typically insufficient to precisely determine the state.

- **2D chamfer distance matching:**
  This is an edged based object detection and recognition technique used in computer vision. This algorithm involves scanning across a given image to give possible regions that may contain people.

- **Silhouette:**
  Is an image of a person, an object or scene represented as a solid shape of a single color, usually black and its edges matching the outline of the subject. The interior of a silhouette is basically featureless, and the whole is typically presented on a light background, usually white, or none at all.

- **Algorithm:**
  In mathematics and computer science, an algorithm is a step-by-step procedure for calculations. Algorithms are used for calculation, data processing, and automated reasoning.

- **ICP Algorithm:**
  ICP (Iterative Closest Point) algorithm is widely used for geometric alignment of three-dimensional models when an initial estimate of the relative pose is known. ICP is often used to reconstruct 2D or 3D surfaces from different scans, to localize robots and achieve optimal path planning.

- **The nearest neighbor algorithm:**
  This algorithm addresses the problem of approximating the value of a function for a non-given point in some space when given the value of that function in points around (neighboring) that point. The algorithm selects the value of the nearest point and does not consider the values of neighboring points at all, yielding a piecewise-constant interpolant. The algorithm is commonly used in real-time 3D rendering to select color values for a textured surface.

1.3 Problem statement

In our modern society with a lot of routine activities, many people find themselves in several positions throughout the day (sitting, standing, bending, stooping, and lying down). It's important to learn how to attain and keep correct posture in each position for good back support, which will result in less back pain.

Many people do not maintain good posture and adequate back support thus adding strain to muscles and put stress on their spine. Over time, the stress of poor posture can change the anatomical characteristics of the spine, leading to the possibility of constricted blood vessels and nerves, as well as problems with muscles, discs and joints. All of these can be major contributors to back and neck pain, as well as headaches, fatigue, and possibly even concerns with major organs and breathing.

Furthermore, Office work often results in poor posture and strain to the lower back. Many people work sitting in an office chair that is not properly fitted to their body and do not provide enough lower back support. Not everyone is aware about choosing an ergonomic office chair that often provides better support than a regular chair and more comfortable for the user.

In addition, the body spine is made for motion, and when standing or sitting (in any type of office chair even an ergonomic office chair) for long periods of time, it is best to change position, stretch and move around regularly throughout the day to recharge stiff muscles. However many people do not apply this especially when one concentrates on a mind catching activity like watching movies, surfing the internet, etc.

The following are examples of common behaviors of bad posture and poor ergonomics that need correction to attain good posture and back support:

- Cradling a phone receiver between the neck and shoulder
- Carrying something heavy on one side of the body
- Wearing high-heeled shoes or clothes that are too tight
- Keeping the head held too high or looking down too much
- Slouching with the shoulders hunched forward
- Lordosis (also called "swayback"): too large of an inward curve in the lower back

All the above problems are experienced because of lack of knowledge about good posture and probably a tool that could monitor human body activities and remind when they have overstayed in one posture to minimize strain to muscles.

With the help of computer vision and the underlying supportive algorithms, human posture recognition can be made possible and reduce common problems of poor posture and improve posture.

1.4 Objectives

The main objective is to develop an approach for recognizing human posture from raw data of a motion sensing device (Microsoft Kinect).

The goal is to build a system that recognize the major human postures (Sitting & Standing) in which most people spend a lot of time and identify common behaviors that lead to common posture problems such as neck and back pain.

Furthermore, to provide proper posture recommendation or warning in case the adopted posture is bad.

1.5 Justification of the study

This project is undertaken in order to bring to the attention of technologists how computer vision with applications in motion capture and activity recognition could be used to promote human health particularly in reducing poor posture in places like classrooms and offices.

Moreover, many specialists are predicting that in the closer future, a new revolution in information technologies will occur. This revolution will be connected with new computer abilities to segment, track, and understand pose, gestures, and emotional expressions of humans. For this, computers must begin to use new types of video sensors that will provide 3D-videos. The Kinect sensor is the first of such new types of sensors.

Furthermore, this project provides a foundation in other areas where it is necessary to control human behavior in different poses.

1.6 Scope, Limitations and Assumptions

“Human posture recognition” in this project will mainly be constrained on sitting and standing postures that mainly affect human health. The scope of the project will encompass the said two postures because many people’s positions can be found either in those two postures. This study also focuses on indoor environments such as office and classroom.

Recognition is carried out on the assumption that the human posture can mainly be described from a human skeleton model.

One of the limitations is that the Kinect sensor needs to be distanced at least 2 meters from the human body in order to provide proper recognition. Furthermore, for optimal performance in full body tracking, the Kinect sensor needs an open space about 13 feet deep by 6 feet wide

2. LITERATURE REVIEW

Human posture recognition is an important task for many applications in different fields, such as surveillance, ambient intelligence, elderly care, human-machine interaction, etc.

Computer vision techniques for human posture recognition have been developed in the last years by using different techniques aiming at recognizing human activities.

The main problems in developing such systems arise from the difficulties of dealing with the many situations that occur when analyzing general scenes in real environments.

2.1 Human Posture

Posture is the position in which one holds his/her body upright against gravity while standing, sitting or lying down.

The human body is a fantastic piece of biological equipment which is essentially a strong movable frame (which we know as the Skeleton) surrounded by a series of strong flexible muscles which are connected to the skeleton by ligaments. This ingenious set up means we are able to move around at our will with a large degree of freedom and control. However due to the nature of our daily lives in this day in age, many of us develop poor posture.

This means we do not hold our body in a good position when doing things such as walking or sitting at a computer desk. This is a concern for many as a good posture provides us with a series of benefits such as a strong spine, and a higher sense of self confidence. It also gives the impression to others we encounter that we are strong, intelligent and capable.

2.1.1 Good Human Posture

Good posture involves training our body to stand, walk, sit and lie in positions where the least strain is placed on supporting muscles and ligaments during movement or weight-bearing activities.

Proper posture has the following benefits:

- Keeps bones and joints in the correct alignment so that muscles are being used properly.
- Helps decrease the abnormal wearing of joint surfaces that could result in arthritis.
- Decreases the stress on the ligaments holding the joints of the spine together.
- Prevents the spine from becoming fixed in abnormal positions.
- Prevents fatigue because muscles are being used more efficiently, allowing the body to use less energy.
- Prevents strain or overuse problems.
- Prevents backache and muscular pain.
- Contributes to a good appearance.

2.1.2 Poor Human Posture

There is also a more serious effect of having a poor posture that is far more relevant to us. Having a poor body posture will effectively reduce someone’s height and make a person look far shorter than he/she actually is. For those of us looking for ways to increase height naturally this is a major problem as height and posture are closely linked and impact on one another.

It can probably be noticed that as people age they can sometimes begin to stoop over and their spines start to hunch forward. This means that people will likely see a good deal of older citizens who look substantially shorter than others. In fact, in truth they are probably no shorter than the others, but the poor posture they have developed gives the impression that they have lost height.

The same can be said of many people these days. Modern working jobs such as typing and writing means that some of us hold our bodies in a bad posture for hours at a time, each and every day.

Over time this can cause bad posture all the time and as we already know posture and height are linked together, they will appear shorter.

2.2 Human Posture Recognition techniques

Different studies have been undertaken in order to establish the most efficient way to detect and track human postures. Most of the research is based on images taken by visible-light cameras, which is a natural way to do it just as what human eyes perform. Some methods involve statistical training based on local features, e.g. gradient-based features and some involve extracting interest points in the image.

The following are some of the techniques:

2.2.1 Human Posture Recognition using convex programming

Jiang, Ze-Nian and Drew (2005) suggest a novel human posture recognition method using convex programming based matching schemes. Instead of trying to segment the object from the background, a novel multi-stage linear programming scheme is developed to locate the target by searching for the best matching region based on an automatically acquired graph template.
The linear programming based visual matching scheme generates relatively dense matching patterns and thus presents a key for robust object matching and human posture recognition.

By matching distance transformations of edge maps, the proposed scheme is able to match figures with large appearance changes.
This method further present objects recognition methods based on the similarity of the exemplar with the matching target. The proposed scheme can also be used for recognizing multiple targets in an image.

The difficulty with this scheme is that background subtraction is not robust and not always available, and the method cannot distinguish postures when body parts are covered by silhouettes.
One method to solve the problem is by extracting range data for the character in the
scene using multiple cameras. But the approach becomes more expensive to deploy. (Jiang, Ze-Nian and Drew, 2005)

2.2.2 Human Posture Tracking using Stereo Vision and 3D Model matching

This method was presented by Pellegrini and Iocchi (2007) as an approach to human posture tracking and classification that aims at overcoming some of the common limitations, thus enlarging the applicability of this technology. The contribution of this technique is to describe a method for posture tracking and classification given a set of data in the form XYZ-RGB, corresponding to the output of a stereo vision based people tracker. The presented method uses a 3D model of human body, performs model matching through a variant of the ICP (Iterative Closest Point) algorithm, tracks the model parameters over time, and then uses a Hidden Markov Model (HMM) to model posture transitions. The resulting system is able to reliably track human

postures, overcoming some of the difficulties in posture recognition, and in particular presenting higher robustness to partial occlusions and to different points of views.
Moreover, the system does not require any off-line training phase, it just uses the first frames (about 10) in which the person is tracked to automatically learn parameters that are then used for model matching. During these training frames we only require that the person is in the standing position (with any orientation) and that his/her head is not occluded.

The approach to human posture tracking and classification presented here is based on stereo vision segmentation. Real-time people tracking through stereo vision has been successfully used for segmenting scenes in which people move in the environment and are able to provide not only information about the appearance of a person (e.g. colors) but also 3D information of each pixel belonging to the person. (Pellegrini and Iocchi, 2007)
In practice a stereo vision based people tracker provides, for each frame, a set of data in the form XYZ-RGB containing a 2 1/2D model and color information of the person being tracked. Moreover, correspondences of these data over time is also available therefore when multiple people are in a scene, we have a set of XYZ-RGB data for each person.

2.2.3 Human Posture Recognition using video sequence

This method presents a new approach to recognize human postures in video sequences comparing two methods. These two methods can be described based on 2D appearances.

The first one uses projections of moving pixels on the reference axis.
The second method decomposes the human silhouette into blocks and learns 2D posture appearances through PCA. Then a 3D model of posture is used to make the previous methods independent of the camera position.

This technique classifies existing methods in human posture recognition in two categories:

- Methods using 2D appearances.
- Methods using 3D models

2D Appearance-Based Methods
The majority of the methods based on 2D appearance use the same schema (Darrell, Gordon, Woodfill and Harville ,1998; Demirdjian, 2002; Ganapathi, Plagemann, Koller and Thrun,

2010 ; Rodgers, Anguelov, Pang and Koller 2006). First they detect the principal parts of the body such as head, hands and feet (the extremities of the body) and based on these detections they search for the secondary parts of the body such as shoulders, elbows and knees (the articulations).

The system Ghost, segments the silhouette from the background (Darrell, Gordon, Woodfill and Harville ,1998). It computes the vertical and horizontal projections of the silhouette to determine the global posture of the person (standing, sitting, crawling-bending and laying) and his orientation relative to the camera (front view, left side view and right side view).

To recognize posture the system computes the projections of the current silhouette and compares them to the model of projection realized for a set of predefined posture and point of view. Then it determines the body parts by analyzing the contour of the silhouette.

Ganapathi, Plagemann, Koller and Thrun (2010) have proposed a method in three steps. The first step consists in determining the center of gravity of the human silhouette. The second step computes the orientation of the upper half of the body. Then the significant points such as feet, hands, elbows, and knees are estimated by using a heuristic contour analysis of the human silhouette.

3D Model-Based Methods

A 3D model is made of geometrical objects such as parallelepipeds, spheres or truncated cones. The model includes the parameters which define the relations between these objects.

This model is defined in a high dimensional phase space (dimension depends on the degrees of freedom of the model).

Systems which use monocular vision are based on a priori knowledge in the form of a human model and the constraints related to it.

Jain and Subramanian (2010) propose an alternative representation of the phase space which supports a more efficient use of the different constraints. This method needs a front parallel torso with respect to the camera. Moreover to deal with real time constraint, one arm is processed.

To handle ambiguities in posture estimation and to estimate precisely the depth, several cameras may be used.

In Ikemura and Fujiyoshi (2011), three cameras are used to observe the person from the top, front and side view. In each image, the significant points (head, hands and feet) are located in 2D. Then two views are selected and the 3D coordinates of each significant point are calculated by triangulation.

Dalal, Triggs and Schmid (2006) use three cameras all around the person. The method compares the projections of a 3D model of a person on an image with the detected silhouettes of the person (binary image). This process is iterated by computing a force that will move the 3D model towards the current silhouette. The final parameters of the 3D model constitute an estimation of the real posture.

Lowe (1999) proposed a system estimating 3D human hand posture. The estimation is based on a 2D image retrieval. More than 16000 hand appearances was generated and stored in a database. The search area is reduced by using an adjacency map in the database. This method is implemented on a PC cluster system consisting of 6 PCs (Pentium III 600 MHz) to achieve real time. The use of these PCs is not adapted for video surveillance applications.

2.3 Recommended Sitting & standing Postures

2.3.1 Seating Posture

In 2006, A study conducted by Scottish and Canadian researchers at Woodend Hospital in Aberdeen, Scotland suggested that sitting up straight hurts our back (Bashir, Torio, Smith, et al. 2006).

The researchers led by Dr Waseem Amir Bashir have found that sitting upright with a straight back and thighs parallel to the floor increases the strain on lumbar discs in the lower back.

The researchers suggested that a look at the spine with new imaging technology reveals how sitting upright with a straight back and thighs parallel to the floor increases the strain on lumbar discs in the lower back.

