BEHAVIORAL ANALYTICS IN VIDEO SURVEILLANCE

PANG JUNN MIN

MASTER OF ENGINEERING SCIENCE

FACULTY OF ENGINEERING AND GREEN TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN
SEPTEMBER 2015

BEHAVIORAL ANALYTICS IN VIDEO SURVEILLANCE

By

PANG JUNN MIN

A dissertation submitted to the Department of Electronic Engineering,
Faculty of Engineering and Green Technology,
Universiti Tunku Abdul Rahman,
in partial fulfillment of the requirements for the degree of
Master of Engineering Science
August 2014

ABSTRACT

BEHAVIORAL ANALYTICS IN VIDEO SURVEILLANCE

PANG JUNN MIN

Video surveillance system is widely used for ensure the public safety and security. Unfortunately, most of the existing surveillance systems do not have additional functions such as human behavioral analytic system for enhancing security of public areas. Therefore, this research has proposed human behavior analysis as an input for video surveillance system by employing Microsoft Kinect sensor. Microsoft Kinect sensor is not affected by variation of illumination. In the proposed algorithm, human body joints data is collected by Microsoft Kinect sensor. It can detect five basic behaviors (sitting, crouching, jumping, walking and running) and two aggressive behaviors (punching and kicking). In addition, it provides fall detection as well. The proposed method is verified with a total of 30 experimental subjects. Every participant has performed all the behaviors with different angles (varies from -180º to 180º) between the Kinect sensor and human body. The proposed algorithm is compared with Ray's Dynamic Time Warping (DTW) gesture matching system. The results show that the proposed algorithm has a higher achievement in terms of accuracy rate and sensitivity. Experimental results show that the proposed method has an average sensitivity of 93.62% for human basic behavior, 97.92% for human aggressive behavior, and 100% for fall detection.

PUBLICATIONS

Based on the work of this dissertation, a conference paper has been published as shown in table below:

No	Category	Title	Publisher	status
1	Conference	Behavioral Analytics in Video Surveillance Print ISBN: 978-1-4799-5685-2 DOI: 10.1109/ICCSCE.2014.7072683	2014 IEEE International Conference on Control System, Computing and Engineering	Published

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Yap Vooi Voon and my co-supervisor Engr. Dr. Soh Chit Siang for their direction, assistance, and guidance. In particular, Dr. Yap Vooi Voon's numerous recommendations and suggestions have been invaluable for the project.

I also wish to thank Dr. Teh Peh Chiong, who has loaned me the equipment to setup the system. Special thanks must also go to Mr. Thong Marn Foo and all the lab officers for their assistance in the laboratory.

Last but not least, I offer my deepest regards and blessings to all of those who supported me in any respect during the completion of the project.

APPROVAL SHEET

This dissertation entitled "**BEHAVIORAL ANALYTICS IN VIDEO SURVEILLANCE"** was prepared by PANG JUNN MIN and submitted as partial fulfillment of the requirements for the degree of Master of Engineering Science at Universiti Tunku Abdul Rahman.

Approved by:

_____

(Asst. Prof. Dr. YAP VOOI VOON)

Date:……..

Assistant Professor/Supervisor

Department of Electronic Engineering

Faculty of Engineering and Green Technology

Universiti Tunku Abdul Rahman

_____

(Asst. Prof. Dr. SOH CHIT SIANG)

Date:……..

Assistant Professor/Co-supervisor

Department of Electronic Engineering

Faculty of Engineering and Green Technology

Universiti Tunku Abdul Rahman

FACULTY OF ENGINEERING AND GREEN TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: _____

SUBMISSION OF THESIS

It is hereby certified that **Pang Junn Min** (ID No: **12AGM07916**) has completed this dissertation entitled "Behavioral Analytics in Video Surveillance" under the supervision of Dr. Yap Vooi Voon (Supervisor) from the Department of Electronic Engineering, Faculty of Engineering and Green Technology, and Dr Soh Chit Siang (Co-Supervisor) from the Department of Electronic Engineering, Faculty of Engineering and Green Technology.

I understand that University will upload softcopy of my dissertation in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

_____

(Pang Junn Min)

DECLARATION

I, Pang Junn Min hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.

_____

(PANG JUNN MIN)

Date_____

Contents

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 Scientific Motivation

Video surveillance system is commonly used to monitor safety and security in public places such as bank, shopping mall and train station. Traditional surveillance technology requires manpower to monitor multiple CCTV screens while more advance surveillance systems can detect and alert the operator regarding events that may cause potential security risks. However, these systems have limited functions. For example, typical features in advanced surveillance systems include motion detection for intrusion detection, and possibly some trajectory analysis in structured environments (Luo et al., 2013) usually are not available in inexpensive and less advanced systems. However, these system can be used to detect aggressive behavior if additional features such as human behavioral analysis is embedded in the system. Nonetheless, one of the challenging part of human behavioral analysis system is body pose recognition which will be discussed in section 1.3.

1.2 Applications

Human behavioral analytic system in video surveillance is important to ensure the security of public. The sub-sections below give an overview of few places where the proposed system will be useful.

1.2.1 Railway Stations/Airports

Human Behavioral analysis system can be installed in public area of airports and railway stations. The safety of the passengers is the main concern for these places. A robust human behavior analysis system will alert the security guards when violent behavioral or a fall event is detected.

1.2.2 Hospitals

The human behavioral analysis system can be integrated with a fall detection system. When a fall event occurs in the patients' room, the proposed system can alert the staff. Detecting a fall event of a patient as soon as possible can help improve the chances of taking the appropriate action as soon as possible.

1.3 Body Pose Recognition/Estimation

The body pose recognition is a very important process in human behavioral analysis system. It allows the system to estimate the subjects' pose before analyzing their behavior. However, body pose recognition is a challenging area of research in field of image processing and analysis. Because of the variety of situations and complexity of the human body for pose observation, recover and recognizing the pose of human body has become a challenging problem in computer vision. In addition, the high number of degrees-of-freedom (DOF) in the body's movement is also a major challenge in recovering pose from video or images. Human body contains more than 20 DOFs, when body parts occlude with each other, it will be very difficult to recover the exact pose. Other difficulties in recovering pose are the variations in lighting, subjects attires or body shapes, as well as camera configuration, and loss of 3D data when observing the pose from 2D image (Disney, 2014).

Body pose estimation has potential in the field of ergonomics studies, robot control, 3D animation, marker-less motion capture (MoCap) for human-computer interfaces and surveillance system (Disney, 2014). Different types of sensors have been created as input to the system for pose recovery such as visible wavelength camera, long-wave thermal infrared camera, time-of-flight camera and laser range scanner camera. Microsoft has released a Kinect sensor which consists of a depth

sensor which is made up of an infrared laser projector and a monochrome CMOS sensor which can capture video data in 3D under variable lighting conditions. It also include an RGB camera and four microphones (John MacCormick). The Microsoft Kinect Sensor can be used to create depth-map which will be discussed in following sections.

1.4 Technical Background

Body pose estimation is a very important process in human aggressive behavioral analysis system. A surveillance system with human aggressive behavioral analysis can help to detect aggressive behavior (Luo, 2013). Basically, a body pose estimation system can be divided into three parts which are (i) RGB image to depth image conversion, (ii) optical motion capture and (iii) body pose estimation; which are shown in Figure 1.1.

Figure 1.1

Flow of Aggressive Behavior Analysis System

1.4.1 Depth Map

Depth map is used to provide a two and a half dimension (2D-plus-Depth) of the scene [1]. It contains information of the depth of the objects from the viewpoint (the distance of the Z axis of the view point). The limitations of depth map can be categorized as follows:[1]

- Occlusion. When one or more objects are occluded with each other, single channel depth maps will not be able to distinguish distance between them.

- Objects' characteristic. Single channel depth maps cannot display information of the surfaces that can reflect or refract such as transparent object and mirrors. This is due to the fact that single channel depth maps can only record the first surface detected.

- Depth map generation method. Depth maps represent the perpendicular distance between the object and the floor plane of the scene. Unfortunately, there will be error when the distance between a flat surface and the floor plane is very small. This method may recognize the flat surface as a part of floor plane, and the distance of the floor plane can be smaller than the actual distance.

1.4.1.1 Depth Map Generation Techniques

There are several techniques that can create a depth map from a camera scene such as monocular technique, binocular technique and structured light method. We will briefly explain these methods as follows:

1. Monocular technique. This technique uses monocular cues to create the depth maps. Monocular cues provide depth information such as motion parallax, depth from motion, kinetic depth effect, perspective, relative and size, when viewing with one eye. Mircosoft Kinect Sensor's Depth from Focus (DFF) method (John MacCormick) and Earl Wong's method (Wong, 2006) uses the monocular technique to create a depth map to get the depth information of the subject.

2. Binocular technique. Using the binocular cues is the same as human viewing a scene with both eyes to create a depth map. Usually this method requires two cameras to act as human eyes. Binocular cues provide depth information such as stereopsis, convergence and shadow stereopsis (Burton, 1945).

3. Structured light method. This method projects a known pattern of infrared light which usually appears as a pattern of grids or horizontal bars. Using vision system depth and surface information of the subject, the depth of the subject is calculated when structured light falls onto it. Microsoft Kinect sensor uses this method to a create depth map (John MacCormick). Therefore, this research proposes to use structured light method to create depth maps as the

Kinect sensor can be used as an input for video surveillance. The reason for choosing this method is that it can provide accurate depth information with no textures of objects. The projected textures can be mixed with the object textures. If there are a lot of object textures, it will be difficulties in finding depth information. This method will be further explained in detail in section 2.3.

1.4.2 Human Tracking

Human tracking can be accomplished in two parts, that is, human detection and human body part tracking. Human tracking is used to extract human body from a video filled with other objects such as animals, buildings, cars and etc. Low-level algorithms such as background updating and foreground segmentation is usually required for human detection. The main challenges faced in human tracking are subjects' attire, occlusion between objects with subject or another subject and various types of posture (Mohamed, 2009). The detail of human tracking will be discussed in the following section.

1.4.2.1 Human Detection

There are numerous techniques available in literature that can be used to detect humans from the scene. These can be divided into two groups; background subtraction and direct detection without pre-processing (Ogale, 2006).

Background subtraction technique gathers the foreground data from the video or image. After analysis, the feature from the data such as motion, shape, and color are used to differentiate the objects in it and classify them into categories like human, animal, vehicle and etc. Several approaches for background subtraction that uses human's feature are shown in Table 1.1.

Table 1.1 Methods Based on Background Subtraction (Ogale, 2006)

Background Subtraction Approach	Human Features
Color/ Ref. Image	Color, Contour, Region Model

Motion/ Frame Difference	Wavelets, Fourier shape, Shape
Motion/ Color	Geometric Pixel Value
Depth	Motion, Shape
Infrared	IR + Color

Direct detection technique can extract the features from the image or video scene to classify them as non-human or human. Using the features such as contours, skin color, motion, or combinations of all features, can be classified into human or non-human categories. Several classifiers and human models are shown in Table 1.2.

Table 1.2 Method Using Direct Detection (Ogale, 2006)

Human Model	**Classifier**
Periodic Motion	Motion Similarity
Geometric Pixel Value	Distance
Shape Template	Chamfer Distance
Shape + Motion	Adaboost Cascade
Optical Flow	SVM (RBF)
History of Gradients	SVM (Linear)

1.4.2.2 Human Body Part Tracking

This process is a pre-processing step for matching the human for a period of time. There are numerous challenges which can affect the accuracy of human body part tracking such as various styles of clothing, partial or full occlusion with object or other subjects and different types of posture. The motion can be categorized into local, that is, feature points or body part motion or global, that is, whole body motion signature. In order to analyze subjects' gesture, human motion features are extracted. When the motion of the body is detected, motion analysis step is used to identify the type of motion. The four main categories of human detectors are (1) Holistic Detector, (2) Part-based Detector, (3) Patch-based Detector and (4) Detector using Multiple Cameras [8].

1. Holistic Detection: These detectors are trained to look for human in the whole video frame. When the image features inside the local search window meet a certain threshold, the detector will start tracking the human. Some methods use global features such as edge template [19], others uses Histogram of oriented gradients [20].

2. Part-based Detector: In this method, a human being is treated as a collection of parts. Part-based detectors are generated by gathering local features such as edgeless features and orientation features. The human hypotheses is then formed by combining all these hypotheses parts. Unfortunately, this approach is very elaborate and difficult to implement (Pedestrian, 2015). This is because a standard procedure for image processing must be followed to perform human body part detection using this approach. The standard procedure is a densely sampled image pyramid that has to be created, then features at each scale can be computed. Classification is carried out for all possible locations, and finally non-maximal suppression will be performed to generate a set of bounding boxes.

3. Patch-based Detector: Leibe [21] proposed a method known as Implicit Shape Model (ISM), which combines both detection and segmentation. During the training process, a codebook of local appearance is learnt. Local features are then extracted during detection process for matching against the codebook entries, and each match casts one vote for the human hypotheses. At the end, the motion of the human will be tracked by further refining these hypotheses. The advantage of this method is that it requires a small number of training images compared to part-base detector.

4. Detection Using Multiple Cameras: Fleuret [22] has developed an approach that integrates multiple calibrated cameras for detecting multiple humans. This method creates a Probability Occupancy Map (POM) which provides an estimation of the probability of each grid cell that can be occupied by a human. Multiple synchronized cameras are used for recording video from different angles at eye level, it can effectively combine a generative model with dynamic programming to follow up to six individuals across thousands of frames despite of occlusions and lighting changes.

1.4.3 Body Pose Estimation

As mentioned in section 1.3 and 1.4 body pose estimation is a very important step for the proposed system. Unfortunately, it still remains a challenge due to occlusion of body parts, various image backgrounds and high dimensional parameter space (Pavlovic et. al., 1997). Various body pose estimation methods have been implemented, such as pictorial structure method which was proposed by Pedro F. Felzenszwalb et. al. (Li, 2011). Pictorial structure method can estimate the global optimal pose with fast pose assumption, however this method cannot be used in real-time applications because of its heavy computational time requirement .

Another well-known method is the Hierarchical method (Pavlovic et. al., 1997; Felzenszwalb and Huttenlocher, 2005;Panagiotakis and Tziritas, 2004). This method is used by Costas et. al. (Felzenszwalb and Huttenlocher, 2005) to track the human body by detecting head, centroid body, legs and arms sequentially. R. Navaratname et al. also used the Hierarchical method to detect upper body pose hierarchically using 2D templates (Panagiotakis and Tziritas, 2004). The hierarchical method can be inaccurate because it focuses on detecting body parts and the hierarchy of searching scheme.

1.5 Behavior Understanding

Behavior understanding is a sub-field of high-level machine vision that is concerned with detecting behavior and detect event based on behaviorious. Behavior of a human can be primitive or complex. This is because the same behavior can be expressed differently by different people. Gesture recognition is a part of behavior understanding that focuses on recognizing human body motion and differentiate whether the gesture is the behavior of interest (Mohamed, 2009).

1.6 Relationship between Behavior and Action

In this research, a human behavioral analytics system using the Microsoft Kinect sensor as the video surveillance system has been proposed. In sociology, behavior basically includs all basic human actions, actions with no meaning, and attitude not directed in front of other people (Minton and Khale, 2014). Behavior is a big field which includs difference of behavior such as caring, kind, polite,

aggressive, irritating and etc. In this research, the aggressive behavior and detecting fall events are the main focus of the proposed system.

As Elizabeth A. Minton et al. (Minton and Khale, 2014) mentioned; behavior is an array of every action and observable emotion made by individuals, organisms or system. Therefore, in this research observable actions include sitting, crouching, walking, running, jumping, kicking, punching and, fall event. The system analyzes all these actions to determine whether the subject is performing aggressive behavior or normal behavior.

1.7 Objectives of Research

As mentioned in the introduction, video surveillance system has been installed in many public areas for monitoring safety and security. However, traditional surveillance technology can be adapted to detect aggressive behavior. Therefore, this research proposed to design and develop a human aggressive behavior analysis system for video surveillance. In this research, the Microsoft Kinect sensor is used to detect and analyse human activity. The advantage of the Kinect sensor is the ability it possesses in detecting human event in the absence of illumination.

It is envisaged that the proposed method will use human gesture recognition and human aggressive behavior analysis based on the Cartesian coordinate point-plane distance equation which will be discussed in detail in Chapter 4.3. It is also envisaged that the proposed method can improve the detection rate for various activities for human behavioral analysis. The added advantage of the proposed system is that is able to detect a behavior in real time which is presented in Chapter 4.

1.8 Dissertation Outline

The rest of the dissertation is organized as follows:

Chapter 2 discusses the human gesture recognition which is an important process for human behavioral analytics system. The proposed human behavioral analytics system can be divided into

behavior analysis system and fall detection system. The literature reviews of techniques in human gesture recognition and fall detection are discussed.

Chapter 3 provides an overview of the developed system including the equipment used and the system setup for this research. In this research, thirty subjects are requested to take part in the evaluation of the proposed system. All the subjects are required to perform five non-aggressive behaviors, two aggressive behaviors and a fall event to evaluate the proposed system. The video of the experiments have recorded.

Chapter 4 discusses the details of the behavior analysis system. The steps that involve in the human behavior analysis system are discussed. The mathematical model of depth-map generation is also discussed in this chapter. In addition, the proposed classifier is also presented.

Chapter 5 presentes the results of human behavior analysis system and fall detection system using Microsoft Kinect sensor. The goal of the evaluation is to show that the system works with real time streaming video. This chapter also demonstrats that the proposed algorithms perform better than the DTW algorithm. The behavioral analysis system for video surveillance has been evaluated in terms of accuracy and sensitivity. This chapter concluded that the Microsoft Kinect sensor is able to resolve the illumination problem.

Chapter 6 provides the conclusion of the dissertation associated with the summary of the result evaluated in this research. In addition, some ideas to further improve the developed system are also discussed.

CHAPTER 2

BEHAVIORAL ANALYTICS IN VIDEO SURVEILLANCE

2.1 Introduction

The main objective of this research is to develop a human behavioral analysis system using a video surveillance. This chapter will discuss human gesture analysis using image processing which is the main process for human behavioral analysis system. A fall detection system is also embedded into the proposed system.

Human gesture recognition is used to identify and interpret human gestures automatically using a set of sensors. The process for human gesture analysis from video scene can be separated into five parts that is, image processing, human body part detection and tracking, behavior understanding, and posture detection (Mohamed, 2009). In order to develop an automatic human gesture recognition system which can work in real-time, many challenges must be resolved. One of the main problems of the gesture recognition system is to deal with a large number of gestures. Gestures recognition involves handling a lot of degrees of freedom (DoF), and variability of 2D appearance depending on the camera view point for the same gesture and resolutions for the temporal dimension (Mohamed, 2009).Therefore, two approaches had been used to resolve the problem, that is, non-vision-based approaches and vision-based approaches. Non-vision-based approach uses instrumented gloves, tracking devices, and vision-based approaches uses one or more cameras. The instrumented gloves are gloves that are embeded with sensors. The rest of the chapter will discuss the human gesture analysis system and fall detection using image processing techniques.

2.2 Image Processing

In imaging science, image processing is a method for processing an image, such as video frame and picture. The output of image processing is an image, where the output of image analysis is a set of parameters or characteristics related to the image. Most of the image processing method will first convert the input image into 2D signal and after that the image is analysed. Algorithms in image processing are basically divided into three levels: (1) low-level vision algorithm that analyse pixels without depth, (2) middle-level vision algorithms, that is, pattern matching, tracking and detect objects, and (3) high-level vision algorithms, that is, interpretation and semantics extraction from images (Mohamed, 2009).

2.3 Structure Light Method

Conversion of live streaming video frame into depth-maps is a very important step for behavioral analysis. As mentioned in section 1.4.1, depth maps will provide the system a 2D image with distance data. It will enable the system to differentiate human and moving objects from the video frame. Section 1.4.1.1, briefly explains the three main techniques for creating a depth-maps. The proposed algorithm, uses structure light method to generate depth-maps as described in section 1.4.1.1. Structure light method is an established depth map conversion method. With this technique, a minimum of one camera is needed to capture the pattern of reflected light which is projected from a projector. The projector will transmit some pattern of light, such as single-point pattern, single-line pattern, and coded pattern (Tong et. al., 2014). After that the triangulation equation is used to calculate the depth information. The triangulation equation will be discussed in section 2.3.1.

Tong Jia et al. (2014) cited that the projector calibration and visible light interference are two main issues in structured light method. Huafen Luo et al. (2013) mentioned that the camera cannot dynamically capture the reflected image due to it calibration difficulty. Therefore, to solve the problem, the authors (Tong et. al., 2014) proposed using an infrared structured light as the light source. The infrared light has lower wavelength compared to visible light, so the reflected infrared light will not be interfering with other reflected light which can cause confusion. As the result, the proposed method used the same light source to create the depth-map.

The proposed method in this research uses a Microsoft Kinect Sensor as the input device for the video surveillance system. As mention in section 1.1, Microsoft Kinect sensor is a low cost range sensor which consists of an RGB camera, an infrared laser emitter and an infrared camera. It combines structured light method with two classic computer vision techniques, that is, depth from focus and depth from stereo to create depth maps.

- Depth From Focus (DFF): Kinect dramatically improves the accuracy of depth from focus method compared to traditional DFF method. It uses the principle that an object gets blurry when it is further away, and a special lens is used in this technique which will be further explained below.

- Depth From Stereo (DFS): Kinect uses the parallax method for DFS. Kinect analyzes the shift of the speckle pattern by projecting from one location and observing from another location.

A special diffraction grating with different focal length in x-axis and y-axis is used to split the single beam which is emitted by the infrared laser source into multiple beams. As a result, a constant pattern of speckles will be created and projected onto the 3D space. The projected laser light pattern will change from circle to an ellipse whose orientation with the depth of the subjects struck as shown in Figure 2.2.

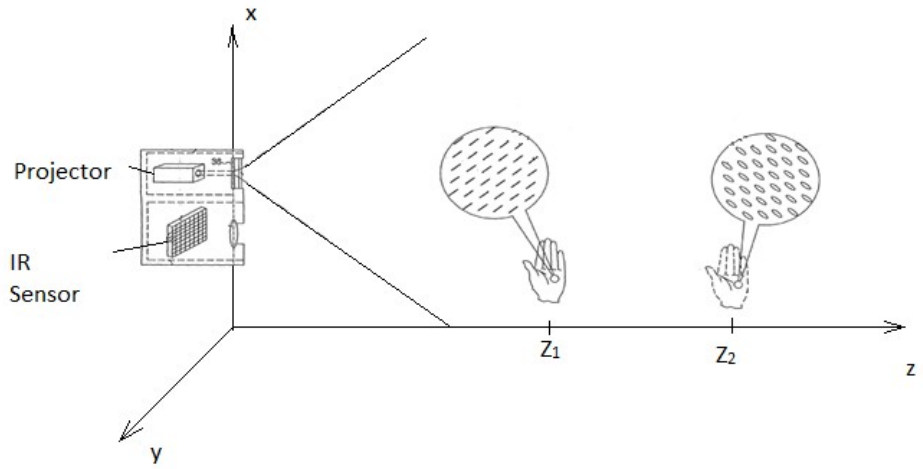


Figure 2.2 Pattern of Speckle Light Projected onto the Scene

A reference pattern image is obtained by capturing at a known distance is stored in the memory of the sensor. The subjects pattern captured by the infrared camera needs the reference pattern to calculate the subjects depth (Pang et. al., 2014). When an object move into the speckle pattern of the infrared laser light, the position of the infrared image is shifted in the direction of the baseline of the laser projector and the perspective center of the infrared image. A disparity image is created by calculating the shift of speckle which is projected on the object using the triangulation equation shown in equation 4.1. Figure 2.3 illustrates the speckle pattern of infrared laser light projected by the Kinect.

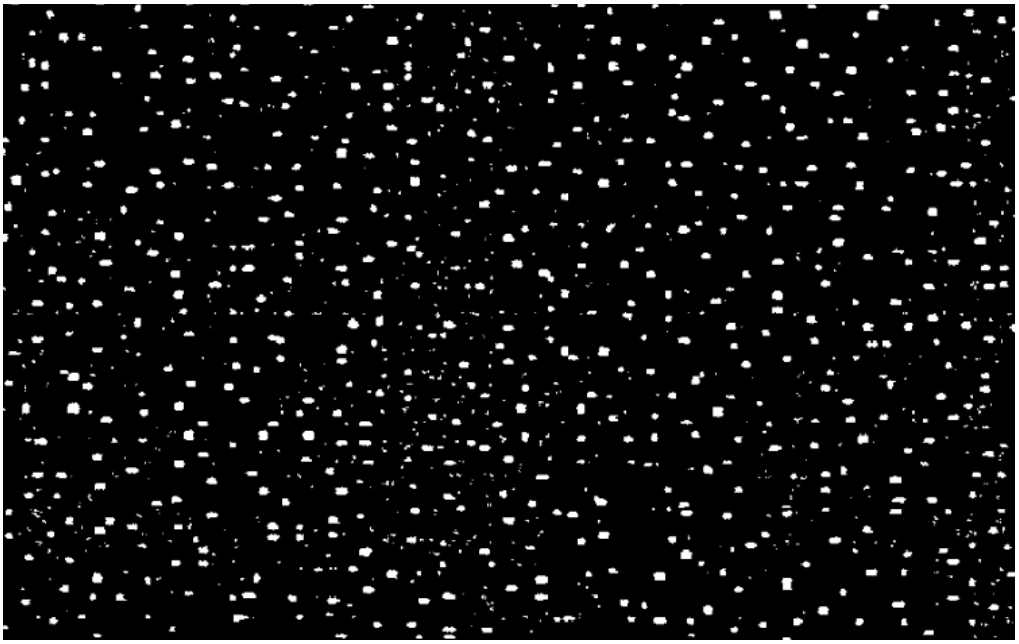


Figure 2.3 Speckle Pattern of Infrared Laser Light

2.3.1 Mathematical Model for Depth Maps

The relationship between the distance of an object at point 'k' from the laser projector and the IR camera and the measured difference 'd' is shown in Figure 2.4. Where C is the IR camera, L is the laser projector, b is the distance between IR camera and laser projector. To express the points of the object in 3D coordinates, a depth coordinate system with its origin at the perspective center of the infrared camera is created. The X-axis of the depth coordinate system is in parallel with reference plane. The Z-axis of the depth coordinate system is pointing towards the reference plane. Y-axis of the depth coordinate system is orthogonal to X-Y plane.

