Faculty of Electronics and Computer Technology and Engineering

KINECT-BASED FALL DETECTION SYSTEM FOR THE ELDERLY

KONG TIAN CHYUAN

Bachelor of Electronics Engineering Technology with Honours

2024

KINECT-BASED FALL DETECTION SYSTEM FOR THE ELDERLY

KONG TIAN CHYUAN

A project report submitted
in partial fulfillment of the requirements for the degree of
Bachelor of Electronics Engineering Technology with Honours

Faculty of Electronics and Computer Technology and Engineering

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2024

UNIVERSITI TEKNIKAL MALAYSIA MELAKA
FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KOMPUTER

BORANG PENGESAHAN STATUS LAPORAN
PROJEK SARJANA MUDA II

Tajuk Projek : Kinect-Based Fall Detection System For The Elderly
Sesi Pengajian : 2023/2024

Saya KONG TIAN CHYUAN mengaku membenarkan laporan Projek Sarjana Muda ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Laporan adalah hakmilik Universiti Teknikal Malaysia Melaka.
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan laporan ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. Sila tandakan (✓):

☐	**SULIT***	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
☐	**TERHAD***	(Mengandungi maklumat terhad yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan.)
√	**TIDAK TERHAD**	

Disahkan oleh:

(TANDATANGAN PENULIS)

(COP DAN TANDATANGAN PENYELIA)

IZARENA BINTI MUSTAFFA
Pensyarah Kanan
Fakulti Teknologi Dan Kejuruteraan Elektronik Dan Komputer (FTKEK)
Universiti Teknikal Malaysia Melaka (UTeM)

Alamat Tetap: 23 Kampung Baru
28400 Mentakab,
Pahang.

Tarikh : 05 Februari 2024

Tarikh : 05 Februari 2024

*CATATAN: Jika laporan ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh laporan ini perlu dikelaskan sebagai SULIT atau TERHAD.

DECLARATION

I declare that this project report entitled “Kinect-Based Fall Detection System For The Elderly” is the result of my own research except as cited in the references. The project report has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature : 

Student Name : KONG TIAN CHYUAN

Date : 05/02/2024

APPROVAL

I hereby declare that I have checked this project report and in my opinion, this project report is adequate in terms of scope and quality for the award of the degree of Bachelor of Electronics Engineering Technology with Honours.

Signature : 

Supervisor Name : PN. IZADORA BINTI MUSTAFFA

Date : 05/02/2024

Signature :

Co-Supervisor :

Name (if any)

Date :

DEDICATION

To my beloved mother, Tan Mee Li, and father, Kong Woei Ming,

and

my dear friend, Soh Teck Wei

ABSTRACT

The growing population of elderly individuals are increasing in correlation to the advancement in healthcare. Along with that, however, are the associated risks, such as incidents of falls. Several studies had shown that older adults experienced at least one fall every year, and it was the main cause of accidental death in older adults aged 65 or more. In Malaysia, many elderly are left alone during the day as their family members are out to work or school. Therefore, a fall detection system was designed to detect when a person experiences a fall or a loss of balance. The system utilizes Kinect sensors and algorithms to detect the movements and postures of an individual, aiming to analyze and identify patterns that indicate a fall event. Skeletons and joints such as heads, shoulder centre, hip centre, ankle left, and right are detected and extracted. The fall algorithm is implemented to obtain the y-coordinate values and threshold values. Several observed fall scenarios including fall to the left side, fall to the right side, fall to the front, fall to the back, and fall while sitting. The result of lower accuracy at short distances from the Kinect sensor can be attributed to its limited field of view and depth perception issues at close range, leading to incomplete or distorted skeleton tracking. Slightly longer distances provide a more optimal range for accurate skeleton tracking and fall detection, while accuracy increases at longer ranges due to increased detail in the captured data. The fall detection accuracy of 100% is obtained throughout the evaluation of all heights of Kinect sensor and lighting conditions.

ABSTRAK

Jumlah populasi warga tua semakin meningkat seiring dengan kemajuan dalam bidang penjagaan kesihatan. Walau bagaimanapun, terdapat risiko yang berkaitan dengan peningkatan ini, seperti kejadian jatuh. Beberapa kajian menunjukkan bahawa orang dewasa tua mengalami sekurang-kurangnya satu kejadian jatuh setiap tahun, dan ia merupakan punca utama kematian akibat kemalangan bagi orang dewasa tua berumur 65 tahun atau lebih. Di Malaysia, ramai warga tua ditinggalkan sendirian sepanjang hari kerana ahli keluarga mereka pergi bekerja atau sekolah. Oleh itu, sistem pengesanan jatuh telah direka untuk mengesan apabila seseorang mengalami jatuh atau kehilangan keseimbangan. Sistem ini menggunakan pengesan Kinect dan algoritma untuk mengesan pergerakan dan posisi individu, dengan tujuan menganalisis dan mengenal pasti corak yang menunjukkan kejadian jatuh. Struktur dan sendi seperti kepala, pusat bahu, pusat pinggul, mata kaki kiri, dan kanan dikesan dan diekstrak. Algoritma jatuh diaplikasikan untuk mendapatkan nilai koordinat Y dan nilai ambang. Beberapa senario jatuh yang diperhatikan termasuk jatuh ke sebelah kiri, jatuh ke sebelah kanan, jatuh ke hadapan, jatuh ke belakang, dan jatuh semasa duduk. Hasil ketepatan yang lebih rendah pada jarak dekat dari penderia Kinect boleh dikaitkan dengan bidang pandangan yang terhad dan isu persepsi kedalaman pada jarak dekat, yang membawa kepada penjejakan rangka yang tidak lengkap atau herot. Jarak yang lebih jauh sedikit memberikan julat yang lebih optimum untuk pengesanan rangka yang tepat dan pengesanan jatuh, manakala ketepatan meningkat pada julat yang lebih panjang disebabkan oleh peningkatan butiran dalam data yang ditangkap. Ketepatan pengesanan jatuh sebanyak 100% diperoleh sepanjang penilaian semua ketinggian sensor Kinect dan keadaan pencahayaan.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my supervisor, Pn. Izadora Binti Mustaffa for the precious guidance, words of wisdom and patient throughout this final year project.

I am also indebted to Universiti Teknikal Malaysia Melaka (UTeM) for the financial support which enables me to accomplish the project. Not forgetting my fellow colleague, Chan Pui Kit for the willingness of sharing his thoughts and ideas regarding the project.

My highest appreciation goes to my parents and family members for their love and prayer during the period of my study. An honourable mention also goes to Dr. Haslinah Binti Mohd Nasir for all the motivation and understanding.

Finally, I would like to thank all the staffs at the faculty, fellow colleagues and classmates, the faculty members, as well as other individuals who are not listed here for being co-operative and helpful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

This chapter aims to establish the framework and presents a brief concept of the project. It focuses on the overview of the project, describes the objectives, briefly the problem, the scope, and the results of the project.

1.2 Background

Fall detection systems had become increasingly important in recent years due to the growing population of elderly individuals and the associated risks of falls [1]. Several studies had shown that elderly people experienced at least one fall every year, and falls was the main cause of accidental death in older adults aged 65 or more. Therefore, fall detection had attracted a lot of attention from researchers and industry professionals [2]. A huge number of materials, equipment, and fall detection methods had been proposed over the years. The equipment used were gyroscope, accelerometer, GPS module and Kinect sensor, while the fall detection methods, such as smartphone-based systems, vision-based systems, and wearable-sensor-based systems. Among these methods, vision-based systems had a great advantage because they did not require the elderly people to wear specific equipment. The Microsoft Kinect device was the system that had been used for fall detection due to its ability to track human body movements accurately.

Fall detection system was designed for use in indoor environments, particularly in homes and public hospitals. Existing fall detection systems often required wearable sensors,

which could be inconvenient for users. There were two main subsystems which are camera and the fall detector module. The camera captured depth information and color images, which were then processed by the fall detector module using available libraries for camera management and computer vision procedures. It had been shown that the system can detect falls with a reliability of 97.3% and an efficiency of 80.0% [3]. The algorithm was designed to provide an accurate solution to detect falls, which is especially important for elderly people living alone at home or those with motor disabilities.

The system could be designed to detect falls and provide immediate assistance by sending an SMS message notification and making an emergency call. The use of the Kinect sensor with robotics could bring new ways to build intelligent systems that could be used to monitor elderly people and raise an alarm in case of falling are detected [4].

The vision-based fall detection system offered several advantages over traditional fall detection methods. The advantage was it provides non-intrusive monitoring, not requiring individuals to wear additional devices or sensors. The Kinect sensor could capture the required data from a distance, making it convenient and user-friendly. One of the advantages was the system operated in real-time, enabling swift response to potential fall incidents. This could significantly reduce the time between the occurrence of a fall and the provision of assistance, potentially minimizing the severity of injuries. Another advantage was the system could collect data over time, allowing for analysis and insights into an individual's movement patterns, which could be useful for long-term monitoring and proactive care.

1.3 Problem Statement

One of the main challenges in the fall detection system is the performance of the detection rate under real-life conditions. It has been shown that promising results in laboratory environments and the accuracy of fall detection systems could be affected by factors such as lighting conditions, flooring surfaces, and obstacles. Another challenge is usability and user acceptance. Fall detection systems must be easy to use and not interfere with daily activities so users can accept them. Power consumption is another issue that must be addressed in fall detection systems. Kinect-based fall detection systems require power to operate continuously. Real-time operations are also important for fall detection systems, as delays in detecting falls can lead to serious injuries [5].