In fact it's better to lean back a bit in a chair, even if looks like slouching.

The research conclusions come from getting a different view of the spine, using a newly designed magnetic resonance imaging machine that allows for a full view of the back while sitting.

Conventional MRI equipment requires the patient to lie down while images are taken, but this puts the spine in an unstressed position, Bashir said. By making images when people are upright in a chair, he said he was able to capture instabilities and deformations not otherwise seen.

In this study, Twenty two volunteers with healthy backs were scanned using a positional MRI machine, which allows patients the freedom to move - so they can sit or stand - during the test.

The patients assumed three different sitting positions: a slouching position, in which the body is hunched forward as if they were leaning over a desk or a video game console, an upright 90-degree sitting position; and a "relaxed" position where they leaned back at 135 degrees while their feet remained on the floor.

The researchers then took measurements of spinal angles and spinal disk height and movement across the different positions.
Spinal disk movement occurs when weight-bearing strain is placed on the spine, causing the disk to move out of place.
Disk movement was found to be most pronounced with a 90-degree upright sitting posture.

It was least pronounced with the 135-degree posture, suggesting less strain is placed on the spinal disks and associated muscles and tendons in a more relaxed sitting position.

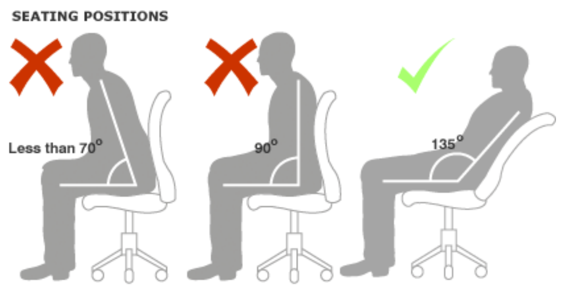


Fig1. Recommendation on seating positions

The "slouch" position revealed a reduction in spinal disk height, signifying a high rate of wear and tear on the lowest two spinal levels.

When they looked at all test results, the researchers said the 135-degree position was the best for backs, and say this is how people should sit.

2.3.2 Standing Posture

In their research review, Kritz and Cronin (2008) state that with good standing posture the body's joints are in a state of equilibrium (vertical and rotational forces balanced) with the least amount of physical energy being used to maintain this upright position.

Thus, as observed in the standing position, if a vertical line tracks from the neck to the tailbone and through the lower limbs, the body will not have to adjust to counter the forces of gravity. Furthermore, Kritz and Cronin (2008) propose that a body in equilibrium theoretically is capable of doing the most efficient work.

Kritz and Cronin (2008) suggest assessing the following checkpoints in standing side posture (this list describes optimal posture):

- head: neutral, with no forward or backward tilt
- hip joints: neutral, neither flexed nor extended
- knee joints: neutral, neither flexed nor hyperextended
- ankle joints: neutral, with leg vertical to sole of foot

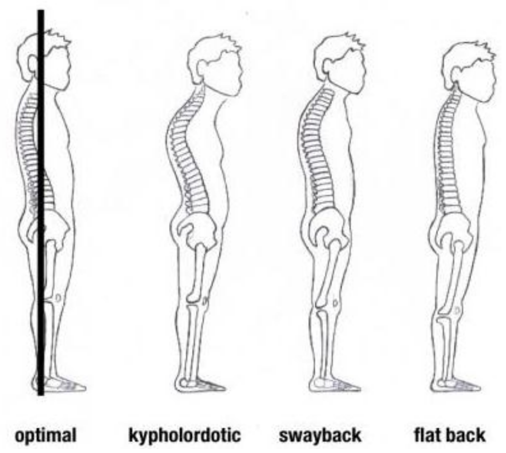


Fig 2. Recommendation on standing positions

2.4 The Microsoft Kinect Sensor

In this work, we attempt to tackle the problem of human posture recognition and tracking using the Microsoft Kinect sensor. We use cues from the RGB and depth streams from the sensor to obtain a model to the human body.
The Microsoft Kinect (Figured below) is a new product launched by Microsoft in November 2010. Its capability to produce depth and RGB streams at a price much lower than traditional

range sensors have made it a sensation in the field of Computer Vision.

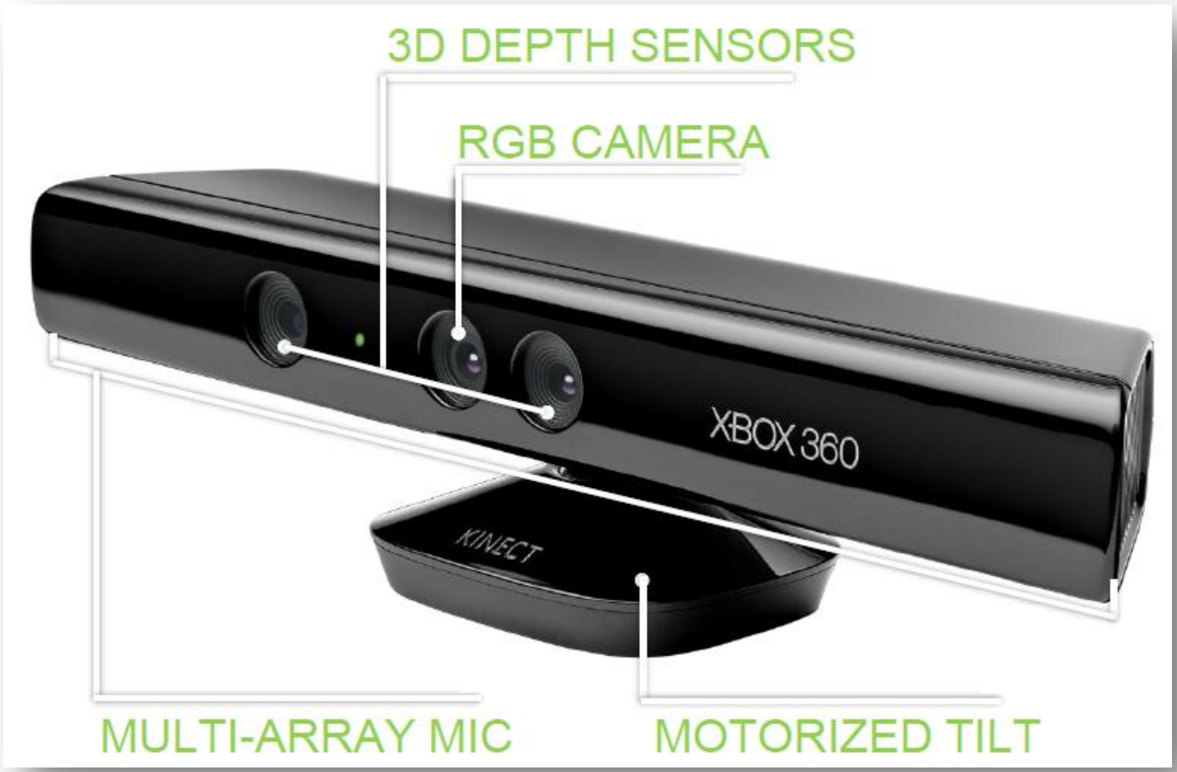


Fig 3. Kinect Sensor Device for XBOX 360

The Kinect Sensor consists of the following:

- **RGB Camera**: A regular video camera that has a resolution of 640x480 at 30 frames per second
- **3D Depth Sensors**: This is a combination of a depth projector (left reticule) and sensor (right reticule) to calculate distance.
- **Motorized Tilt**: The motorized tilt enables one to adjust the camera up or down 27 degrees to ensure that the camera has the best possible view of a scene
- **Multi-Array Microphone**: A four-microphone array that is mounted as a single microphone in Windows.

The RGB camera measures the distance of any given point from the sensor using the time taken by near-IR light to reflect from the object.

In addition to it, an IR grid is projected across the scene to obtain deformation information of the grid to model surface curvature.

The Kinect SDK merely provides a way to read and manipulate the sensor streams from the Kinect device. When a human object is determined in an image frame, the Kinect SDK provides Skeleton Data that can be used for further manipulation.

Skeleton data obtained from Kinect streams can be illustrated as below:

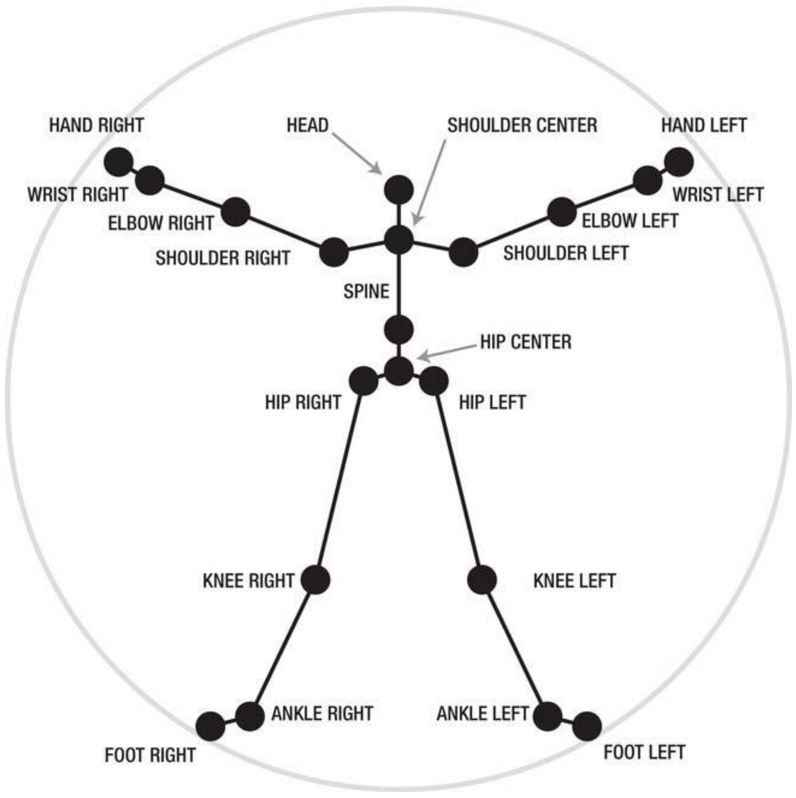


Fig 4. Illustrated skeleton joints

The Kinect streams can provide detection of up to 6 skeletons and tracking of a maximum of 2 skeletons per frame. Each Skeleton object consists of Joints. These collectively make a set of Joint structures that describe the trackable joints (head, hands, elbow and others) of a skeleton.

The application obtains streams from the sensor at a rate of 30 frames per second (fps). From every frame, the joints Shoulder center, Hip center and knees are used to determine the current posture of the skeleton.
The skeleton tracking engine of the SDK follows and reports on twenty points or joints on each skeleton.

Each joint has a Position, which reports the X, Y, and Z planes of the joint. The X and Y values are relative to the skeleton space. Consider figure below:

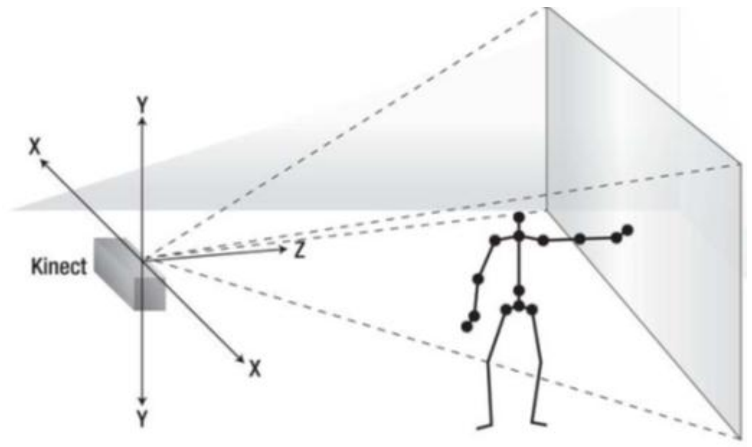


Fig. 5 Skeleton Space

The advantage of a skeleton model is that the Kinect sensor may continuously track the parts of the human body that are necessary for recognition (Two knees, One hip, Two shoulders, Two hands, Head) which collectively give information about the current human posture.

3. METHODOLOGY

This chapter discusses the main technique that will be used to achieve the outlined objectives of this project.

3.1 The proposed technique

We propose presents a novel human posture recognition method using depth information taken by the Kinect sensor device.
This technique involves processes achieved by the Kinect Sensor (Human Detection) and the ones involved in achieving our objectives (Posture Determination).
The project proposes a model based approach, which detects humans using a 2-D body contour model and a 3-D body surface model.
Then, a segmentation scheme is proposed to segment the human from his/her surroundings and extract the whole contours of the figure based on detection points.

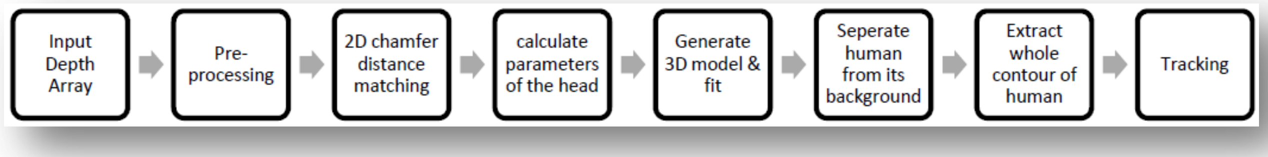

Input Depth Array
Pre-processing
2D chamfer distance matching
calculate parameters of the head
Generate 3D model & fit
Seperate human from its background
Extract whole contour of human
Tracking

Fig 6. Overview of the human detection method

The algorithm utilizes depth information only. It can also be combined with traditional gradient based approaches to give faster and more accurate detection. The detection algorithm can also serve as an initial step of the research on pose estimation and tracking recognition using depth information.