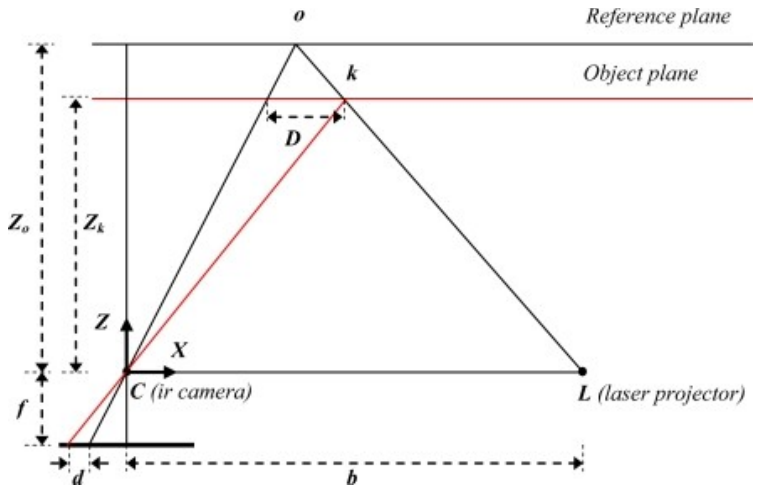


Figure 2.4 Relation between Depth and Measure Displacement

Assume there is an object **k** with distance $\boldsymbol{Z_k}$ is in between the reference plane and the sensor. The reference plane has a distance $\boldsymbol{Z_0}$ from the sensor. The image plane of the infrared camera captured the speckle strike on the object. From Figure 2.4, it is observed that there will be a displacement ***d*** in ***X*** direction when the object get closer or further to the Kinect sensor. Using similarity of triangles;

$$\frac{D}{b}=\frac{Z_0-Z_k}{Z_0} \tag{2.1}$$

and

$$\frac{d}{f}=\frac{D}{Z_k} \tag{2.2}$$

where ***D*** is the displacement of the IR camera view point from the reference plane to point ***k***. ***b*** is the distance between IR camera and IR laser light source, and ***f*** is the focal length of the infrared camera. To calculate the object distance Z_k, ***D*** is substitute from equation 2.2 into equation 2.1:

$$Z_k = \frac{Z_0 bf}{bf + d Z_0} \tag{2.3}$$

The parameters Z_0, ***f***, and ***b*** is a constant which can be determined by calibration. Equation 2.3 is a basic mathematical equation for computing the derivation of depth from the observed disparity. The **X** and **Y** coordinates of the each point can be calculated from its image coordinates and scale using the equation 2.4 and 2.5:

$$X_k = \frac{-Z_k}{f}(x_k - x_0 + \delta x) \tag{2.4}$$

$$Y_k = \frac{-Z_k}{f}(y_k - y_0 + \delta y) \tag{2.5}$$

assuming that the object image coordinate system is parallel with the sensor. The depth coordinate system, x_k and y_k are the image coordinates of the object points, x_0 and y_0 are the coordinates of the view point, and δx and δy are the lens distortion correction.

2.3.2 Calibration for the Mathematical Model

As mention in section 2.3.1, there are total five calibration parameters used by the mathematical model to calculate the 3D coordinates from the raw image.

- **b** is length between IR camera and IR laser light source
- $\mathbf{Z_0}$ is distance from the sensor
- $\boldsymbol{f}$ is focal length of IR camera
- $\boldsymbol{x_0}$, $\boldsymbol{y_0}$ are the principle point offsets
- $\boldsymbol{\delta x}$, $\boldsymbol{\delta y}$ are the lens distortion correction.

The last three calibration parameters stated above are determined by the standard calibration of the IR camera. In practice, the size of the disparity images generated by the internal processor of Kinect is smaller than the image size from the infrared sensor. Therefore the calibration parameters of the IR sensor does not directly correspond to the disparity images. In addition, due to the limited bandwidth of the USB connection, the images of the IR sensor are resized and cropped to size corresponding to the disparity images.

The calibration parameter of the IR camera refer to the reduced IR image instead of the actual IR image. Therefore, the variable $\boldsymbol{d}$ in Equation 2.3 is replaced with $\boldsymbol{md'+n}$ whereby $\boldsymbol{d'}$ is the normalized disparity where **m** and **n** are the parameters of the linear normalization as shown in Equation 2.6 to increase the accuracy of the system.

$$Z_k^{-1} = \left(\frac{m}{fb}\right) d' + \left(Z_0^{-1} + \frac{n}{fb}\right) \tag{2.6}$$

Equation 2.6 is the linear expression between the inverse depth of the object and its corresponding normalized displacement. The coefficients of the equation can be estimated if the distance of the object to the sensor is known. Unfortunately, **b** and $\mathbf{Z_0}$ cannot be determinee separately due to the inclusion of the normalize parameters. The calibration parameters representing the relation between the object coordinates (**X, Y, Z**) and image parameters(**x, y,** $\boldsymbol{d'}$). Finally, the point cloud shown in Figure 2.3 is created from each disparity image once the calibration parameters are estimated.

2.3.3 Adding Color to Point

The orientation of the RGB camera relative to the depth coordinate system is one of the crucial element in integrating depth and color data. As mentioned in section 2.3.1, the origin of the depth coordinate system is at the perspective center of the IR camera. Therefore, the orientation parameters can be estimated by the stereo calibration of two cameras. The parameters estimated include the three rotation between the camera coordinate system of the RGB camera and IR camera, and the 3D position of the perspective center of the RGB camera within the coordinate system of the IR camera. Furthermore, the focal length, principal point offsets and the lens distortion of the RGB camera are estimated too.

The image size of the RGB camera is reduced to be more convenient to perform stereo calibration instead of using actual image. The parameters represent the relation between the 3D coordinates of each point and its corresponding pixel-coordinates in the reduced RGB image. The color are added to the point cloud by projecting each 3D point onto the RGB image and interpolating the color when the parameters are estimated.

2.4 Estimation Pose

In this section, the estimating of a joint position from the depth map generated from section 4.2 is discussed. There are two steps involved in the process of estimating pose from depth image, that is, finding body part and computing the joint position which will be further explain in following section.

(a) Training data

(b) Real Data

Figure 2.5 Training and Real Data. Depth and ground truth of body parts variety in pose, shape, clothing, and crop (John)

2.4.1 Finding Body Parts

Microsoft Kinect Sensor provides an intermediate body part representation (Shotton et. al., 2011) which can be used to identify several localized body part labels that cover whole body, as color-coded in

Error: Reference source not found. Microsoft Kinect color coded the human body part with different color to differentiate the body part. Some of the body parts are labeled with particular skeletal joints of interest, while other body parts fill the gaps or used in combination to calculate other joints such as hip and spine joints of the subjects.

The body parts are specified in a texture map which are retargeted to combine the various characters while rendering. The Kinect's classifier learned the fully labeled data that generated from both depth and body part images (Shotton et. al., 2011). Kinect assumes that there are total 31 body parts for a human body which is separated into left and right part due to its own limitation. This will allow the classifier to differentiate left and right side of the body.

2.4.1.1 Depth Image Features

To find out the body parts of the subjects, a depth comparison features is used as shown in Equation 2.7. This equation will calculate the difference of depth between two pixels.

$$f_\theta(I,x)=d_I\left(x+\frac{u}{d_I(x)}\right)-d_i\left(x+\frac{v}{d_I(x)}\right) \qquad (2.7)$$

For a given pixel *x* in image *I*, the features are computed. Where $d_I(x)$ is the depth of the pixel, θ = (u,v) is offset to second pixel. To ensure the features are depth invariant, $\frac{1}{d_I(x)}$ is used to normalization the offsets. At a given pixel on the body in the image, a fixed world space offset will show whether the pixel is near or far from the camera. If an offset pixel is out of the bounds of the image or fall on the background, the depth probe $d_I(x')$ will become a large positive value.

Figure 2.6 shows the two features for different pixel of the body. The yellow crosses represent the pixel x being classified. The red circles are the offset pixels as stated in Equation 2.7. For example for f_{θ_1} in Figure 2.6(a) and Figure 2.6(b) one of the offset pixel is on top of the pixel being classified and another have the same location with the pixel being classified. The feature f_{θ_1} in Figure 2.6(a) will provide a large positive value than feature f_{θ_1} in Figure 2.6(b). This is because pixel $\boldsymbol{\theta_1}$ in (a) is higher than pixel $\boldsymbol{\theta_1}$ in (b), and both point fall on the upper body of tracked human. The feature f_{θ_2} can be used to help detect thin vertical structures such as an arm.

(a)

(b)

Figure 2.6 Depth Image Features

2.4.1.2 Randomized Decision Forest (RDF)

If the depth image features in the section above are used individually, it just show which part of body the pixel belong to, but with the RDF, they are sufficient to accurately recognize all trained part. RDF is a group of *T* randomized decision tree as show in Figure 2.7. Each tree consist of split nodes and leaf nodes. Every split node consists feature f_θ and threshold τ. To indentify the pixel **x** in image ***I***, the system will repeat evaluation with Equation 2.7 from the root of the tree, and comparing the result with the threshold τ in the current state to determine which path to follow. When the evaluation reached the leaf node of the tree, a learned distribution $P_t(c|I,x)$ for all body parts which are labeled as *c* is stored. The average of the distributions for all trees in RDF is calculate using Equation 2.8 for final classification (Shotton et. al., 2011).

$$P(c|I,x)=\frac{1}{T}\sum_{t=1}^{T}P_t(c\vee I,x) \qquad (2.8)$$

The Microsoft Kinect trained each tree in RDF with different set of randomly synthesized images. To make sure the pixel is distributed across whole body parts, a random subset of 2000 example pixels from every image is chosen.

Figure 2.7 Randomized Decision Forests. Every tree consists of split node (blue line) and leaf node (green circle). Red arrow show the paths that taken after calculate with Equation 2.8 and compare with threshold τ. (John)

2.4.1.3 Inferring Joints Position

As mentioned above, Microsoft Kinect sensor measures each pixel information to recognize body parts. The information will be shared across pixel to generate reliable inferring positions of 3D skeletal joints. The Kinect Sensor would perform a check to detect potential failure and uses each pixel information to carry out a self-initialization.

The global 3D centers of probability mass for each part is accumulated, using the known calibrated depth to share the information across pixels. Unfortunately, the quality of global estimation could be severely degraded due to outlying pixels. Therefore, to overcome this problem, Microsoft Kinect uses a local mode-finding approach based on mean shift with a weighted Gaussian kernel. Equation 2.9 is the density estimator for each body part (Shotton et. al., 2011).

$$f_c(\hat{x}) \propto \sum_{i=1}^{N} w_{ic} \exp\left(-\left\| \frac{\hat{x} - \hat{x}_i}{b_c} \right\|^2\right) \tag{2.9}$$

$$w_{ic} = P\left(c \middle| I, x_i\right) * d_I(x_i)^2 \tag{2.10}$$

where $\hat{x}$ is a coordinate in 3D world space, pixel weighting $\boldsymbol{\omega}_{ic}$ (Equation 2.10) is considering inferred body part probability at the pixel with the world surface area, N is number of image pixels. $\hat{x}_i$ the reprojection of image pixel $\mathbf{x_i}$ onto world space with depth $\boldsymbol{d_I(x_i)}$, and the learned each part bandwidth $\boldsymbol{b_c}$.

Equation 2.10 ensures that the density estimates are depth invariant, and it can significantly improve the accuracy of joints prediction. The mean shift is used for finding modes in the density efficiently.

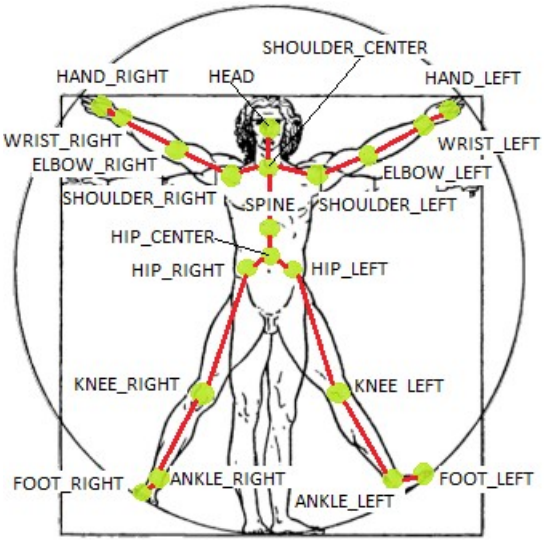


Figure 2.8 Human Body Joints Detected by Kinect

2.5 Human Gesture Analysis

Human gesture analysis is concerned with automatically recognition of human gesture from the video scenes which is an important part of human behavioral analysis (Mohamed, 2009). As mentioned in section 2.1, there are two known approaches for analyse human gestures: (1) non-vision-based technology and (2) vision based technology. Both technologies have advantages and disadvantages. For example, vision based technology does not require user cooperation but it will be more difficult to configure and may faces problem with occlusion. Non-vision-based technology requires user cooperation, unfortunately it will be uncomfortable to the user who has to wear a sensor device for a long period of time. Non-vision-based technology is more precise compared to vision based technology (Mohamed, 2009), because when there is occlusion then vision based technology may fail analyse that body part. Table 2.3 shows the comparison between two technologies (Mohamed, 2009).

Table 2.3 Comparison between Non-Vision-based and Vision-based Technology

	Vision-based	Non-vision-based
User cooperation	Yes	No
User Intrusive	Yes	No
Flexible to configure	Yes	No
Flexible to use	No	Yes
Occlusion Problem	Yes	No
Precise	Yes	Yes/No

2.5.1 Classifier for Human Gesture Recognition

Basically, the problem of gesture recognition can be divided into two sub-problems: (1) the decision problem and (2) gesture representation problem which is cited in Mohamed's thesis (Mohamed, 2009). Several classifiers have been implemented for decision making in gesture recognition. For static gesture (postures), a general classifier (linear/ non-linear classifiers) or a template matcher will be able to recognize it. However, dynamic gesture requires special classifiers for decision making such as Hidden Markov Models (HMM), Support Vector Machines (SVM), Dynamic Time Warping (DTW) and Finite State Machines (FSM).

1) Hidden Markov Models (HMM): Markov Model is a stochastic mathematical model, which generate random sequences of outcome according to certain probabilities [Mahmoud, 2009]. The HMM is a statistical Markov model using Markov process with hidden states. The output of the model is visible to the observer but the states in the model are hidden from the observer. Each state has probability distribution over the possible output. Figure 2.9 shows an example of HMM. HMM is widely used in gesture recognition and understanding systems (Technology and application, 2012; Noury et. al., 2003). The advantage of this model is that, it provides better compression than a Markov Model, which enables more sequences to be identified. Unfortunately, HMM needs to be trained with a set of seed sequences and basically it requires larger number of samples than Markov Model which can slow down the process (Shotton et. al, 2011).

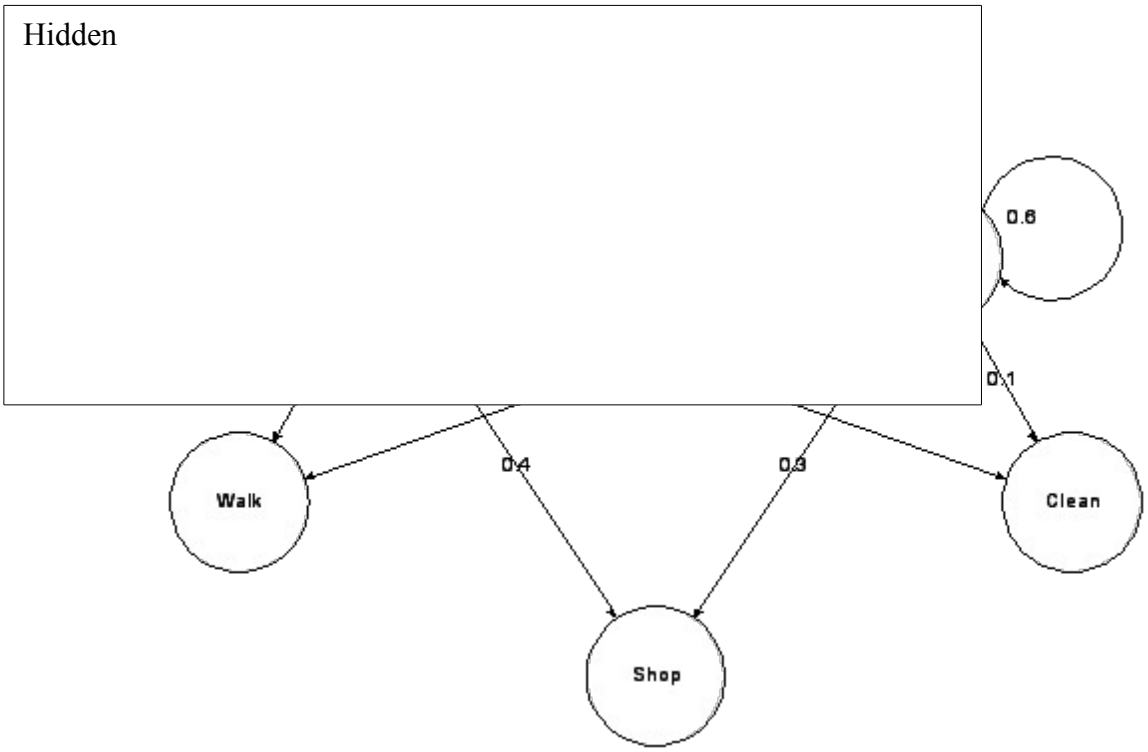


Figure 2.9 Example of HMM (Pattern Recognition, 2011).

2) Support Vector Machines (SVM): SVMs are supervised learning models with associated learning algorithms for optimal modeling of the data (Lin, 2014). It is used for regression analysis and classification. It also separates the data class into a clear gap that is as wide as possible by learning the decision function. New data is mapped into the same place. Predication is carried out to establish where it belongs based on the side of the gap they fall on. The main feature of SVM is that, it can be integrated into the kernel. SVM has the flexibility in choosing the level of the threshold separating the data. The disadvantage of SVM is that, it is time consuming and requires larger data size for both training and testing (Rivera, 1996).

3) Dynamic Time Warping (DTW): A well-known technique for computing the similarity between two sequences which may vary in time or speed. Basically DTW is used to compute the optimal match between two time-depended sequences (Nirjon et. al., 2014; Ratanamahatana et. al, 2004). It allows the system to match and recognize an action with varying speed.

4) Finite State Machine (FSM): A mathematical model for designing both computer programs and sequential logic circuit. The model will only stay in one state known as the current state. Basically, a FSM is defined by a list of its states and the condition of each transition. When the machine is triggered by an event or condition, it will change from current state to another state which is known as the transition. The FSM is easy to use because the state and the flow of the model can be represented graphically. Unfortunately, the graphical representation can be confusing when the FSM consist of many states and transition paths.

2.5.2 Gesture Representation Algorithms

Numerous gesture representation methods have been implemented to detect and model human body parts motion. These methods can be divided into two main algorithms (Rehm et. al., 2008): (1) 3D model-based algorithms and (2) appearance based algorithms. In addition, all implemented methods can be sub-divided into two types (Mohamed, 2009), that is, the spatial and the temporal aspects. Both are modeled separately which is known as posture-automation and motion models. Figure 2.10 shows the different representation of gestures.

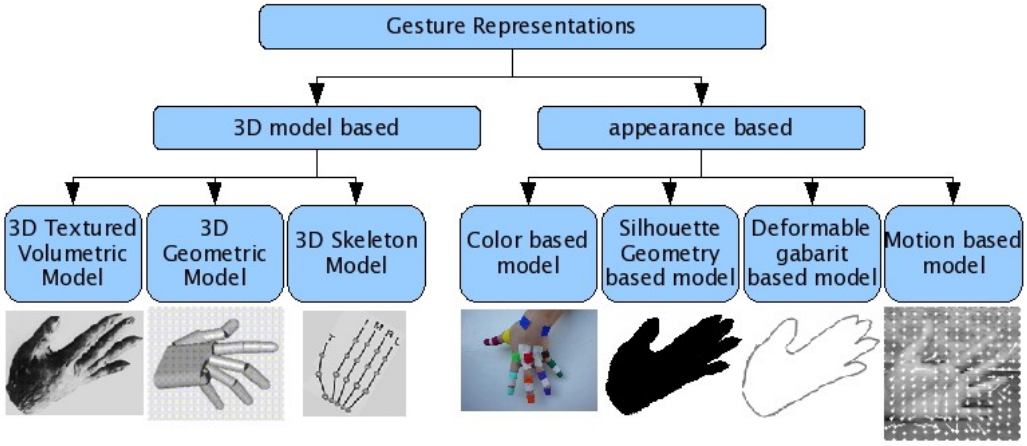


Figure 2.10 Different Representations of Gesture (Mohamed, 2009)

2.5.2.1 3D Model based Algorithms

The 3D model-based algorithms can acquire 3D spatial description, that is, palm position or joint angles, from the human body. One or more cameras are involved when the gesture recognition process is running. The algorithm will compute the parameters of the model that matches spatially to the real target. The target motion can be tracked and the model parameters are updated accordingly. It will continuously check to determine whether it matches the transition of the temporal model. It is cited that 3D model-based algorithms have high recognition rate for human gesture recognition (Boulay, 2007; Ye et. al., 2004). Several 3D models have been implemented, for example volumetric model and 3D skeleton model are two main models (Mohamed, 2009).

- 3D skeleton model: This model is commonly used because of its simplicity and high adaptability. Instead of dealing with many parameters and using intensive processing of the 3D models, the model deals with the degree of freedom and information regarding articulation of body joints.

- Volumetric model: This method is usually used in computer animation and computer vision. It contains many details of the human body, that is, skeleton and skin surface information. The model will synthesize the 3D model of human body in question and checking its parameters to see if the model and the human body are the same in the visual image (Ashraf et. al., 2014). The disadvantage is that this model is too complex to be used in real time.

2.5.2.2 Appearance based Algorithms

The appearance based algorithms do not need the spatial description of the human body parts, it extracts the data from an image or video using a template database. These models can be divided into two main categories, that is, deformable 2D template models and image sequence models.

The deformable templates are used to estimate the object's contour by using a sets of points of the contour of the objects as interpolation nodes. These models are generally used for hand-tracking, but it can also be used as a classifier of a simple gesture (Ashraf et. al., 2014). There are two other

methods which use similar algorithm. These are color-based models and geometry based models (Mohamed, 2009). The color-based methods mark the human body park with different color to track the motion. For an example, Bretzner et. al. (2002) used hierarchical models, particle filtering and multi-scale color features to create a hand gesture recognizer. Binary silhouette based models use the geometrical properties such as perimeter, convexity, surface, and bounding box to recognize gesture. Ahmet Birdal et al. used this method to recognize hand gesture by analyzing the geometric properties of the bounding box of the hand (Khoshelham and Elberink, 2011). Vladimir I. et.al. (1997) made a comparison between 3D based models and deformable based models. On the other hand, 3D based models extract important parameters such as joints angles or body part position by analyzing 3D features of the body parts.

In image sequences approach, a sequences of representative image n-tuples will model every gesture which is required. In mathematics, an n-tuple is a series of n element where n is a positive integer. The representative image n-tuples consist numerous elements which are the view of the same body part from different directions. This method usually use monoscopic or stereoscopic technique, which was discussed in section 1.4.1.1, to acquire input image for camera. Features derived from images or the whole image are used as the parameters (Vladimir I. et.al., 1997). One of the well known algorithm in image sequences approach is motion history images (MHIs). MHIs accumulate the motion of every single pixel in the image within a temporal window and form 2D images with the data. The intensity of the pixel in MHIs algorithms are affected by the period of the motion observed at the same pixel position.

2.5.3 Review of Human Gesture Analysis

There are numerous techniques implemented for human gesture analysis using image processing (Mohamed, 2009). These are HMM based methods, SVM based methods, DTW based methods, FSM based methods, and accelerometer based methods (Mohamed, 2009; Elmezain et. al., 2009; Nirjon et. al., 2014; Hong et. al., 2000; Rehm et. al., 2008). Mahmoud Elmezain (2009) proposed a method which combine both SVM and HMM based method in the same algorithm (Elmezain et. al., 2009). In this approach, SVM classifier is used to identify the hand shape of the segmented hand. After that the HMM classifier is used to identify the hand gesture from stereo color image sequences. The gesture identifying algorithm gets the start and end points of the gesture from the video scenes using the difference observation probability value and maximal for both gesture models and non-gesture model. The end result is a system that can execute both hand gesture identification and recognition simultaneously.

In a different research, Shahriar Nirjon (Nirjon et. al., 2014) implemented four supervised classifiers to identity four aggressive actions, that is hitting kicking, pushing, and throwing using Microsoft Kinect sensor. Each classifier will identity an action from the skeleton frame generated by the kinect. All classifiers are developed using DTW as the kernel for SVM. The classifier is trained by the feature vectors which include the relative position and orientation of the body, and moving speed.

2.6 Posture Detection

Posture detection is a process which recognizes the posture of the tracked person by analyzing the data of a studied frame. Posture detection is a sub-field of gesture recognition since a posture is a "static" gesture (Mohamed, 2009). Posture recognition process involves people detection and gesture recognition. Marcos Zuniga (Zuniga, 2009) cited that when performing people detection, the posture is of interest. On the other hand, some gesture recognition algorithms need to execute posture recognition to analyse the gesture. For example, Bernard Boulay (Birdal and Hassanpour, 2008) proposed an approach to recognize human posture that consists of posture detection and posture temporal filtering. The posture is executed before posture temporal filtering.

2.7 Fall Detection

Automatically detecting falls with video surveillance system has become a major area for computing vision system in recent years (Jansen and DEklerck, 2006). The growing population of the elderly in many parts of the world has triggered the development of surveillance systems to ensure safety of the elderly. Many fall detection systems are able to detect human fall from the scenes automatically in real time situation. Numerous fall detection systems have been implemented which can be divided into two main categories; wearable device based and vision based. Figure 2.11 shows the block diagram for wearable device based system and Figure 2.12 shows the block diagram for vision based system.

Figure 2.11 Block Diagram for Wearable Device based Fall Detection System

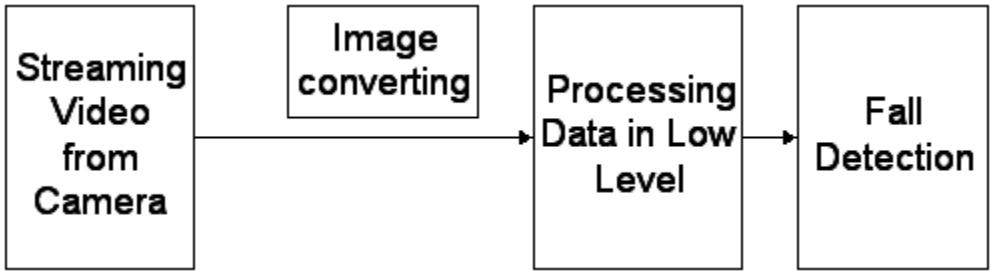


Figure 2.12 Block Diagram for Vision Based Fall Detection System

2.7.1 Wearable Device Based Approaches

Wearable device based approaches use embedded sensors to detect the motion and location of the human body joints. These approaches can be further categorized into motion based and posture based methods. Accelerometers are usually used in these approaches. An accelerometer is a device that can be used to measure the acceleration of different parts of a body.

Mathie et al. (2004) proposed a fall detection that system used an integrated approach of waist mounted accelerometer. This method is able to detect a fall event when there is a sudden increased in negative acceleration. In another work, Toshiyo Tamura et al. (2009) used a wearable airbag which can be triggered when the acceleration and the angular velocity of the subject exceeded a threshold level. This fall detection system can help to reduce the number of injuries of the subject from fall. In 2005, Jay Chen et al. (2006) created a low power, wireless sensor network using a small, non-invasive sensor node. The device reduces the burden of the network by performing the acceleration sampling sequentially. The angle of change is calculated by the dot product of acceleration vectors obtained from the orientation data during a fall, and the acceleration vectors are calculated from average of one-second windows. Chia-Chi Wang et al. (2008) proposed a system using an accelerometer which is placed on the head. The proposed system checks with reference velocities, where the reference velocities is calculated by differentiating the acceleration from the accelerometer. When the reference velocity exceeds the predefined threshold, a fall is detected. However, the limitation of this system is a fall event can only be detected if the subjects are wearing the accelerometer on the head.