Another key challenge in fall detection is the imperfection of collected data, which could be caused by various factors, such as sensor noise, calibration errors, and environmental interference. Another challenge is sensor technologies' diversity or low reliability, which can lead to inconsistent results and false alarms. Furthermore, some challenges typical to other frameworks with data fusion requirements such as selecting appropriate sensors, designing effective algorithms for data fusion, and dealing with missing or incomplete data. In addition, there is no standard dataset available for evaluating fall detection systems, which makes it difficult to compare different approaches. It is significant to develop more sophisticated algorithms for data fusion and explore new sensor technologies [6].

1.4 Objective

The project aims to achieve the following objectives: -

i. To design a non-invasive, reliable, and accurate solution to detect falls among older adults.

ii. To develop an algorithm that can accurately detect falls based on the data collected by the Kinect sensor, including skeletal tracking data and color data.

iii. To analyze and evaluate real-time fall detection and alerts to caregivers or emergency services to reduce response time and improve outcomes.

1.5 Scope and Limitations

The project focuses on Kinect-based fall detection system for the elderly. Kinect sensors can provide reliable depth sensing and tracking capabilities, allowing them to detect human body movements accurately. This makes it suitable for detecting falls and distinguishing them from other activities. In the same way, Kinect-based fall detection systems can provide real-time monitoring of individuals, enabling immediate response in case of a fall. Additionally, Kinect uses depth-sensing cameras to capture movement data, eliminating the need for wearable devices or physical contact. This non-intrusive setup makes it more comfortable and convenient for users.

On the other hand, Kinect has a specific field of view and range, which may limit its effectiveness in large rooms or areas where individuals are far away from the device. Multiple Kinect devices or careful placement may be required to cover larger spaces adequately. Kinect-based fall detection can sometimes produce false positives or negatives like any other automated system. Certain movements or actions, such as sudden movements or bending down, may trigger false alarms, while actual falls may occasionally go

undetected. Continuous refinement and testing are necessary to minimize such errors. Moreover, Kinect-based systems capture and analyze individuals' visual data, raising privacy concerns. Adequate measures should be taken to ensure data security, privacy, and compliance with applicable regulations.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Extensive research and investigation have been conducted to accomplish this project. Data and studies relevant to the project were collected from various sources including books, articles, journals, and websites. The information served as a valuable reference to ensure the feasibility of completing the project within the allocated time. The studies and information gathered focused on significant and relevant topics about the project.

This part explored several thesis and publication journals from the Google Scholar website were explored. There were a few keywords to find the related information which were “fall detection”, “Kinect”, “elderly people”, “wearable sensor”, and “IoT application”. The literature review focused on the fall detection system integrated with the application and sensors to identify and alert individuals or caregivers when a person experiences a fall. Fourteen articles were chosen to focus on the development of fall detection system.

2.2 Wearable-Sensor-Based Fall Detection System

2.2.1 Quaternion Algorithm

The wearable device was placed on the patient's waist to analyze the human body's acceleration. The system monitored the human body's movements from normal daily activities by an effective quaternion algorithm and automatically sent requests for help to caregivers with the patient's location. The advantages of this system included timely and reliable surveillance to mitigate the adverse effects of falls, which was especially important

for elderly individuals who may experience a decline in physical fitness. However, the choice of threshold was essential to distinguish falling from heavily lying activity, and a sufficient sample number collected from subjects with different ages and gender was necessary to improve reliability and robustness [7].

Figure 2.1: Fall location [7].

2.2.2 IoT and Big Data

3D-axis accelerometer was used and embedded into a 6LowPAN device wearable to collect real-time data from the movements of older people. The sensor readings were processed and analyzed using a decision trees-based Big Data model running on a Smart IoT Gateway to detect falls [8]. If a fall was detected, an alert would activate, and notifications would send to the groups responsible for caring for older people. The advantages of this system included its ability to provide high efficiency in fall detection and its use of low-

power wireless sensor networks, smart devices, big data, and cloud computing. It also automatically sends notifications to caregivers in case of falls. However, some disadvantages included the need for older people to always wear the device, which might be uncomfortable or inconvenient for them. There might also be privacy concerns regarding collecting and analyzing personal data. Moreover, there might be false alarms due to other movements that resemble falls.

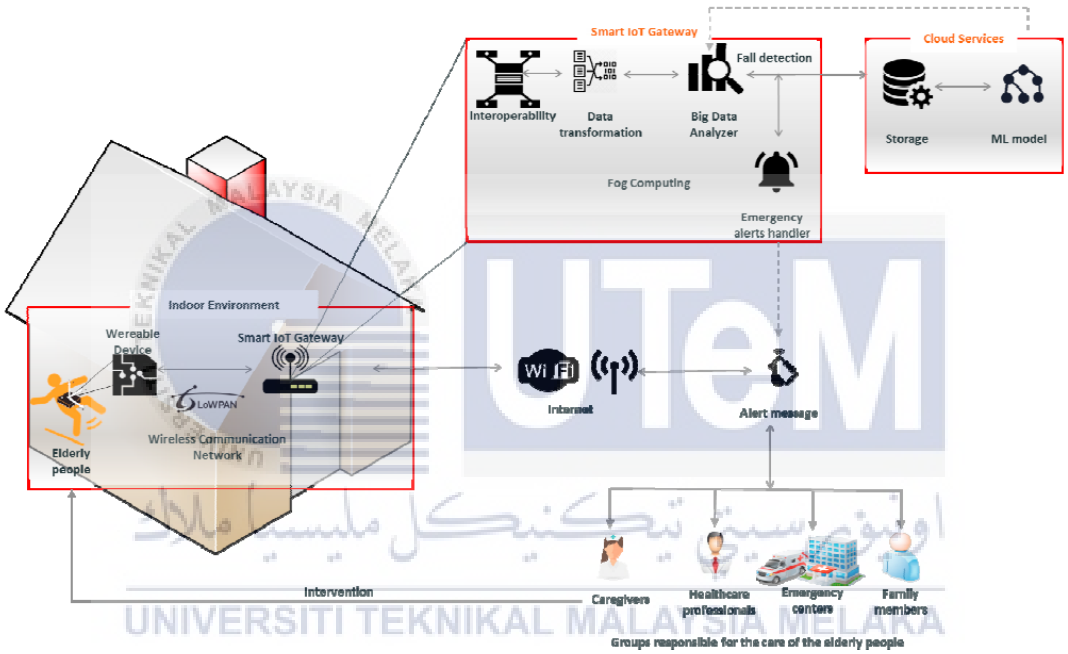


Figure 2.2: Fall detection system architecture [8].

2.2.3 Machine Learning Algorithm

The wearable device integrated with 3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer, and barometer sensor. These sensors worked together to obtain highly accurate information about the posture and altitude of the subject. The accelerometer, gyroscope, and magnetometer were implemented an AHRS (Attitude and Heading Reference System) that estimated the device's orientation and provided dynamic vertical

acceleration values. The barometer sensor measured pressure and temperature to calculate altitude, which was then fused with vertical displacement data from the AHRS and acceleration data using a complementary filter to obtain an accurate altitude estimation.

Machine learning techniques was used to detect accurately falls. The algorithm involved several stages which were data acquisition, feature extraction, selection, and classification. The sensor data was first acquired from the accelerometer, gyroscope, magnetometer, and barometer sensors and then preprocessed to remove noise and artifacts. Next, a set of features was extracted from the preprocessed data, including time-domain, frequency-domain, and statistical features. A feature selection algorithm was then used to select the most relevant features for fall detection. Finally, a machine learning classifier, a support vector machine (SVM), is trained on the selected features to classify falls and non-falls. The proposed algorithm achieved a high accuracy rate of 98.3% in detecting falls, demonstrating its effectiveness in accurately detecting falls and minimizing false alarms [9].

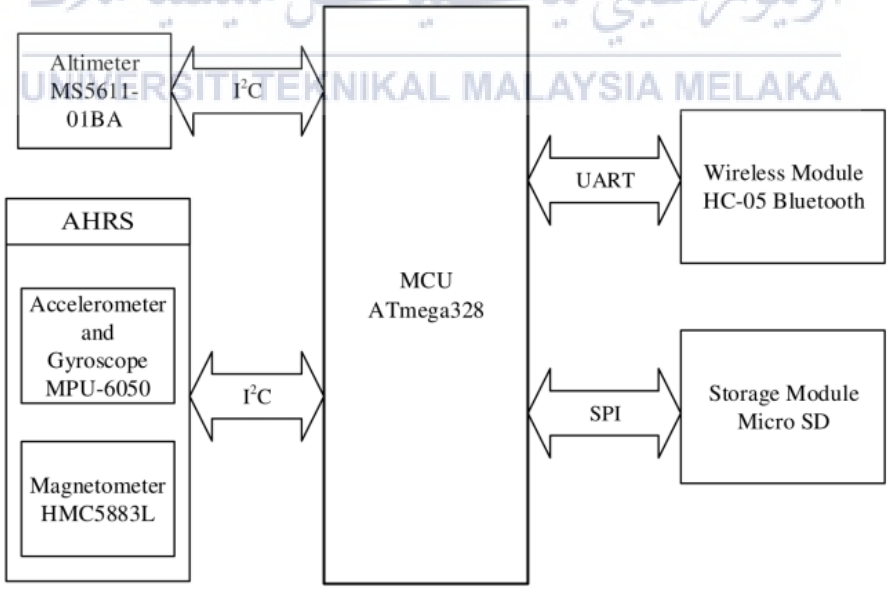


Figure 2.3: AHRS and barometer sensor based fall detector system [9].

2.3 Smartphone-Based Fall Detection System

2.3.1 Threshold-Based Algorithm

Android devices were used with sensors and communication services and incorporated a threshold-based algorithm enhanced by a kNN classifier. Personalization implementation was proposed to improve accuracy. The system achieved high fall detection accuracy, with 97.53% sensitivity and 94.89% specificity, comparable to related works [10].