3.1.1 Overview of the method

This section provides an overview of steps that the Kinect device applies to obtain depth information from a given scene. These steps are summarized in Fig. 6 above.
Given an input depth array, we first reduce noise and smooth the array for later process. We use a 2-stage head detection process to locate the people. We first explore the boundary information embedded in the depth array to locate the candidate regions that may indicate the appearance of people. The algorithm used here is 2D chamfer distance matching. It scans across the whole

image and gives the possible regions that may contain people. Each of these regions are examined using a 3D head model, which utilizes the relational depth information of the array for verification. The parameters of the head are extracted from the depth array and use the parameter to build a 3D head model. Then matching of the 3D model against all the detected regions is done to make a final estimation. Also, a region growing algorithm is developed to find the entire body of the person and extract his/her whole body contour.

3.1.2 3D chamfer distance matching

i) Preprocessing

To prepare the data for our processing, some basic pre-processing is needed. In the depth image taken by the Kinect, all the points that the sensor is not able to measure depth are offset to 0 in the output array and regarded as a kind of noise. To avoid its interference, and recover its true depth value, it is supposed that the space is continuous, and the missing point is more likely to have a similar depth value to its neighbors. With this assumption, all the 0 pixels are regarded as vacant and need to be filled.

The nearest neighbor interpolation algorithm is used to fill these pixels and get a depth array that has meaningful values in all the pixels. Then a median filter is used with a 4×4 window on the depth array to make the data smooth.

ii) 2D chamfer distance matching

The first stage of the method is to use the edge information embedded in the depth array to locate the possible regions that may indicate the appearance of a person. 2D chamfer distance matching is used in this stage for quick processing. Also, chamfer distance matching is a good 2D shape matching algorithm that is invariant to scale, and it utilizes the edge information in the depth array which means the boundary of all the objects in the scene. Canny edge detector is used to find all edges in the depth array. To reduce calculation and reduce the disturbance from the surrounding irregular objects, all the edges whose sizes are smaller than a certain threshold are eliminated. (Here, the size of the edge is determined by the pixels it contained.)

Results of chamfer distance matching are shown in Fig. below

Binary head template shown in Fig. 7 (d) is used to match the template to the resulted edge image. To increase the efficiency, a distance transform is calculated before the matching process.

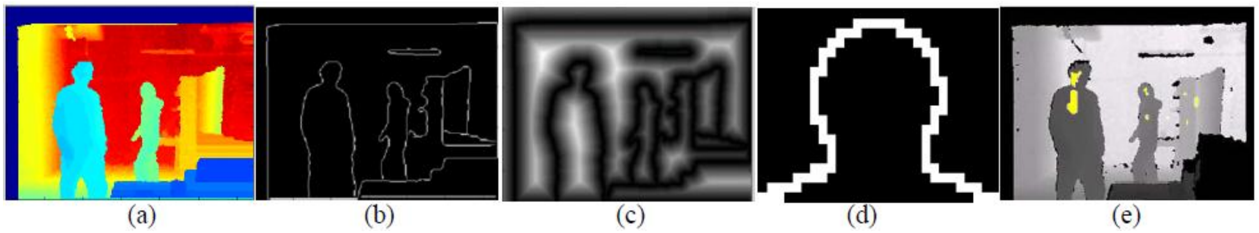


Fig 7. Results of 2D chamfer distance matching

Intermediate results of 2D Chamfer match:

(a) Shows the depth array after noise reduction.

(b) Gives the binary edge image calculated using Canny edge detector and then eliminate small edges.

(c) Shows the distance map generated from edge image (b). Match the binary head template (d) to (c) gives the head detection result

(e). Yellow dots indicate the detected locations.

3.1.3 3D model fitting

At this section all the regions that are detected by the 2D chamfer matching algorithm are examined:

i) Compute the parameters of the head

In order to generate the 3D model to fit on the depth array, we must know the true parameter of the head that within a certain range that is defined by of the head:

$$R = 1.33h / 2.$$

Here, h is the height of the head obtained from 2D chamfer matching algorithm and R is the search radius.

Next, we search for the head within a circular region defined by radius R in the edge image. If there is a circular edge in this region that satisfied all the constraints, e.g. size pass a certain threshold, it is decided that a head is detected.

The next thing to do is to find the true radius of the head. It happened to be that the distance map we calculated at 2D chamfer matching stage can be used to estimate the radius of the head. Recall that the pixels in the distance map contain the distances from this pixels to the closest data pixels in the edge image, considering the head is a circular like shape, the value of the center of the head on the distance map is just an approximation of the radius of the head. So we can take this directly as our estimation of the true radius of the head.

ii) Generate the 3D model

Considering the calculation complexity of 3D model fitting is comparatively high, the model should be view invariant so that there are no several different models or rotate the model and run several times. The model should generalize the characteristics of the head from all views: front, back, side and also higher and lower. To meet these constraints and make it simplest, we use a hemisphere as the 3D head model.

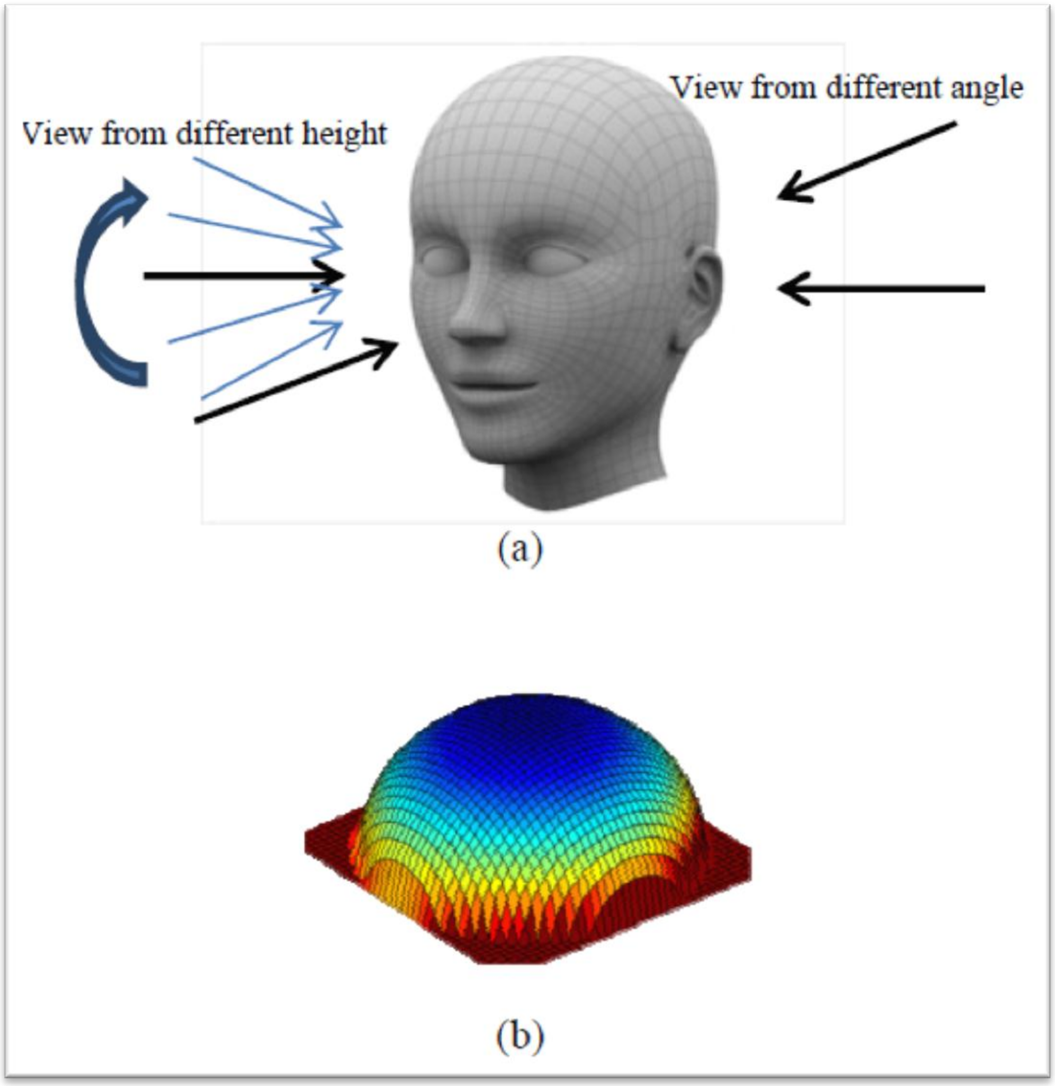


Fig 8. 3D head model. (a) illustrates the demands of the head model: The model should be invariant to different views. (b) shows the hemisphere model that is used as the 3D head model. A threshold is used to decide wether the region is actually a head. Fig. 9 below illustrates some of the steps in this stage and shows the result of the 3D matching.

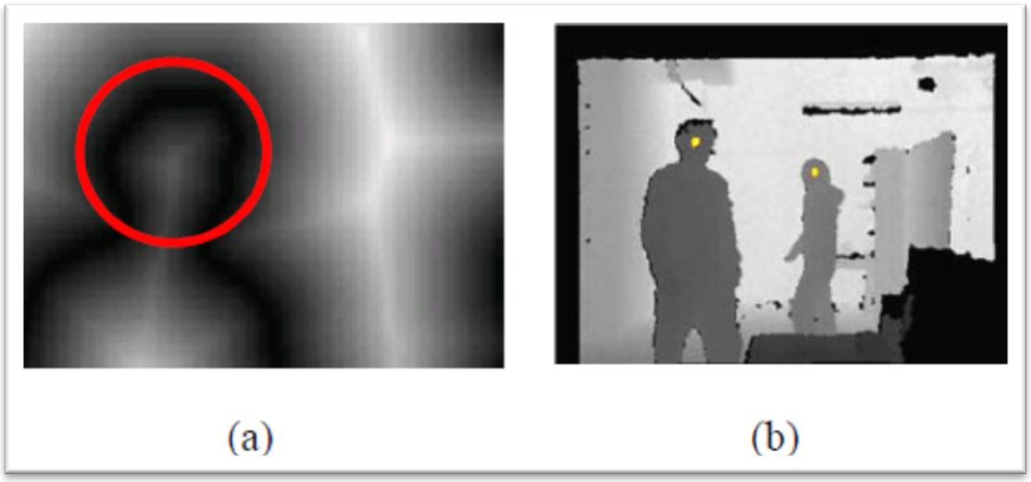


Fig. 9 (a) illustrates the process of estimating the true parameter of the head from the distance map. Input of the 3D model fitting is the output of the 2D chamfer matching in Fig. 7(e). Output of 3D model fitting is shown in (b). Yellow dots indicate the center of the head detected.

iii) Fitting

Next, the model is fitted onto the regions detected from previous steps.

3.1.4 Extract contours

The overall contour of the person is extracted so it is possible to track his/her hands and feet and recognize the posture.

In an RGB image, despite the person is standing on the ground, it is less a problem to detect the boundary between a feet and the ground plane using gradient feature.

However, in a depth array, the values at the person's feet and the local ground plane are the same depth with the person. To resolve this issue, we take advantages of the fact that persons' feet generally appear upright in a depth array regardless of the posture.

3.1.5 Tracking

Finally, preliminary results on tracking are given using depth information based on the detection result. Tracking in RGB image is usually based on color, the assumption is that the color of the same object in different time frames should be similar. But in depth images we don't have such color information. What we have is the 3D space information of the objects, so that we can measure the movements of the objects in a 3D space.

The tracking algorithm is based on the movements of the objects. It is assumed that the coordinates and speed of the same objects in neighboring frames change smoothly, i.e. there should not be big jumps in coordinates or speed. First, identify the center of the detected blob. Then, calculate the 3D coordinates and speed of the persons in each frame. The coordinates are given in the depth array directly; the speed is calculated from the coordinates of neighboring frames.

3.1.6 Posture Determination

The output of Kinect depth data is a collection of skeleton joints coordinates that can be used to determine the posture of a given person in a scene.

The main approach applied in the application to determine Seating and standing postures is the Two joint method (Webb and Ashley 2012). In our application, this method is used to calculate the angle at the Hip. With only two points (Knee and Shoulder center), it is possible to form a triangle. The angles of these triangles are calculated using the laws of trigonometry. Consider triangle below:

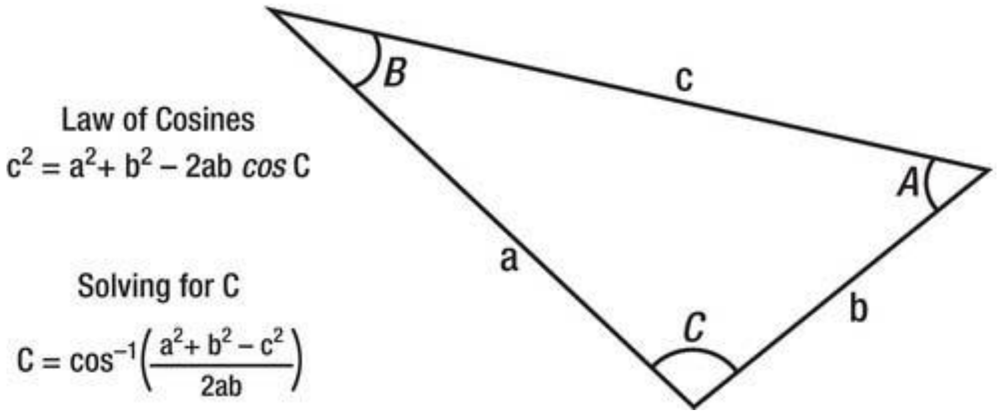

Law of Cosines
$c^2 = a^2 + b^2 - 2ab\,\cos C$
Solving for C
$C = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$
B
c
A
a
C
b

Fig 10. Calculation of angle using laws of trigonometry

Knowing the coordinates of each point in the triangle means that we know the length of each side, but no angle values. Applying the Law of Cosines formula gives us the value of any desired angle.