As mentioned, the accelerometer provide the acceleration details of movements of the body parts, therefore, it is possible to ascertain the subjects' physical activity and inactivity. The information of the behavior is used to understand the relationship between movement frequency, duration and intensity. Sixsmith et al. (2004) used an array of low cost infrared detectors to design a wearable system known as Smart Inactivity Monitor using Array-Based Detectors (SIMBAD). The subject motion is monitored to detect the dynamic characteristics of a fall event. The inactivity period of the subjects were analysed, and the period of inactivity is calculated within a certain view window and checked whether the period is acceptable for different locations. Both S. Srinivasan et al. (2007) and Youngbum Lee et al. (2007) used motion sensors embedded with wireless accelerometer sensor modules to monitor the subject active and inactive motion. This approach managed to achieve 93% detection rate in detecting fall event.

Chin-Feng Lai et al. (2010) developed a system which combined several tri-axial accelerometers for sensing the joints of injured body parts. The tri-axial accelerometer is a device that can detect the acceleration in three dimensions simultaneously. The proposed system detects a fall when the acceleration transmitted from the sensor significantly exceeds the threshold acceleration. The system is also able to determine the level of injury by comparing the impact acceleration and normal acceleration. Maarit Kangas et al.[48] also used a similar approach. With the Kangas approach the tri-axial accelerometer, is located at the waist, wrist, and the head of the subject. The acceleration of these three parts are tracked and checked whether they have reach the threshold level. The threshold acceleration of all these parts are different and they are used to optimize the fall detection. The results

showed that by analyzing only the acceleration of waist and head, the system is able to distinguish between fall event and activities of daily living.

It has been cited that, by using multichannel accelerometer the system is able differentiate between posture and basic motion patterns (Mubashir et. al., 2012). Therefore, it is possible to detect a fall by analyzing the posture. Bostjan Kaluza et al. (2009) presented a posture-based fall detection algorithm using some small and low-cost wireless tags attached to the ankles, hips, wrists, elbows, and shoulder. The authors used the reconstruction of a subject's posture to detect fall together with abnormal behaviors. The posture of the subject is then reconstructed in a 3D plane by tracking the wireless tag with the motion capture system. The wireless tags enable the locations of these body parts to be determined. This system applied the acceleration thresholds along with velocity data to detect the fall, however this system is too elaborate and complex.

2.7.2 Vision Based Approaches

Many fall detection systems are created using vision based approaches. One of the reasons why the vision based approach is so popular is that it does not require the subjects to wear specific sensor to detect fall. The vision based approaches can be divided into four main categories that is, spatiotemporal methods, posture method, inactivity/change of shape and 3D head position analysis. These four methods will be discussed in the following sections.

2.7.2.1 Spatiotemporal

Spatiotemporal features are used in shape modeling to provide key information of tracked subject activities to detect various events. An efficient and precise shape modeling algorithm is usually an important part for image analysis. Homa Foroughi et al. (2008) derived an algorithm which can detect a fall by combining integrated time motion images (ITMI) with Eigen space approach. ITMI is a spatiotemporal database which consists of the time of the motion occurrence and information of the motion such as the velocity. Eigen space technique is then applied to ITMI to collect the Eigen-motion. MLP Neural Network classifier is also used to differentiate normal activities and fall event. In 2009, Guangyi Shi et al. (2009) proposed a mobile human airbag system to protect subjects from injuries when a fall occurs. The system consists of gyroscopes, 3D MEMs accelerometers, a Bluetooth module and a Micro Controller Unit. A high-speed camera is used to record human motion and analysis

whether any fall event has occured. This system used a gyro threshold for detecting lateral fall. The author uses a SVM classifier to differentiate normal behavior and fall event. A digital signal processing system is included in this method for real time application. The results from this work shows that the SVM classifier is able to work in real time condition.

2.7.2.2 Inactivity/Change of Shape

With the inactivity/change of shape approach, fall events are detected based on the analysis of change of subjects shape as well as subjects' inactivity. Homa Foroughi et al. (2008) applied an best-fit approximated ellipse around the body to detect the change of body shape. The projection histograms of the segmented silhouette is then analysed, and the position of subjects' head are tracked to detect different behavior. A MLP Neural Network is used to analyse the extracted feature vectors to determine a fall event and motion classification.

Shaou-Gang Miaou et al. (2006) developed a fall detection system which uses an omni-camera called MapCam. The personal information of every subject such as height, weight, electronic health history are used to reduce faulty detection and increase sensitivity of the system. Background subtraction method is used to perform object segmentation from the image. A bounding box is applied to enclose the tracked subject. The ratio of the subject's height and weight is calculated for every single frame. The system analyse the ratio for the last six continuously frame. A fall event is detected when the ratio for the first three frames within six continuous frames is greater than 1, and the last three frames within six continuous frames is less than 1. The results show that the detection rate of the fall detection system using this approach is low. The experimental results show a 79.8% detection rate when the personal information are provided, and 68% without personal information.

2.7.2.3 Posture

The use of posture information can increase accuracy of a fall detection system. The posture of the subject is calculated by analyzing the body parts position. Rita Cucchiara et al. (2005) proposed a fall detection system which analyse human behaviors by considering the posture of the tracked human. The projection histograms of the tracked human are calculated and compared with a set of posture samples. The advantage of this system is that it resolved the occlusion problem when tracking a human.

Nicolas Thome et al. (2006) implemented motion modeling system using a Hierarchical Hidden Markov Models (HHMM) with two layers of states. The first layer consists of two states which corresponds to the human posture that is an upright standing pose and a lying pose. The first layer is dedicated to detect sudden change such as a fall by analyzing the motion features which correspond to a big movement within a small time period. At the second layer, the authors studied the relationship between the 3D angle and their projection with the image plane. The system derive theoretical properties after performing an initial image metric rectification to handle the error angle which is generated in the process of formatting an image for a standing posture. This allows the system to declare other posture as "non-standing", hence the system can accurately detect a person falling down from sitting or walking position. However, the system is time consuming when trying to differentiate a fall event from sitting position.

2.7.2.4 Analyze 3D Head Position

Head position analysis will track the subjects' head and look for the occurrence of large movement within a certain time period. To track the subjects' head with different magnitude of movement, different state models are used. C. Rougier et al. (2005) proposed a fall detection system using a monocular camera for streaming video. Rougier conjecture that a fall event occurs when the subjects' head is still visible in the streaming image and a sudden large movement occur with specific 3D trajectory data. The 3D head trajectory contains data such as 3D velocity of tracked head. Rougier et al. is able to distinguish fall event from daily actions with the proposed algorithm.

Bart Janse et al. (2006) proposed the principle that when a fall event occurs, the vertical acceleration of the subjects' head is faster than the horizontal acceleration. This principle is used in the proposed system for fall detection and jumping detection. The model uses data extracted from the image that is obtained by 3D visual approaches along with a context model. The context model detects a fall event with different methods based on time, location and duration of a fall event.

2.7.3 Comparison between Wearable Device based and Vision based Approaches

In this section, advantages and disadvantages of wearable device based and vision based approaches will be discussed. The wearable devices based approaches have advantage in installation and setup, that is, they are much easier to setup compared to vision based approaches. The cost for

wearable devices based approaches is lower than another approaches. Unfortunately, the wearable devices can be affected by electrical noise and wireless signal, which can cause the device to be disconnected from the system and therefore, unable to detect a fall event. In additional, it might not be comfortable for the subjects to wear the device for the long period of time.

The vision based system has advantage in resolving the problem of intrusion. Most of the vision based systems can detect fall in real time by using normal computing platforms and low costs cameras. Furthermore, the vision based approaches have higher accuracy and robustness when compared with wearable device based approaches. The disadvantages of these approaches are the complexity in creating the system as well as the range of detection depending on the camera range. Vision based approaches are usually complex to develop due to problems such as body part occlusion, and low resolution camera. In addition, the computing device speed has to be resolved in order to develop a robust system. Table 2.4 briefly shows the comparison between two approaches.

Table 2.4 Comparison between Wearable Devices and Vision Based Approaches

Approach	**Cost**	**Intrusion**	**Accuracy**	**Setup**	**Robust**
Wearable Devices	Low	Yes	Scenario Dependent	Easy	No
Vision Based	High	Low	High	Medium	Yes

2.8 Summary

This chapter discussed the mathematical model of depth-map generation in Kinect .The function of human gesture recognition for human behavioral analysis system for video surveillance system is also discussed in this chapter. Numerous gesture recognition techniques such as Mahmoud Elmezain's method and Shahriar Nirjon's method were discussed in section 2.5. The literature review reveals that various challenges (occlusion, deal with a large number of gestures and camera's resolution) still exist when using gesture recognition. One of the most challenging problem is occlusion between subject with other subjects/objects. The mathematical model of depth-map generation.

In additional, numerous well known classifier methods for example, Hidden Markov Models, Support Vector Machines, Finite State Machines and Dynamic Time Warping models were also discussed. Gesture representation algorithms like 3D Model-based Algorithms and Appearance based Algorithms for gesture recognition are also discussed.

Two main approaches that is, wearable device based and vision based fall detection are also discussed in this chapter. The discussion revealed that accelerometers are usually used in wearable device based approaches. Vision based approaches usually use cameras along with several classifiers for fall detection. The comparison between two approaches were also presented in this chapter.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter will provide an overview of the behavioral analytic system using IR camera and RGB camera as these are built into the Microsoft Kinect Sensor which is used as the input device for the proposed system. The details of the system are discussed in this chapter.

3.2 System Setup

The proposed behavioral analytic system uses a Microsoft Kinect Sensor to receive live streaming audio and RGB-depth data. The Microsoft Kinect Sensor has one RGB camera, one infrared camera and four array microphone which can be used in 3D motion capture, voice recognition and facial recognition. The infrared camera consists of an infrared projector and is integrated with a monochrome CMOS sensor. With the infrared camera Microsoft Kinect Sensor is able to capture 3D video data under any light condition which will be further discussed in section 2.3.

The resolution of the color image recorded by the RGB camera of Kinect sensor is 640 x 480 (8-bit VGA resolution) with a Bayer color filter at a frame rate of 9 to 30 fps recommended by Microsoft. Bayer color filter is a color filter array used to organize the RGB color filters on a square grid of photosensors as shown in Figure 3.13 [63]. The pattern of the color filters are 50% of green (G), 25% of red (R) and 25% of blue (B).

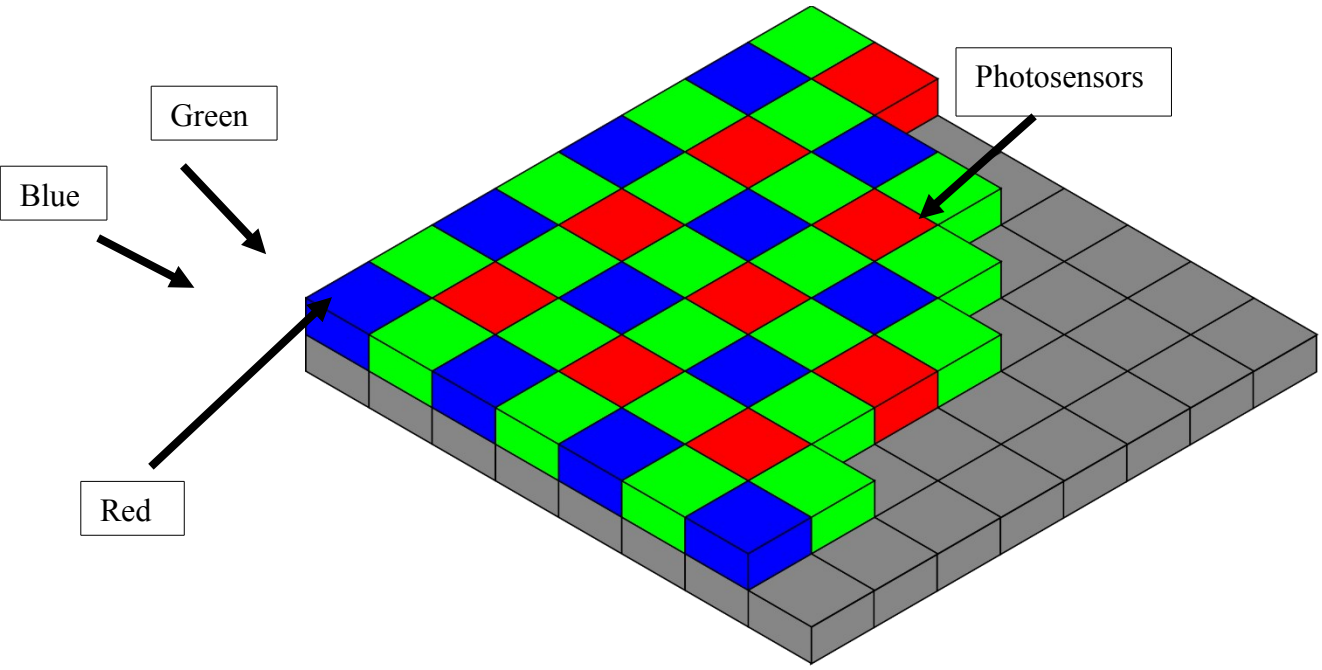


Figure 3.13 Bayer Color Filters on the Pixel Array of an Image Sensor

The resolution of the depth-maps recorded by the Infrared camera of Kinect sensor is 640 x 480 (8-bit VGA resolution) with 11-bit depth, which provides 2048 levels of sensitivity. Unfortunately, the Microsoft Kinect Sensor has a limitation with the detecting range. The detection range of the Kinect sensor is 1m to 3.5m. Subjects that fall in the range of 3.5m to 4.3m are still detectable but the accuracy of the system will decrease. Therefore, an experiment is conducted to evaluate all basic human behaviors, fall detection and human aggressive behaviors within a range of 1m to 4m. The Kinect sensor is placed at a height of 1m from the floor, because it is within the range of height (0.6m to 1.8m) recommend by Microsoft. Figure 3.14 shows the experimental setup for Kinect Sensor.

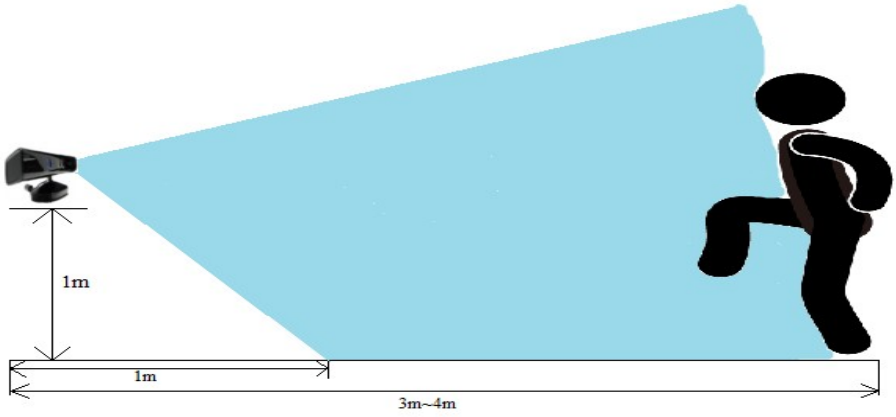


Figure 3.14 Experimental Setup for Human Behavioral Analysis System

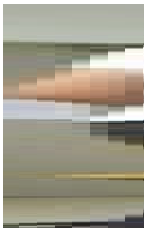
The computer utilized in the experimental setup is an Intel core I7 3630QM with 8 GB RAM. Data sent to the computer from the Kinect sensor is then analyzed by proposed algorithm using Microsoft Visual C# 2010 Express. Some samples of the code is shown below; more sample code can be found in Appendix C.

```
public MainWindow()
{
        InitializeComponent();
}
```

```
private void HABclassifier(double Lleg_velocity, double footL_distance, double
Rleg_velocity, double footR_distance , double Lhand_velocity , double
Rhand_velocity)
{
        if (Lleg_velocity > 2.3 && footL_distance > 0.4 || Rleg_velocity > 2.3 &&
        footR_distance > 0.4)
      kicking_flag = true;
    else
      kicking_flag = false;

        if (Lhand_velocity >= 1.75 && Lhand_up || Rhand_velocity >= 1.65 &&
        Rhand_up)
      sfighting_flag = true;
    else
      sfighting_flag = false;
}
```

(a) Microsoft Kinect Sensor

(b) Laptop



(c) Behavior detection

Figure 3.15 Basic View of Behavioral Analysis System

3.3 Experiment Subjects

The proposed system is evaluated with 30 subjects with different height, width, gender and attire as shown in Figure 3.16. The live streaming video of the subjects are recorded within Microsoft Kinects detectable range (1m to 4m). The sample of the video sequences is shown in

Figure 3.16 Some Samples of Subjects with Different Attire

All subjects are evaluated for five basic behaviors, that is, sitting, crouching, jumping, walking and running, and two aggressive behaviors that is punching and kicking at different angles and illumination. In addition, the system is also able to detect a fall event at different angle and illumination. All the subjects are requested to perform some random action such as waving hand, standing up quickly from crouching pose, and shaking hand. Figure 3.17 shows the experimental environment for evaluating the system. Figure 3.18 shows some video sample sequence when detecting a behavior.

Figure 3.17 Evaluation Environment

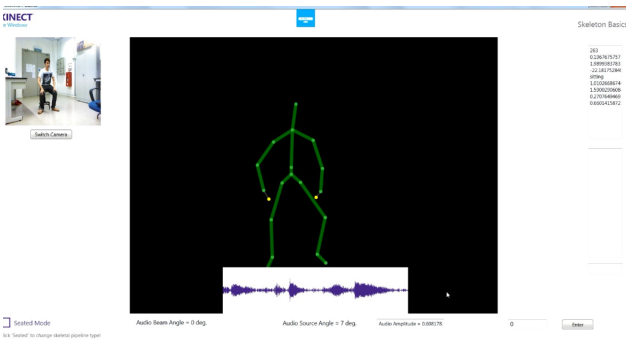


Figure 3.18 Video Sequences Sample

3.4 System Limitation

It is observed that the Microsoft Kinect sensor has two limitations. Firstly, Microsoft Kinect Sensor can detect maximum six subjects in a single frame, however only two subjects' skeleton joints with the nearest distance to the Kinect sensor can be displayed. The second limitation is, which is discussed in section 3.2, the detectable range of the Kinect Sensor. The accuracy of the proposed system decreases when the subject fall out of the detectable range of the Kinect Sensor.

3.5 Summary

This chapter gives an overview of the proposed system including the equipment used and the system setup. Thirty subjects are requested to evaluate the behavior analytic of the system. All the subjects are required to perform five non-aggressive behaviors, two aggressive behaviors and a fall event to evaluate the proposed system.

CHAPTER 4

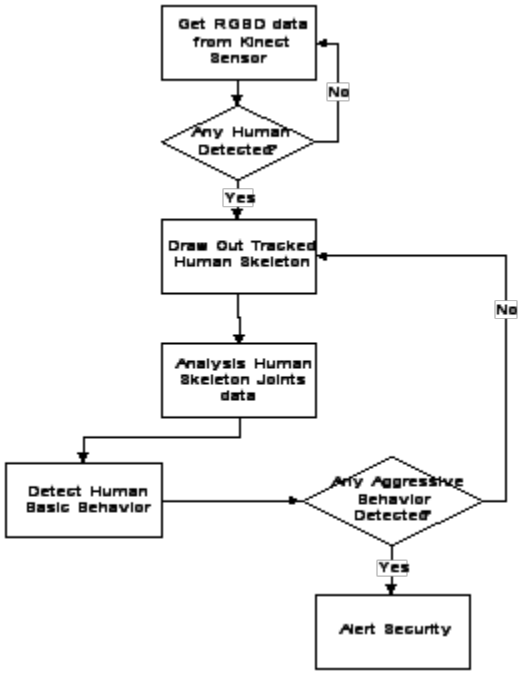
ALGORITHM

4.1 Introduction

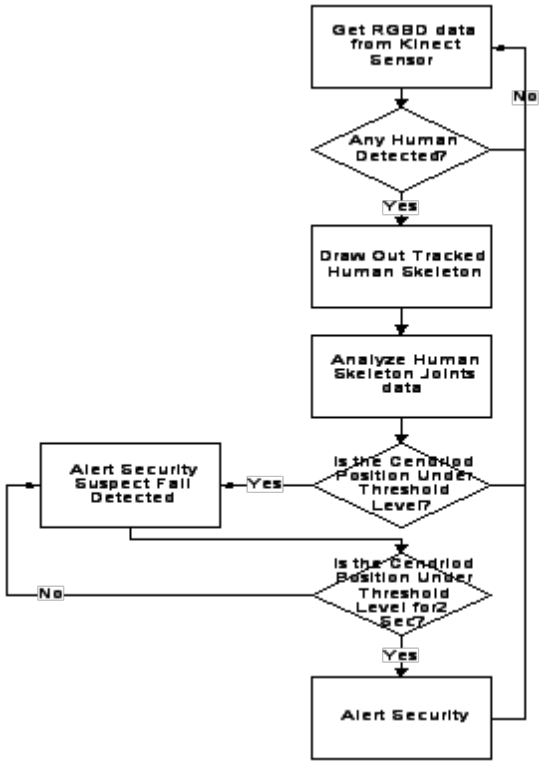
The main focus of this chapter is the detailed discussion of the two main modules used in the proposed algorithm. Flowchart of each module will be explained in detail. The first module is the algorithm of human behavioral analysis system using Microsoft Kinect sensor. The second module is human fall detection system using Microsoft Kinect sensor.

4.1.1 Flow Chart

Figure 4.19 shows the flow chart of the proposed system.



(a) Flow Chart of Human Behavior Analysis



(b) Flow Chart of Human Fall Detection

Figure 4.19 Flow Chart of Behavior Analysis System

The followings steps are involved in analyzing human behavior:

1. Get the color data with depth data from Microsoft Kinect Sensor. Kinect will create a depth-map using structure light method discussed in section 2.3.

2. Detect whether there are any humans in the region.

3. If there is any human detected, Microsoft Kinect will draw out the skeleton joints of the tracked human. The floor plane coordinates of the tracked human will be evaluate using the floor plane equation which parameters are modify as shown in Equation 4.1.

4. The skeleton joints position is calculated using Cartesian coordinate point-plane distance formula shown in Equation 4.2. The velocity of shoulder center, both legs and hand are evaluated using Eqaution 4.3.

5. Classifiers will analysis the human skeleton joints data from step 4 and compare the features with the thresholds shown in Figure 4.21 to Figure 4.24. The proposed classifiers will be further explained in section 4.3.

6. If an aggressive behavior or a fall event occurs, the system will alert the operator.

4.2 Proposed Algorithm for Human Behavioral Analysis System

A robust illumination invariant human behavioral analysis system is discussed in this section. The proposed system used a Microsoft Kinect sensor as a video surveillance system. The proposed system also uses depth-maps which are generated by using the Microsoft Kinect sensor built-in IR sensor. The advantage of Microsoft Kinect sensor is that the system will work even when there is no light at all. In the proposed algorithm, the live streaming video captured by Kinect sensor is converted into depth-maps using structure light method which will be further explained in the following section. Secondly, the system draws out the estimated skeleton joints of the tracked human. After that, the position of skeleton joints are calculated with Cartesian coordinate point-plane equation. The behavior of the tracked human is recognized by using a proposed classifier. The block diagram of the proposed algorithm is shown here;

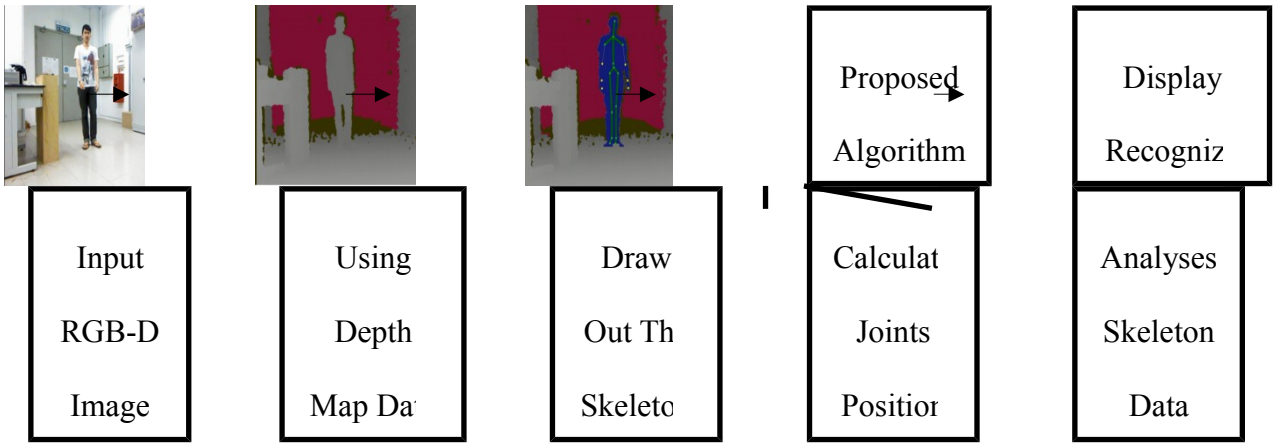


Figure 4.20 Block Diagram of Proposed Algorithm

4.3 Proposed Classifier

The classifier of the proposed behavioral analysis system will be discussed in this section. The first 3 steps of the proposed system are explained detail in section 2.3 and 2.4. To recognize the behavior of the tracking human, the proposed classifier will analyze the actual position of the tracked human body joints, which is calculated using the inferring joints position features mentioned in section 2.4.1.3.

To calculate the human joints position, a floor plane equation is needed to estimate the coordinate of the floor in the real world. Although Microsoft Kinect library provided its own parameter for Equation 4.1 to calculate the coordinate of the floor plane using the floor plane equation. However, the Kinect does not always detected the floor especially when the subject is standing on the stairs. As the result, Microsoft Kinect is not able to estimate the value for **A**, **B**, **C** and **D** in equation 4.1. Therefore, a floor plane equation with modified parameter has been developed as show in Equation 4.1.

$$Ax + By + Cz + D = 0 \tag{4.1}$$

where $\boldsymbol{A} = 0$, $\boldsymbol{B} = \cos\theta$, $\boldsymbol{C} = \sin\theta$, $\boldsymbol{D} = 1$, θ = the calibration angle of the Microsoft Kinect Sensor. The modified floor plane equation parameters are defined according to experimental data carried out in this research. The parameter $\boldsymbol{A}$ is defined as 0 because we assumming all the subjects are in Y-Z axis 2D space. The Cartesian coordinate point-plane distance equation is integrated into the element in floor plane equation as shown in Equation 4.2. This equation is used to calculate the joints position of tracked human.

$$d = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}} \tag{4.2}$$

where (***x, y, z***) is the coordinate element of the joints position, ***d*** is the joint position. Human body joints position data for every 20 frames is used to calculate the velocity. Velocity of a few body joints such as, head, shoulder center, both hands and legs is calculated using the Equation 4.3

$$V_i = \frac{1000}{N-1} \sum_{i=1}^{N-1} \frac{d_{i+1} - d_i}{t_{i+1} - t_i} \qquad (4.3)$$

where **N** is the number of frame, **d** is the distance calculated by using Equation 4.2, and **t** is the time for each frame.