The advantages of this system included its use of widely available technology (Android devices), its high accuracy in detecting falls, and its potential for personalization implementation. However, there were also some potential disadvantages to consider. For example, the system might not be as effective in detecting falls if the user was not carrying their smartphone at the time of the fall or cannot press an emergency button on their device. On the other hand, some users might hesitate to use a fall detection system that always required carrying their smartphone.

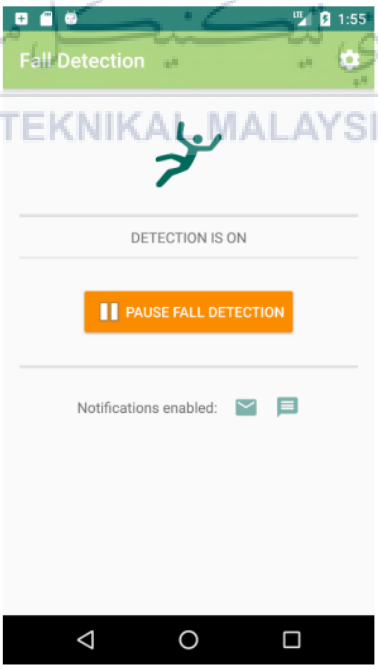


Figure 2.4: Fall detection app [10].

2.3.2 Deep Learning Algorithm

SmartFall was developed using deep learning algorithms to analyze accelerometer data and detect falls. The system was designed to be user-friendly and non-intrusive, allowing seniors to wear the smartwatch and carry their smartphones as they went about their daily activities. SmartFall has several advantages over traditional fall detection systems, including its low cost, high accuracy, and ability to collect data on activities of daily living (ADLs). However, there were also some potential disadvantages to consider. For example, the battery life of the smartwatch and smartphone might be reduced when running SmartFall continuously alongside other apps. In addition, some seniors might be hesitant to wear a smartwatch or carry a smartphone around all day for fall detection purposes. In short, SmartFall represented a new development in the health IoT applications field to improve seniors' safety [11].

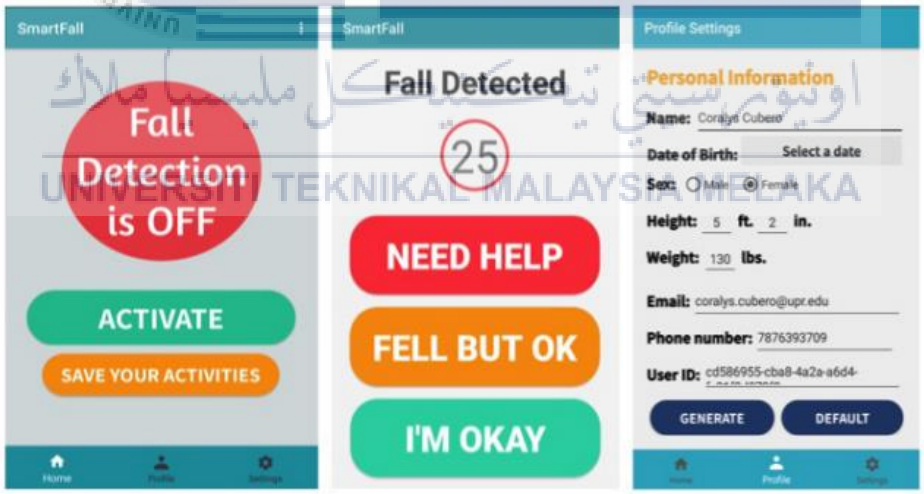


Figure 2.5: SmartFall user interface [11].

2.3.3 Machine Learning Algorithm

The need for a fall detection and alert system was discussed to reduce the response time of emergency responders, improve personalized fall prevention strategies, and reduce the risk of secondary complications and functional decline associated with falls. Previous studies on fall detection had been conducted in lab settings or using simulated falls excluding real-world falls and their causes. The results of a prospective study investigated the performance of a smartphone-based fall detection and alert system that uses the smartphone's built-in accelerometer and gyroscope. A machine learning algorithm and an online component were used to send notifications in real-time to researchers or caregivers. Preliminary results showed that the system detected 73% of falls and had a specificity of over 99.9%, making it a promising approach for real-life fall detection [12]. The system's advantages were its portability, low cost, and the fact that users don't need to wear dedicated sensors. However, limitations included that the system was smartphone-dependent and required signal reception to detect and notify about falls, and battery life could be a restricting factor for data collection.

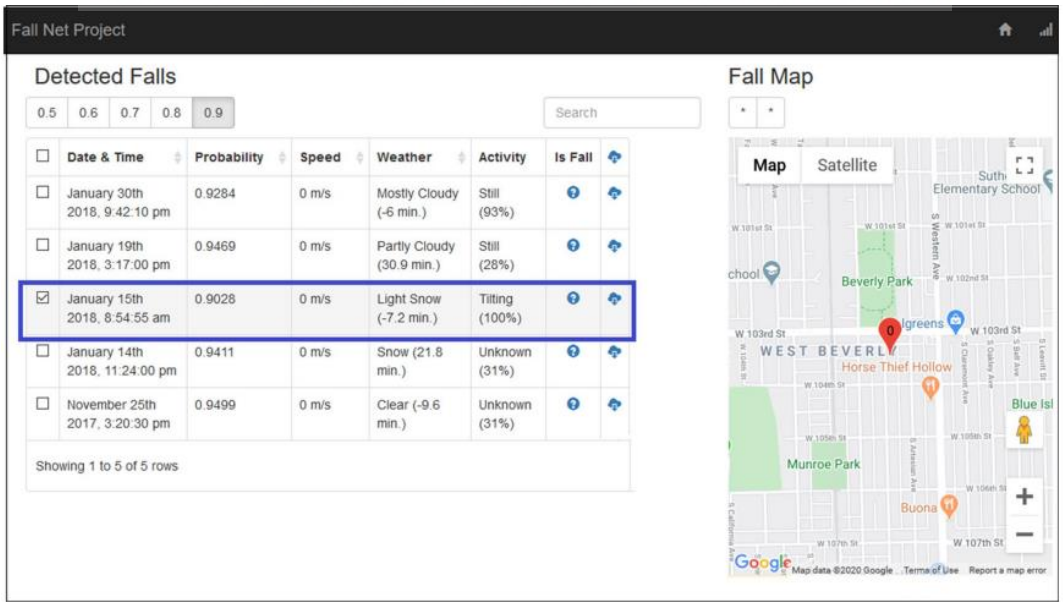


Figure 2.6: Web portal data summary [12].

2.4 Kinect-Based Fall Detection System

2.4.1 Position Algorithm and Velocity Algorithm

Microsoft Kinect sensor was used to detect, and report falls among older adults. The system employed two algorithms for fall detection. The first algorithm analyzed a single frame to determine if a fall had occurred, while the second algorithm used time series data to differentiate between falls and slowly lying down on the floor. This distinction was crucial for accurate detection. In addition to detecting falls, the system offered various reporting options. Reports could be sent as emails or text messages, and they could include pictures taken during and after the fall. This visual documentation provided valuable information for further analysis and assessment. The system incorporated a voice recognition system that allows users to cancel erroneous alerts [13].

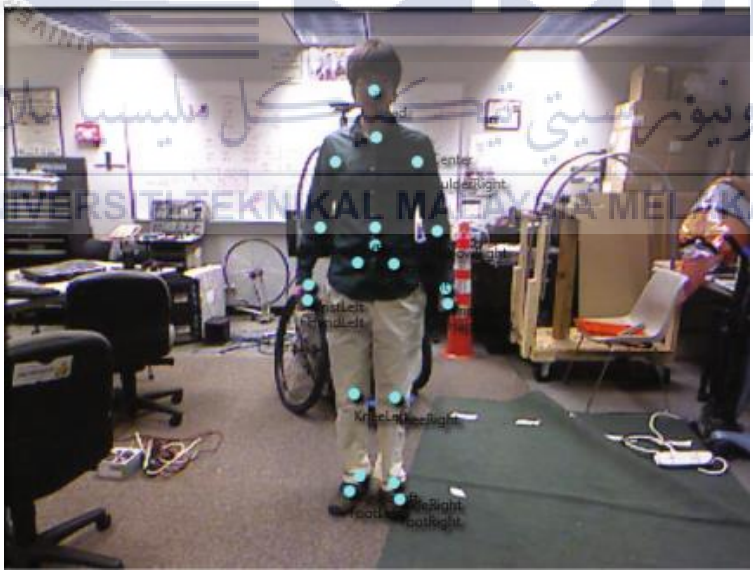


Figure 2.7: Joints of skeleton detection [13].

2.4.2 Point Cloud

A Kinect camera was used to capture depth data of the individual in the living environment. The height change acceleration of the human point cloud was used to detect possible fall activity, which was transmitted in real-time to an Arduino microprocessor that notified the guardian. The system eliminated the need for wearable devices, had a low false-positive rate, and operated 24/7 despite external illumination. The experimental results showed high accuracy in detecting falls, with an overall accuracy rate of 99%. The system contributed to reducing the number of false-positive detections. The system's advantages were precise detection, non-invasiveness, and easy implementation. The disadvantages were the need for an existing Kinect camera, system limitations on detecting falls in certain scenarios, and the possible risk of false negatives during quick fall events. In short, the system could enhance the safety and quality of life for elderly individuals living alone [14].