The Law of Cosines states that

c2 = a2 + b2 -2abcosC

Where C is the angle opposite side c. This formula derives from the commonly known Pythagorean theorem of c2 = a2 + b2.

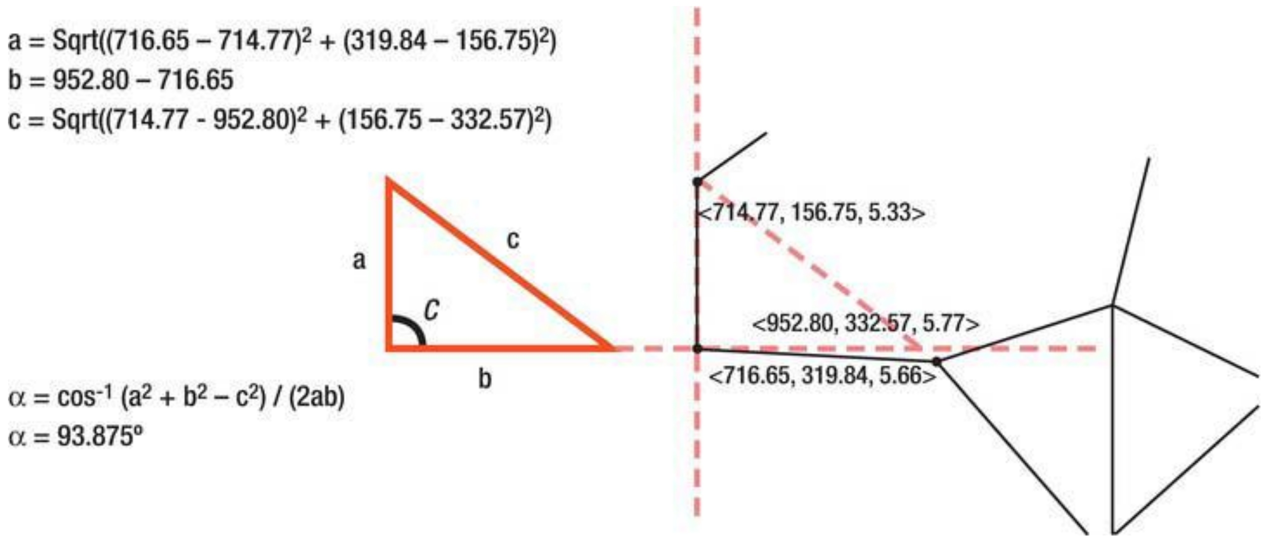
Calculations on the joint points give the values for a, b, and c.

The unknown is angle C. Transforming the formulas to solve for the unknown angle C yields:

C = cos-1((a2 + b2 – c2) / 2ab).

Arccosine (cos-1) is the inverse of the cosine function, and returns the angle of a specific value.

For example the below can illustrate how we can obtain the values of a, b, c and angle C given coordinates of shoulder and knee



$a = \mathrm{Sqrt}((716.65 - 714.77)^2 + (319.84 - 156.75)^2)$
$b = 952.80 - 716.65$
$c = \mathrm{Sqrt}((714.77 - 952.80)^2 + (156.75 - 332.57)^2)$
a
c
C
b
$\alpha = \cos^{-1} (a^2 + b^2 - c^2) / (2ab)$
$\alpha = 93.875^\circ$
<714.77, 156.75, 5.33>
<952.80, 332.57, 5.77>
<716.65, 319.84, 5.66>

Fig 11. Calculation of Joints vectors and Hip angle

The angle obtained at Hip is then used to determine the posture

3.2 Software Development Methodology

This section mainly refers to the software development methodology that will be applied in designing and developing the Human posture recognition application.

A software development methodology or system development methodology in software engineering is a framework that is used to structure, plan, and control the process of developing an information system.
Such of these methodologies include waterfall, prototyping and spiral.

This project will be implemented using the Rapid Application Development (RAD) methodology.

3.2.1 Rapid application development (RAD) methodology

This is a software development methodology that uses minimal planning in favor of rapid prototyping. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

The main phases of this methodology illustrated as shown below:

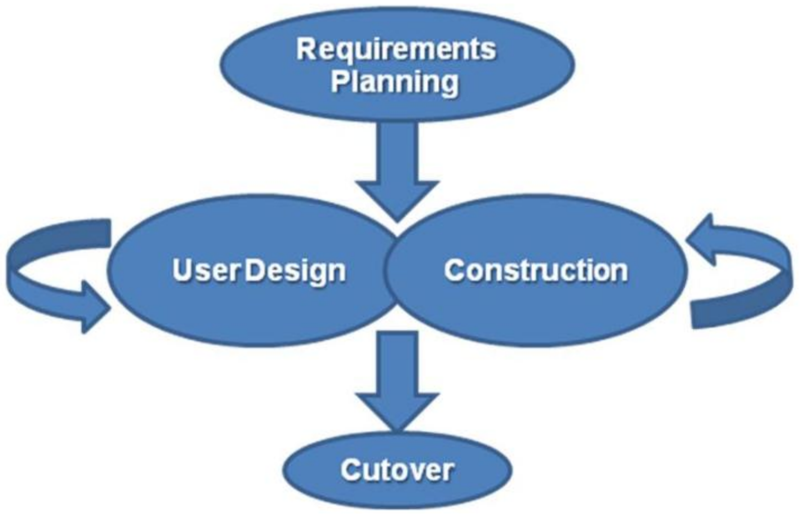


Fig 10. RAD Methodology steps

i) **Requirements Planning phase** – combines elements of the system planning and systems analysis phases of the System Development Life Cycle (SDLC). Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.

ii) **User design phase** – during this phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. *User Design* is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.

iii) **Construction phase** – focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.

iv) **Cutover phase** – resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner. Its tasks are data conversion, full-scale testing, system changeover, user training.

This methodology will be suitable for this project because of its various pros that include:

- Minimizes feature creep by developing in short intervals resulting in miniature software projects and releasing the product in mini-increments.
- Creates minimalist solutions (i.e., needs determine technology) and delivers less functionality earlier; per the policy that 80% today is better than 100% tomorrow.

3.3 Tool and Method for Data collection

The tools of data collection translate the research objectives into specific questions/ items, the responses to which will provide the data required to achieve the research objectives.
The main technique that will be employed in data collection for this project is the Observation technique.

Observation schedule is a form on which observations of an object or a phenomenon are recorded. The items to be observed are determined with reference to the nature and objectives of this study.

The observation schedule for this project will involve the monitoring the amount of time a person is one posture (e.g. sitting) and constantly compares to the maximum amount of time (40 min) recommended to be in one posture in order to avoid muscle strain.

The observation techniques in this project will mainly be concerned with collecting data such as

- Types of postures (sitting or standing) recognized over a certain period of monitoring

- Frequency of the recognized postures
- Time (in minutes)
- Coordinates and or angles of various body joints such as knee, hip and shoulders.

The observation data listed above can collectively be used later to achive the objectives of this study.

3.4 Resources and budget

3.4.1 Project resources

The following resources are useful for the success and completion of this project:

- A desktop or laptop PC with windows environment
- The Microsoft Kinect for PC
- Microsoft Visual Studio (Software development environment)
- C# programming language skills
- In door environment and person to track

3.4.2 Project Budget

The cost of the entire project can be estimated as below:

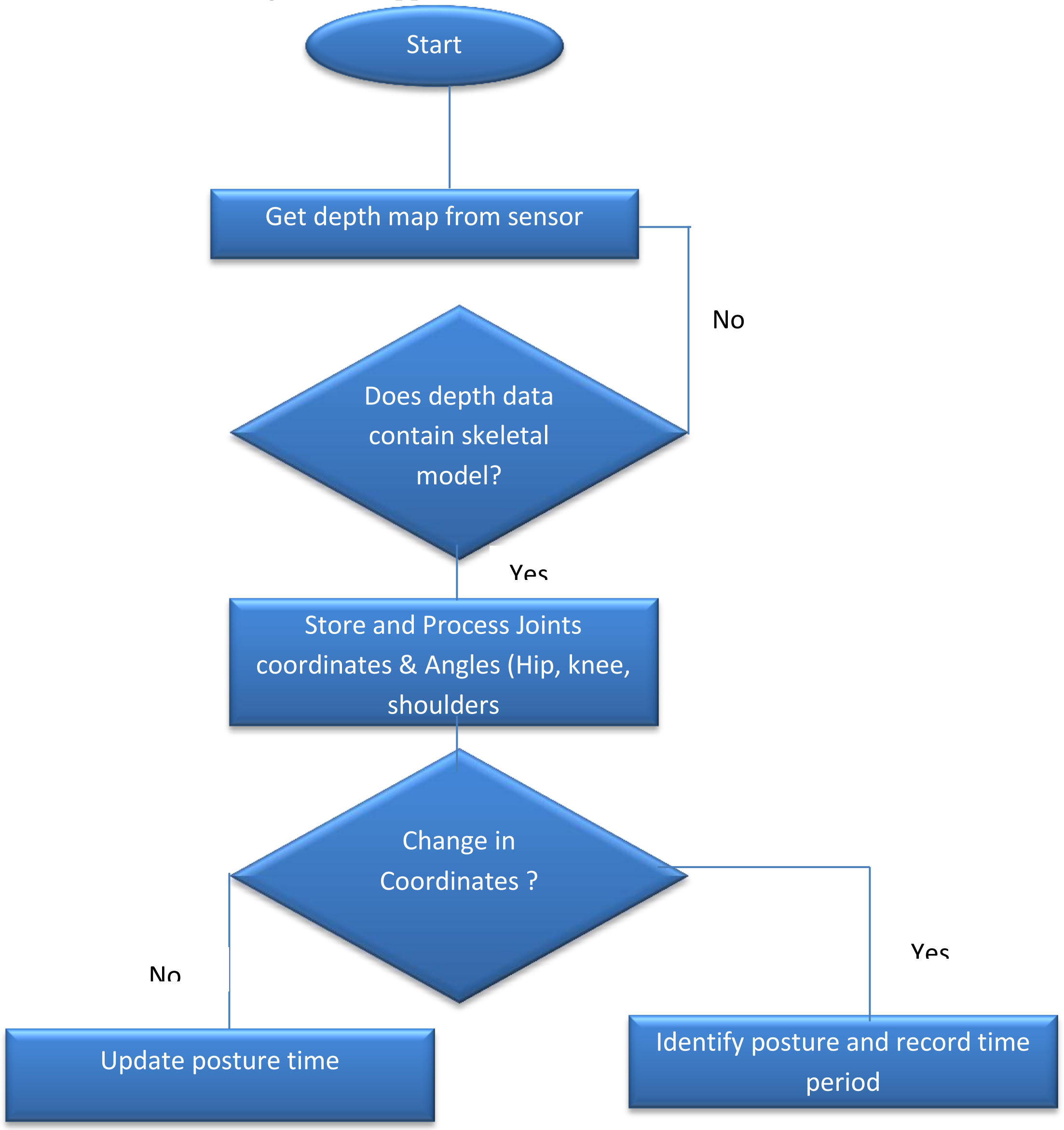
RESOURCE	**Cost (KES)**
Laptop PC	30,000.00
Kinect for PC device	21,000.00
Stationery	1,500.00
Miscellaneous	500.00
TOTAL	**53,000.00**

3.5 Implementation

The application was mainly developed using the Kinect SDK library. This SDK requires Microsoft .Net Framework 4.0 The application was built in Visual Studio 2010. The Kinect SDK can be downloaded at http://www.kinectforwindows.org/download/. The code in this project was written in WPF 4 and C#

3.5.1 The design flow

The design of the application can be illustrated as below:


Start
Get depth map from sensor
No
Does depth data contain skeletal model?
Yes
Store and Process Joints coordinates & Angles (Hip, knee, shoulders
Change in Coordinates ?
No
Yes
Update posture time
Identify posture and record time period

4. Discussion and Findings

In this section we describe the experiments performed to evaluate our method and achieve our objectives. We show both qualitative and quantitative results on our datasets and compare our approach with a window-based human detection algorithm by Ikemura and Fujiyoshi (2011)

4.1 Human Detection experiment

We evaluate our method using a sequence of depth arrays taken by the Kinect for XBOX 360 in indoor environment. We took the sequence in our lab with at most two persons presented in the scene. There are tables, chairs, shelves, computers, an overhead lamp and so on presented in the scene. The people have a variety of poses, and they have interaction with others or the surrounding objects.

There are 98 frames in the test set and the frame rate is 0.4s/frame. The size of the depth array is 1200 ×900 and the resolution is about 10mm.

To better illustrate our image frames, we scale the depth array and plot using color map as in Fig. 11. The depth is measured in millimeters and the points that failed to be measured are offset to 0. Our detection method performs well on our indoor dataset. Fig. 11 shows some of the results of our algorithm.