Using the analysis of the joints position and velocity of joints, the system is able to differentiate between a normal behavior and an aggressive behavior with the proposed classifier which explained in this section. For an example, if one of the human subjects moves his hand with a velocity higher than a threshold level and touches another subject's body, it is interpreted by the algorithm as punching. Punching behavior is not detected by the system in the following two conditions: if the hand velocity is high but no body contact takes place or body contact occurs but the velocity does not reach the threshold level. Table 4.5 shows the joints position and velocity for each behavior concerned. The proposed algorithm will consider eight joints out of twenty skeleton joints generated by Kinect as stated in Table 4.5. The behavioral analytic system used 8 joints to differentiate behavior, and the fall detection system used 4 joints to detect fall event.

Table 4.5 Joints Position and Velocity of Each Behavior Concerned

No.		Joints Position	Joints Velocity
1	Jumping	Head, Both Leg	Shoulder Center
2	Walking	Both Leg	Shoulder Center
3	Running	Both Leg	Shoulder Center
4	Sitting	Hip	-
5	Crouching	Hip	-
6	Fall Event	Head, Spine	Shoulder Center
7	Punching	Both Hand, Spine	Both Hand
8	Kicking	Both Leg	Both Leg

The classifier is trained on a set of randomly daily behavior image to recognize specific pose. The image set consists eight subsets for use in training to detect each pose. Each subset consists of 210 images which are captured from 30 subjects of different gender, height, and attire. The data of the experiments are gathered, and plotted as shown in Figure 4.21 to Figure 4.24. Figure 4.21 and Figure 4.22 show the mean velocity of some joints for each behaviors. In Figure 4.22, the shoulder center's velocitys are sparated into two; Y-axis only and 3-axis, because the system is only considering the Y-axis velocity to increase the accuracy rate of detecting jumping behavior. Figure 4.23 and Figure 4.24 show the mean position of some joints for each behaviors.

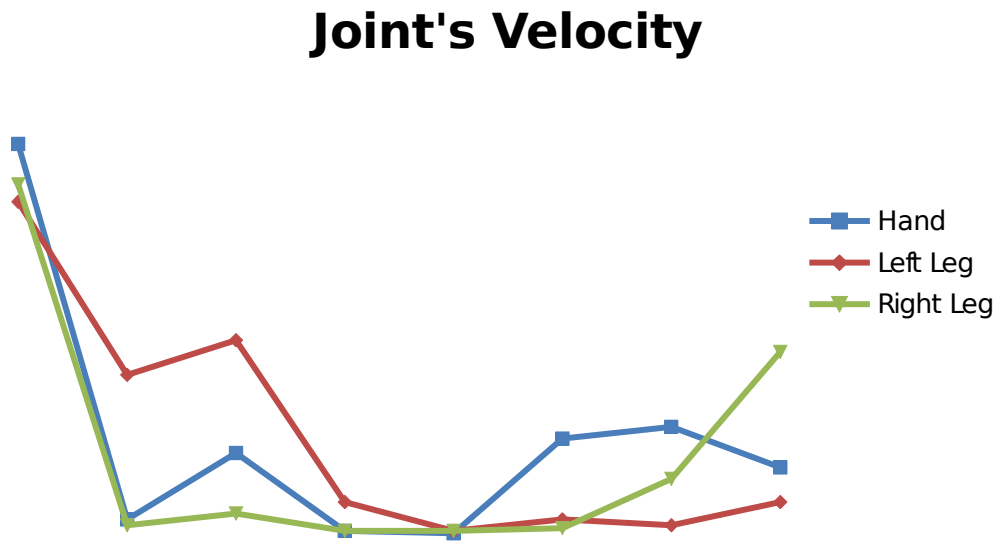


Figure 4.21 Mean of Joints' Velocity for Concerned Points for Different Action

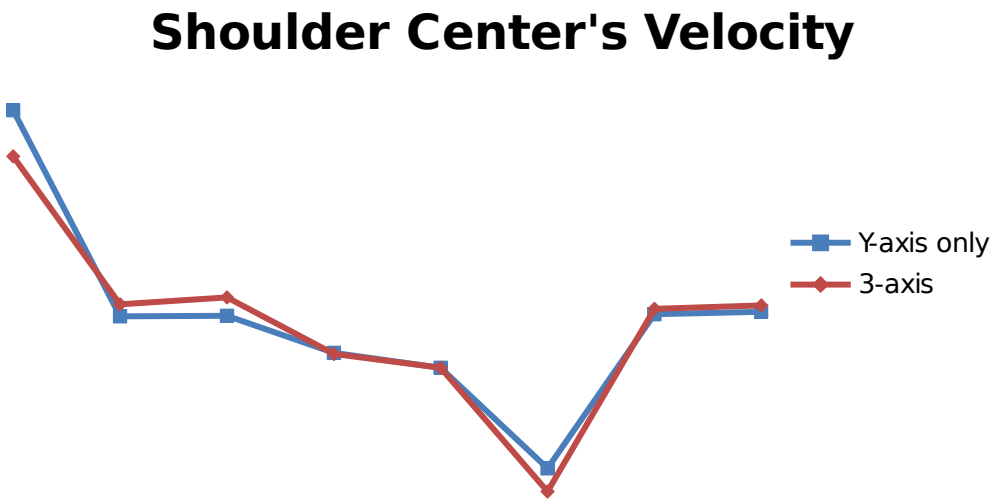


Figure 4.22 Mean of Shoulder Center's Velocity

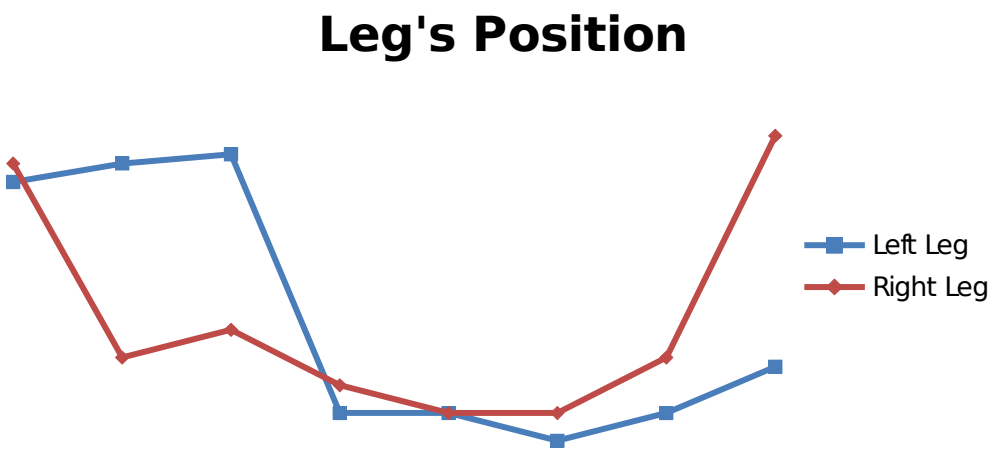


Figure 4.23 Mean of Leg's Position Graph

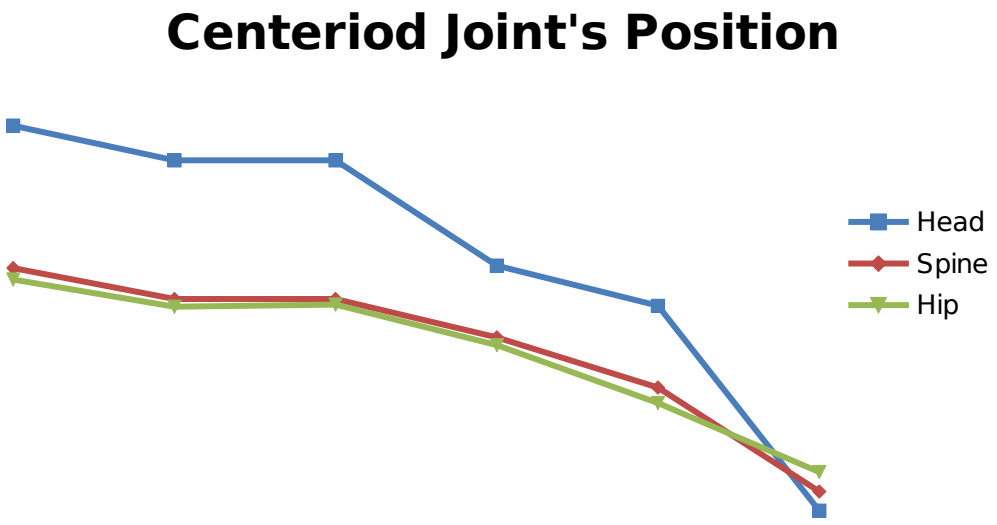


Figure 4.24 Mean Body Centeroid Joints' Position Graph

4.3.1 Finite State Machine

The proposed classifier is developed based on finite state machine model which is mentioned in section 2.5.1. The proposed system consists of three different classifier to detect human basic behaivor, human aggressive behavior and fall event. Figure 4.25 shows the state diagram for human basic behavior. Figure 4.26 shows the state diagram for aggressive behavior detection. Figure 4.27 shows the state diagram for fall detection.

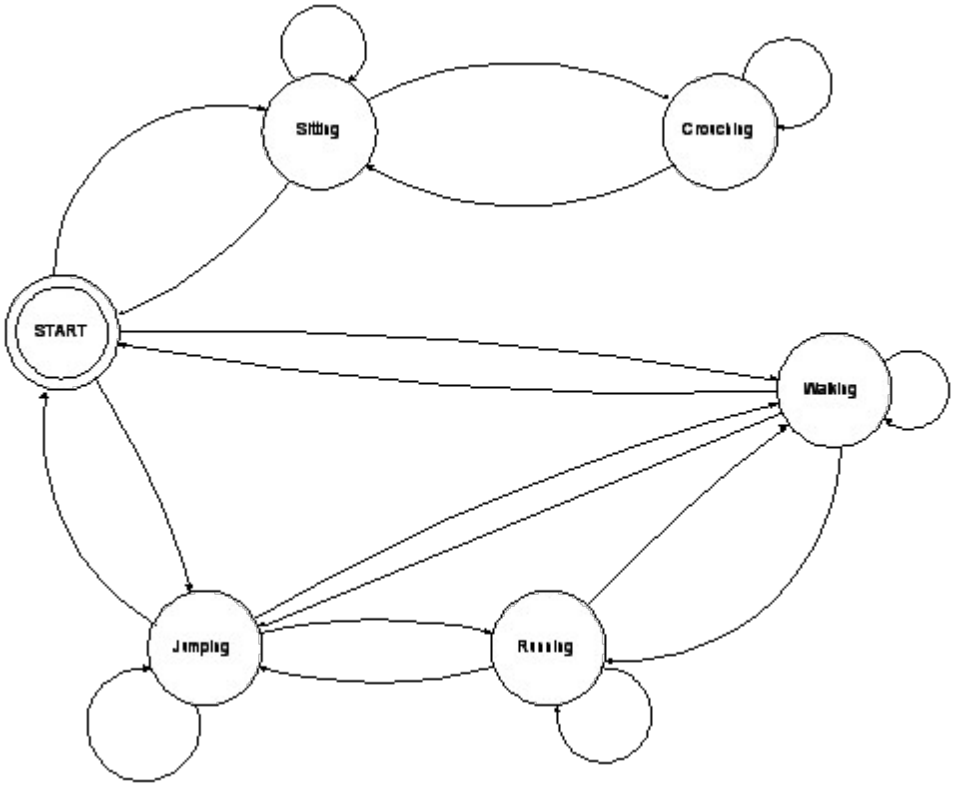


Figure 4.25 State Diagram for Human Basic Behavior

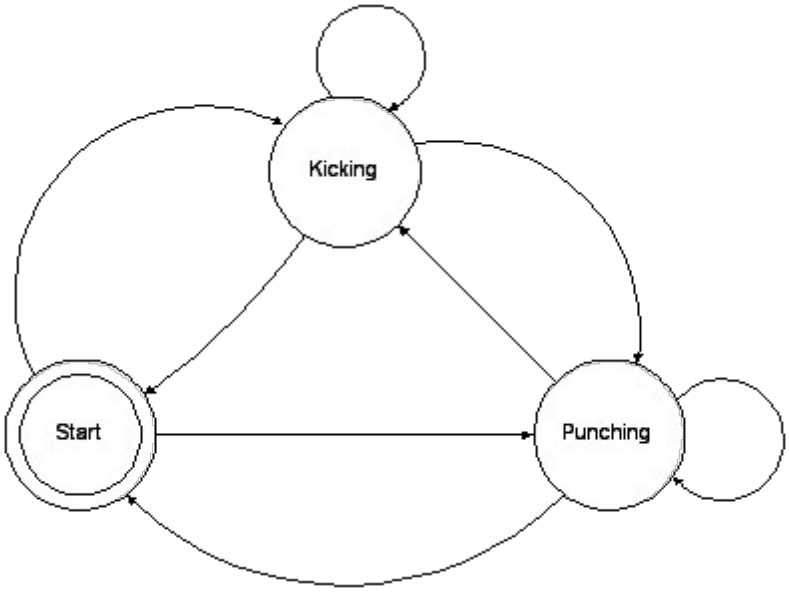


Figure 4.26 State Diagram for Human Aggressive Behavior

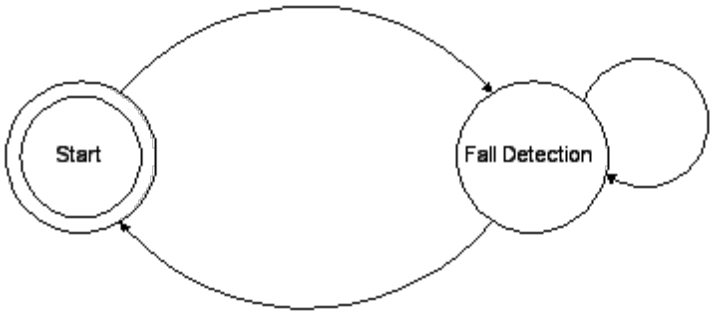


Figure 4.27 State Diagram for Fall Detection

As mentioned in section 4.3, the classifier is trained with 30 subjects. There are 23 males and 7 females involved in the experiment. The age range of the subjects are 23 to 27 years old. The classifier learnt the data from Figure 4.21 to Figure 4.24 to perform the classification by treating them as the

threshold level for the Finite State Diagram. Each state has a different set of joints position and velocity. It is observed from Figure 4.25 that the three behaviors jumping, running and walking are correlated, and sitting and couching pose are correlated. The simplified pseudo-codes of the proposed system are shown below:

Pseudo-code for human aggressive behavior analysis:

```
private void HABclassifier (double Lleg_velocity, double footL_distance,
double Rleg_velocity, double footR_distance, double Lhand_velocity,
double Rhand_velocity)
{
        if (Lleg_velocity > 2.3 && footL_distance > 0.4 || Rleg_velocity >
        2.3 && footR_distance > 0.4)
                kicking_flag = true;
        else
                kicking_flag = false;

        if(kicking_flag && leg_touch)
                f_kicking_flag = true;
        else
                f_kicking_flag = false;

        if (Lhand_velocity >= 1.75 && Lhand_up &&body_contact_flag||
        Rhand_velocity >= 1.65 && Rhand_up &&body_contact_flag)
                sfighting_flag = true;
        else
                sfighting_flag = false;
}
```

l bserved that the HAB
o recognize aggressive behavior. When the leg or hand of one subject touched another subject body part, the body_contact_flag and leg_touch will be true. As the result, the aggressive behavior will be only recognized if the velocity and position of hand or leg reach the threshold level and have body contact.

```
private void HBBclassifier(double height, double hip_distance, double
footL_distance, double footR_distance, double avg_velocity, double
head_distance, double javg_velocity)
{
        if (hip_distance <= (height * 0.545) && hip_distance > (height *
        0.42) )
                sit_flag = true;
        else
                sit_flag = false;

        if (hip_distance <= (height * 0.42) && hip_distance >= (height *
        0.25))
                crouching_flag = true;
        else
                crouching_flag = false;

        if ((avg_velocity > 1.3 && head_distance > 1.6 && hip_distance <
        0.65 * height && hip_distance > 0.61 * height && footL_distance
        > 0.25) || (avg_velocity > 1.3 && head_distance > 1.6 &&
        hip_distance < 0.65 * height && hip_distance > 0.61 * height &&
        footR_distance > 0.25))
                running_flag = true;
        else
                running_flag = false;

        if (avg_velocity <= 1.3 && !sit_flag && !crouching_flag && !jflag
        && avg_velocity > 0.65 && head_distance > 1.6 && hip_distance
        < 0.65 * height && hip_distance > 0.61 * height)
                walking_flag = true;
        else
                walking_flag = false;

        if (javg_velocity > 15 && footL_distance > 0.38 * height &&
        footR_distance > 0.38 * height && !sit_flag && !crouching_flag)
                jflag = true;
        else
                jflag = false;

}
```

From the pseudo-code for human daily behavior analysis, it observed that the HBB classifier able to detect five behavior, that is sitting, crouching, running, walking and jumping. The features used in

recognize sitting and crouching are the same but with different threshold level. The conditions to detect other three behaviors are more complex compare to recognize sitting and crouching.

Pseudo-code for fall event detection:

```
private void HFDclassifier (double head_distance, double spine_distance,
double avg_velocity)
{
        if ((head_distance <= 0.65|| spine_distance <= 0.45)&&
        avg_velocity < -5)// fall detection
                fallflag1 = true;
        else
        {
                fallflag1 = false;
                fallflag2 = false;
        }


}
```

when the head_distance or spine_distance is below the threshold level and the avg_velocity of shoulder center is lower than -5. The proposed system will first defined the fall event as a suspect fall event. If the subject does not move for 4 second, the system will on confirm it is a fall event.

4.4 Summary

This chapter discussed the detail of the behavior analysis system. The steps that are involved in human behavior analysis system are discussed. In additional, the proposed classifier is presented too. The result of the behavior analysis system will be discussed in the next chapter.

CHAPTER 5

RESULTS AND ANALYSIS

5.1 Introduction

This chapter will discuss the results of the proposed system. As mention in topic 1.7 this research has proposed a robust human behavioral analysis system for video surveillance using Microsoft Kinect sensor as the input device. A statistical model of local motion descriptors is used as the base for the proposed framework, which provides a measure for behavioral analysis. This chapter will discuss the results gathered from different evaluations using a database which consist 30 subjects. Each of the subjects has different height, skin color, and wearing a different attire.

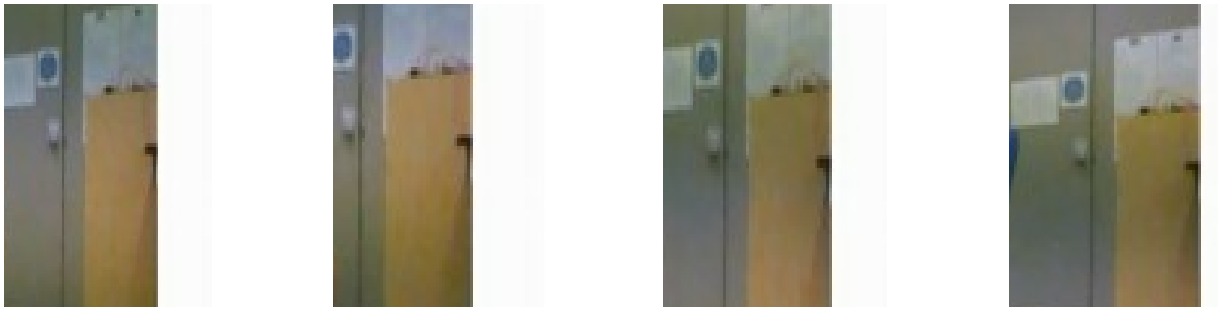
All subjects are evaluated for five basic behaviors that are; sitting, crouching, jumping, walking and running, two aggressive behaviors that are; punching and kicking; and fall event in various of angle (from 180º to -180º) and illumination. Random behaviors such as hand shake, waving hand, standing up rapidly from crouching position are performed to evaluate the accuracy and confusion rate of the system. The confusion rate is a measure of how the system is confused by some behavior like crouching and sitting. The physical size of the subjects is proportional to the distance in between subjects and Kinect sensor. The proposed system can be divided into two parts, that is, human behavior analysis and fall detection. To benchmark the proposed human behavior analysis system, the performance is compared with DTW-based gesture matching algorithm for Kinect which is proposed by Shahriar Nirjon's method (2014).

5.2 Human Behavior Analysis

The results of the proposed human behavior analysis algorithm are presented in this section. The proposed human behavior analysis system can be separated into two parts, that is, normal behavior analysis and aggressive behavior analysis. All evaluations were carried out within the range of 1m to 3.5m under two illumination conditions that is, lights on and lights off. This is because the detectable range of Microsoft Kinect sensor is 1m to 3.5m.

5.2.1 Normal Behavior Recognition

The proposed algorithm can detect 5 normal behaviors, that is sit, crouch, jump, run, and walk in real time from live streaming video. Figure 5.28 shows the subject in sitting position at different angles. Figure 5.29 shows the subject in crouching pose at different angle. Figure 5.30 shows the subject in jumping pose at different angles. Figure 5.31 shows the subject in walking and running pose at different angles.



0 degree 90 degree 180 degree 270 degree

Figure 5.28 Examples of Detected Sitting Pose

0 degree 90 degree 180 degree 270 degree

Figure 5.29 Examples of Detected Crouching Pose

Normal Jump

High Jump

Spin when Jumping

Figure 5.30 Examples of Detected Jumping Pose

0 degree

90 degree

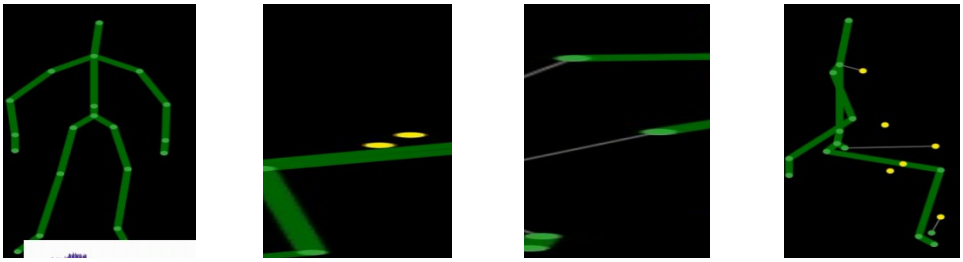
180 degree

270 degree

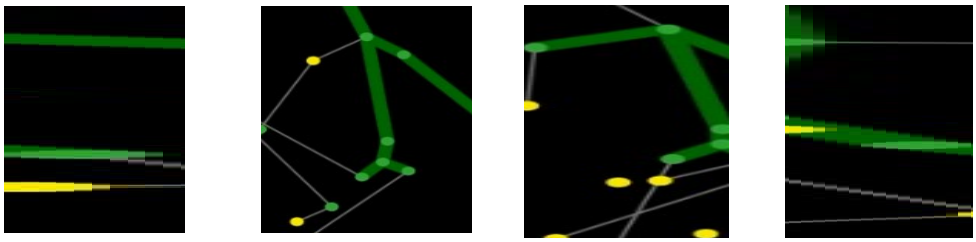
Figure 5.31 Examples of Detected Walking and Running Pose

Figure 5.32 shows the skeleton of the subject from Figure 5.1 to Figure 5.4. The skeleton data that is extracted by Microsoft Kinect. In addition, the Kinect can recognize behavior under no lighting environment. However, when the body parts are occluded by own body parts or objects, the Kinect is unable to estimate the position of the joints correctly. The proposed algorithm has to overcome this problem. It is observed from Figure 5.32 that the cendriod joints (head, hip, spine) of the tracked

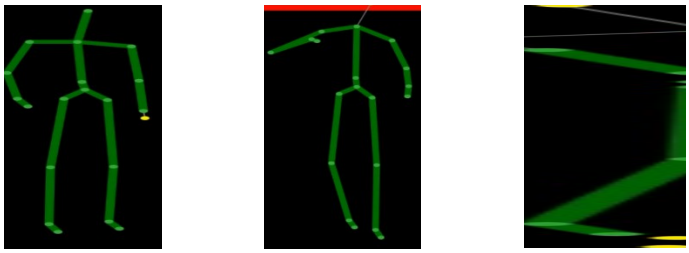
skeleton always inferred even if occlusion occurs. Therefore, the proposed method analyses eight joints from all twenty skeleton joints, that is, head, knees, hip, spine, both legs, and hands.



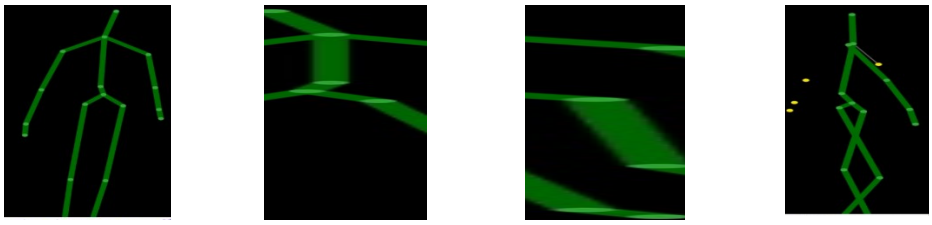
(a) Sitting Pose



(b) Crouching Pose



(c) Jumping Pose



(d) Walking and Running Pose

Figure 5.32 Samples images of Skeleton Data Extracted by Kinect (Basic Behavior)

5.2.1.1 Result Comparison between Proposed and DTW-based Method

The proposed method has been compared with DTW-based gesture recognition method and the results are shown in Table 5.6, Table 5.7 and Table 5.8. It can be observed that the proposed method is able to increase the sensitivity and accuracy for recognizing human basic behavior (HBB). The sensitivity of the proposed method can be calculated using equation 5.1. and the accuracy of proposed system can be calculated using equation 5.2.

Testing is conducted to establish the sensitivity and accuracy of the proposed system. With the reference to Figure 5.28 and Figure 5.32(a), a sitting pose can be positive, that is, the pose is recognized by the system as sitting. It can be negative, if the system fails to recognized it as sitting pose.

- True positive (TP): Recognize sitting pose when the subject is sitting.
- False positive (FP): Recognize sitting pose when the subject is not sitting.
- True negative (TN): Fail to recognize sitting pose when the subject is not sitting.
- False negative (FN): Fail to recognize sitting pose when the subject is sitting.

$$sensitivity = \frac{number\ of\ true\ positives}{number\ of\ true\ positives + number\ of\ false\ negatives} \quad (5.1)$$

$$accuracy = \frac{number\ of\ true\ positives + number\ of\ true\ negative}{\sum Total\ population} \quad (5.2)$$

Where total population is the summation of TP, FP, TN, and FN

Table 5.6 Comparison of Sensitivity for 30 Subjects

	Sit	Crouch	Jump	Walk	Run	Fall Down
Proposed Method (light condition, %)	100	97.40	78.13	86.21	100	100
Proposed Method (no light condition, %)	95.18	97.59	76.76	84	100	100
DTW Method (%)	46.67	38.67	<0.1	<0.1	<0.1	<0.1

Table 5.6 shows the sensitivity for both methods for 450 images from thirty subjects. The sensitivity for jumping, walking, running and falling down using the DTW method are very low, hence negligible. This is because the DTW method could not detect the composition of a series of actions. It can only detect gesture by matching the position of the body joints without considering the height of the body from the floor plane. It can be concluded that the DTW gesture matching method is not able to detect jumping and fall down actions. In addition, Table 5.6 also shows that the sensitivity of the proposed system is dropped when the environment is in no light condition. This is because of the RGB camera of the proposed system is not functioning when the environment is dark. As the result, the system only relay on the infrared sensor instead of combining RGB camera and infrared sensor data which will slightly decrease the sensitivity of the system.