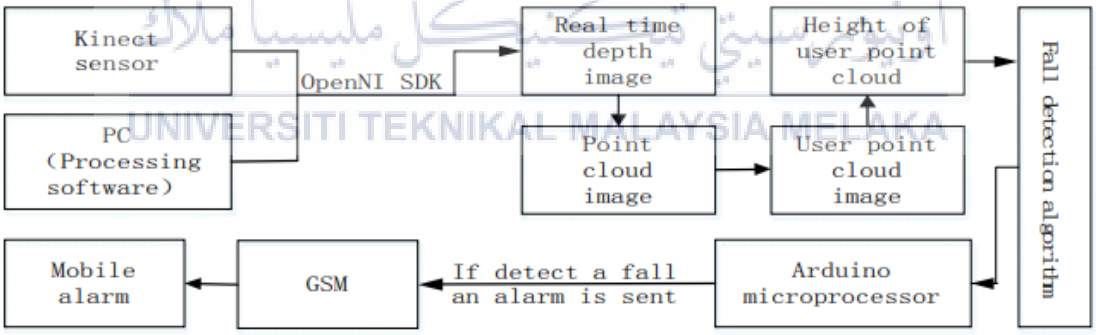


Figure 2.8: Point cloud system diagram [14].

2.4.3 Thresholding Algorithm

The platform utilized data from a Microsoft Kinect v2 sensor to monitor the movements of elderly individuals and detect falls in real time. The application could also

capture and store a wide range of image data, store positional information in CSV files for further offline processing, calculate the real-time acceleration of body joints, and calculate the angle between arbitrarily chosen joints for simple gesture detection. The modular platform allowed for the integration of other data sources to create a comprehensive monitoring system. The advantages of this platform included its real-time fall detection capabilities, ability to capture and store a wide range of image data, and modular design that allows for customization and integration with other data sources. However, some potential disadvantages included the need for a Microsoft Kinect v2 sensor, which might be costly or difficult to obtain, as well as potential privacy concerns related to collecting and storing personal data. In addition, the platform might require technical expertise to set up and maintain. Overall, this platform could provide peace of mind to families and caregivers of elderly individuals living alone [15]

Figure 2.9: Environment testing model used in Kinect-based platform [15].

2.4.4 Deep Learning Technique on 3D Skeleton

This system combined traditional algorithms with neural networks. A skeleton information extraction algorithm was used to transform depth information into skeleton

information and extract important joints related to fall activity. The skeleton-based method is modified with seven highlight feature points. The deep neural network architecture is used to classify the extracted features and detect falls. The experimental results showed that the proposed system had higher accuracy than traditional algorithms and consumed less energy than pure neural network methods. The advantages of this system were its high accuracy, low energy consumption, and potential for use in homecare systems to protect older people from falls. However, the system requires more memory to store parameters than traditional algorithms, and it might not be suitable for real-time applications due to its computational complexity [16].

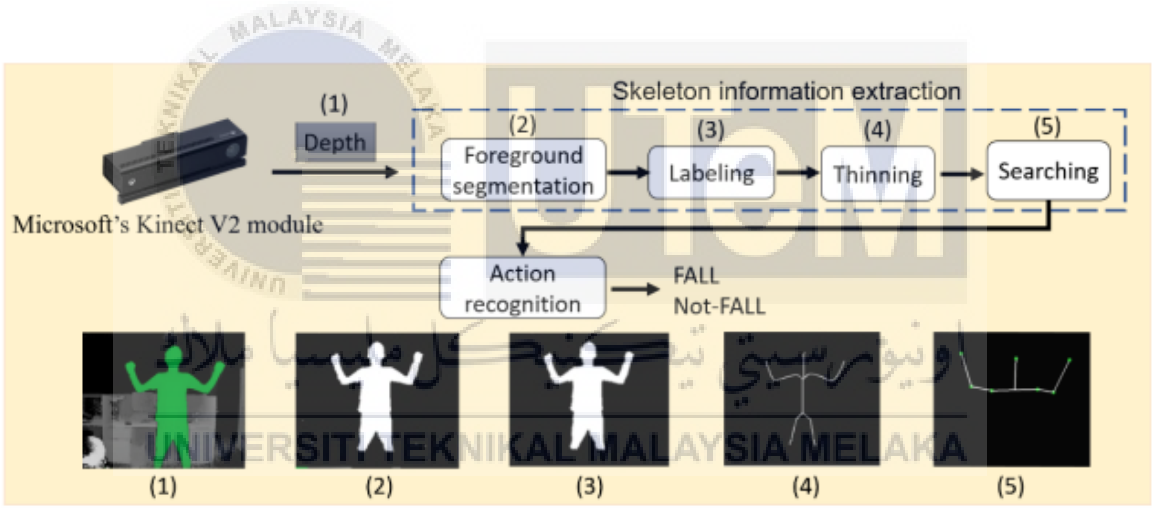


Figure 2.10: 3D skeleton using deep learning system [16].

2.4.5 Human Posture Recognition

Microsoft Kinect V2 sensor technology was developed to create a natural human-computer interaction method that enables people to communicate with computers easily and naturally. By processing depth information and using skeleton tracking technology, the optimized BP neural network was trained to recognize standing, sitting, and lying positions, with fall detection based on this. The research showed that a neural network could help to

achieve reliable and accurate results, even under real-world conditions. The system developed used non-wearable techniques through the Kinect V2 camera and skeleton tracker to recognize human body posture. Human motion recognition based on Kinect sensors has shown excellent value in the healthcare field, specifically in medical rehabilitation and remote care for older people. On the other hand, the output of skeleton tracking algorithms in real-world applications was not always stable and accurate, and the recognition result of lying posture was not satisfactory. However, the proposed system was based only on depth maps and did not use color information, which guaranteed the person's privacy and worked in poor light conditions [17].

Figure 2.11: Experimental setup [17].

2.4.6 Reinforcement Learning Algorithm

Microsoft Kinect Sensor was used to detect falls and an automatic fall notification system based on Reinforcement Learning to notify caregivers or emergency services. The proposed system aims to overcome the issues found in existing fall detection and notification systems, such as obtrusiveness, lack of accuracy and robustness, and inadequate feedback systems. The advantages of this system included its non-obtrusiveness, high accuracy and

robustness, and efficient notification system that avoids unnecessary disturbance to caregivers or emergency services. Moreover, the proposed system could be customized for different living environments. However, there were also some potential disadvantages to consider. Using a Microsoft Kinect Sensor might limit the range of detection compared to other sensors. Furthermore, the proposed system might require technical expertise to set up and maintain. Overall, the use of reinforcement learning for automatic fall notification might require significant computational resources [18].

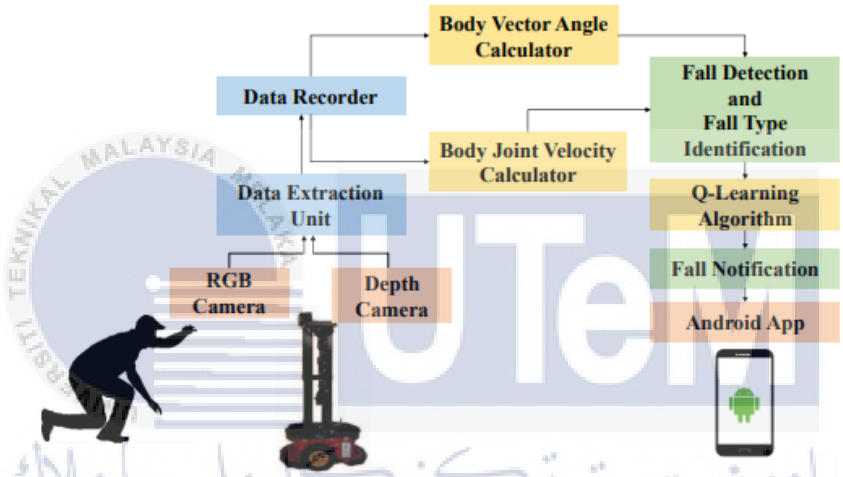


Figure 2.12: Reinforcement learning system [18].

2.4.7 Background Subtraction Algorithm

The depth images were collected from a Microsoft Kinect sensor. A background subtraction algorithm was used to subtract the background and segment daily activities and falls to train the model. The model was trained using a decision tree, and ground truthing to ensure fall confidence. The results were then processed for analysis.

The advantages of this proposed technique included its non-invasive nature, low cost, and high accuracy in detecting falls. In addition, it could be easily installed in individual

homes without requiring additional infrastructure. However, there were some limitations to this technique as well. For example, it might not be able to detect falls outside of the sensor's range or if the person falls in a way not captured by the depth images Furthermore, it might not be suitable for detecting falls in crowded environments where multiple people are present [19].

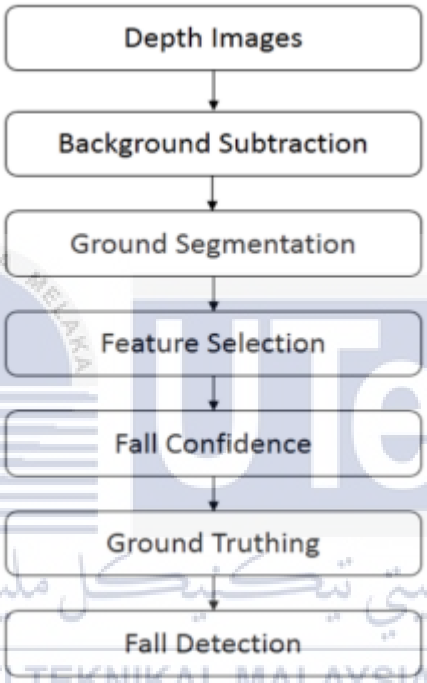


Figure 2.13: Methodology flowchart.

2.4.8 Skeleton-Based and Thresholding Algorithm

The Skeleton-based method was used to detect falls by calculating the distances of every joint with the floor plane. The algorithm stated that a fall event is detected if the distance of joints from the floor plane is less than 0.3 meters. In addition, the skeleton joint's average velocity and the algorithm's threshold value were used to determine fall status. The

system's quantitative measurement identified ideal conditions for detecting falls, and the parameters were tested to detect human conditions for different postures.