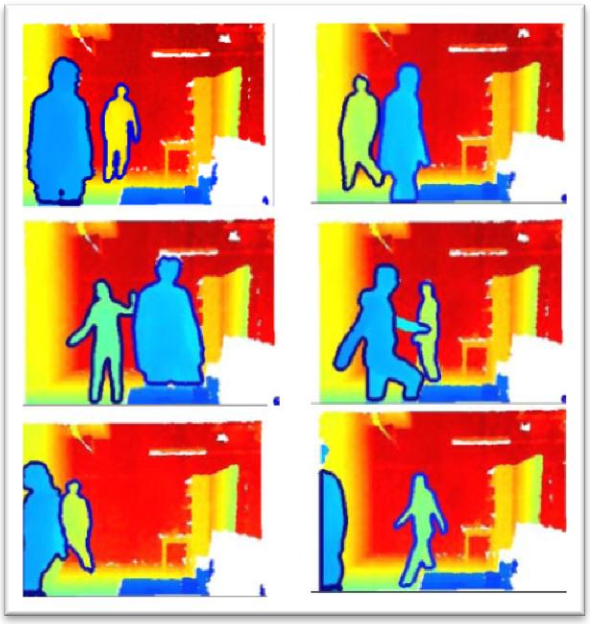


Fig 11 Examples of the human detection result.

Fig. 12 shows the preliminary tracking results based on our detection result. 15 consecutive frames are shown, which includes two people walking past each other, one person gets occluded and appears again. The detection method performs well in most cases. We do not have any FP instances but only a few FN detections. It happened when the person's head is occluded by another person or half of the body is out of the frame, as shown in Fig. 12.

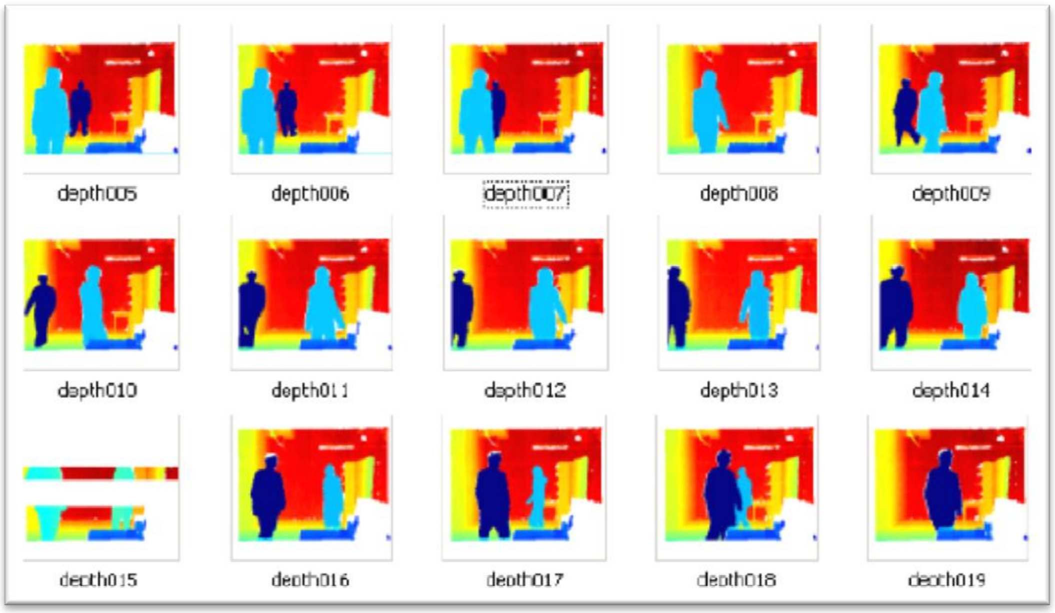


Fig 12. Tracking result

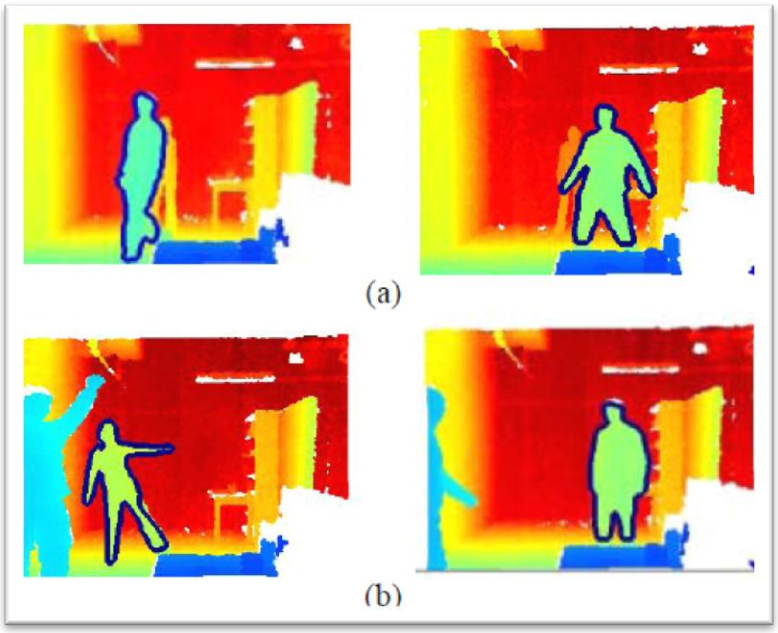


Fig. 13. FN examples. (a) The person behind is not detected. (b) The person on the left edge of the image is not detected.

We evaluate our method with different accuracy metrics, as shown in Table 2. The precision, recall and accuracy are defined as follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP= Number of True Positives

FP= Number of False Positives

FN= Number of False Negatives

TN = Number of True Negatives

We compare our algorithm with the one provided by Ikemura and Fujiyoshi (2011).

TP	**TN**	**FP**	**FN**
169	266	0	7
Precision		**Recall**	
100%		96.0%	
Accuracy			
98.4%			

Table 2 accuracy of our Algorithm

	Precision	**Recall**	**Accuracy**
Our	100%	96.0%	98.4%
Ikemura	90.0%	32.9%	85.8%

Table 3 Comparison of performance

The table above shows the comparison of performances of both methods. There are about 0 to 500 windows extracted from each frame and we subsample them and use the odd number of frames for training and even number of frames for testing. There are 770 positive examples and 2922 negative examples in the training set and 738 positive examples and 2930 negative examples in the test set. Note that the unit here is window not frame.

From Table 3 we can see that our algorithm outperforms Ikemura and Fujiyoshi (2011)'s algorithm on this dataset. The main reason is that Ikemura and Fujiyoshi (2011) window-based algorithm is better at handling the instances when the people in the frame are in an upright

position. However, people in this dataset are presented in all kinds of postures and rotations. The recall of Ikemura and Fujiyoshi (2011) algorithm is low because it is a window-based method which has a large false negative rate. The high false negative rate actually does not deteriorate his performance because the same person would appear in a lot of scanning windows. The algorithm will produce true positive when the person is well centered in the window, and it will classify the rest of the windows which the persons are not in
the center as negative frames. And that is the cause of the high false negative rate. But the person in the image is successfully detected in this case.

4.2 Human Posture Recognition Experiment

4.2.1 The Application: Human Posture Recognition in Action

The Human Posture Recognition application works through a series of steps namely Detection, Posture determination and Posture tracking.
The application consists of recognition and recommendation functionalities. It provides real time update of the current posture of the detected person and possible recommendations to the recognized posture.
The application furthermore records the time consumed for every posture change so as to provide for alerts such as desktop alerts, email and SMS.

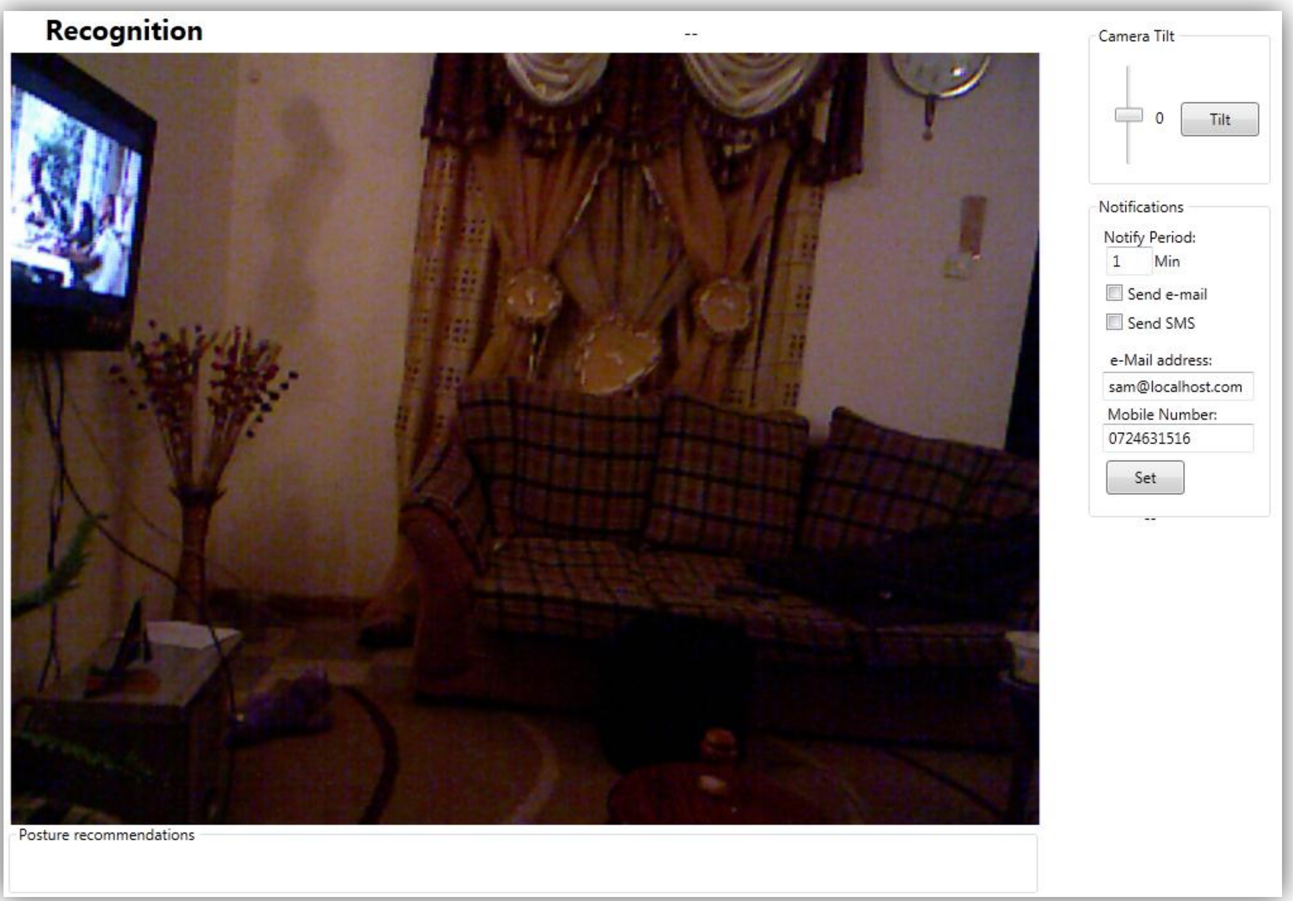


Fig 14: Human Posture Recognition application main window, without human detection on scene

i) Detection

The detection process is facilitated through the use of depth information fed from the Kinect device to perform head detection. The image fed is then mapped against a 3D image template.

Upon detection of human being on scene, the application displays a skeleton frame that synchronizes with the main image frame.

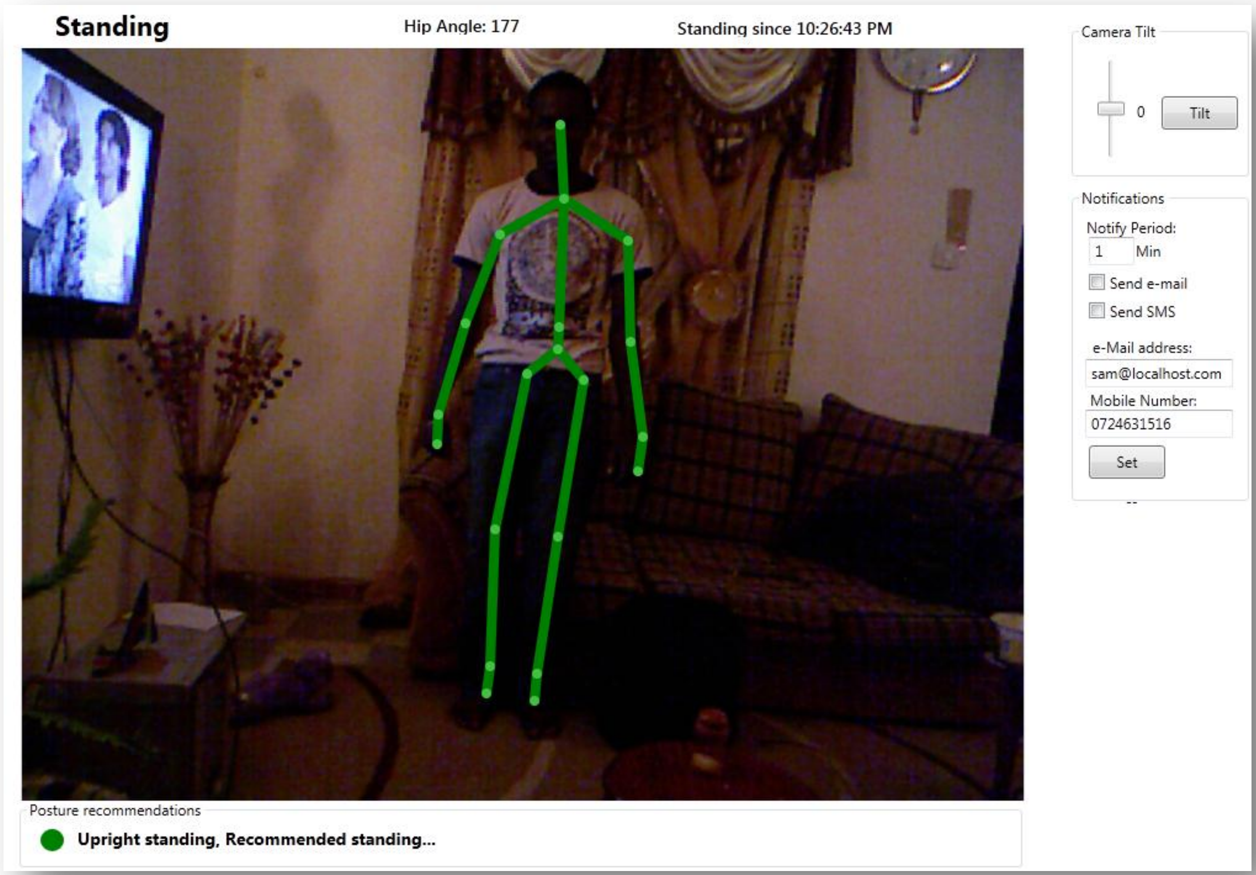


Fig 15: HPR window with detected human being on scene in standing pose

ii) Posture determination and Results finding:

When a presence of a person is detected in scene, body joints coordinates required for determination of posture are located in scene. These include knee, hip and shoulders.