Table 5.7 Comparison of Accuracy for 30 Subjects for Four Angle (Sitting Pose)

	Sitting			
Degree (°)	**0**	**90**	**180**	**270**
Proposed Method (light condition, %)	100	100	100	100

Proposed Method (no light condition, %)	85.71	95.24	95.24	100
DTW Method (%)	100	13.33	100	6.68

Table 5.8 Comparison of Accuracy for 30 Subjects for Four Angle (Crouching Pose)

	Crouching			
Degree (°)	**0**	**90**	**180**	**270**
Proposed Method (light condition, %)	100	100	100	100
Proposed Method (no light condition, %)	100	90.48	100	95.24
DTW Method (%)	100	0	50	0

From Table 5.7 and Table 5.8, it is observed that the DTW cannot perform well at angle 90º (face to right) and 270º (face to left) for both poses. Therefore, this research conclused that the Kinect is unable to estimate the body joints accurately when there are occlusion between body parts and joints. As a result, the accuracy of DTW is low. However, the proposed method can detect these poses, even when the body joints are occluded. Table 5.7 and Table 5.8 also show that the accuracy of the proposed system in no light condition had dropped, because the RGB camera of Kinect sensor is not functioning in dark. Microsoft Kinect need to combine RGB camera with IR camera to enhance the accuracy of estimating body joints.

5.2.2 Aggressive Behavior Recognition

The proposed algorithm can detect two aggressive behavior, that is, punching and kicking in real time from live streaming video. Figure 5.33 show examples of detectable kicking pose. Figure

5.34 shows the examples of detectable punching pose. It is observed that the proposed system can detect human aggressive behavior regardless to subjects' attire, height, gender and skin color. The skeleton joints for Figure 5.33 and Figure 5.34 are shown in Figure 5.35.

Figure 5.33 Examples of Detected Kicking Pose

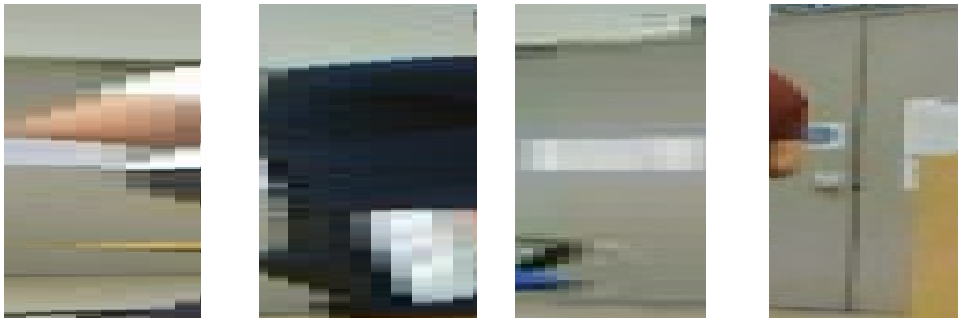


Figure 5.34 Examples of Detected Punching Pose

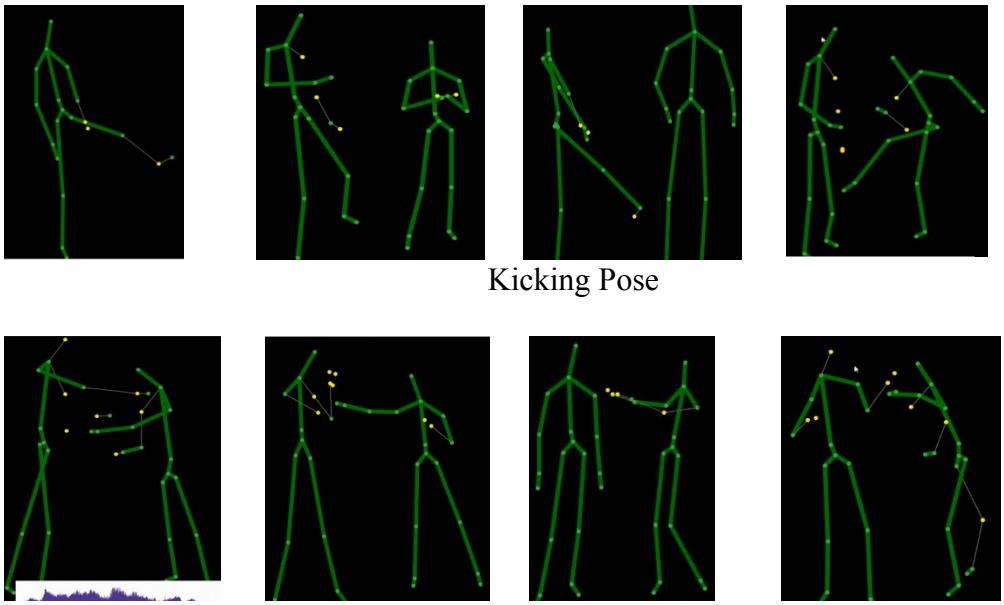


Figure 5.35 Samples images of Skeleton Data Extracted by Kinect (Aggressive Behavior)

5.2.2.1 Result Comparison between Proposed and DTW-based Method

The results of comparison between proposed algorithm and DTW-based method is shown in Table 5.9. The sensitivity of human aggressive behavior(HAB) can be calculated using equation 5.1. From Table 5.9, it can be seen that the proposed system in light/no light condition has higher sensitivity than the DTW gesture matching system. The sensitivity rate of human aggressive behavior is not effected by no light condition. This is due to the classifier of human aggressive behavior is more complex comepared to other behaviors, since this research is focusing in detect aggressive behavior. It can also be observe that the sensitivity of the proposed system in light condition is 18.05% higher for punching and 44.44% higher for kicking compared to DTW-based method. Even when the proposed system is in no light condition, the sensitivity is 17.46% higher for punching and 44.44% higher for

kicking compare to DTW-based method. Similarly, experiment subjects are required to repeat the pose at different angle. To test the sensitivity of the system, subjects are required to perform some random behaviors, that is, hand shaking, waving hand, and touching.

Table 5.9 Sensitivity for HAB for the Proposed Method and DTW Method

	Punching	**Kicking**
Proposed (light condition)	95.83%	100%
Proposed (no light condition)	95.24%	100%
DTW	77.78%	55.56 %

In order to evaluate the performance of proposed method, this research has created a confusion matrix as shown in

. It is observed that few behaviors can confuse the system such as sitting with crouching, jumping with walking, kicking with running, and so on. It is also observed that the proposed system can distinguish each behavior except for jumping with walking. This is because when the subjects jump from one location to another location, displacement is involved, and so there is a 7.41% chance that the system will detect that behavior as walking behavior.

Table 5.10 Confusion Matrix for the Proposed Method

	Sit	**Crouch**	**Jump**	**Walk**	**Run**	**Punch**	**Kick**
Sit	100%	0	0	0	0	0	0
Crouch	0	100%	0	0	0	0	0
Jump	0	0	92.59%	7.41%	0	0	0
Walk	0	0	0	100%	0	0	0
Run	0	0	0	0	100%	0	0
Punch	0	0	0	0	0	100%	0
Kick	0	0	0	0	0	0	100%

5.3 Fall Detection System

The results of the proposed fall detection algorithm is discussed in this section. The comparison results between DTW-based algorithm and the proposed algorithm is shown in Table 5.6. As mentioned in section 5.2.1.1, DTW-based algorithm need to match the position of the body joints from the training set, without considering the height of the joints from the floor plane to recognize posture. Therefore, the sensitivity of DTW-based algorithm is very low. Both C. Rougier [23] and C. Kawatsu [24] methods also used Microsoft Kinect sensor. Their results show that the fall detection system using Kinect sensor is still able to detect a fall in all situations, except the computing speed is slower than proposed method by 0.3 to 0.4 sec.

5.3.1 Result for Fall Detection System

In this section, 30 subjects participated in a test under two lighting condition, that is with and without light as shown in Figure 5.36. Figure 5.37 shows the fall detected in no light condition. C. Rougier [23], C. Kawatsu [24] and the proposed method can detect a fall from live streaming video. The comparison of the three methods are shown in Table 5.6.

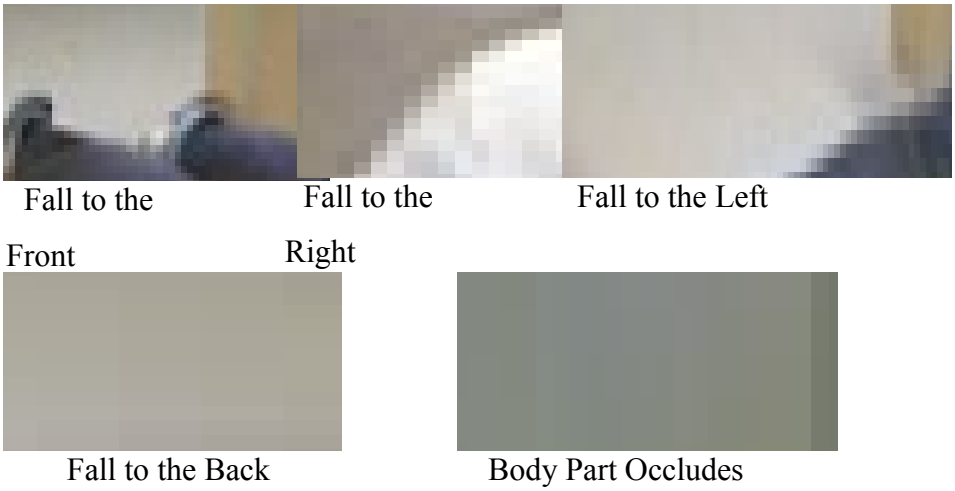
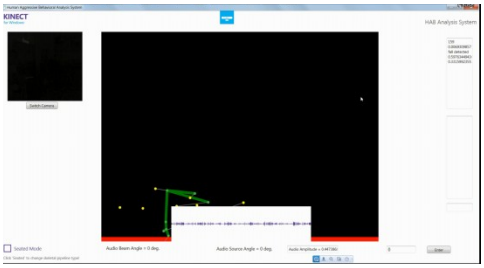
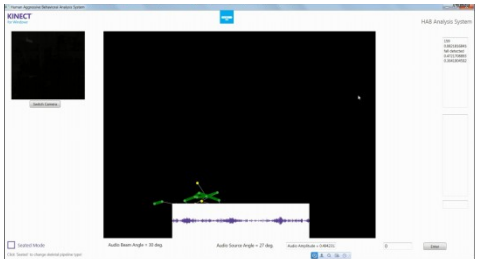


Figure 5.36 Sample Images of Detected Fall



Fall to the Right



Fall to the Back

Fall to the Front

Figure 5.37 Sample Images of Fall Detection under No Light Condition

As shown in Figure 5.36 and Figure 5.37, the proposed system can detect a fall in various lighting condition. In additional, it can be observed that the system is able to detect a fall even when the body parts are occluded.

Table 5.11 Comparison between 3 Methods

	Operate Time	Accuracy
C. Rougier's Method (using Depthmap)	2~3 Sec	100%
C. Kawatsu's Method (concern on 20 skeleton joints)	Real Time	100%
Proposed Method (concern on 2	Real Time	100%

skeleton joints)		

Based on Table 5.11, it is observed that C. Rougier's method cannot perform under real time condition as the method is implemented in Matlab, which is a programming platform that need larger computing cycle for heavy task compare to other platforms. Kawatsu's Method is slower than proposed method used in this research for 0.3 to 0.4 sec, because it take into account all of 20 skeleton joints generated by Kinect. From the table, it can observed that C. Kawatsu's Method and the proposed method can operate under real time conditions.

5.4 Summary

This chapter presented the results of human behavior analysis system and fall detection system using Microsoft Kinect sensor. The goal of the experiments is to show the system working with real time streaming video. This chapter also shows that the proposed algorithms perform better than the DTW algorithm as shown in Table 5.6 to Table 5.9. The behavioral analysis system for video surveillance has been evaluated in terms of accuracy and sensitivity. This section concluded that the Microsoft Kinect sensor is able to resolve the illumination problem.

CHAPTER 6

DISCUSSION AND CONCLUSION

6.1 Introduction

In the introduction of this dissertation, it was highlighted that a system that can detect aggressive behavior can be embedded into a video surveillance system. The dissertation also highlighted that the challenge of human behavioral analysis is the body pose recognition. Therefore, this dissertation agrues that it is justified to put in effort to research behavioral analysis to ensure safety in public places. A fall detection system is also integrated into the developed system to detect fall event of human.

A summary will be given in this chapter, based on the results obtained in this research. Several ideas to improve the system will be discussed in section 6.5.

6.2 Body Pose Recognition

The Body Pose Recognition is the main process in human behavioral analysis system. The proposed system executes the body pose recognition before analyzing the subjects' behavior. Several of techniques used for body pose recognition are discussed in Chapter 1. The disadvantages and advantages of each techniques are also discussed in Chapter 1. The challenges faced in body pose recognition are difficult to recover and recognize the pose of human body due to the complexity of the human body for pose observation. Secondly, the variety of situations and the high number of degrees-of-freedom (DOF) in the body's movement. To overcome these challenges, a video surveillance system using the Microsoft Kinect Sensor as input is developed. The Microsoft Kinect Sensor using the structure light method to create depth-map for recognizing human body in various illumination.

6.3 Fall Event Detection

A fall detection system is intergrated in the proposed system to prevent serious injury in fall event. Fall detection system is a system that automatically detects fall event from video stream. In Chapter 2 of this dissertation, two approaches used in fall detection system are discussed, that is, wearable device based approaches and vision based approaches. The disadvantages and advantages of each technique is discussed in detail.

6.4 Contributions

A robust human behavioral analytic system in video surveillance has developed. The results of the proposed system shows that the system had improved the accuracy and sensitivity in human behavior analysis. The system is able to analyze the human behavior under light and no light condition. The Microsoft Kinect sensor is adopted in the proposed algorithm to generate depth-map from the 3D space using structure light method.

In this research, the parameters of the floor plane equation used, that is, equation 4.1 to calculate the floor position of the subjects is modified. However, the equaton is repeated here for convenience:

$$Ax + By + Cz + D = 0 \tag{4.1}$$

The reason is because, Microsoft Kinect does not always detect the floor position especially when the subject is standing on the stairs. The position of the body joints are calculated using the Cartesian coordinate point-plane distance equation.

Secondly, the proposed system developed a novel classifier using finite state machine as the kernel. The proposed classifier can analyze the features of the body joints and identifies what behaviors the subject is performing. The performance of the proposed system has been evaluated with 30 subjects

with different attire, height, weight and gender. Sensitivity of the system in light condition for sitting pose is 100%, crouching pose is 97.4%, jumping pose is 78.13%, walking pose is 86.21%, running pose is 100%, fall event is 100%, punching 95.83% and kicking is 100% which shown in chapter 5. The result shows that the proposed system archive higher sensitivity compare to the DTW algorithm.

Thirdly, a confusion matrix table is created too. The confusion matrix table shows the rate of how the system is confused by some similar behavior such as crouching and sitting. And the results indicate that the system has 7.41% chance of detecting a jumping behavior as a walking behavior. The accuracy for sitting and crouching pose is higher than the DTW algorithm. The experimental results for the proposed algorithm in terms of sensitivity and accuracy shows that the proposed system can perform well compared to DTW algorithm, 18.05% more for punching and 44.44% more for kicking.

6.5 Future Work

In this research, a human behavioral analytic system in video surveillance has been proposed. The proposed algorithm have solved the illumination problem and increased the sensitivity of behavioral analysis. However, there is still a problem that has not been solved, that is, the limitations of the Microsoft Kinect Sensor. One of the limitation of the Microsoft Kinect sensor is, although the sensor is able to track six persons, but only two persons that are nearer to the sensor body joints will be analyzed. Therefore, the proposed system can only recognize two subjects behavior.

In order to overcome this problem, it is proposed that a better Kinect sensor be used. A better Kinect sensor can detect more than two subjects' body joints. Another method is to use two or more Kinect sensors. This will enable the proposed system to detect more subjects in a crowded environment.

In conclusion, this research demonstrated that it is feasible to embeded a human behavior analytics system into a video surveillance system. The human behavior analytics system can detect human aggressive behavior, and alert the operator to take action.

REFERENCES

Ashraf Abbas M. AL-Modwahi, Halimah Badioze Zaman, and Azlina Ahmad, 2014. Evaluation of an Intelligent Video Surveillance Model Based on Human Behaviour Detection and Analysis. *International Journal of Computer Science Issues (IJCSI)*, Vol. 11 Issue 3, pp. 33.

B. Kaluza and M. Lustrek, 2009. Fall Detection and Activity Recognition Methods for the Confidence Project: A Survey. *12th International Multi-conference Information Society, Vol. A*, 2008, pp. 22–25.

B. Leibe, E. Seemann, and B. Schiele, 2005. Pedestrian detection in crowded scenes. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference*, 20 - 25 June 2005 San Diego, CA, USA, pp 878 – 885 vol. 1.

Bernard Boulay, 2007. Human Posture Recognition for Behaviour. PhD Thesis. Universit´e Nice Sophia Antipolis. English.

Birdal A., and Hassanpour R., 2008. Region based hand gesture recognition. *16th International conference in central Europe on computer graphics, visualization and computer vision*, pp 1–7.

Boulay, B., 2007. *Human posture recognition for behaviour understanding*. PhD thesis, Universit´e de Nice-Sophia Antipolis.

Burton HE, 1945. The optics of Euclid. *Journal of the Optical Society of America 35 (5): 357–372*. doi:10.1364/JOSA.35.000357.

C. Panagiotakis and G. Tziritas, 2004. Recognition and tracking of the members of a moving human body. *Articulated Motion and Deformable Objects*, pages 86–98.

C. Papageorgiou and T. Poggio, 2000. A Trainable Pedestrian Detection system. *International Journal of Computer Vision (IJCV)*, page 1:15-33.

C. Rougier, E. Auvinet, J. Rousseau, M. Mignotte, and J. Meunier, 2011. Fall detection from depth map video sequences, *Proc. ICOST*, pp. 121–128.

Chen J., Karric Kwong, Chang D., and Luk J., 2006. Wearable Sensors for Reliable Fall Detection. *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference*, 17 – 18 Jan 2006 Shanghai, China, pp. 3551-3554.

Chia-Chi Wang, Chih-Yen Chiang, Po-Yen Lin and, Yi-Chieh Chou, 2008. Development of a Fall Detecting System for the Elderly Residents. *Bioinformatics and Biomedical Engineering, 2008. ICBBE 2008. The 2nd International Conference*, 16 -18 May 2008 Shanghai, China, pp. 1359 – 1362.

C-J. Lin and R. C. Weng, 2004. *Simple probabilistic predictions for support vector regression*. Technical report, Department of Computer Science, National Taiwan University.

Cucchiara R., Grana C., Prati A., and Vezzani R., 2005. Probabilistic posture classification for human-behavior analysis. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on (Volume: 35, Issue: 1)*, pp. 42 – 45.

Depth Map, 2014 [Online]. Available at: http://en.wikipedia.org/wiki/Depth_map [Accessed: 1 March 2015]

Disney Research, 2014, *Human pose estimation* [Online]. Availabe at: http://cs.brown.edu/~ls/Publications/SigalEncyclopediaCVdraft.pdf [Accessed: 1 March 2015]

Earl Wong, 2006. A New Method for Creating a Depth Map for Camera Auto Focus Using an All in Focus Picture and 2D Scale Space Matching. *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on (Volume: 3)*, 14 -19 May 2006, Toulouse, pp. 3.

Elizabeth A. Minton and Lynn R. Khale, 2014. Belief Systems, Religion, and Behavioral Economics. *New York: Business Expert Press LLC. ISBN 978-1-60649-704-3*.

F. Fleuret, J. Berclaz, R. Lengagne, and P. Fua, 2008. Multicamera People Tracking with a Probabilistic Occupancy Map. *Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume: 30, Issue: 2)*, pp. 262 -2782.

Foroughi H., Aski B. S., and Pourreza H., 2008. Intelligent Video Surveillance for Monitoring Fall Detection of Elderly in Home Environments. *Intelligent Video Surveillance for Monitoring Fall Detection of Elderly in Home Environments*, 24 – 27 Dec 2008, Khulna, pp. 219 – 224.

Foroughi H., Naseri A., Saberi A., and Yazdi H. S., 2008. An Eigenspace-Based Approach for Human Fall Detection Using Integrated Time Motion Image and Neural Network. *Signal Processing, 2008. ICSP 2008. 9th International Conference*, 26 – 29 Oct 2008 Beijing, China, pp. 1499 – 1503.

Guangqi Ye, J. J. Corso, and G. D. Hager, 2004. Gesture Recognition Using 3D Appearance and Motion Features. *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04*, 27 – 02 June 2004, pp. 160.

Guangyi Shi, Cheung Shing Chan, Wen Jung Li, and Kwok-Sui Leung, 2009. Mobile human airbag system for fall protection using MEMS sensors and embedded SVM classifier. *Sensors Journal, IEEE (Volume: 9, Issue: 5)*, pp. 495 – 503.

Huafen Luo, Bingtuan Gao, Jing Xu, and Ken Chen, 2013. An approach for structured light system calibration. *Cyber Technology in Automation, Control and Intelligent Systems (CYBER), 2013 IEEE 3rd Annual International Conference*, 26 – 29 May 2013 Nanjing, China, pp. 428 – 433.

Jansen B. and Deklerck, 2006.Context Aware Inactivity Recognition for Visual Fall Detection. *Pervasive Health Conference and Workshops, 2006*, 29 – 1 Dec 2006, Innsbruck, pp. 1 – 4

John MacCormick, *How does the Kinect work?* [Online]. Available at: http://www.cs.bham.ac.uk/~vvk201/Teach/Graphics/kinect.pdf [Accessed: 1 March 2015]

Junn Min Pang, Vooi Voon Yap, and Chit Siang Soh, 2014. Human Behavioral Analytics System For Video Surveillance. *Control System, Computing and Engineering (ICCSCE), 2014 IEEE International Conference*, 28 – 30 Nov 2014 Batu Rerringhi, Penang, pp. 23 – 28.

Kangas M., Konttila A., Winblad I., and Jamsa T., 2007. Determination of Simple Thresholds for Accelerometry-Based Parameters for Fall Detection. *29th IEEE Annual International Conference on Engineering in Medicine and Biology Society (EMBS)*, 22 - 26 Aug 2007, Lyon, pp. 1367 – 1370.

Kawatsu C, Li J, Chung C, 2012. Development of a fall detection system with microsoft kinect. *Kim JH, Matson ET, Myung H, Xu P (eds) Robot Intelligence Technology and Applications 2012*, Springer Berlin Heidelberg, Advances in Intelligent Systems and Computing, vol 208

Kourosh Khoshelham and Sander Oude Elberink, 2011. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors 2012, 12(2)*, pp. 1437-1454.

Lai Chin-Feng, Sung-Yen Chang, Han-Chieh Chao, and Yueh-Min Huang, 2010.Detection of Cognitive Injured Body Region Using Multiple Triaxial Accelerometers for Elderly Falling. *Sensors Journal, IEEE (Volume: 11, Issue: 3),* pp. 763 – 770.

Lars Bretzner, Ivan Laptev, and Tony Lindeberg, 2002. Hand Gesture Recognition using Multi-Scale Colour Features, Hierarchical Models and Particle Filtering. *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference*, 12 – 21 May 2002, Washington, DC, USA, pp. 423-428.

Laura Auria, Rouslan A. Moro, 2008. *Support Vector Machines (SVM) as a Technique for Solvency Analysis.* Discussion Papers, DIW Berlin, August 2008.

M. Mubashir, L. Shao, and L. Seed, 2012. A survey on fall detection: Principles and approaches. *Neurocomputing*, vol. 100, pp. 144–152.

Mahmoud Elmezain, Ayoub Al-Hamadi, Omer Rashid and Bernd Michaelis, 2009. *Posture and Gesture Recognition for Human-Computer Interaction*. In: Advanced Technologies, Kankesu Jayanthakumaran (Ed.), ISBN: 978-953-307-009-4, InTech, DOI: 10.5772/8221.

Marcos ZUNIGA, 2009. *Incremental Learning of Events in Video using Reliable Information*. PhD thesis, Universite Nice Sophia Antipolis.

MATHIE, M. J., COSTER, A. C. E, LOVELL, N. H., and CELLER, B. G., 2004. Accelerometry: providing an integrated, practical method for long-term, ambulatory monitoring of human movement. *Physiol. Neas*., 25, pp. R1-R20.

Matthias Rehm, Nikolaus Bee, and Elisabeth Andre, 2008. Wave Like an Egyptian — Accelerometer Based Gesture. *BCS-HCI '08 Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction - Volume 1*, British Computer Society Swinton, UK, UK pp. 13 – 22.

Max Weber, 1991.The Nature of Social Action in Runciman. *W.G. 'Weber: Selections in Translation' Cambridge University Press*, p7.

Mohamed B´echa KAˆA NICHE, 2009. *Human Gesture Recognition*. PhD thesis, Universite Nice Sophia Antipolis.

N. Dalal and B. Triggs, 2005. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference*, 25 June 2005 San Diego, CA, USA, pp 886 – 893 vol. 1.

Neeti A. Ogale, 2006. *A survey of techniques for human detection from video*. Master thesis, University of Maryland.

Noury, N., Barralon, P., Virone, G., Boissy, P., Hamel, M. & Rumeau, P., 2003, A smart sensor based on rules and its evaluation in daily routines. *Engineering in Medicine and Biology Society, 2003. Proceedings of the 25th Annual International Conference of the IEEE*, Vol. 4, pp. 3286–3289.

P. F. Felzenszwalb and D. P. Huttenlocher, 2005. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79.

Pavlovic V. I., Sharma R., and Huang T. S., 1997. Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume: 19, Issue: 7), pp. 677 – 695

Pedestrian detection, 2015 [Online]. Availabe at: http://en.wikipedia.org/wiki/Pedestrian_detection [Accessed: 1 March 2015]

Pengyu Hong, Matthew Turk, and Thomas S. Huang, 2000. Gesture modeling and recognition using finite state machines. *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference*, 28 -30 March, Grenoble, pp. 410 – 415.

Ramanan Navaratnam, Arasanathan Thayananthan, Philip H. Torr, Roberto Cipolla, 2005. Hierarchical part-based human body pose estimation, *Proceedings of the British Machine Vision Conference (BMVC'05)*, Sept 2005, Oxford, United Kingdom.

Ratanamahatana, C.A. & Keogh, E., 2004. Everything You Know about Dynamic Time Warping is Wrong. *Third Workshop on Mining Temporal and Sequential Data, in conjunction with the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, Seattle, WA.

Rey Rivera, 1996, *Strengths and weaknesses of hidden Markov models* [Online]. Available at: http://compbio.soe.ucsc.edu/html_format_papers/tr-94-24/node11.html [Accessed: 1 March 2015]

RGB "Bayer" Color and MicroLenses [Online]. Available at: http://www.siliconimaging.com/RGB %20Bayer.htm

Rougier, C., Meunier, J., 2005. Demo: Fall Detection Using 3D Head Trajectory Extracted From a Single Camera Video Sequence. *Journal of Telemedicine and Telecare 11(4)*.