The advantages of this system included its affordability, non-invasiveness, and ability to detect falls accurately. However, some disadvantages included its reliance on the visible floor plane and its inability to detect falls in certain situations, such as when a person is sitting or lying down. Moreover, the system might produce false alarms if there are sudden movements or objects obstructing the Kinect sensor's view. [20].

Figure 2.14: Fall detection model [20].

2.4.9 Skeleton Detection Algorithm

Skeleton detection of the human figure was used as algorithm with the help of sensors in Kinect, which ran in real-time scenarios. The fall detection system could communicate the current scenario of the patient to family members and provide data logs for future analysis by doctors. The algorithm was unobtrusive, meaning it would not have any

environmental interaction with the target people, such as wearable devices or environment-mounted sensors. The hardware specifications and fall detection algorithm, including skeleton diagram extraction, merging depth images and skeleton in global coordinates, and monitoring the motion state of the human body from RGB are detailed. The simulation results demonstrated the algorithm's real-time performance with a reasonable fall detection success rate.

Advantages of the proposed method included not requiring an individual to wear any wearable devices and it being faster and less intrusive. Disadvantages included limitations in relation to individuals with mental health conditions like Alzheimer's disease as falls from those with the condition are not due to accident but as result of the condition [21].

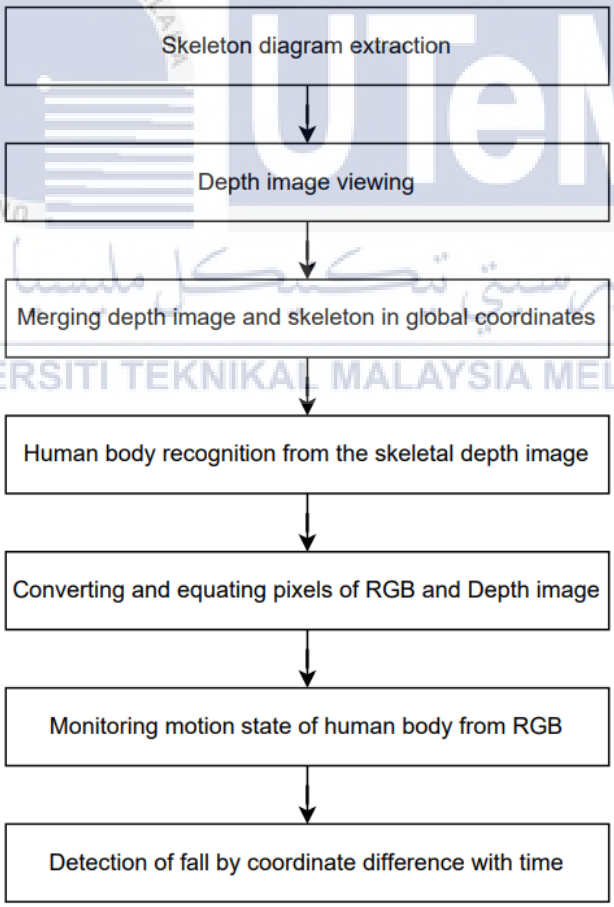


Figure 2.15: Algorithm procedures.

2.5 Sensor Comparison from Previous Work Related to the Project

Sensor selection is important to ensure the efficiency of fall detection. Each sensor plays a unique role in fall detection, contributing to the overall efficiency of the system. The choice of sensor depends on the specific requirements of the fall detection system.

Table 2.1: Sensor comparison.

No	Year	Sensor	Purpose and Specifications
1	2021	Xbox 360 Microsoft Kinect [21]	• Tracks motion and gestures that comprises of camera and infrared sensors
2	2021	Accelerometer [12]	• Measure the acceleration and determine the changes in velocity and movement.
3	2021	Gyroscope [12]	• Measure the angular velocity and provides information of rotational movement.
4	2016	Barometer sensor [9]	• Measure changes in atmospheric pressure to estimate the changes in altitude.
5	2016	Magnemometer [22]	• Measure the Earth's magnetic field to estimate the orientation of detector.

2.6 Journal Comparison from Previous Work Related to the Project

Fall detection system has been significant advancements over the years. There is a growing trend towards integrating system into smartphones, machine learning, context-aware technique, and Internet of Things (IoT).

Table 2.2: Journal comparison.

No	Year	Title	Software	Hardware	Finding
1	2021	Kinect Sensor Based Human Fall Detection System Using Skeleton Detection Algorithm [21]	• Kinect SDK	• Kinect sensor	• Human body recognition from depth image and extracted skeleton diagram • Observation on difference of coordinate with time
2	2021	Smartphone-based Online System for Fall Detection with Alert Notifications and Contextual Information of Real-life Falls [12]	• Purple Robot App	• Accelerometer • Gyroscope	• Integration of threshold-based algorithm and machine learning algorithm • Data is stored in cloud server and a web portal is developed for exploration

3	2020	A Fall Detection and Emergency Notification System for Elderly [18]	• Kinect SDK	• Kinect sensor	• Differentiation between prone position, crawl position, and kneel position • Implementation of Q-Learning algorithm
4	2020	An Elderly Fall Detection System using Depth Images [19]	• Kinect SDK	• Kinect sensor	• Calculation of background subtraction and segmentation of depth image • Cross validation technique is used
5	2020	An Analysis of Kinect-Based Human Fall Detection System [20]	• Visual Studio IDE • Kinect SDK	• Kinect sensor	• Integration of velocity and distance of joint points to floor plane • Analysis of different light intensity and distance are carried out
6	2019	Kinect-Based Platform for Movement Monitoring and Fall-Detection of Elderly People [15]	• Kinect SDK	• Kinect sensor • Computer	• Observation on acceleration of joints and angle between joints • Implementation of threshold-based algorithm

7	2019	Implementation of Fall Detection System Based on 3D Skeleton for Deep Learning Technique [16]	• Kinect SDK	• Kinect sensor	• Implementation of NVIDIA Jetson TX2 platform. Development of Neural Network with NTU RGB+D dataset on computation of fall detection
8	2019	Human Posture Recognition and Fall Detection Using Kinect V2 Camera [17]	• Kinect SDK • NITE SDK	• Kinect sensor	• Integration of Neural Network with Kinect dataset to detect different poses
9	2018	Smartfall: A Smartwatch-Based Fall Detection System using Deep Learning [11]	• IoT application	• Microsoft Band 2 • Nexus 5X smartphone	• Evaluation on deep learning algorithm and machine learning algorithm
10	2018	Fall Detection System for Elderly People using IoT and Big Data [8]	• Contiki OS	• 3D-axis accelerometer • Rasberry Pi	• Implementation of wireless Ipv6 (6LowPAN) technology and cloud services

11	2018	Design and Development of the Fall Detection System based on Point Cloud [14]	• OpenNI SDK	• Kinect sensor • Arduino microprocessor	• Extraction of point cloud image and observation on height change acceleration of point cloud, height of point cloud, and recovery time
12	2017	A Smartphone-Based Fall Detection System for the Elderly [10]	• Android Studio IDE	• Accelerometer	• Incorporation of threshold-based algorithm and kNN classifier of machine learning algorithm
13	2016	A Wearable Fall Detector for Elderly People Based on AHRS and Barometric Sensor [9]	• MATLAB	• Accelerometer • Gyroscope • Magnetometer • Barometer sensor	• The accelerometer, gyroscope, and magnetometer implement an AHRS (Attitude and Heading Reference System) that estimates the orientation of the device and provides dynamic values of vertical acceleration. • The barometer sensor measures pressure and temperature to calculate altitude, which is then fused with vertical displacement data from the

					AHRS and acceleration data using a complementary filter to obtain an accurate altitude estimation.
14	2015	Development of a Wearable-Sensor-Based Fall Detection System [7]	• Map in Web browser	• Accelerometer • SIM908 module • MCU MSP430F1611 • GPS antenna	• Threshold-based algorithm on acceleration and rotational angle • Quaternion algorithm is used to calculate the rotational angle of accelerometer coordinate
15	2014	An Analysis on Human Fall Detection using Skeleton from Microsoft Kinect [2]	• Kinect SDK	• Kinect sensor	• Floor plane detection using V-disparity method and Kinect-based plane detection method • Computation on distance of joints to floor plane, velocity, and angle of each joint • Implementation of SVM technique

16	2014	A Real-Time Fall Detection System in Elderly Care Using Mobile Robot and Kinect Sensor [4]	• Kinect SDK	• Kinect sensor • LEGO Mindstorm robot • Laptop • Nokia 6200	• Implementation of gesture recognition and speech recognition system • Development of person-following system in LEGO Mindstorm
17	2014	Fall Detection in Indoor Environment with Kinect Sensor [3]	• Kinect SDK	• Kinect sensor	• Implementation of Kalman filter to estimate height and width-depth speed
18	2014	Fall Detection for Elderly Using Anatomical-Plane-Based Representation [1]	• Kinect SDK	• Kinect sensor	• Support Vector Machine (SVM) and constraint Dynamic Time Warping (cDTW) as classification layers of fall detection
19	2012	Development of Fall Detection with Microsoft Kinect [13]	• Kinect SDK	• Kinect sensor	• Integration of position algorithm and velocity algorithm • Involvement of voice recognition system to reduce false alert notification

2.7 Summary

This chapter presented the fall detection methods categorised as smartphone-based, wearable-sensor, and vision-based systems. Depth information and RGB data were captured using an accelerometer, gyroscope, and Kinect sensor for further analysis. Deep learning, machine learning and thresholding algorithms were used to develop fall detection systems.