- **Sitting posture recognition**

For sitting posture recognition, algorithm is based on obtaining three coordinates (xs, ys, zs), (xh, yh, zh), and (xk, yk, zk) of the positions of the human Shoulder (denoted as S), Hip (denoted as H), and Knee (denoted as K).

A sitting posture is related to the angle a between the line HK (from hip to knee) and the line HS (from hip to shoulder).

By considering the angle HK – HS, Observations and results obtained from the application about sitting posture can be tabulated as follows:

Angle	**Sitting posture**
10-80	Sitting: Bending forward
85-100	Straight sitting
133-138	Leaning back: best recommended sitting position
+145	Sleeping or tendency to standing pose

Consider screenshot below:

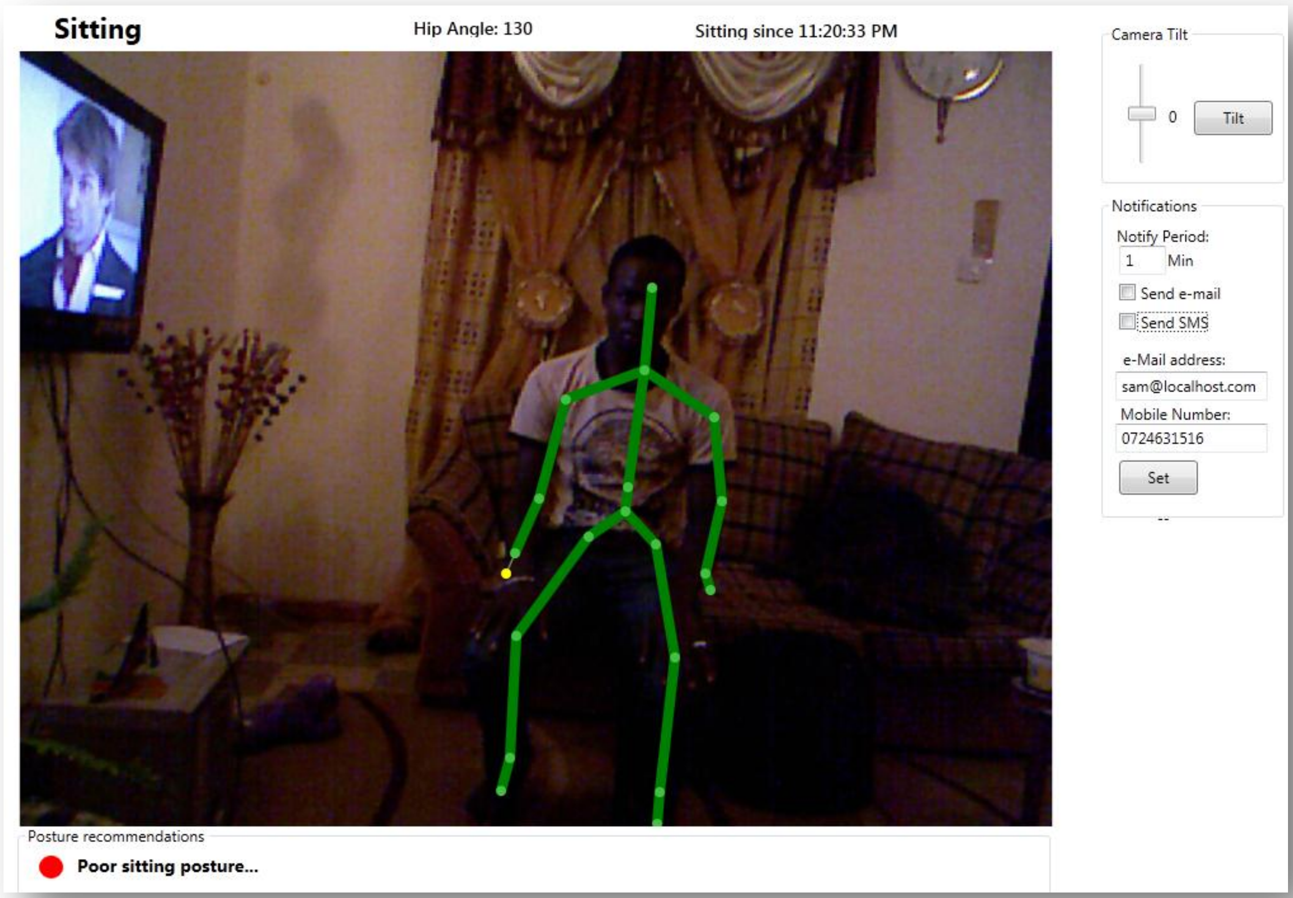


Fig 16: Object in sitting pose at hip angle 30 and application recommendation

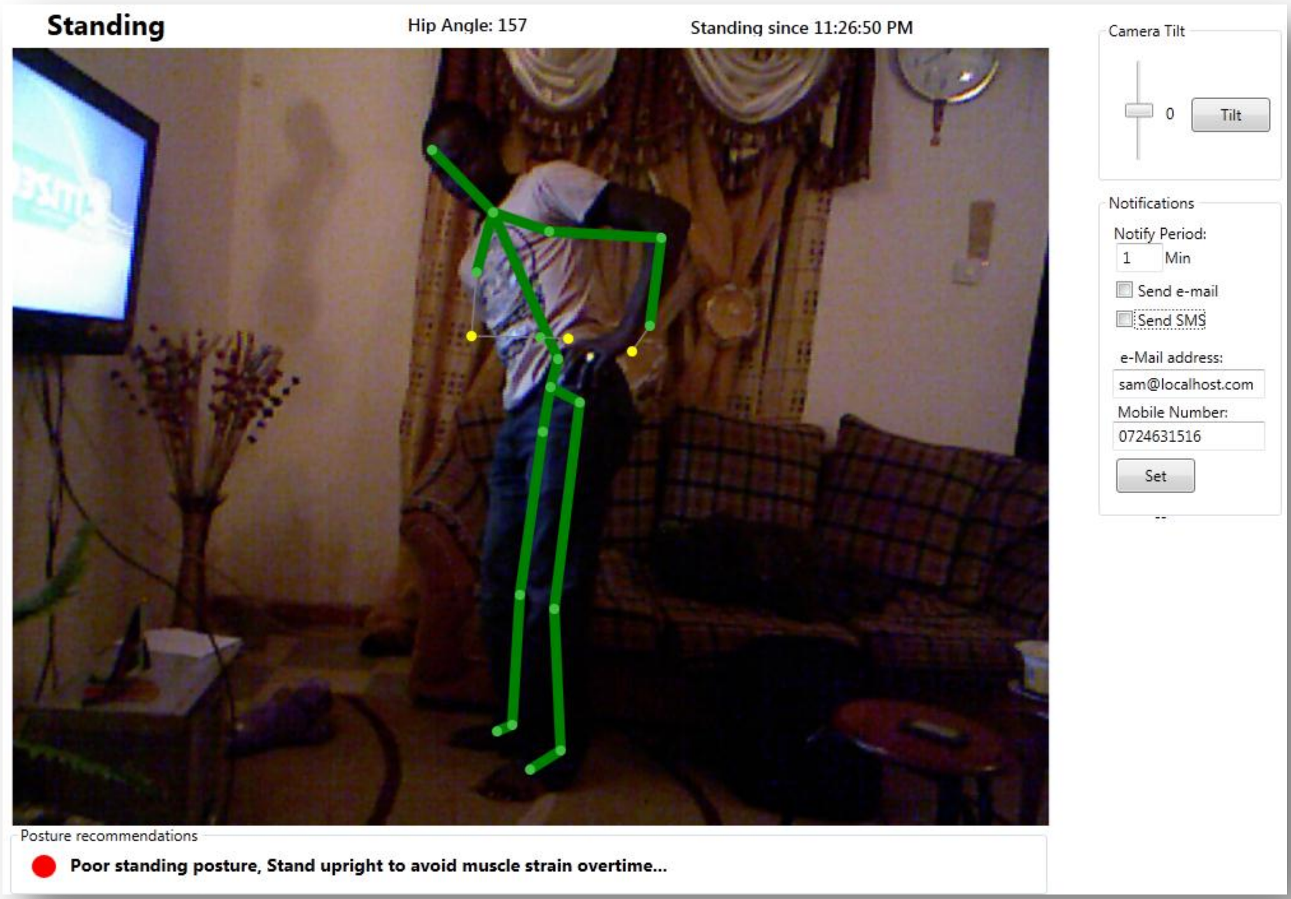
➢ **Standing posture recognition**

Standing posture recognition is also determined by the angle HK – HS as in sitting above and furthermore the depth values of the knee and hip joints of the body.

By observing the values of the angle from the application, results obtained can be tabulated as below:

Angle	**Standing posture**
150<x<175	Standing: Bending forward
± 180	Straight standing
+190	Leaning back

Consider screenshot below for standing posture:


Standing

Hip Angle: 157

Standing since 11:26:50 PM

Camera Tilt

0

Tilt

Notifications

Notify Period:

1 Min

Send e-mail

Send SMS

e-Mail address:

sam@localhost.com

Mobile Number:

0724631516

Set

Posture recommendations

● **Poor standing posture, Stand upright to avoid muscle strain overtime...**

Fig 17: Object in poor standing pose with recommendation

Recommendation in this application is carried out based on Bashir, Torio, Smith et al. (2006) and Kritz and Cronin (2008) researches for sitting and standing postures respectively as discussed earlier.

CONCLUSIONS

Value of Study, limitations and recommendations

In this paper, a new approach to human posture recognition has been presented to bring out the essential abilities of the Kinect device beyond the gaming experience that it is known for.
With that as a backdrop, here are a few ways in which Kinect and Human Posture Recognition could leap beyond this current project application.

1. Health and medicine

Many researchers at Microsoft and beyond have blogged at length about Kinect applications in telemedicine, neurological processes, physical therapy and medical training.
Kinect can also be applied to post-stroke patients or the cybertherapy pilots that is being implemented at various militaries.

2. Special needs children and adults

Kinect has proved how a different interface can literally open up new worlds. Research into computerized gaming and autism already reveals the potential for mirroring applications to improve facial recognition. Kinect hacks and human behavior analysis might take this further.

4. Education

Educational technology has grown in recent years through electronic publishing, learning management systems, and new models for virtual degrees earned online. Kinect's video and interface can allow for distributed teaching.

5. Participatory art

The idea of having Kinect-based puppet prototype will increase interactivity. As the Kinect platform matures, groups of people dancing in public and private spaces could explore new art forms that blend virtual and physical spaces. Flash mobs could get a lot more distributed.

6. Advertising and e-commerce

The Kinect can already detect multiple people in field. The Xbox 360 already enables users to watch live and recorded sports on ESPN TV channel. As more intelligence is built into the platform, combining those two capacities could go far beyond changing channels. It could lead to recognizing different users, profiles and creating a more interactive watching experience.

Limitations

The human posture recognition approach gives good results. However, the approach has some limitation.

The main limitations experienced during the observation of implementation are as follows:

- Kinect elevation angle:

For proper recognition the device requires a certain angle of about 45 degrees sideways from the object.

- Distance:

Kinect device works best at a distance of about 0.8 – 4 meters.

It is AC powered and not suitable for outdoors environments

- Multiple tracking:

The device can track at most 2 people at a time and detect at most 6 people in a scene.

In this application, recommendation and recognition is carried only on the first detected skeleton.

For a commercial application of human posture recognition, it can be recommended that multiple devices can be used for monitoring in order to increase the efficiency of the application.

Due to the above listed challenges and other aspects like variations in pose and clothing attire, the observation at runtime could suggest that the application has an accuracy level of 95%. This can be derived from getting the amount of time the application output is correct against the total runtime.