S. Srinivasan, J. Han, D. Lal, A. Gacic, 2007. Towards Automatic Detection of Falls Using Wireless Sensors. *29th IEEE Annual International Conference on Engineering in Medicine and Biology Society (EMBS)*, 22 - 26 Aug 2007, Lyon, pp. 1379–1382.

Shahriar Nirjon, Chris Greenwood, Carlos Torres, Stefanie Zhou, and John A. Stankovic, 2014. Kintense: A Robust, Accurate, Real-Time and Evolving System for Detecting Aggressive Actions from Streaming 3D Skeleton Data. *IEEE Pervasive Computing and Communications (PerCom)*, 24-28 March, Budapest, pp. 2 - 10.

Shaou-Gang Miaou, Pei-Hsu Sung, and Chia-Yuan Huang, 2006. A Customized Human Fall Detection System Using Omni-Camera Images and Personal Information. *Distributed Diagnosis and Home Healthcare, 2006. D2H2. 1st Transdisciplinary Conference*, 2 – 4 April 2006, Arlington, VA, pp. 39 – 42.

Shifeng Li, Huchuan Lu, Xiang Ruan, and Yen-wei Chen, 2011. Pose estimation and body segmentation based on hierarchical searching tree. *Image Processing (ICIP), 2011 18th IEEE International Conference*, 11 – 14 Sept 2011, Brussels, pp. 1289 – 1292.

Shotton J., Fitzgibbon A., Cook M., and Sharp T., 2011. Real-Time Human Pose Recognition in Parts from Single Depth Images. *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference*, 20 – 25 June 2011 Providence, RI, pp. 1297 – 1304.

Sixsmith A. and Johnson N., 2004. A smart sensor to detect the falls of the elderly. *Pervasive Computing, IEEE (Volume: 3, Issue: 2)*, pp. 42 – 47.

Tamura T., Yoshimura T. Sekine Masaki, and Uchida M., 2009 .A Wearable Airbag to Prevent Fall Injuries. *Information Technology in Biomedicine, IEEE Transactions on (Volume: 13, Issue: 6)*, pp. 910 – 914.

Thome N. and Miguet S, 2006. A HHMM-Based Approach for Robust Fall Detection. *Control, Automation, Robotics and Vision, 2006. ICARCV '06. 9th International Conference*, 5 – 8 Dec 2006 Singapore, pp. 1 – 8.

Tong Jia, ZhongXuan Zhou, and HaiHong Gao, 2014. Depth Measurement Based on Infrared Coded Structured Light. *Journal of Sensors Volume 2014*, Article ID 852621, 8 pages.

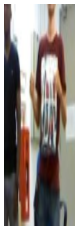
Vladimir I. Pavlovic, Rajeev Sharma, and Thomas S. Huang, 1997. Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume: 19, Issue: 7)*, pp. 677 – 695

Youngbum Lee, Jinkwon Kim, Muntak Son, and Myoungho Lee, 2007. Implementation of Accelerometer Sensor Module and Fall Detection Monitoring System based on Wireless Sensor Network. *29th IEEE Annual International Conference on Engineering in Medicine and Biology Society (EMBS)*, 22 - 26 Aug 2007, Lyon, pp. 2315–2318.

APPENDIX A

Experiment Subjects





APPENDIX B

Publication

Junn Min Pang, Vooi Voon Yap, and Chit Siang Soh, 2014. Human Behavioral Analytics System For Video Surveillance. *Control System, Computing and Engineering (ICCSCE), 2014 IEEE International Conference*, 28 – 30 Nov 2014 Batu Rerringhi, Penang, pp. 23 – 28.

Link for the paper:
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7072683&filter%3DAND(p_IS_Number%3A7072673)

Human Behavioral Analytics System For Video Surveillance

*Junn Min Pang, Vooi Voon Yap, Chit Siang Soh
1 Faculty of Engineering and Green Technology
2 Universiti Tunku Abdul Rahman
3 Kampar, Malaysia
4 *jmpang@outlook.com

***Abstract*—Video surveillance system is an important system to monitor safety and security for public place. Unfortunately, most of them do not have additional function. In this paper, we have proposed an algorithm for detect human aggressive behavior by using Microsoft Kinect sensor. In our proposed algorithm, Cartesian coordinate formula is used for calculate position of body joints detected by Microsoft Kinect sensor. By analysis position of body joints, we can differentiate whether it is a normal or aggressive behavior. Experiment results show that our proposed method has an average accuracy of 95.83% for punching pose, and almost perfectly detected for kicking and normal behavior.**

***Index Terms*—Human basic behavior analysis, human aggressive behavior analysis, fall detection, kinect**

I. INTRODUCTION

Video surveillance system is commonly used to monitor safety and security at public places such as bank, shopping mall and train station. Traditional surveillance technology requires manpower to monitor multiple CCTV screens. While more advance systems can detect and alert the operator on events that cause potential security risks, they have limited functionalities. For example, typical features in advanced surveillance systems include motion detection for intrusion detection, and possibly some trajectory analysis in structured environments.

There are numerous techniques employed to detect human body and motion from video (background subtraction, optical flow, depth map [1,2,3]).In the context of human behavior recognition, we focus on fast detection of aggressive behavior in any kind of environment. Earlier work done can only detect one or two persons in in-door environment with static background [1]. In addition to using video data, a multi-modal based smart surveillance system named CASSANDRA combined audio and video sensor to detect aggressive behavior in public environments [2]. Sergio Escalera proposed Pose recovery (PR) as the first step for Human Behavior Analysis (HBA) [3]. In this method, Microsoft Kinect sensor is used to obtain RGBDT (Red, Green, Blue, Depth and time) data in order to perform several PR method (Random Forest, RF Graph Cuts, Multiscale scanning window, Gabor filter and etc). The justification for using this approach is due to the difficulty to detect events related to the build-up or enactment of aggressive behavior by a single sensor data. Therefore, with this understanding, the method reported in this paper not only combines audio and video sensing but also includes human speech recognition system for more accurate detection of human aggressive behavior. Unlike previous works, the system reported in this paper can also detect specific words used by people involved in verbal argument.

Recently, much research has been carried out to detect human falling using image sensor. One of the basic methods is to draw a bounding box around the person in a single image and subsequently analyze it [4]. Unfortunately, this method will fail if the camera is not placed sideways or occluded by other objects. Other method used is by combining Motion History Image (MHI) and change in the human shape [5]. In this approach, all large motion of human will be detected in the first stage. After a motion is detected, shape of the human will be analyzed to detect whether it is a falling motion or a normal motion. This method will fail when a person falls and subsequently shows large motion behavior due to his or her injury.

The latest system developed by Kawatsu *et al*. for human fall detection utilized Microsoft Kinect sensor [6]. In this method, the authors used 20 body joints provided by the Microsoft Kinect sensor to calculate the velocity of the body and the position of all the joints. This is to analyze and to determine whether the motion is a fall. The system developed in this paper also adopted the Kinect sensor but it is slightly different from the method employed by Kawatsu *et al.* [6]. Our system analyzes a few body joints rather than all 20 body joints and hence it is expected to be less computationally intensive. In addition, it includes speech recognition function to detect someone calling for help.

Since Kinect sensor provides a synchronized RGBD (Red, Green, Blue, and Depth) images, it can be used for many functions (such as Object Tracking & recognition, Human Activity Analysis, hand Gesture Analysis and Indoor 3D Mapping) [7]. The Kinect Sensor contains an RGB camera, depth sensor (infrared sensor) and 4 microphone arrays. Various applications can be implemented using the software tool that is made available. Our system is able to detect human basic behaviors (such as running, walking, sitting and etc), human falling and human aggressive behaviors (such as