CHAPTER 3

METHODOLOGY

3.1 Introduction

In this project, the depth map of the scene is created from depth data. This depth map provides information about the distances of objects from the sensor. Therefore, the 3D structure of the environment is perceived. Moreover, RGB image provides visual context and enables Kinect sensor to recognize and track objects and people based on the appearance.

On the other hand, data analysis and storage involve processing and organizing the collected data from the Kinect sensor. Data analysis techniques such as computer vision algorithms are applied to extract useful information from the raw data. The processed data can be stored in a database or a file system for future reference or analysis. In addition, real-time monitoring allows system to continuously capture and process data in real-time. Hence, immediate feedback and interaction from the user. This enables real-time tracking of people or objects, gesture recognition, and responsive feedback within the system. By analyzing the depth data and tracking the movements of individuals, the Kinect sensor can also identify specific patterns or poses associated with a fall event. Once a fall is detected, the system will generate message or alarm notification to inform caregivers or emergency services.

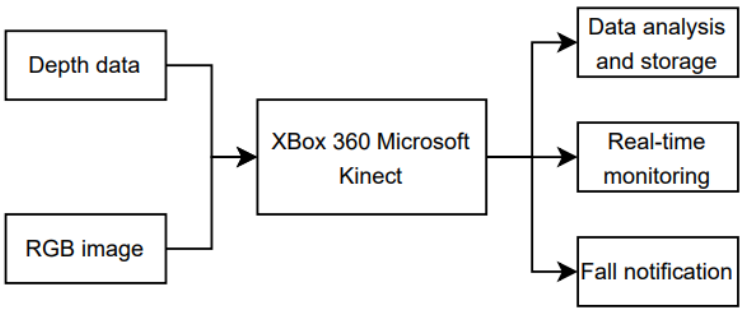


Figure 3.1: Block diagram.

3.2 Hardware

- Kinect sensor

  Kinect sensor is a depth-sensing camera device developed by Microsoft. It was originally created as an accessory for the Xbox gaming console but has also found applications in other fields, such as robotics, healthcare, and computer vision research.

Figure 3.2: Xbox 360 Microsoft Kinect.

- Kinect sensor utilizes a combination of cameras and infrared sensors to capture depth information and track human movement. It consists of some main components:

  1. RGB Camera

     The Kinect sensor includes a traditional RGB camera that captures color images like a regular camera. This camera is useful for capturing visual information and can be used for applications like gesture recognition or video conferencing.

  2. Depth Sensor

     The Kinect sensor employs an infrared depth sensor that projects an infrared pattern into the scene and measures the time it takes for the pattern to bounce back. This allows the sensor to calculate the distance of objects from the camera, generating a depth map of the environment.

3. Infrared Projector

The Kinect sensor emits an infrared light pattern that is invisible to the human eye. This pattern combined with the depth sensor, allows the sensor to accurately measure distances and create a detailed depth image.

4. Microphone Array

The Kinect sensor includes an array of microphones that capture audio from the surrounding environment. This enables applications to incorporate voice commands and perform speech recognition.

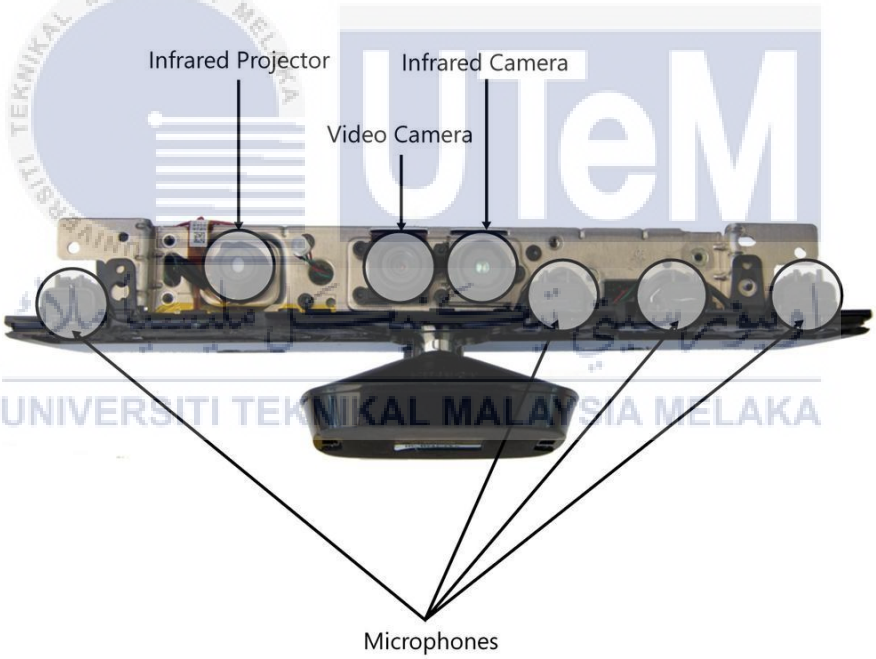


Figure 3.3: Sensor arrangement of Xbox 360 Microsoft Kinect

3.3 Software

- Microsoft Kinect SDK (Software Development Kit) is a software package provided by Microsoft for developing applications that utilize the capabilities of the Kinect

sensor. The Kinect SDK enables developers to create applications that can track and interpret human body movement, recognize faces and voices, and capture depth information. There are several features of the Kinect SDK:

1. Skeletal Tracking

   The SDK provides real-time tracking of human body movements, allowing developers to track the position and orientation of individual joints and create applications that respond to gestures and actions.

2. Voice Recognition

   The Kinect SDK includes speech recognition technology, enabling developers to build applications that can understand and respond to voice commands.

3. Gesture Recognition

   The SDK includes built-in gesture recognition capabilities, allowing developers to recognize predefined gestures such as swipes, waves, and specific poses. This enables users to interact with applications through natural hand and body movements.

- Microsoft Visual Studio is an integrated development environment (IDE) that provides a comprehensive set of tools for building software applications. It is widely used by developers to create a wide range of applications, including desktop, web, mobile, cloud, and gaming applications. There are several features of Microsoft Visual Studio:

  1. Code Editor

     Visual Studio offers a powerful and feature-rich code editor that supports multiple programming languages such as C#, C++, Python, JavaScript, and

more. It provides features like syntax highlighting, code completion, context-aware code suggestions, and refactoring tools to improve productivity and code quality.

2. Debugging

   Visual Studio includes advanced debugging capabilities that help developers identify and fix issues in their code. It offers features like breakpoints, step-through debugging, watch windows, and real-time code execution analysis, making it easier to locate and resolve bugs.

3. Mobile Development

   Visual Studio supports mobile application development for platforms like Android, iOS, and Windows. It provides tools like Xamarin for cross-platform development, iOS and Android emulators, and integration with mobile-specific services like push notifications and app distribution.

3.3.1 Flowchart

Figure 3.4 shows the system flowchart. Firstly, depth and RGB data are acquired from the Kinect sensor. Skeletons are detected and joints such as heads, shoulder centre, hip centre, left ankle, and right ankle are extracted. Initial head-to-ankle distance is observed. If the initial head-to-ankle distance is zero, y-coordinates value from head to ankle is calculated. The calculated value is then multiplied with height threshold factor to obtain distance threshold value. If the initial head-to-ankle distance is not zero, current head-to-ankle distance is calculated. Absolute difference between initial and current head-to-ankle distance is calculated and compared with distance threshold value. Another position threshold value is compared with y-coordinates value of each joints. “Fall is detected” is displayed when absolute difference between initial and current head-to-ankle distance is

greater than distance threshold value, and y-coordinates values of each joints are lower than position threshold. Instead, “Fall is not detected” is displayed. In addition, if the y-coordinates value of one hand or both hand is greater than y-coordinates value of head, “Help Command Detected!! Check for help” is displayed.