The work described in this paper can be extended to consider other activities (for example gestures)

REFERENCES

1. Bashir, W., Torio, T., Smith, F., et al. 2006. The way you sit will never be the same! Alterations of lumbosacral curvature and intervertebral disc morphology in normal subjects in variable sitting positions using whole-body positional MRI. *Presented at: Radiological Society of North America 2006 Meeting; November 27, 2006; Chicago, Illinois.*

2. Dalal, D. and Trigges, B. 2005. ‘Histograms of oriented gradients for human detection’, CVPR, 1, pp. 886-893.

3. Dalal, N.,Triggs, B. and Schmid. C. 2006. Human detection using oriented histograms of flow and appearance, in: *European Conference on Computer Vision*, Graz, Austria, May 7–13, 2006

4. Darrell, T., Gordon, G., Woodfill, J. and Harville, M. 1998. "Integrated Person Tracking using Stereo, Color, and Pattern Detection," Proceedings IEEE Conference on Computer Vision and Pattern Recognition, Santa Barbara, June 1998

5. Demirdjian, T. D. 2002.’ 3-D Articulated Pose Tracking for Untethered Diectic Reference’,ICMI, pp. 267-272

6. Ganapathi,V.C., Plagemann, Koller, D. and Thrun, S. 2010. Real time motion capture using a single time-of-flight camera. *Proceedings of CVPR* 2010. pp.755~762

7. Ikemura, S., Fujiyoshi, S. 2011. Real-Time Human Detection using Relational Depth Similarity Features. ACCV 2010, *Lecture Notes in Computer Science*, 6495, pp. 25-38

8. Jain, H.P. and Subramanian, A. 2010. Real-time upper-body human pose estimation using a depth camera. *HP Technical Reports*, HPL-2010-190

9. Jiang, H., Ze-Nian, L. and Drew, M.S. 2005. *Human Posture Recognition with Convex Programming*, Available http://www.cs.sfu.ca/~li/papers-on-line/Hao-ICME-05.pdf [Accessed 17 May 2012]

10. Kritz, M.F., & Cronin, J. 2008. Static posture assessment screen of athletes: Benefits and considerations. *Strength and Conditioning Journal*, 30 (5), 18–27.

11. Lowe, D.G. 1999. Object Recognition from Local Scale-Invariant Features. *Proceedings of the International Conference on Computer Vision*. Pp. 1150–1157

12. Pellegrini, S. and Iocchi, L. 2007. *Human Posture Tracking and Classification through Stereo Vision and 3D Model Matching*, Available http://www.dis.uniroma1.it/~iocchi/publications/iocchi-jivp08.pdf [Accessed 17 May 2012]

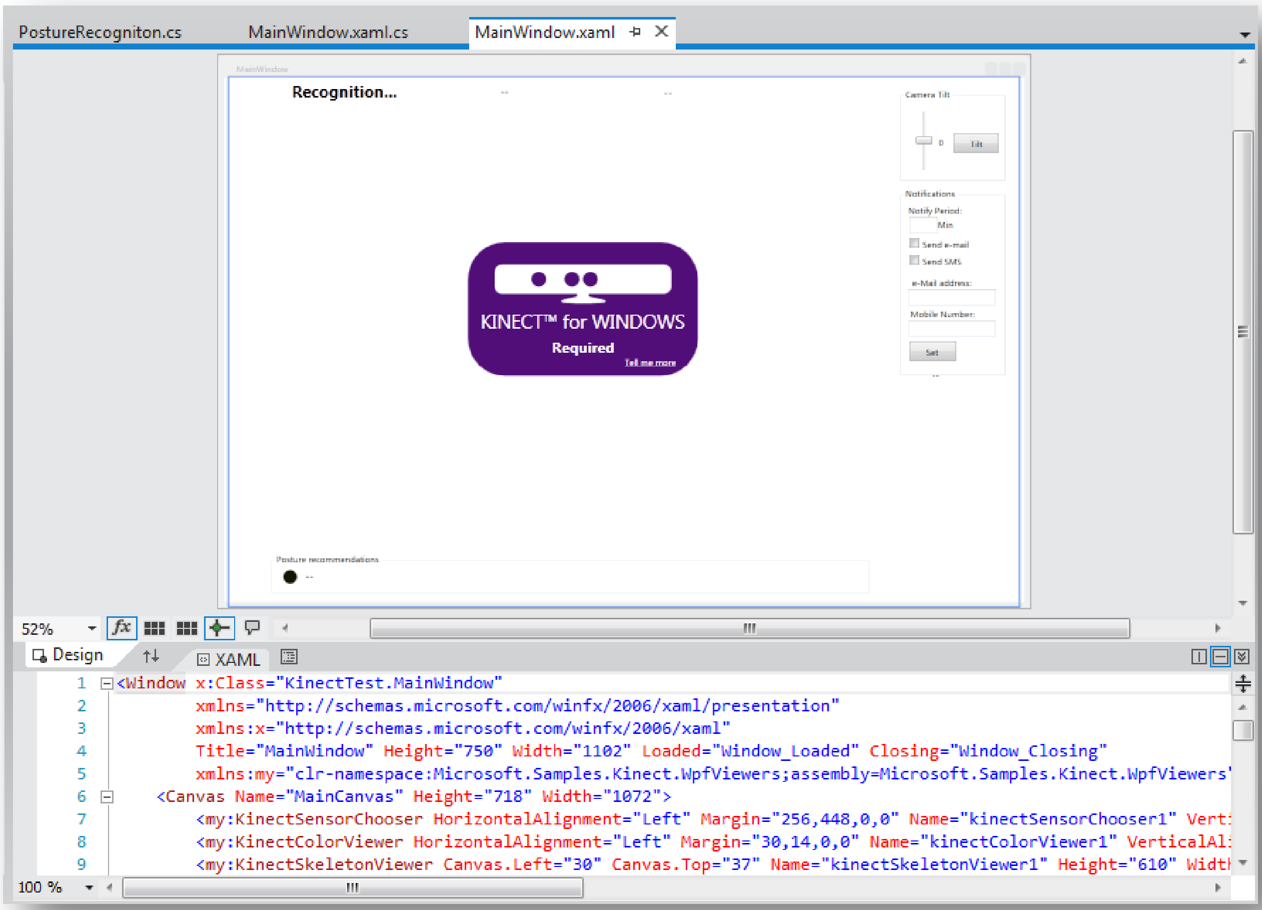
13. Rodgers,J., Anguelov, D., Pang, H.C. and Koller, D. 2006.' Object pose detection in range scan data'. In Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2006

14. Webb, J. and Ashley, J. 2012. *Beginning Kinect Programming with the Microsoft Kinect SDK*,New York, NY: Apress

Appendices

➔ Design and Source code

Human Posture Recognition was developed as a WPF4 application in visual studio. The main application window is as below:



```
<Window x:Class="KinectTest.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="750" Width="1102" Loaded="Window_Loaded" Closing="Window_Closing"
        xmlns:my="clr-namespace:Microsoft.Samples.Kinect.WpfViewers;assembly=Microsoft.Samples.Kinect.WpfViewers'
    <Canvas Name="MainCanvas" Height="718" Width="1072">
        <my:KinectSensorChooser HorizontalAlignment="Left" Margin="256,448,0,0" Name="kinectSensorChooser1" Verti
        <my:KinectColorViewer HorizontalAlignment="Left" Margin="30,14,0,0" Name="kinectColorViewer1" VerticalAl
        <my:KinectSkeletonViewer Canvas.Left="30" Canvas.Top="37" Name="kinectSkeletonViewer1" Height="610" Widt
```