APPENDIX C

Code for Porposed Behavioral Analytics System

```
public MainWindow()
{
    InitializeComponent();
        this.energyBitmap = new WriteableBitmap(EnergyBitmapWidth, EnergyBitmapHeight, 96, 96,
        PixelFormats.Indexed1, new BitmapPalette(new List<Color> { Colors.White,
        (Color)this.Resources["KinectPurpleColor"] }));
}


/// <summary>
/// Execute startup tasks
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void WindowLoaded(object sender, RoutedEventArgs e)
{
 // Create the drawing group we'll use for drawing
 this.drawingGroup = new DrawingGroup();

 // Create an image source that we can use in our image control
 this.imageSource = new DrawingImage(this.drawingGroup);
  Image.Source = this.imageSource;

 // Look through all sensors and start the first connected one.
 // This requires that a Kinect is connected at the time of app startup.
 // To make your app robust against plug/unplug,
  // it is recommended to use KinectSensorChooser provided in Microsoft.Kinect.Toolkit (See
 components in Toolkit Browser).
 foreach (var potentialSensor in KinectSensor.KinectSensors)
 {
  if (potentialSensor.Status == KinectStatus.Connected)
  {
   this.sensor = potentialSensor;
   break;
  }
```

```
}

if (null != this.sensor)
{

 // Turn on the color stream to receive color frames
            this.sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);

// Allocate space to put the pixels we'll receive
 this.colorPixels = new byte[this.sensor.ColorStream.FramePixelDataLength];

// This is the bitmap we'll display on-screen
 this.colorBitmap = new WriteableBitmap(this.sensor.ColorStream.FrameWidth,
 this.sensor.ColorStream.FrameHeight, 96.0, 96.0, PixelFormats.Bgr32, null);

this.image1.Source = this.colorBitmap;

 // Add an event handler to be called whenever there is new color frame data
this.sensor.SkeletonFrameReady += this.SensorSkeletonFrameReady;

// Turn on the skeleton stream to receive skeleton frames
this.sensor.SkeletonStream.Enable();

 // Add an event handler to be called whenever there is new color frame data
this.sensor.ColorFrameReady += this.SensorColorFrameReady;

 // Start the sensor!
 try
 {
  this.sensor.Start();
 }
 catch (IOException)
 {
  this.sensor = null;
 }
 //sensor_angle = sensor.ElevationAngle;
}

if (null == this.sensor)
{
 this.statusBarText.Text = Properties.Resources.NoKinectReady;
}

// Initialize foreground pixels
this.foregroundPixels = new byte[EnergyBitmapHeight];
for (int i = 0; i < this.foregroundPixels.Length; ++i)
{
 this.foregroundPixels[i] = 0xff;
```

```
}

this.waveDisplay.Source = this.energyBitmap;

CompositionTarget.Rendering += UpdateEnergy;

 this.sensor.AudioSource.BeamAngleChanged += this.AudioSourceBeamChanged;
 this.sensor.AudioSource.SoundSourceAngleChanged +=
 this.AudioSourceSoundSourceAngleChanged;

// Start streaming audio!
this.audioStream = this.sensor.AudioSource.Start();

 // Use a separate thread for capturing audio because audio stream read operations
// will block, and we don't want to block main UI thread.
this.reading = true;
this.readingThread = new Thread(AudioReadingThread);
this.readingThread.Start();

this.RecognizeSpeechAndWriteToConsole();
sensor.ElevationAngle = 0;
sensor_level.Text = sensor.ElevationAngle.ToString();
}

/// <summary>
/// Execute shutdown tasks
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void WindowClosing(object sender, System.ComponentModel.CancelEventArgs e)
{
   // Tell audio reading thread to stop and wait for it to finish.
 this.reading = false;
 if (null != readingThread)
 {
  readingThread.Join();
 }

   if (null != this.sensor)
   {
    CompositionTarget.Rendering -= UpdateEnergy;

        this.sensor.AudioSource.BeamAngleChanged -= this.AudioSourceBeamChanged;
        this.sensor.AudioSource.SoundSourceAngleChanged -=
        this.AudioSourceSoundSourceAngleChanged;
    this.sensor.AudioSource.Stop();

    this.sensor.Stop();
```

```
    this.sensor = null;
   }
  }

    /// <summary>
    /// Event handler for Kinect sensor's SkeletonFrameReady event
    /// </summary>
    /// <param name="sender">object sending the event</param>
    /// <param name="e">event arguments</param>
    private void SensorSkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
    {
        Skeleton[] skeletons = new Skeleton[0];
        using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
        {
            if (skeletonFrame != null)
            {
                skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
                skeletonFrame.CopySkeletonDataTo(skeletons);
            }
        }

        using (DrawingContext dc = this.drawingGroup.Open())
        {
            // Draw a transparent background to set the render size
            dc.DrawRectangle(Brushes.Black, null, new Rect(0.0, 0.0, RenderWidth, RenderHeight));

            if (skeletons.Length != 0)
            {
                if (human_check ==1)
                    update2 = true;
                foreach (Skeleton skel in skeletons)
                {
                    RenderClippedEdges(skel, dc);
                    if (first == 0) first = skel.TrackingId;
                    else if (first != 0 && first != skel.TrackingId && second == 0) second =
skel.TrackingId;
                    else if ((first != skel.TrackingId && second != skel.TrackingId ))
                    {
                        cflag = true;
                        if (fflag == true)
                        {
                            fflag = false;
                            cflag = false;
                            sflag = false;
                            second = skel.TrackingId;
                            //update2 = true;
                            //human_check = 2;
                        }
```

```
                    else if (sflag == true)
                    {
                        sflag = false;
                        cflag = false;
                        fflag = false;
                        first = skel.TrackingId;
                        update1 = true;
                        //update2 = true;
                        //human_check = 1;
                        //cycle2 = 0;
                    }
                }
                if (first == skel.TrackingId && cflag == true)
                    fflag = true;

                else if (second == skel.TrackingId && cflag == true)
                    sflag = true;

                    if (skel.TrackingState == SkeletonTrackingState.Tracked)
                    {
                        this.DrawBonesAndJoints(skel, dc);
                        using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
                        {
                            this.DisplaySkelPosition(skel, first, second, skel.TrackingId,
skeletonFrame,human_check);
                        }
                    }
                    else if (skel.TrackingState == SkeletonTrackingState.PositionOnly)
                    {
                        dc.DrawEllipse(
                        this.centerPointBrush,
                        null,
                        this.SkeletonPointToScreen(skel.Position),
                        BodyCenterThickness,
                        BodyCenterThickness);
                    }
                }
            }

            // prevent drawing outside of our render area
            this.drawingGroup.ClipGeometry = new RectangleGeometry(new Rect(0.0, 0.0,
RenderWidth, RenderHeight));

            //this.DisplaySkelPosition(skeletons[0]);
        }
    }

    /// <summary>
```

```
        /// Draws a skeleton's bones and joints
        /// </summary>
        /// <param name="skeleton">skeleton to draw</param>
        /// <param name="drawingContext">drawing context to draw to</param>
        private void DrawBonesAndJoints(Skeleton skeleton, DrawingContext drawingContext)
        {
            // Render Torso
            this.DrawBone(skeleton, drawingContext, JointType.Head, JointType.ShoulderCenter);
            this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter, JointType.ShoulderLeft);
            this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter,
JointType.ShoulderRight);
            this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter, JointType.Spine);
            this.DrawBone(skeleton, drawingContext, JointType.Spine, JointType.HipCenter);
            this.DrawBone(skeleton, drawingContext, JointType.HipCenter, JointType.HipLeft);
            this.DrawBone(skeleton, drawingContext, JointType.HipCenter, JointType.HipRight);

            // Left Arm
            this.DrawBone(skeleton, drawingContext, JointType.ShoulderLeft, JointType.ElbowLeft);
            this.DrawBone(skeleton, drawingContext, JointType.ElbowLeft, JointType.WristLeft);
            this.DrawBone(skeleton, drawingContext, JointType.WristLeft, JointType.HandLeft);

            // Right Arm
            this.DrawBone(skeleton, drawingContext, JointType.ShoulderRight, JointType.ElbowRight);
            this.DrawBone(skeleton, drawingContext, JointType.ElbowRight, JointType.WristRight);
            this.DrawBone(skeleton, drawingContext, JointType.WristRight, JointType.HandRight);

            // Left Leg
            this.DrawBone(skeleton, drawingContext, JointType.HipLeft, JointType.KneeLeft);
            this.DrawBone(skeleton, drawingContext, JointType.KneeLeft, JointType.AnkleLeft);
            this.DrawBone(skeleton, drawingContext, JointType.AnkleLeft, JointType.FootLeft);

            // Right Leg
            this.DrawBone(skeleton, drawingContext, JointType.HipRight, JointType.KneeRight);
            this.DrawBone(skeleton, drawingContext, JointType.KneeRight, JointType.AnkleRight);
            this.DrawBone(skeleton, drawingContext, JointType.AnkleRight, JointType.FootRight);

            //this.DisplaySkelPosition(skeleton);

            // Render Joints
            foreach (Joint joint in skeleton.Joints)
            {
                Brush drawBrush = null;

                if (joint.TrackingState == JointTrackingState.Tracked)
                {
                    drawBrush = this.trackedJointBrush;
                }
                else if (joint.TrackingState == JointTrackingState.Inferred)
```

```
            {
                drawBrush = this.inferredJointBrush;
            }

            if (drawBrush != null)
            {
                drawingContext.DrawEllipse(drawBrush, null, this.SkeletonPointToScreen(joint.Position),
JointThickness, JointThickness);
            }
        }
    }

    /// <summary>
    /// Maps a SkeletonPoint to lie within our render space and converts to Point
    /// </summary>
    /// <param name="skelpoint">point to map</param>
    /// <returns>mapped point</returns>
    private Point SkeletonPointToScreen(SkeletonPoint skelpoint)
    {
        // Convert point to depth space.
        // We are not using depth directly, but we do want the points in our 640x480 output resolution.
        DepthImagePoint depthPoint =
this.sensor.CoordinateMapper.MapSkeletonPointToDepthPoint(skelpoint,
DepthImageFormat.Resolution640x480Fps30);
        return new Point(depthPoint.X, depthPoint.Y);
    }

    /// <summary>
    /// Draws a bone line between two joints
    /// </summary>
    /// <param name="skeleton">skeleton to draw bones from</param>
    /// <param name="drawingContext">drawing context to draw to</param>
    /// <param name="jointType0">joint to start drawing from</param>
    /// <param name="jointType1">joint to end drawing at</param>
    private void DrawBone(Skeleton skeleton, DrawingContext drawingContext, JointType
jointType0, JointType jointType1)
    {
        Joint joint0 = skeleton.Joints[jointType0];
        Joint joint1 = skeleton.Joints[jointType1];

        // If we can't find either of these joints, exit
        if (joint0.TrackingState == JointTrackingState.NotTracked ||
            joint1.TrackingState == JointTrackingState.NotTracked)
        {
            return;
        }

        // Don't draw if both points are inferred
```

```
            if (joint0.TrackingState == JointTrackingState.Inferred &&
                joint1.TrackingState == JointTrackingState.Inferred)
            {
                return;
            }

            // We assume all drawn bones are inferred unless BOTH joints are tracked
            Pen drawPen = this.inferredBonePen;
            if (joint0.TrackingState == JointTrackingState.Tracked && joint1.TrackingState ==
JointTrackingState.Tracked)
            {
                drawPen = this.trackedBonePen;
            }

            drawingContext.DrawLine(drawPen, this.SkeletonPointToScreen(joint0.Position),
            this.SkeletonPointToScreen(joint1.Position));
        }

/// <summary>
/// Display human skeleton data
/// </summary>
/// <param name="skeleton"></param>
/// <param name="first"></param>
/// <param name="second"></param>
/// <param name="skelcount"></param>
/// <param name="skeletonframe"></param>
/// <param name="skeleton_length"></param>
private void DisplaySkelPosition(Skeleton skeleton , int first, int second, int skelcount, SkeletonFrame
skeletonframe, int skeleton_length)
{

double  degree = (sensor.ElevationAngle/360) *(2*Math.PI);

if (reset_flag)// reset for first user
{
  stopwatch.Restart();
  reset_flag = false;
  ftotal_velocity = 0;
  Lhand_total1 = 0;
  Rhand_total1 = 0;
  Lleg_total = 0;
  Rleg_total = 0;
}

if (reset2_flag || (human_check == 1))//reset for second user
{
  stopwatch2.Restart();
  reset2_flag = false;
```

```
    stotal_velocity = 0;
    Lhand_total2 = 0;
    Rhand_total2 = 0;
    Lleg_total2 = 0;
    Rleg_total2 = 0;
    sskeleton = new Skeleton [N];
    cycle2 = 0;
}

double A = 0;
double B = Math.Cos(degree);
double C = Math.Sin(degree);
double D = 1;



double jointx1 = A * skeleton.Joints[JointType.Head].Position.X;
double jointy1 = B * skeleton.Joints[JointType.Head].Position.Y;
double jointz1 = C * skeleton.Joints[JointType.Head].Position.Z;

double jointx2 = A * skeleton.Joints[JointType.FootLeft].Position.X;
double jointy2 = B * skeleton.Joints[JointType.FootLeft].Position.Y;
double jointz2 = C * skeleton.Joints[JointType.FootLeft].Position.Z;

double jointx3 = A * skeleton.Joints[JointType.HipCenter].Position.X;
double jointy3 = B * skeleton.Joints[JointType.HipCenter].Position.Y;
double jointz3 = C * skeleton.Joints[JointType.HipCenter].Position.Z;

double jointx4 = A * skeleton.Joints[JointType.FootRight].Position.X;
double jointy4 = B * skeleton.Joints[JointType.FootRight].Position.Y;
double jointz4 = C * skeleton.Joints[JointType.FootRight].Position.Z;

double jointx5 = A * skeleton.Joints[JointType.KneeLeft].Position.X;
double jointy5 = B * skeleton.Joints[JointType.KneeLeft].Position.Y;
double jointz5 = C * skeleton.Joints[JointType.KneeLeft].Position.Z;

double jointx6 = A * skeleton.Joints[JointType.KneeRight].Position.X;
double jointy6 = B * skeleton.Joints[JointType.KneeRight].Position.Y;
double jointz6 = C * skeleton.Joints[JointType.KneeRight].Position.Z;



double jointx7 = A * skeleton.Joints[JointType.Spine].Position.X;
double jointy7 = B * skeleton.Joints[JointType.Spine].Position.Y;
double jointz7 = C * skeleton.Joints[JointType.Spine].Position.Z;

double ftotal1 = jointx1 + jointy1 + jointz1 + D;
double ftotal2 = jointx2 + jointy2 + jointz2 + D;
double ftotal3 = jointx3 + jointy3 + jointz3 + D;
```

```
double ftotal4 = jointx4 + jointy4 + jointz4 + D;
double ftotal5 = jointx5 + jointy5 + jointz5 + D;
double ftotal6 = jointx6 + jointy6 + jointz6 + D;
double ftotal7 = jointx7 + jointy7 + jointz7 + D;
double ftotal_d = A * A + B * B + C * C;

double head_distance = ftotal1 / (float)System.Math.Sqrt(ftotal_d);
double footL_distance = ftotal2 / (float)System.Math.Sqrt(ftotal_d);
double hip_distance = ftotal3 / (float)System.Math.Sqrt(ftotal_d);
double footR_distance = ftotal4 / (float)System.Math.Sqrt(ftotal_d);
double kneeL_distance = ftotal5 / (float)System.Math.Sqrt(ftotal_d);
double kneeR_distance = ftotal6 / (float)System.Math.Sqrt(ftotal_d);
double spine_distance = ftotal7 / (float)System.Math.Sqrt(ftotal_d);

if (skeleton.TrackingId == first)
{
  human_count1 = true;
  human_stay += 1;
  if (human_stay > 2)
  {
    human_count2 = false;
    human_stay = 0;
  }
}
else if (skeleton.TrackingId == second)
{
  human_count2 = true;
  if (human_stay == 1)
    human_stay = 0;
}

if (human_count1 && !human_count2)//check whether there are one or two person
  human_check = 1;
else if (human_count1 && human_count2)
  human_check = 2;

if(skeleton.Joints[JointType.HandLeft].Position.Y > (skeleton.Joints[JointType.Spine].Position.Y +
0.2))//detect hand over spine position
  Lhand_up = true;
else
  Lhand_up = false;

if (skeleton.Joints[JointType.HandRight].Position.Y > (skeleton.Joints[JointType.Spine].Position.Y +
0.2))//detect spine over hip position
  Rhand_up = true;
else
  Rhand_up = false;
```

```
if (verbal_aggresive)
{
  verbal_timer += 1;
  textBox3.Text = "Aggressive Verbal Detected";
}
if (verbal_timer >= 150)
{
  verbal_timer = 0;
  verbal_aggresive = false;
}

if (first == skelcount)/////////first user start here
{
  if (head_distance <= 0.65|| spine_distance <= 0.45)// fall detection
    fallflag1 = true;
  else
  {
    fallflag1 = false;
    fallflag2 = false;
  }
sit_crouching_detect(head_distance,fheight, hip_distance, footL_distance,footR_distance);
fskeleton[cycle1] = skeleton;

if (cycle2 > 1)
{
  touching_check(skeleton, sskeleton[cycle2 - 1]);
}
else
{
  body_contact_flag = false;
  stick_flag = false;
  leg_touch = false;
}

if (cycle1 >= 1)//calculate velocity for few point of human skeleton
{
  ftotal_velocity += 1000 *
  Math.Sqrt(Math.Pow(fskeleton[cycle1].Joints[JointType.ShoulderCenter].Position.X -
  fskeleton[cycle1 - 1].Joints[JointType.ShoulderCenter].Position.X, 2)
   + Math.Pow(fskeleton[cycle1].Joints[JointType.ShoulderCenter].Position.Z - fskeleton[cycle1 -
  1].Joints[JointType.ShoulderCenter].Position.Z, 2));

  jump_velocity1 += 1000 * (fskeleton[cycle1].Joints[JointType.ShoulderCenter].Position.Y -
  fskeleton[cycle1 - 1].Joints[JointType.ShoulderCenter].Position.Y);

  Lhand_total1 += 1000 *
  Math.Sqrt(Math.Pow(fskeleton[cycle1].Joints[JointType.HandLeft].Position.X - fskeleton[cycle1 -
  1].Joints[JointType.HandLeft].Position.X, 2)
```

```
    + Math.Pow(fskeleton[cycle1].Joints[JointType.HandLeft].Position.Y - fskeleton[cycle1 -
    1].Joints[JointType.HandLeft].Position.Y, 2)
    + Math.Pow(fskeleton[cycle1].Joints[JointType.HandLeft].Position.Z - fskeleton[cycle1 -
    1].Joints[JointType.HandLeft].Position.Z, 2));

    Rhand_total1 += 1000 *
    Math.Sqrt(Math.Pow(fskeleton[cycle1].Joints[JointType.HandRight].Position.X - fskeleton[cycle1 -
    1].Joints[JointType.HandRight].Position.X, 2)
    + Math.Pow(fskeleton[cycle1].Joints[JointType.HandRight].Position.Y - fskeleton[cycle1 -
    1].Joints[JointType.HandRight].Position.Y, 2)
    + Math.Pow(fskeleton[cycle1].Joints[JointType.HandRight].Position.Z - fskeleton[cycle1 -
    1].Joints[JointType.HandRight].Position.Z, 2));

    Lleg_total += 1000 * Math.Sqrt(Math.Pow(fskeleton[cycle1].Joints[JointType.FootLeft].Position.X -
    fskeleton[cycle1 - 1].Joints[JointType.FootLeft].Position.X, 2)
    + Math.Pow(fskeleton[cycle1].Joints[JointType.FootLeft].Position.Y - fskeleton[cycle1 -
    1].Joints[JointType.FootLeft].Position.Y, 2)
    + Math.Pow(fskeleton[cycle1].Joints[JointType.FootLeft].Position.Z - fskeleton[cycle1 -
    1].Joints[JointType.FootLeft].Position.Z, 2));

    Rleg_total += 1000 * Math.Sqrt(Math.Pow(fskeleton[cycle1].Joints[JointType.FootRight].Position.X
    - fskeleton[cycle1 - 1].Joints[JointType.FootRight].Position.X, 2)
    + Math.Pow(fskeleton[cycle1].Joints[JointType.FootRight].Position.Y - fskeleton[cycle1 -
    1].Joints[JointType.FootRight].Position.Y, 2)
    + Math.Pow(fskeleton[cycle1].Joints[JointType.FootRight].Position.Z - fskeleton[cycle1 -
    1].Joints[JointType.FootRight].Position.Z, 2));

  }

  cycle1 += 1;
  if (cycle1 == N / 2 || cycle1 == N)
  {
    javg_velocity1 = jump_velocity1 / ((N / 2) - 1);
    jump_velocity1 = 0;
  }

  if (cycle1 == N)
  {
    stopwatch.Stop();
    reset_flag = true;
    favg_velocity = (ftotal_velocity/ (stopwatch.Elapsed.Milliseconds / N)) / (N - 1);
    Lhand_velocity1 = (Lhand_total1 / (stopwatch.Elapsed.Milliseconds / N)) / (N - 1);
    Rhand_velocity1 = (Rhand_total1 / (stopwatch.Elapsed.Milliseconds / N)) / (N - 1);
    Lleg_velocity1 = (Lleg_total / (stopwatch.Elapsed.Milliseconds / N)) / (N - 1);
    Rleg_velocity1 = (Rleg_total / (stopwatch.Elapsed.Milliseconds / N)) / (N - 1);

    cycle1 = 0;
    j += 1;
```

```
}

jump_check(javg_velocity1, kneeL_distance, kneeR_distance,fheight);//check whether user 1 is
jumping
walking_running_detect(favg_velocity, head_distance, footL_distance, footR_distance, fheight,
hip_distance );// walk and run detect for user 1
HABclassifier(Lleg_velocity1, footL_distance, Rleg_velocity1, footR_distance, Lhand_velocity1 ,
Rhand_velocity1);//human aggressive behavior classifier

if (body_contact_flag && sfighting_flag && fight_timer1 < 100)
{
  fighting_flag1 = true;
  fight_timer1 = 0;
}
if (fighting_flag1 && fight_timer1 > 100 && !fallflag1 && !fallflag2)
{
  fighting_flag1 = false;
  fight_timer1 = 0;
}

if (kicking_flag && leg_touch && kick_timer1 < 100)
{
  f_kicking = true;
  kick_timer1 = 0;
}
if (f_kicking && kick_timer1 > 100)
{
  f_kicking = false;
  kick_timer1 = 0;
}

if (jflag && jump_timer1 < 25)
{
  jump_timer1 = 0;
  jump_hold1 = true;
}
else if (jump_timer1 >= 25)
{
  jump_hold1 = false;
  jump_timer1 = 0;
}

///update user1 height
if (head_distance > fheight && !jump_hold1 && footL_distance < 0.33 || head_distance > fheight
&& !jump_hold1 && footR_distance < 0.33)//get the max height for user 1
{
  update1 = true;
}
```

```
if (update1)
{
  update1 = false;
  fheight = head_distance;
}


if (!fallflag1 && !fallflag2 && !jump_hold1 && !sit_flag && !crouching_flag && !running_flag
&& !walking_flag && !sfighting_flag && !kicking_flag && !body_contact_flag && !leg_touch && !
fighting_flag1 && !stick_flag && !surrender_flag && !f_kicking)
{
  timecount1 = 0;

if (cycle2 < 1)
{
if (Lhand_up & Rhand_up)
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  Lhand_velocity1 + "\n" + Rhand_velocity1 + "\n" + human_check+ "\n" + "Both Hand Up" + "\n" +
  hip_distance + "\n" + head_distance + "\n" + footL_distance + "\n" + kneeL_distance + "\n" +
  Lleg_velocity1;
else if (Lhand_up & !Rhand_up)
{
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  Lhand_velocity1 + "\n" + Rhand_velocity1 + "\n" + human_check+ "\n" + "Left Hand Up" + "\n" +
  hip_distance + "\n" + head_distance + "\n" + footL_distance + "\n" + kneeL_distance + "\n" +
  Lleg_velocity1;
}
else if (!Lhand_up & Rhand_up)
{
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  Lhand_velocity1 + "\n" + Rhand_velocity1 + "\n" + human_check+ "\n" + "Right Hand Up"  + "\n" +
  hip_distance + "\n" + head_distance + "\n" + footL_distance + "\n" + kneeL_distance + "\n" +
  Lleg_velocity1;
}
else
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  Lhand_velocity1 + "\n" + Rhand_velocity1 + "\n" + human_check+ "\n" +
  stopwatch.Elapsed.Milliseconds  + "\n" + hip_distance + "\n" + head_distance + "\n" +
  footL_distance + "\n" + kneeL_distance + "\n" + Lleg_velocity1;
}
else
{
if (Lhand_up & Rhand_up)
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  Lhand_velocity1 + "\n" + Rhand_velocity1 + "\n" + human_check+ "\n" + "Both Hand Up" + "\n" +
  skeleton.Joints[JointType.Spine].Position.X + "\n" + sskeleton[cycle2 -
  1].Joints[JointType.Spine].Position.X + "\n" + skeleton.Joints[JointType.Spine].Position.Z + "\n" +
  sskeleton[cycle2 - 1].Joints[JointType.Spine].Position.Z;
```

```
else if (Lhand_up & !Rhand_up)
{
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  Lhand_velocity1 + "\n" + Rhand_velocity1 + "\n" + human_check+ "\n" + "Left Hand Up" + "\n" +
  skeleton.Joints[JointType.Spine].Position.X + "\n" + sskeleton[cycle2 -
  1].Joints[JointType.Spine].Position.X + "\n" + skeleton.Joints[JointType.Spine].Position.Z + "\n" +
  sskeleton[cycle2 - 1].Joints[JointType.Spine].Position.Z;
}
else if (!Lhand_up & Rhand_up)
{
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  Lhand_velocity1 + "\n" + Rhand_velocity1 + "\n" + human_check
  + "\n" + "Right Hand Up" + "\n" + skeleton.Joints[JointType.Spine].Position.X + "\n" +
  sskeleton[cycle2 - 1].Joints[JointType.Spine].Position.X + "\n" +
  skeleton.Joints[JointType.Spine].Position.Z + "\n" + sskeleton[cycle2 -
  1].Joints[JointType.Spine].Position.Z;
}
else
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  Lhand_velocity1 + "\n" + Rhand_velocity1 + "\n" + human_check+ "\n" +
  stopwatch.Elapsed.Milliseconds + "\n" + skeleton.Joints[JointType.Spine].Position.X + "\n" +
  sskeleton[cycle2 - 1].Joints[JointType.Spine].Position.X + "\n" +
  skeleton.Joints[JointType.Spine].Position.Z + "\n" + sskeleton[cycle2 -
  1].Joints[JointType.Spine].Position.Z;
}
}
else if (fallflag1 && timecount1 < 150 && !fallflag2)
{
   textBox1.Text = first + "\n" + favg_velocity + "\n" + "suspect fall detected" + "\n" + timecount1 +
   "\n" + head_distance + "\n" + spine_distance;
   timecount1 += 1;
}
else if (fallflag1 && timecount1 >= 150 || fallflag2)
{
  textBox1.Text = first + "\n" + favg_velocity + "\n" + "fall detected" + "\n" + head_distance + "\n" +
  spine_distance;
}
else if (jump_hold1)
{
  timecount1 = 0;
  jump_timer1 += 1;
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  "jumping" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
            }
else if (sit_flag)
{
  timecount1 = 0;
```

```
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  "sitting" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
}
else if (crouching_flag && !surrender_flag)
{
  timecount1 = 0;
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  "crouching" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
}
else if (walking_flag && Math.Abs(javg_velocity1)< 6)
{
  timecount1 = 0;
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  "walking" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
}
else if (running_flag && Math.Abs(javg_velocity1) < 6)
{
  timecount1 = 0;
  textBox1.Text = first + "\n" + favg_velocity + "\n" + fheight + "\n" + javg_velocity1 + "\n" +
  "running" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
}
else if (fighting_flag1 && !fallflag1 && !fallflag2)
{
  textBox1.Text = "First is" + "\n" + "punching Second";
  fight_timer1 += 1;
}
else if (sfighting_flag && !body_contact_flag && !kicking_flag && !fallflag1 && !fallflag2)
{
  timecount1 = 0;
  textBox1.Text = "Suspect Aggressive Detected!!" + "\n" + "Alert!!!" + "\n" + Lhand_velocity1 + "\n"
  + Rhand_velocity1;
}
else if (body_contact_flag && !sfighting_flag)
{
  timecount1 = 0;
  if (cycle2 > 0)
  textBox1.Text = "first touched second" + "\n" + fskeleton[cycle1].Joints[JointType.Spine].Position.X
  + "\n" + sskeleton[cycle2 - 1].Joints[JointType.Spine].Position.X+ "\n" +
  fskeleton[cycle1].Joints[JointType.Spine].Position.Z + "\n" + sskeleton[cycle2 -
  1].Joints[JointType.Spine].Position.Z+ "\n" +
  fskeleton[cycle1].Joints[JointType.HandLeft].Position.X + "\n" +
  fskeleton[cycle1].Joints[JointType.HandLeft].Position.Z;
}
```

```
else if (kicking_flag && !leg_touch && !sit_flag && !crouching_flag && !f_kicking && fallflag1
&& fallflag2)
  textBox1.Text = " First is Kicking.";
else if (f_kicking && !sit_flag && !crouching_flag && !fallflag1 && !fallflag2)
{
  textBox1.Text = " first kicking"+ "\n"+ "second";
  kick_timer1 += 1;
}



}
else if (second == skelcount)///////second user start here
{
if (head_distance <= 0.65 || spine_distance < 0.45)// fall detection
  fallflag3 = true;
else
{
  fallflag3 = false;
  fallflag4 = false;
}
sit_crouching_detect(head_distance,sheight, hip_distance,footL_distance,footR_distance);

sskeleton[cycle2] = skeleton;

if (cycle1 > 1)
{
  touching_check(skeleton, fskeleton[cycle1 - 1]);
}
else
{
  body_contact_flag = false;
  leg_touch = false;
  stick_flag = false;
}
if (cycle2 >= 1)
            {
stotal_velocity += 1000 *
Math.Sqrt(Math.Pow(sskeleton[cycle2].Joints[JointType.ShoulderCenter].Position.X - sskeleton[cycle2
- 1].Joints[JointType.ShoulderCenter].Position.X, 2)
+ Math.Pow(sskeleton[cycle2].Joints[JointType.ShoulderCenter].Position.Z - sskeleton[cycle2 -
1].Joints[JointType.ShoulderCenter].Position.Z, 2));

jump_velocity2 += 1000 * (sskeleton[cycle2].Joints[JointType.ShoulderCenter].Position.Y -
sskeleton[cycle2 - 1].Joints[JointType.ShoulderCenter].Position.Y);

Lhand_total2 += 1000 * Math.Sqrt(Math.Pow(sskeleton[cycle2].Joints[JointType.HandLeft].Position.X
- sskeleton[cycle2 - 1].Joints[JointType.HandLeft].Position.X, 2)
```

```
+ Math.Pow(sskeleton[cycle2].Joints[JointType.HandLeft].Position.Y - sskeleton[cycle2 -
1].Joints[JointType.HandLeft].Position.Y, 2)
+ Math.Pow(sskeleton[cycle2].Joints[JointType.HandLeft].Position.Z - sskeleton[cycle2 -
1].Joints[JointType.HandLeft].Position.Z, 2));

Rhand_total2 += 1000 *
Math.Sqrt(Math.Pow(sskeleton[cycle2].Joints[JointType.HandRight].Position.X - sskeleton[cycle2 -
1].Joints[JointType.HandRight].Position.X, 2)
+ Math.Pow(sskeleton[cycle2].Joints[JointType.HandRight].Position.Y - sskeleton[cycle2 -
1].Joints[JointType.HandRight].Position.Y, 2)
+ Math.Pow(sskeleton[cycle2].Joints[JointType.HandRight].Position.Z - sskeleton[cycle2 -
1].Joints[JointType.HandRight].Position.Z, 2));

Lleg_total2 += 1000 * Math.Sqrt(Math.Pow(sskeleton[cycle2].Joints[JointType.FootLeft].Position.X -
sskeleton[cycle2 - 1].Joints[JointType.FootLeft].Position.X, 2)
+ Math.Pow(sskeleton[cycle2].Joints[JointType.FootLeft].Position.Y - sskeleton[cycle2 -
1].Joints[JointType.FootLeft].Position.Y, 2)
+ Math.Pow(sskeleton[cycle2].Joints[JointType.FootLeft].Position.Z - sskeleton[cycle2 -
1].Joints[JointType.FootLeft].Position.Z, 2));

Rleg_total2 += 1000 * Math.Sqrt(Math.Pow(sskeleton[cycle2].Joints[JointType.FootRight].Position.X
- sskeleton[cycle2 - 1].Joints[JointType.FootRight].Position.X, 2)
+ Math.Pow(sskeleton[cycle2].Joints[JointType.FootRight].Position.Y - sskeleton[cycle2 -
1].Joints[JointType.FootRight].Position.Y, 2)
+ Math.Pow(sskeleton[cycle2].Joints[JointType.FootRight].Position.Z - sskeleton[cycle2 -
1].Joints[JointType.FootRight].Position.Z, 2));
}

  cycle2 += 1;
if (cycle2 == N / 2 || cycle2 == N)
{
  javg_velocity2 = jump_velocity2 / ((N / 2) - 1);
  jump_velocity2 = 0;
}

if (cycle2 == N)
{
  stopwatch2.Stop();
  reset2_flag = true;
  savg_velocity = (stotal_velocity / (stopwatch2.Elapsed.Milliseconds / N)) / (N - 1);
  Lhand_velocity2 = (Lhand_total2 / (stopwatch2.Elapsed.Milliseconds / N)) / (N - 1);
  Rhand_velocity2 = (Rhand_total2 / (stopwatch2.Elapsed.Milliseconds / N)) / (N - 1);
  Lleg_velocity2 = (Lleg_total2 / (stopwatch2.Elapsed.Milliseconds / N)) / (N - 1);
  Rleg_velocity2 = (Rleg_total2 / (stopwatch2.Elapsed.Milliseconds / N)) / (N - 1);

  j += 1;
}
```

```
jump_check(javg_velocity2, kneeL_distance, kneeR_distance,sheight);//check whether user 2 is
jumping
walking_running_detect(savg_velocity, head_distance, footL_distance, footR_distance, sheight,
hip_distance);//detect whether user 2 is running or walking
HABclassifier(Lleg_velocity2, footL_distance, Rleg_velocity2, footR_distance, Lhand_velocity2,
Rhand_velocity2);// Human aggressive behavior analysis

if (body_contact_flag && sfighting_flag && fight_timer2 < 100 && !fallflag3 && !fallflag4)
{
  fighting_flag2 = true;
  fight_timer2 = 0;
}
if (fighting_flag2 && fight_timer2 > 100)
{
  fighting_flag2 = false;
  fight_timer2 = 0;
}

if (kicking_flag && leg_touch && kick_timer2 < 100)
{
  s_kicking = true;
  kick_timer2 = 0;
}
if (s_kicking && kick_timer2 > 100)
{
  s_kicking = false;
  kick_timer2 = 0;
}

if (jflag && jump_timer2 < 25)
{
  jump_timer2 = 0;
  jump_hold2 = true;
}
else if (jump_timer2 >= 25)
{
  jump_timer2 = 0;
  jump_hold2 = false;
}

///update user2 height
if (head_distance > sheight && !jump_hold2 && footL_distance < 0.33 || head_distance > sheight
&& !jump_hold2 && footR_distance < 0.33)
{
   update2 = true;
}

if (update2)
```

```
{
  sheight = head_distance;
  update2 = false;
}
if (!fallflag3 && !fallflag4 && !jump_hold2 && !sit_flag && !crouching_flag && !running_flag
&& !walking_flag && !kicking_flag && !body_contact_flag && !sfighting_flag && !leg_touch && !
stick_flag && !fighting_flag2 && !s_kicking)
{
timecount2 = 0;
if (cycle1 < 1)
{
if (Lhand_up & !Rhand_up)
 textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
 skeleton_length+ "\n" + "Left Hand Up" + "\n" + hip_distance + "\n" + Lhand_velocity2 + "\n" +
 Rhand_velocity2+ "\n" + kneeL_distance + "\n" + kneeR_distance;
else if (!Lhand_up & Rhand_up)
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  skeleton_length"\n" + "Right Hand Up" + "\n" + hip_distance + "\n" + Lhand_velocity2 + "\n" +
  Rhand_velocity2"\n" + kneeL_distance + "\n" + kneeR_distance;
else if (Lhand_up & Rhand_up)
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  skeleton_length"\n" + "Both Hand Up" + "\n" + hip_distance + "\n" + Lhand_velocity2 + "\n" +
  Rhand_velocity2+ "\n" + kneeL_distance + "\n" + kneeR_distance;
else
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  skeleton_length + "\n" + stopwatch2.Elapsed.Milliseconds + "\n" + hip_distance + "\n" +
  Lhand_velocity2 + "\n" + Rhand_velocity2"\n" + kneeL_distance + "\n" + kneeR_distance;
}
else
{
if (Lhand_up & !Rhand_up)
textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
skeleton_length"\n" + "Left Hand Up" +"\n"+ hip_distance + "\n" + Lhand_velocity2 + "\n" +
Rhand_velocity2+ "\n" + kneeL_distance + "\n" + kneeR_distance;
else if (!Lhand_up & Rhand_up)
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  skeleton_length+ "\n" + "Right Hand Up" + "\n" + hip_distance + "\n" + Lhand_velocity2 + "\n" +
  Rhand_velocity2+ "\n" + kneeL_distance + "\n" + kneeR_distance;
else if (Lhand_up & Rhand_up)
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  skeleton_length+ "\n" + "Both Hand Up" + "\n" + hip_distance + "\n" + Lhand_velocity2 + "\n" +
  Rhand_velocity2+ "\n" + kneeL_distance + "\n" + kneeR_distance;
else
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  skeleton_length + "\n" + stopwatch2.Elapsed.Milliseconds + "\n" + hip_distance + "\n" +
  Lhand_velocity2 + "\n" + Rhand_velocity2
  + "\n" + kneeL_distance + "\n" + kneeR_distance;
```

```
}
}
else if (fallflag3 && timecount2 < 150 && !fallflag4)
{
  textBox2.Text = second + "\n" + savg_velocity + "\n" + "suspect fall detected" + "\n" + timecount2  +
  "\n" + head_distance + "\n" + spine_distance;
  timecount2 += 1;
}
else if (fallflag3 && timecount2 >= 150 || fallflag4)
{
  textBox2.Text = second + "\n" + savg_velocity + "\n" + "fall detected" + "\n" + head_distance + "\n"
  + spine_distance;
}
else if (jump_hold2)
{
  jump_timer2 += 1;
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" + "\n"
  + "jumping" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance+ "\n" +
  kneeL_distance;
}
else if (sit_flag)
{
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  "sitting" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
}
else if (crouching_flag)
{
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  "crouching" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
}
else if (walking_flag && Math.Abs(javg_velocity2)< 6)
{
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  "walking" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
}
else if (running_flag && Math.Abs(javg_velocity2) < 6)
{
  textBox2.Text = second + "\n" + savg_velocity + "\n" + sheight + "\n" + javg_velocity2 + "\n" +
  "running" + "\n" + hip_distance + "\n" + head_distance + "\n" + kneeR_distance + "\n" +
  kneeL_distance;
}
else if (fighting_flag2 && !fallflag3 && !fallflag4)
{
  textBox2.Text = "Second is Punching";
  fight_timer2 += 1;
```

```
}
else if (sfighting_flag && !body_contact_flag && !kicking_flag && !fallflag3 && !fallflag4)
   textBox2.Text = "Suspect fight detect on second!";
else if (kicking_flag && !leg_touch && !sit_flag && !crouching_flag && !s_kicking && !fallflag3
&& !fallflag4)
  textBox2.Text = "Second is kicking";
else if (s_kicking && !sit_flag && !crouching_flag && !fallflag3 && !fallflag4)
{
   textBox2.Text = "Second kicking" + "\n" + "first!";
   kick_timer2 += 1;
}
else if (leg_touch && !kicking_flag)
  textBox2.Text = "Second leg touch first";
else if (body_contact_flag && !sfighting_flag && !fighting_flag2)
  textBox2.Text = "Second touching first";


}
}

/// <summary>
/// Recognize Speech with the word added
/// </summary>
private void RecognizeSpeechAndWriteToConsole()
{
    _recognizer = new SpeechRecognitionEngine();
    _recognizer.RequestRecognizerUpdate(); // request for recognizer update
    _recognizer.LoadGrammar(new Grammar(new GrammarBuilder("help"))); // load a "help" grammar
    _recognizer.RequestRecognizerUpdate(); // request for recognizer update
    _recognizer.LoadGrammar(new Grammar(new GrammarBuilder("fuck"))); // load a "fuck" grammar
    _recognizer.RequestRecognizerUpdate(); // request for recognizer update
    _recognizer.LoadGrammar(new Grammar(new GrammarBuilder("what the hell"))); // load a "what
    the hell" grammar
    _recognizer.RequestRecognizerUpdate(); // request for recognizer update
    _recognizer.SpeechRecognized += _recognizeSpeechAndWriteToConsole_SpeechRecognized; // if
    speech is recognized, call the specified method
    _recognizer.SetInputToDefaultAudioDevice(); // set the input to the default audio device
    _recognizer.RecognizeAsync(RecognizeMode.Multiple); // recognize speech asynchronous

}

/// <summary>
/// speech that can be detected
/// </summary>
/// <param name="e"></param>
private void _recognizeSpeechAndWriteToConsole_SpeechRecognized(object sender,
SpeechRecognizedEventArgs e)
{
```

```
    if (e.Result.Text == "help")
    {
      //Console.WriteLine("help");
      textBox3.Text = "help";
      if (fallflag1)
       fallflag2 = true;
      if (fallflag3)
       fallflag4 = true;
      }
      else if (e.Result.Text == "fuck")
      {
       textBox3.Text = "fuck";
       verbal_aggresive = true;
      }
      else if (e.Result.Text == "what the hell")
      {
       textBox3.Text = "what the hell";
       verbal_aggresive = true;
    }
}

 /// <summary>
 /// detect whether the tracked human is sit or crouching
 /// </summary>
/// <param name="height"></param>
/// <param name="hip_distance"></param>
private void sit_crouching_detect(double head_position,double height,double hip_distance, double
footL_distance, double footR_distance)
{
 if (height >= 1.9)
{
   if (hip_distance <= (height * 0.545) && hip_distance > (height * 0.42) )
          sit_flag = true;
   else
          sit_flag = false;

   if (hip_distance <= (height * 0.42) && hip_distance >= (height * 0.25))
         crouching_flag = true;
   else
         crouching_flag = false;
}
else if (height >= 1.8)
{
   if (hip_distance <= (height * 0.565) && hip_distance > (height * 0.42))
         sit_flag = true;
   else
         sit_flag = false;
```

```
  if (hip_distance <= (height * 0.42) && hip_distance >= (height * 0.25))
        crouching_flag = true;
  else
        crouching_flag = false;
}
else if (height >= 1.7)
{
  if (hip_distance <= (height * 0.585) && hip_distance > (height * 0.42) && footL_distance < 0.35
  || hip_distance <= (height * 0.585) && hip_distance > (height * 0.42) && footR_distance < 0.35)
        sit_flag = true;
  else if (hip_distance <= (height * 0.635) && hip_distance > (height * 0.42) && footL_distance >
  0.35
  || hip_distance <= (height * 0.635) && hip_distance > (height * 0.42) && footR_distance > 0.35)
        sit_flag = true;
  else
        sit_flag = false;

  if (hip_distance <= (height * 0.42) && hip_distance >= (height * 0.25))
        crouching_flag = true;
  else
        crouching_flag = false;
}
else if(height <= 1.7)
{
  if (hip_distance <= (height * 0.605) && hip_distance > (height * 0.42) && footL_distance < 0.35
  || hip_distance <= (height * 0.605) && hip_distance > (height * 0.42) && footR_distance < 0.35)
           sit_flag = true;
  else if (hip_distance <= (height * 0.655) && hip_distance > (height * 0.42) && footL_distance >
  0.35
  || hip_distance <= (height * 0.655) && hip_distance > (height * 0.42) && footR_distance > 0.35)
           sit_flag = true;
  else
          sit_flag = false;

  if (hip_distance <= (height * 0.42) && hip_distance >= (height * 0.25))
         crouching_flag = true;
  else
         crouching_flag = false;
 }
}

/// <summary>
/// detect a person is walking or running
/// </summary>
/// <param name="avg_velocity"></param>
/// <param name="head_distance"></param>
/// <param name="footL_distance"></param>
/// <param name="footR_distance"></param>
```

```
/// <param name="height"></param>
/// <param name="hip_distance"></param>
private void walking_running_detect(double avg_velocity, double head_distance, double
footL_distance, double footR_distance ,double height, double hip_distance)
 {
if (height >= 1.9)
{
    if (avg_velocity > 1.3 && head_distance > 1.6 && hip_distance < 0.65 * height && hip_distance >
    0.61 * height)
        running_flag = true;
    else
        running_flag = false;

    if (avg_velocity <= 1.3 && !sit_flag && !crouching_flag && !jflag && avg_velocity > 0.65 &&
    head_distance > 1.6 && hip_distance < 0.65 * height && hip_distance > 0.61 * height)
        walking_flag = true;
    else
         walking_flag = false;
}
else if(height < 1.9)
{
  if (avg_velocity > 1.3 && head_distance > 1.6 && hip_distance < 0.66 * height && hip_distance >
  0.61 * height)
      running_flag = true;
  else
      running_flag = false;

  if (avg_velocity <= 1.3 && !sit_flag && !crouching_flag && !jflag && avg_velocity > 0.65 &&
  head_distance > 1.6 && hip_distance < 0.66 * height && hip_distance > 0.61 * height)
     walking_flag = true;
  else
     walking_flag = false;
}
}

/// <summary>
/// checking whether someone is jumping
/// </summary>
/// <param name="javg_velocity"></param>
/// <param name="footL"></param>
/// <param name="footR"></param>
/// <param name="height"></param>
private void jump_check(Double javg_velocity, Double footL, Double footR, Double height)
{
if (height > 1.9)
{
  if (javg_velocity > 15 && footL > 0.38 * height && footR > 0.38 * height && !sit_flag && !
  crouching_flag)
```

```
        jflag = true;
    else
        jflag = false;
}
if (height > 1.8)
{
    if (javg_velocity > 15 && footL > 0.37 * height && footR > 0.37 * height && !sit_flag && !
    crouching_flag)
        jflag = true;
    else
        jflag = false;
}
else if (height > 1.7)
{
    if (javg_velocity > 15 && footL > 0.36 * height && footR > 0.36 * height && !sit_flag && !
    crouching_flag)
        jflag = true;
    else
        jflag = false;
}
else if (height > 1.6)
{
    if (javg_velocity > 15 && footL > 0.35 * height && footR > 0.35 * height && !sit_flag && !
    crouching_flag)
        jflag = true;
    else
        jflag = false;
}
}

/// <summary>
/// Check whether touched each other
/// </summary>
/// <param name="skeleton1"></param>
/// <param name="skeleton2"></param>
private void touching_check(Skeleton skeleton1, Skeleton skeleton2)
{


if (Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.Spine].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.Spine].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.Spine].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.Spine].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
```

```
skeleton2.Joints[JointType.Spine].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.Spine].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.HipCenter].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.HipCenter].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.HipCenter].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.HipCenter].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
skeleton2.Joints[JointType.HipCenter].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.HipCenter].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.ShoulderCenter].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.ShoulderCenter].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.ShoulderCenter].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.ShoulderCenter].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
skeleton2.Joints[JointType.ShoulderCenter].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.ShoulderCenter].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.Head].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.Head].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.Head].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.Head].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
skeleton2.Joints[JointType.Head].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.Head].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.ElbowLeft].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.ElbowLeft].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.ElbowLeft].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.ElbowLeft].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
```

```
skeleton2.Joints[JointType.ElbowLeft].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.ElbowLeft].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.ElbowRight].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.ElbowRight].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.ElbowRight].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.ElbowRight].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
skeleton2.Joints[JointType.ElbowRight].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.ElbowRight].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.ShoulderLeft].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.ShoulderLeft].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.ShoulderLeft].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.ShoulderLeft].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
skeleton2.Joints[JointType.ShoulderLeft].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.ShoulderLeft].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.ShoulderRight].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.ShoulderRight].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.ShoulderRight].Position.Z) < 0.15
 || Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.ShoulderRight].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
skeleton2.Joints[JointType.ShoulderRight].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.ShoulderRight].Position.Z) < 0.15
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.WristLeft].Position.X) < 0.10 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.WristLeft].Position.Y) < 0.10 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.WristLeft].Position.Z) < 0.10
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.WristLeft].Position.X) < 0.10 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
```

```
skeleton2.Joints[JointType.WristLeft].Position.Y) < 0.10 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.WristLeft].Position.Z) < 0.10
|| Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.X -
skeleton2.Joints[JointType.WristRight].Position.X) < 0.10 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Y -
skeleton2.Joints[JointType.WristRight].Position.Y) < 0.10 &&
Math.Abs(skeleton1.Joints[JointType.HandLeft].Position.Z -
skeleton2.Joints[JointType.WristRight].Position.Z) < 0.10
|| Math.Abs(skeleton1.Joints[JointType.HandRight].Position.X -
skeleton2.Joints[JointType.WristRight].Position.X) < 0.10 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Y -
skeleton2.Joints[JointType.WristRight].Position.Y) < 0.10 &&
Math.Abs(skeleton1.Joints[JointType.HandRight].Position.Z -
skeleton2.Joints[JointType.WristRight].Position.Z) < 0.10)
   body_contact_flag = true;
else
   body_contact_flag = false;

if (Math.Abs(skeleton1.Joints[JointType.Spine].Position.X -
skeleton2.Joints[JointType.Spine].Position.X) <= 0.2 &&
Math.Abs(skeleton1.Joints[JointType.Spine].Position.Z - skeleton2.Joints[JointType.Spine].Position.Z)
<= 0.2)
   stick_flag = true;
else
   stick_flag = false;

if (Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.Spine].Position.X) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.Spine].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.Spine].Position.Z )< 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.Spine].Position.X) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
skeleton2.Joints[JointType.Spine].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.Spine].Position.Z )< 015
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.HipCenter].Position.X) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.HipCenter].Position.Y )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.HipCenter].Position.Z )< 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.HipCenter].Position.X) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
```

```
skeleton2.Joints[JointType.HipCenter].Position.Y )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.HipCenter].Position.Z )< 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.ShoulderCenter].Position.X) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.ShoulderCenter].Position.Y )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.ShoulderCenter].Position.Z )< 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.ShoulderCenter].Position.X) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
skeleton2.Joints[JointType.ShoulderCenter].Position.Y )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.ShoulderCenter].Position.Z )< 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.Head].Position.X) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.Head].Position.Y )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.Head].Position.Z )< 0.15
|| Math.Abs(skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.Head].Position.X) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.FootRight].Position.Y -
skeleton2.Joints[JointType.Head].Position.Y) < 0.15 &&
Math.Abs(skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.Head].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.KneeLeft].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.KneeLeft].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.KneeLeft].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.KneeLeft].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
skeleton2.Joints[JointType.KneeLeft].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.KneeLeft].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.KneeRight].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.KneeRight].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.KneeRight].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.KneeRight].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
```