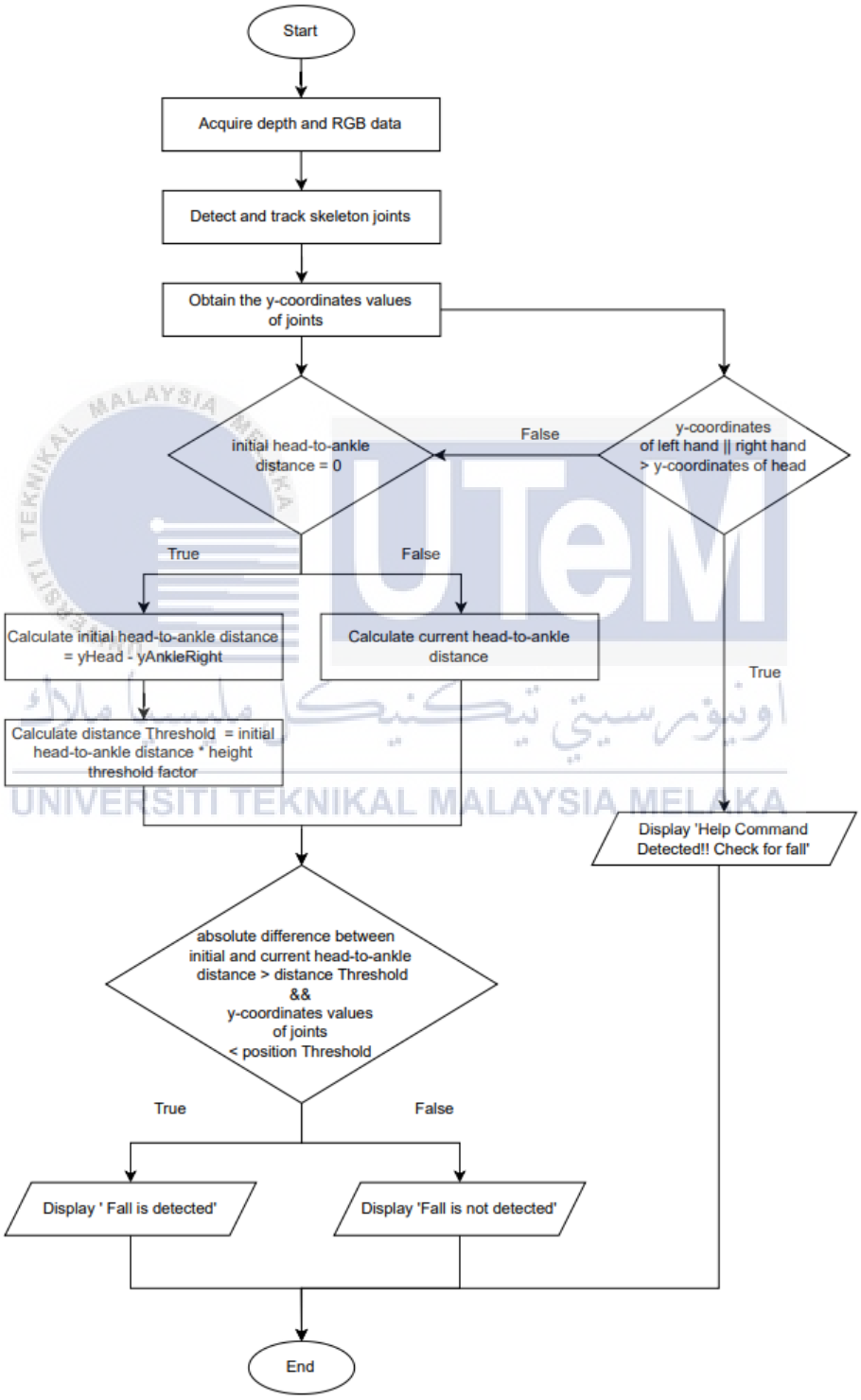


Figure 3.4: System flowchart.

3.3.2 Fall Algorithm

In the pseodocode, initial head-to-ankle distance is determined. If the initial head-to-ankle distance is zero, y-coordinates value from head to ankle is calculated. It means that this is the first frame where the person is detected. The calculated value is then multiplied with height threshold factor value of 0.7 to obtain distance threshold value.

If the initial head-to-ankle distance is not zero, current head-to-ankle distance is calculated. It means that the person has been detected in previous frames, so the initial head-to-ankle distance and the distance threshold have already been calculated. Absolute difference between initial and current head-to-ankle distance is calculated and compared with distance threshold value. Another position threshold value of 0.3 is compared with y-coordinates value of each joints.

Fall event is considered when absolute difference between initial and current head-to-ankle distance is greater than distance threshold value, and y-coordinates values of each joints are lower than position threshold. Instead, safe event is considered.


```
Function IsPersonFalling(skeleton):
    yHead = skeleton. Head.Y
    yShoulderCenter = skeleton. ShoulderCenter.Y
    yHipCenter = skeleton. HipCenter.Y
    yAnkleLeft = skeleton. AnkleLeft.Y
    yAnkleRight = skeleton. AnkleRight.Y


If Head_Ankel1 == 0:
    Head_Ankel1 = abs (yHead -yAnkelRight)
    thr= Head_Ankel1 * 0.7
Else
    Head_Ankel2= abs (yHead-yAnkelRight)

If (abs (Head_Ankel1- Head_ Ankel2) > thr) AND (yHead < Threshold) AND
   (yShoulderCenter < Threshold):
   If (yHipCenter < Threshold) AND (yAnkelLeft < Threshold) AND (yAnklRight <
      Threshold):
      Return "Fall is detected"
   Else
      Return "Safe"
```

Figure 3.5: Fall algorithm.

3.3.3 Gesture Recognition

The system can only detect falls within the Kinect's field of view, therefore gesture recognition is developed to improve fall detection accuracy. The gesture is programmed to detect the action of one hand or both hands being raised above the head. The Kinect sensor continually analyses the movements of the person in the field of view. When a person raises one hand or both hands above the head, the system will interpret it as a potential fall event. Upon detecting the gesture, the system triggers the notification to check for fall event.

3.4 Experimental Setup

Proprietary connecter of Xbox 360 Microsoft Kinect is plugged into corresponding port on the USB adapter. End of USB adapter is plugged into laptop. Microsoft Kinect is then connected to power supply and tested with Kinect developer toolkit. Figure 3.6 shows that the Kinect sensor is positioned above and is facing towards the person. The measurement from the person to the Kinect sensor represents the horizontal distance between the Kinect sensor and the person. The vertical measurement pointing downwards from the Kinect sensor indicates the height at which the Kinect sensor is placed above the ground.

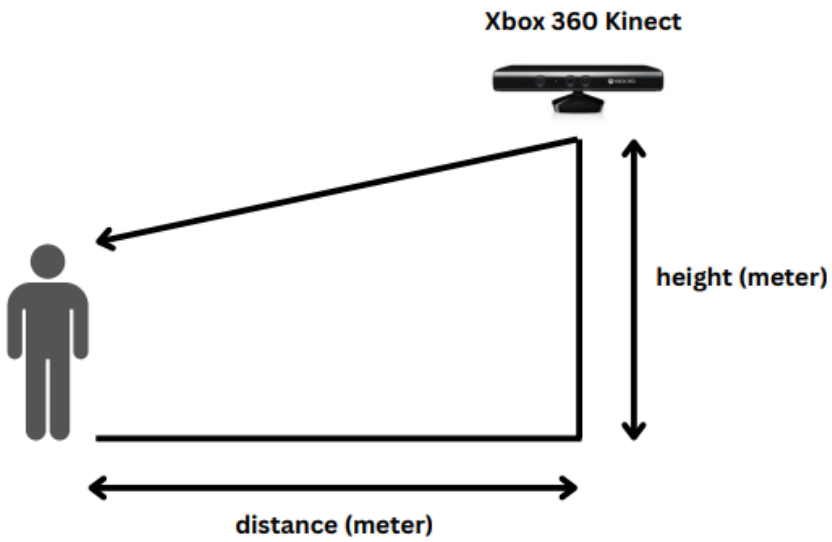
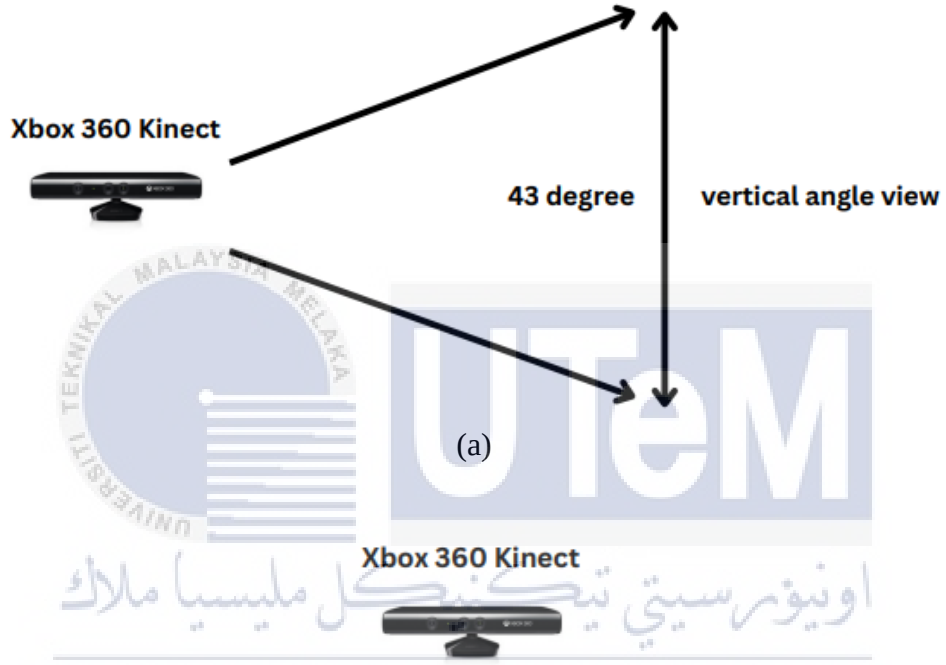
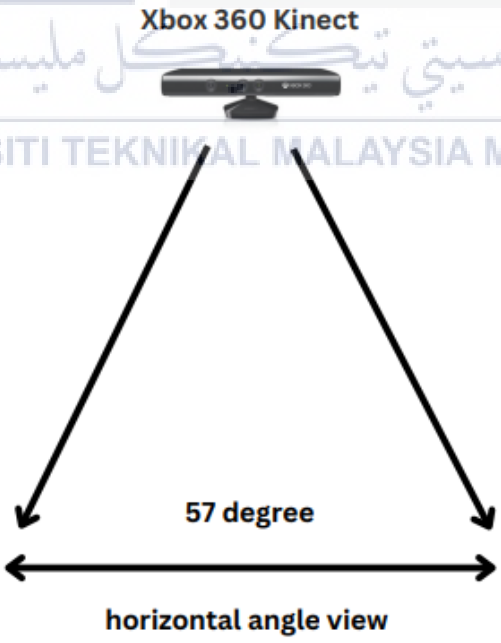


Figure 3.6: Experimental setup.

Figure 3.7 (a) shows the vertical field of view of Kinect sensor where the sensor can perceive objects within a 43-degree vertical range in front of it. Figure 3.7 (b) shows the angle at which the Kinect sensor can detect motion across the horizontal plane. The sensor must be placed in such a way that the area where motion is to be detected falls within this 57-degree field.



(a)



(b)

Figure 3.7: (a) Vertical angle view (b) Horizontal angle view of Kinect sensor.

Figure 3.8 shows the floor plan of a house with markings indicating the placement and field of view of a Kinect sensor. The Kinect sensor is placed in two potential locations within the house, as indicated by the blue circles. The blue triangles indicate the extent of the area covered by the sensor's camera.