Fig 18. HPR, Design Main window

The main Window XAML source:

```
<Window x:Class="KinectTest.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="750" Width="1102" Loaded="Window_Loaded"
Closing="Window_Closing"
        xmlns:my="clr-
namespace:Microsoft.Samples.Kinect.WpfViewers;assembly=Microsoft.Samples.Kinect.WpfViewers
" WindowState="Maximized">
    <Canvas Name="MainCanvas" Height="718" Width="1072">
```

```
        <my:KinectSensorChooser HorizontalAlignment="Left" Margin="256,448,0,0"
Name="kinectSensorChooser1" VerticalAlignment="Top" Width="312" Height="181"
Canvas.Left="68" Canvas.Top="-224" />
        <my:KinectColorViewer HorizontalAlignment="Left" Margin="30,14,0,0"
Name="kinectColorViewer1" VerticalAlignment="Top" Height="610" Width="869"
Kinect="{Binding ElementName=kinectSensorChooser1, Path=Kinect}" Canvas.Top="23" />
        <my:KinectSkeletonViewer Canvas.Left="30" Canvas.Top="37"
Name="kinectSkeletonViewer1" Height="610" Width="869" Kinect="{Binding
ElementName=kinectSensorChooser1, Path=Kinect}" />
        <TextBlock Canvas.Left="85" Canvas.Top="3" Height="37" Name="textBlock1"
Text="Recognition..." Width="247" FontWeight="ExtraBold" FontSize="22" />
        <Label Canvas.Left="585" Canvas.Top="6" Content="--" Grid.Column="1" Height="27"
Name="lblLastTime" Width="314" FontSize="14" FontWeight="SemiBold" />
        <Label Canvas.Left="949" Canvas.Top="391" Content="--" Grid.Column="1" Height="28"
Name="lblHeadDist" Width="99" />
        <Label Canvas.Left="363" Canvas.Top="4" Grid.Column="1" Height="27"
Name="lblHipAngle" Width="162" Content="--" FontSize="14" FontWeight="SemiBold" />
        <GroupBox Canvas.Left="909" Canvas.Top="14" Header="Camera Tilt" Height="128"
Name="groupBox1" Width="146"></GroupBox>
        <Grid Height="110" Width="124" Canvas.Left="927" Canvas.Top="32">
            <Grid.RowDefinitions>
                <RowDefinition Height="216*" />
                <RowDefinition Height="0*" />
            </Grid.RowDefinitions>
            <Slider Height="88" HorizontalAlignment="Left" Margin="4,10,0,0"
Name="slider1" VerticalAlignment="Top" Width="34" Orientation="Vertical" Minimum="-27"
Maximum="27" ValueChanged="slider1_ValueChanged" />
            <Button Content="Tilt" Height="27" HorizontalAlignment="Left"
Margin="56,44,0,0" Name="button1" VerticalAlignment="Top" Width="62" Click="button1_Click"
/>
            <Label Content="0" Height="28" HorizontalAlignment="Left" Margin="31,43,0,0"
Name="label1" VerticalAlignment="Top" />
        </Grid>
        <GroupBox Canvas.Left="909" Canvas.Top="149" Header="Notifications" Height="256"
Name="groupBox2" Width="146">
            <Grid Width="126">
                <CheckBox Content="Send e-mail" Height="16" HorizontalAlignment="Left"
Margin="5,53,0,0" Name="chkSendMail" VerticalAlignment="Top" IsChecked="False"
Checked="chkSendMail_Checked" Unchecked="chkSendMail_Unchecked" />
                <CheckBox Content="Send SMS" Height="16" HorizontalAlignment="Left"
Margin="5,76,0,0" Name="chkSMS" VerticalAlignment="Top" IsChecked="False"
Checked="chkSMS_Checked" Unchecked="chkSMS_Unchecked" />
                <TextBox Height="23" HorizontalAlignment="Left" Margin="2,123,0,0"
Name="txtemail" VerticalAlignment="Top" Width="120" Text="" />
                <TextBox Height="23" HorizontalAlignment="Left" Margin="2,164,0,0"
Name="txtMobile" VerticalAlignment="Top" Width="120" Text="" />
                <Label Content="e-Mail address:" Height="28" HorizontalAlignment="Left"
Margin="3,100,0,0" Name="label2" VerticalAlignment="Top" Width="103" />
                <Label Content="Mobile Number:" Height="28" HorizontalAlignment="Left"
Margin="1,143,0,0" Name="label3" VerticalAlignment="Top" Width="103" />
                <Button Content="Set" Height="27" HorizontalAlignment="Left"
Margin="5,193,0,0" Name="btnSet" VerticalAlignment="Top" Width="62" Click="btnSet_Click"
/>
                <TextBox Height="23" HorizontalAlignment="Left" Margin="5,24,0,0"
Name="txtNotifyPeriod" Text="" VerticalAlignment="Top" Width="37" />
                <Label Content="Notify Period:" Height="28" HorizontalAlignment="Left"
Margin="-2,3,0,0" Name="label4" VerticalAlignment="Top" Width="103" />
                <Label Content="Min" Height="28" HorizontalAlignment="Left"
Margin="38,22,0,0" Name="label5" VerticalAlignment="Top" Width="38" />
            </Grid>
        </GroupBox>
        <GroupBox Canvas.Left="55" Canvas.Top="645" Header="Posture recommendations"
Height="57" Name="recGroup" Width="816">
            <Grid>
                <Label Height="28" HorizontalAlignment="Left" Margin="37,1,0,0"
Name="lblrecommendations" VerticalAlignment="Top" Width="761" Content="--" FontSize="14"
FontWeight="Bold" />
                <Ellipse Height="18" HorizontalAlignment="Left" Margin="12,8,0,0"
Name="recEllipse" Stroke="{x:Null}" VerticalAlignment="Top" Width="19" Fill="#FF141404" />
            </Grid>
        </GroupBox>
    </Canvas>
</Window>
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;
using Coding4Fun.Kinect;
using Coding4Fun.Kinect.Wpf;
using PostureRecognition;
using System.Net.Mail;


namespace KinectTest
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        double recognitionHeadDistance = 0;
        DateTime lastTime = DateTime.MaxValue;
        string CurrentPosition = "None";
        bool IsClosing = false;
        bool Notified = false;
        const int SkeletonCount = 6;
        Skeleton[] allskeletons = new Skeleton[SkeletonCount];
        DepthImageFrame _LastDepthFrame;
        SmtpClient _client = new SmtpClient("sam-pc", 25);
        smsDataContext db = new smsDataContext();

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {

            kinectSensorChooser1.KinectSensorChanged += new
DependencyPropertyChangedEventHandler(kinectSensorChooser1_KinectSensorChanged);
```

```
            this.Title = "Human Posture Recognition using Kinect";
            lastTime = DateTime.Now;
            lblHipAngle.Content = "";
            chkSendMail.IsChecked = Properties.Settings.Default.SendEmail;
            chkSMS.IsChecked = Properties.Settings.Default.SendSMS;
            txtemail.Text = Properties.Settings.Default.email;
            txtMobile.Text = Properties.Settings.Default.SMSNumber;
            txtNotifyPeriod.Text = Properties.Settings.Default.NotifyPeriod.ToString();
            recEllipse.Fill = null;
        }

        void _sensor_AllFramesReady(object sender, AllFramesReadyEventArgs e)
        {

            if (IsClosing)
            {
                return;
            }
            Skeleton first = GetFirstSkeleton(e);

            if (first == null)
            {
                lblLastTime.Content = "--";
                textBlock1.Text = "Recognition";
                lblHipAngle.Content = "";
                lblrecommendations.Content = "";
                recEllipse.Fill = null;
                return;
            }

         //Try get distance
            //double xPos = first.Joints[JointType.Head].Position.X;
            //double yPos = first.Joints[JointType.Head].Position.Y;

            //using (DepthImageFrame frame = e.OpenDepthImageFrame())
            //{
            //    if (frame != null)
            //    {
            //        short[] pixelData = new short[frame.PixelDataLength];
            //        frame.CopyPixelDataTo(pixelData);
            //        int stride = frame.Width * frame.BytesPerPixel;
            //        if (pixelData != null && pixelData.Length > 0)
            //        {
            //            int pixelIndex = (int)(xPos + ((int)yPos * frame.Width));
            //            int depth = pixelData[pixelIndex] >>
DepthImageFrame.PlayerIndexBitmaskWidth;
            //            //int depthInches = (int)(depth * 0.0393700787);
            //            //int depthFt = depthInches / 12;
            //            lblHeadDist.Content = depth + "mm";
            //        }
            //    }
            //}

            double angle;
            if (first.Joints[JointType.ShoulderCenter].TrackingState ==
JointTrackingState.Tracked && first.Joints[JointType.HipCenter].TrackingState ==
JointTrackingState.Tracked)
            {
                if (first.Joints[JointType.KneeRight].TrackingState ==
JointTrackingState.Tracked)
                {
                    angle = 
Recogniton.CalculateAngle(first.Joints[JointType.ShoulderCenter], 
first.Joints[JointType.HipCenter], first.Joints[JointType.KneeRight]);
                    lblHipAngle.Content = "Hip Angle: " + (int)angle;
                    TimeSpan postureDuration = DateTime.Now - lastTime;

                    if (angle < 40)
                        textBlock1.Text = "Sitting";
                    if (40 < angle && angle < 145)// || 100 < angle && angle < 180)
                    {
                        //check if the person bends right knee while standing to avoid 
confusion with sitting
                        if (IsStanding(first) == true)
                        {
                            return;
                        }
                        CheckPosition();
                        if (textBlock1.Text != "Sitting")
                        { CurrentPosition = "Sitting"; lastTime = DateTime.Now; 
lblLastTime.Content = CurrentPosition + " since " + lastTime.ToString("hh:mm:ss tt"); }
                        textBlock1.Text = "Sitting";
                        CheckPostureRecommendations("Sitting", angle);
                    }

                        if (angle >= 150)
                        {
                            if (textBlock1.Text != "Standing")

                            { CurrentPosition = "Standing"; lastTime = DateTime.Now; 
Notified = false; lblLastTime.Content = CurrentPosition + " since " + 
lastTime.ToString("hh:mm:ss tt"); }
                            textBlock1.Text = "Standing";
                            CheckPostureRecommendations("Standing", angle);
                        }
                    //if  (lblLastTime.Content.ToString !="--")
                    //{
                    //    lblLastTime.Content = lblLastTime.Content + "(" + 
postureDuration.TotalSeconds + " sec)";
                    //}
                }

            }

        }

        void CheckPostureRecommendations(string posture, double angle)
        {
            //recommendations for standing posture
            if (posture == "Standing")
            {
                if (angle > 150 && angle <= 167)
                {
                    lblrecommendations.Content = "Poor standing posture, Stand upright to 
avoid muscle strain overtime...";
                    recEllipse.Fill = Brushes.Red;
                }
                if (angle > 167 && angle <= 180)
                {
                    lblrecommendations.Content = "Upright standing, Recommended 
standing...";
                    recEllipse.Fill = Brushes.Green;
                }
            }
            //recommendations for sitting posture
            if (posture == "Sitting")
            {
                if (angle > 45 && angle <= 80)
                {
                    lblrecommendations.Content = "Poor sitting posture [Bending forward], 
lean back to avoid muscle strain overtime...";
                    recEllipse.Fill = Brushes.Red;
                }
                if (angle > 80 && angle <= 125)
                {
                    lblrecommendations.Content = "upright sitting,not recommended for 
healthy back...";
                    recEllipse.Fill = Brushes.Red;
                }
                if (angle > 128 && angle <= 138)
                {
                    lblrecommendations.Content = "Proper Sitting, recommended sitting 
posture...";

                    recEllipse.Fill = Brushes.Green;
                }
                if (angle > 138 && angle <= 148)
                {
                    lblrecommendations.Content = "Poor sitting posture...";
                    recEllipse.Fill = Brushes.Red;
                }
            }

        }

        void CheckPosition()
        {
            //int PositionTime=
            if (textBlock1.Text == "Sitting")
            {
                TimeSpan postureTime = DateTime.Now - lastTime;

                if (postureTime.TotalMinutes >= Properties.Settings.Default.NotifyPeriod 
&& Notified==false)
                {

                    try
                    {
                        if (Properties.Settings.Default.SendSMS == true)
                        {
                            Notified = true;
                            //send text message to user...
                            ozekimessageout sms=new ozekimessageout();
                            sms.msg="Boss, too much sitting..Mind your health!!";
                            sms.receiver=Properties.Settings.Default.SMSNumber;
                            sms.status="send";
                            db.ozekimessageouts.InsertOnSubmit(sms);
                            db.SubmitChanges();
                        }

                        if (Properties.Settings.Default.SendEmail == true)
                        {
                           Notified = true;
                          _client.Send("admin@localhost.com", 
Properties.Settings.Default.email , "HPR Notification", "Boss...Sitting for too long Mind 
your health!!");
```

```
                          //MessageBox.Show("You have been sitting for more than " +
Properties.Settings.Default.NotifyPeriod +" min", "HPR", MessageBoxButton.OK);
                        }


                    }
                    catch (Exception)
                    {

                       // throw;
                    }
                    //send notification
                    Notification notification=new Notification();
                    notification.Msg = "Oops! You have been in that position for more than
" + Properties.Settings.Default.NotifyPeriod + " min";
                    db.Notifications.InsertOnSubmit(notification);
                    db.SubmitChanges();

                }

            }
        }

        bool IsStanding(Skeleton first)
        {
            bool Standing=false;
            if (first.Joints[JointType.KneeLeft].TrackingState ==
JointTrackingState.Tracked)
            {
                double angle =
Recogniton.CalculateAngle(first.Joints[JointType.ShoulderCenter],
first.Joints[JointType.HipCenter], first.Joints[JointType.KneeLeft]);
                if (angle > 150)
                {
                    Standing = true;
                }
            }
            return Standing;
        }

        Skeleton GetFirstSkeleton(AllFramesReadyEventArgs e)
        {
            using (SkeletonFrame skeletonFrameData = e.OpenSkeletonFrame())
            {
                if (skeletonFrameData == null)
                {
                    return null;
                }


                skeletonFrameData.CopySkeletonDataTo(allskeletons);

                //get the first tracked skeleton
                Skeleton first = (from s in allskeletons
                                  where s.TrackingState == SkeletonTrackingState.Tracked
                                  select s).FirstOrDefault();

                return first;

            }
        }
        void CheckHeadDistance(AllFramesReadyEventArgs e)
        {

        }

        void kinectSensorChooser1_KinectSensorChanged(object sender,
DependencyPropertyChangedEventArgs e)
        {
            KinectSensor Oldsensor = (KinectSensor)e.OldValue;
            StopKinect(Oldsensor);

            KinectSensor newSensor = (KinectSensor)e.NewValue;

            if (newSensor == null)
            {
                return;
            }

            //var parameters = new TransformSmoothParameters
            //{
            //    Smoothing = 0.3f,
            //    Correction = 0.0f,
            //    Prediction = 0.0f,
            //    JitterRadius = 1.0f,
            //    MaxDeviationRadius = 0.5f
            //};

            //newSensor.SkeletonStream.Enable(parameters);
            //newSensor.ColorStream.Enable();
            //newSensor.DepthStream.Enable();
            newSensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
            newSensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
            newSensor.SkeletonStream.Enable();
                    newSensor.AllFramesReady += new
EventHandler<AllFramesReadyEventArgs>(_sensor_AllFramesReady);
                   // _sensor.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(_sensor_SkeletonFrameReady);
                   // newSensor.DepthFrameReady += new
EventHandler<DepthImageFrameReadyEventArgs>(newSensor_DepthFrameReady);
                    try
                    {
                        newSensor.Start();
                    }
                    catch (System.IO.IOException)
                    {

                        kinectSensorChooser1.AppConflictOccurred();
                    }

                    }

            void newSensor_DepthFrameReady(object sender, DepthImageFrameReadyEventArgs e)
               {
                   //if (this._LastDepthFrame != null)
                   //{
                   //    this._LastDepthFrame.Dispose();
                   //    this._LastDepthFrame = null;
                   //}
                   //this._LastDepthFrame = e.OpenDepthImageFrame();
                   //if (this._LastDepthFrame != null)
                   //{
                   //    this._LastDepthFrame.CopyPixelDataTo(this._DepthImagePixelData);
```

```
                    //    this._RawDepthImage.WritePixels(this._RawDepthImageRect,
this._DepthImagePixelData,
                    //    this._RawDepthImageStride, 0);
                    //}
                }

        void StopKinect(KinectSensor sensor)
        {
            if (sensor != null)
            {
                sensor.Stop();
                sensor.AudioSource.Stop();
            }
        }


       private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            IsClosing = true;
            StopKinect(kinectSensorChooser1.Kinect);
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            kinectSensorChooser1.Kinect.ElevationAngle = (int)slider1.Value;
        }

        private void slider1_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            label1.Content = (int)slider1.Value;
        }

        private void chkSendMail_Checked(object sender, RoutedEventArgs e)
        {
            Properties.Settings.Default.SendEmail = true;
            Properties.Settings.Default.Save();
        }

        private void chkSendMail_Unchecked(object sender, RoutedEventArgs e)
        {
            Properties.Settings.Default.SendEmail = false;
            Properties.Settings.Default.Save();
        }

        private void btnSet_Click(object sender, RoutedEventArgs e)
        {
            Properties.Settings.Default.email = txtemail.Text;
            Properties.Settings.Default.SMSNumber = txtMobile.Text;
            Properties.Settings.Default.NotifyPeriod =
Convert.ToInt16(txtNotifyPeriod.Text);
            Properties.Settings.Default.Save();
        }

        private void chkSMS_Checked(object sender, RoutedEventArgs e)
        {
            Properties.Settings.Default.SendSMS = true;
            Properties.Settings.Default.Save();
        }

        private void chkSMS_Unchecked(object sender, RoutedEventArgs e)
        {
            Properties.Settings.Default.SendSMS = false;
            Properties.Settings.Default.Save();
        }
    }
}
```

The Posture Recognition Module (Class)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Kinect;

namespace PostureRecognition
{
    public static class Recogniton
    {


        /// <summary>
        /// Calculate angle between vectors HipShoulder and KneeShoulder
        /// </summary>
        /// <param name="shoulder">shoulder position</param>
        /// <param name="hip">hip position</param>
        /// <param name="knee">knee position</param>
        /// <returns></returns>
        public static double CalculateAngle(Joint shoulder, Joint hip, Joint knee)
        {
            double angle = 0;
            double shrhX = shoulder.Position.X - hip.Position.X;
            double shrhY = shoulder.Position.Y - hip.Position.Y;
            double shrhZ = shoulder.Position.Z - hip.Position.Z;
            double hsl = vectorNorm(shrhX, shrhY, shrhZ);
            double unrhX = knee.Position.X - hip.Position.X;
            double unrhY = knee.Position.Y - hip.Position.Y;
            double unrhZ = knee.Position.Z - hip.Position.Z;
            double hul = vectorNorm(unrhX, unrhY, unrhZ);
            double mhshu = shrhX * unrhX + shrhY * unrhY + shrhZ * unrhZ;

            double x = mhshu / (hul * hsl);
            if (x != Double.NaN)
            {
                if (-1 <= x && x <= 1)
                {
                    double angleRad = Math.Acos(x);
                    angle = angleRad * (180.0 / 3.1416);
                }
                else
                    angle = 0;


            }
            else
                angle = 0;


            return angle;
        }
        // RecognizeSittingPoseByAngle
        /// <summary>
        ///  RecognizeSittingPoseByAngle
        ///  0 ~ 40  down   sleeping
        ///  40~80   down   non-concentrating
        ///  80~100  down   concentrating
        ///   up     raising hand
        /// </summary>
        /// <param name="angle">angle between vectors HipShoulder and KneeShoulder</param>
        /// <returns></returns>
        public static string RecognizeSittingPoseByAngle(double angle)
        {
            string result = null;
            if (angle < 40)
                result = "Sitting -- Bending forward";
            if (40 < angle && angle < 120)
                result = "Sitting";
            if (angle >= 145)
                result = "Standing";
            return result;
        }
        /// <summary>
        /// Euclidean norm of 3-component Vector
        /// </summary>
        /// <param name="x"></param>
        /// <param name="y"></param>
        /// <param name="z"></param>
        /// <returns></returns>
        public static double vectorNorm(double x, double y, double z)
        {
            return Math.Sqrt(Math.Pow(x, 2) + Math.Pow(y, 2) + Math.Pow(z, 2));

        }
    }
}
```