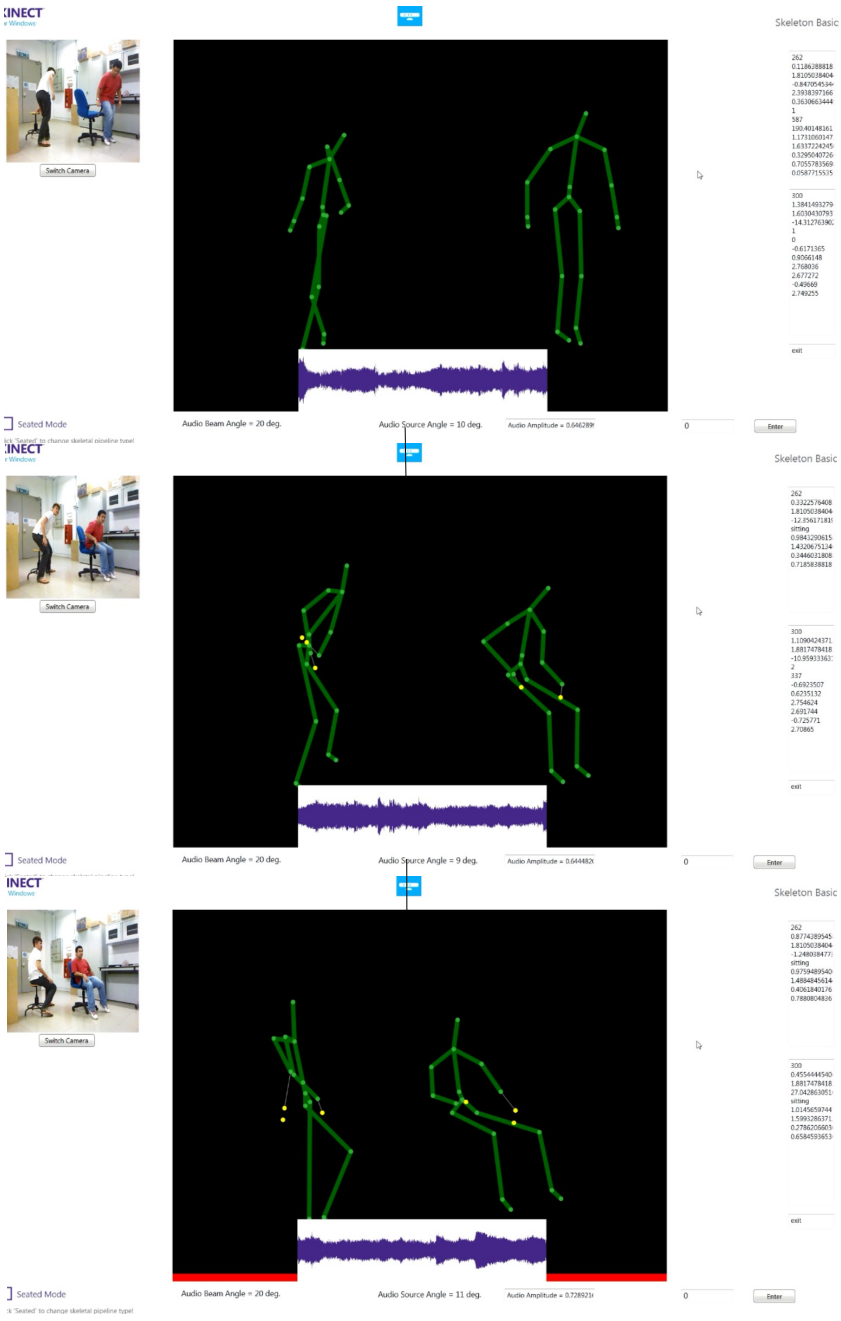
```
skeleton2.Joints[JointType.KneeRight].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.KneeRight].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.ElbowLeft].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.ElbowLeft].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.ElbowLeft].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.ElbowLeft].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
skeleton2.Joints[JointType.ElbowLeft].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.ElbowLeft].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.ElbowRight].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.ElbowRight].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.ElbowRight].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.ElbowRight].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
skeleton2.Joints[JointType.ElbowRight].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.ElbowRight].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.WristLeft].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.WristLeft].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.WristLeft].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.WristLeft].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
skeleton2.Joints[JointType.WristLeft].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.WristLeft].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.X -
skeleton2.Joints[JointType.WristRight].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Y -
skeleton2.Joints[JointType.WristRight].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootLeft].Position.Z -
skeleton2.Joints[JointType.WristRight].Position.Z) < 0.15
|| Math.Abs( skeleton1.Joints[JointType.FootRight].Position.X -
skeleton2.Joints[JointType.WristRight].Position.X )< 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Y -
```

```
skeleton2.Joints[JointType.WristRight].Position.Y) < 0.15 &&
Math.Abs( skeleton1.Joints[JointType.FootRight].Position.Z -
skeleton2.Joints[JointType.WristRight].Position.Z) < 0.15)
   leg_touch = true;
else
   leg_touch = false;
}

/// <summary>
/// Human Aggressive Behavioral Analysis code start from here
/// </summary>
/// <param name="Lleg_velocity"></param>
/// <param name="footL_distance"></param>
/// <param name="Rleg_velocity"></param>
/// <param name="footR_distance"></param>
/// <param name="Lhand_velocity"></param>
/// <param name="Rhand_velocity"></param>
private void HABclassifier(double Lleg_velocity, double footL_distance, double Rleg_velocity, double
footR_distance , double Lhand_velocity , double Rhand_velocity)
{
if (Lleg_velocity > 2.3 && footL_distance > 0.4 || Rleg_velocity > 2.3 && footR_distance > 0.4)
   kicking_flag = true;
else
      kicking_flag = false;

if (Lhand_velocity >= 1.75 && Lhand_up || Rhand_velocity >= 1.65 && Rhand_up)
   sfighting_flag = true;
else
   sfighting_flag = false;
}
}
}
```

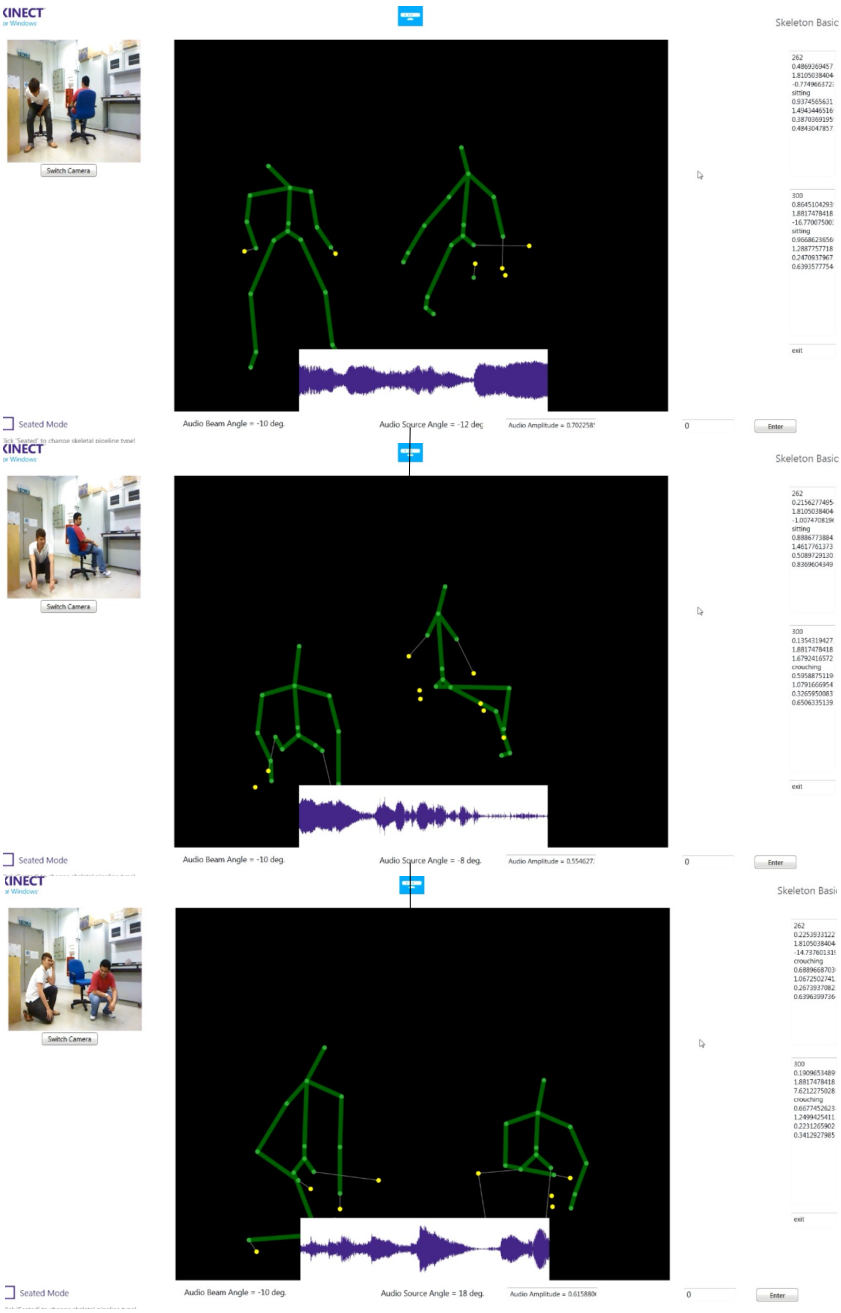
APPENDIX D

Samples of Video Sequence for Detectable Behavior

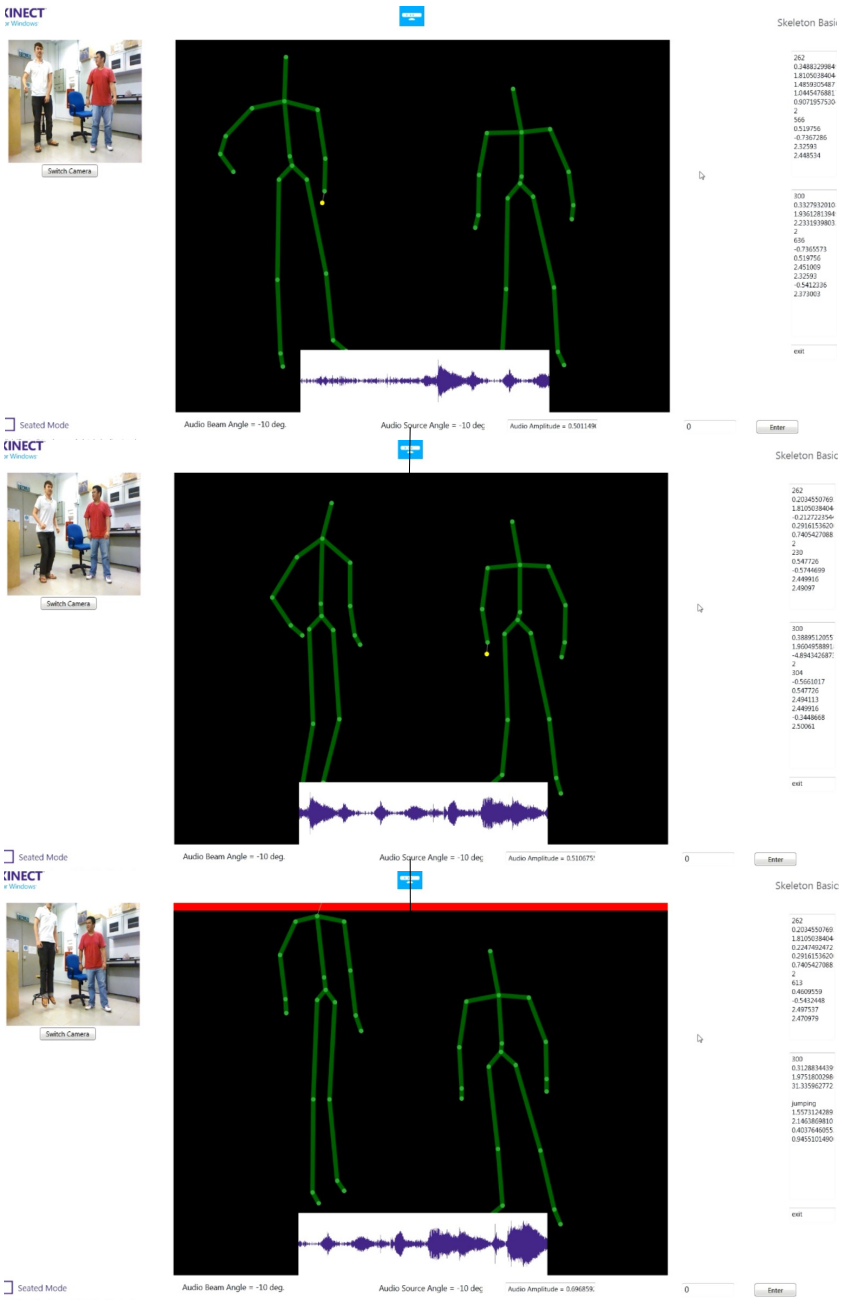
Sitting


Skeleton Basic
Switch Camera
Seated Mode
Audio Beam Angle = 20 deg.
Audio Source Angle = 10 deg.
Enter
Skeleton Basic
Switch Camera
Seated Mode
Audio Beam Angle = 20 deg.
Audio Source Angle = 9 deg.
Enter
Skeleton Basic
Switch Camera
Seated Mode
Audio Beam Angle = 20 deg.
Audio Source Angle = 11 deg.
Enter

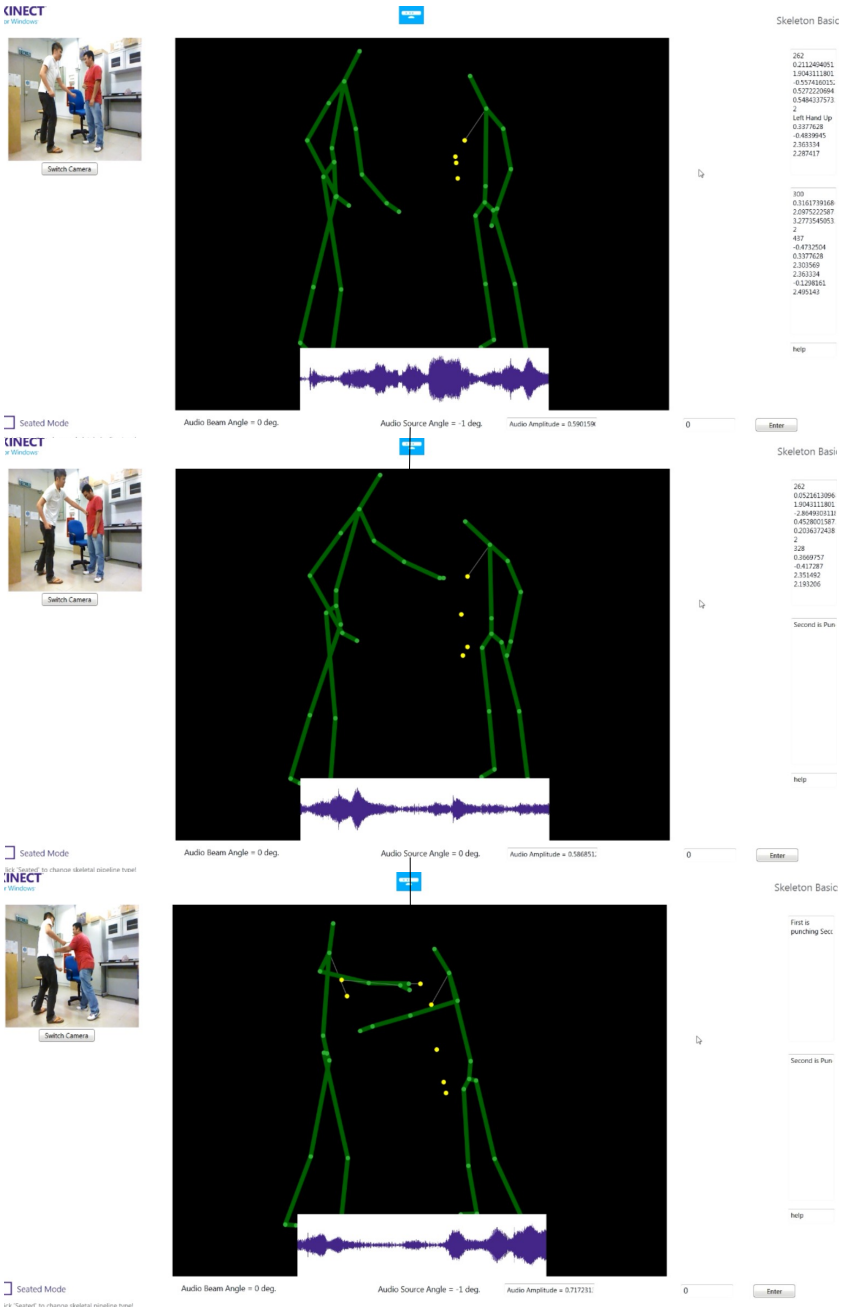
Crouching


Skeleton Basic
Switch Camera
Seated Mode
Audio Beam Angle = -10 deg.
Audio Source Angle = -12 deg
Audio Amplitude = 0.702258
Enter
Skeleton Basic
Switch Camera
Seated Mode
Audio Beam Angle = -10 deg.
Audio Source Angle = -8 deg.
Audio Amplitude = 0.554027
Enter
Skeleton Basi
Switch Camera
Seated Mode
Audio Beam Angle = -10 deg.
Audio Source Angle = 18 deg.
Audio Amplitude = 0.615880
Enter

Jumping


Switch Camera
Skeleton Basi
Seated Mode
Audio Beam Angle = -10 deg.
Audio Source Angle = -10 deg
Enter
Switch Camera
Skeleton Basic
Seated Mode
Audio Beam Angle = -10 deg.
Audio Source Angle = -10 deg
Enter
Switch Camera
Skeleton Basic
Seated Mode
Audio Beam Angle = -10 deg.
Audio Source Angle = -10 deg
Enter

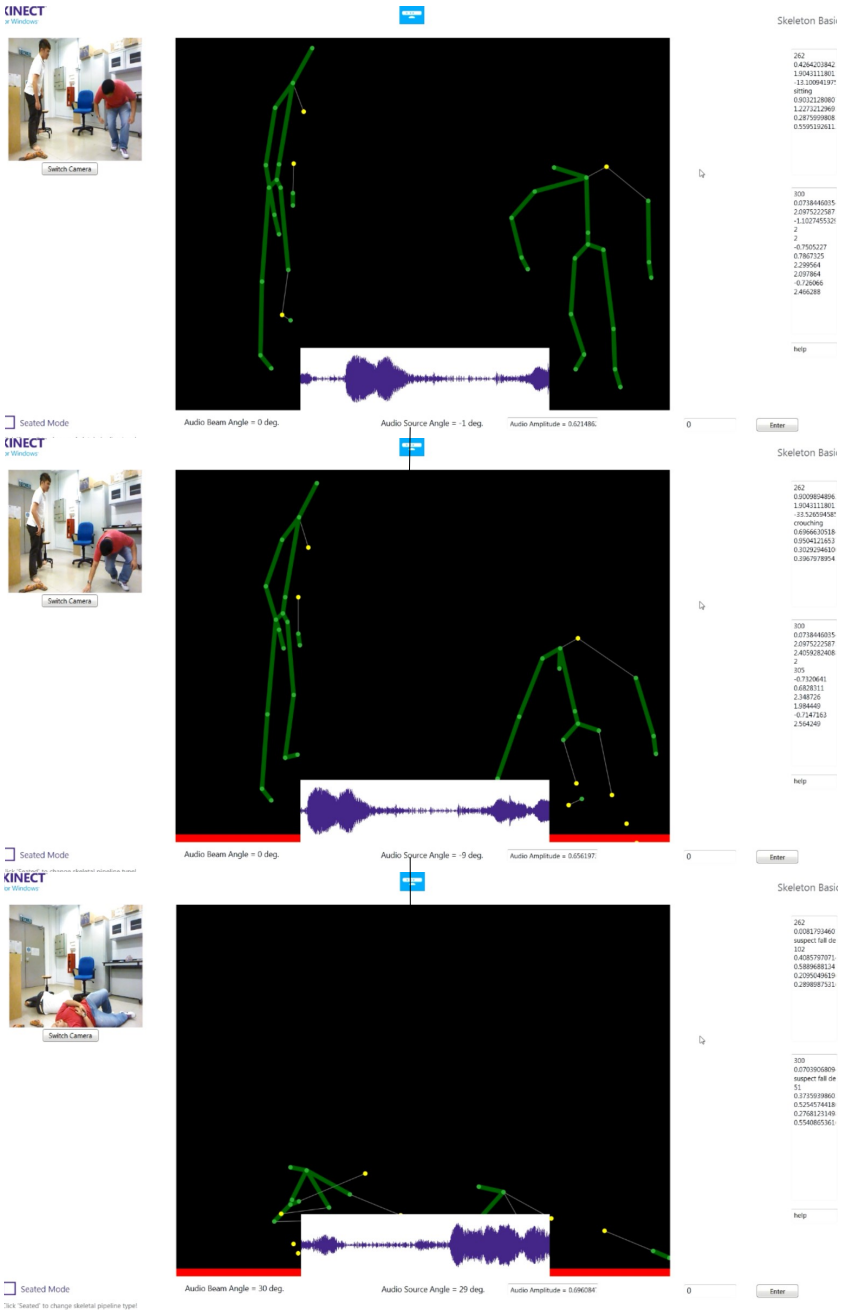
Punching


Switch Camera
Skeleton Basic
Seated Mode
Audio Beam Angle = 0 deg.
Audio Source Angle = -1 deg.
Enter
Switch Camera
Skeleton Basi
Seated Mode
Audio Beam Angle = 0 deg.
Audio Source Angle = 0 deg.
Enter
Switch Camera
Skeleton Basic
Seated Mode
Audio Beam Angle = 0 deg.
Audio Source Angle = -1 deg.
Enter

Kicking


Switch Camera
Seated Mode
Audio Beam Angle = 0 deg.
Audio Source Angle = -9 deg.
Enter
Switch Camera
Seated Mode
Audio Beam Angle = 0 deg.
Audio Source Angle = -2 deg.
Enter
Switch Camera
Seated Mode
Audio Beam Angle = 20 deg.
Audio Source Angle = 16 deg.
Enter

Fall Event


Switch Camera
Seated Mode
Audio Beam Angle = 0 deg.
Audio Source Angle = -1 deg.
Audio Amplitude = 0.621486
Enter
Switch Camera
Seated Mode
Audio Beam Angle = 0 deg.
Audio Source Angle = -9 deg.
Audio Amplitude = 0.656197
Enter
Switch Camera
Seated Mode
Audio Beam Angle = 30 deg.
Audio Source Angle = 29 deg.
Audio Amplitude = 0.696084
Enter