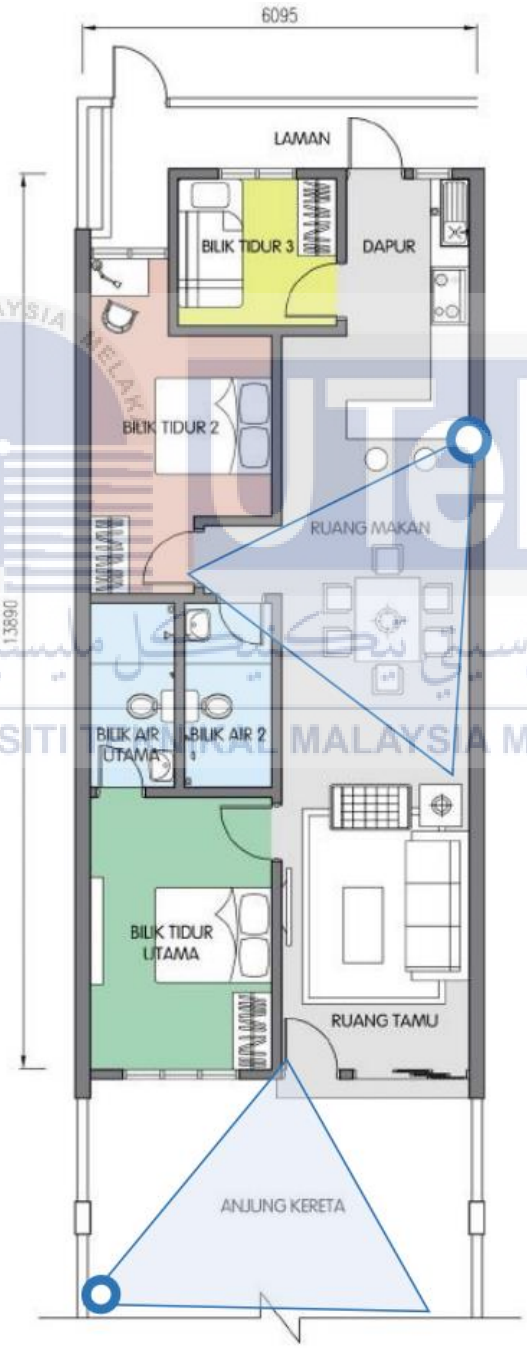


Figure 3.8: Kinect sensor location.

3.5 Summary

This chapter presented the use of Microsoft Kinect SDK, Microsoft Visual Studio, and Xbox 360 Microsoft Kinect. This chapter also presented the algorithm and flowchart of fall detection system.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter focuses on the results and analysis on the implementation of the coding and application of fall detection. The simulation results are evaluated and validated to determine the possible fall event.

4.2 Result and Analysis

Fall detection is simulated and evaluated throught different scenarios. Fall detection accuracy is analysed from the aspects of distances, heights and lighting conditions.

Figure 4.1: Real-time experimental setup.

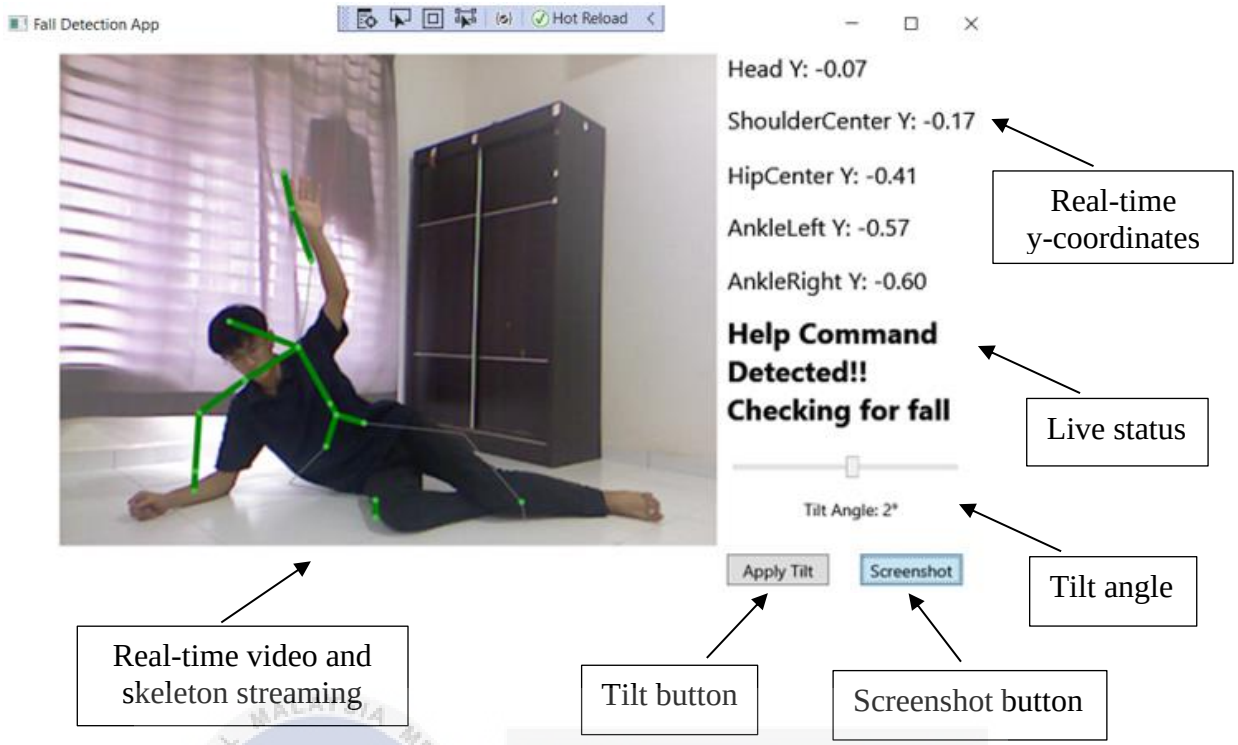


Figure 4.2: Output window.

4.2.1 Fall Detection Accuracy of Different Distances

Fall detection accuracy of different distances is observed and analysed. Distance of 0.5m, 1.0m, 1.5m, 2.0m and 2.5m are evaluated. At extremely close range of 0.5 meters, the fall detection accuracy is the lowest, which is 20%. The Kinect sensor struggles with accurate skeleton tracking. The limited field of view and potential depth perception distortions at such a close distance result in incomplete or inaccurate data, leading to the lowest fall detection accuracy.

Table 4.1: Fall detection accuracy of different distances.

Distance of Kinect Sensor (m)	No. of simulated falls scenarios	No. of detected falls scenarios	No. of undetected falls scenarios	Accuracy (%)
0.5	5	1	4	20
1.0	5	4	1	80

1.5	5	5	0	100
2.0	5	5	0	100
2.5	5	5	0	100

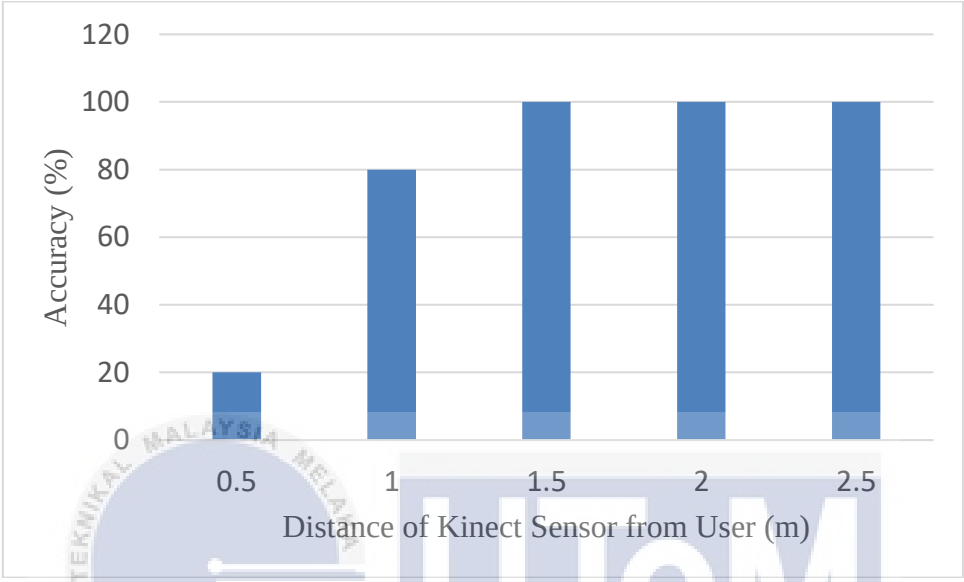


Figure 4.3: Fall detection accuracy of different distances.

Moving slightly further to 1.0 meters, the fall detection accuracy increases to 80%. The sensor has a better field of view at this distance, allowing for more effective and complete skeleton tracking. Thus, the accuracy of fall detection is higher compared to 0.5 meters. At a more optimal range of 1.5 meters, the Kinect sensor provides a comprehensive view of the body and captures detailed movements and postures accurately. The sensor achieves a good balance at 1.5 meters, leading to fall detection accuracy of 100%.

As the distance increases further to 2.0 meters and 2.5 meters, the resolution and detail in the captured data increase. The increased distance leads to finer details necessary for accurate fall detection.

In short, the result of lower accuracy at 0.5 meters with the Kinect sensor can be attributed to its limited field of view and depth perception issues at close range, leading to incomplete or distorted skeleton tracking. Slightly longer distances provide a more optimal

range for accurate skeleton tracking and fall detection, while accuracy increaces at longer ranges due to increased detail in the captured data. If the distance increases further to the limit of the effective range of Kinect sensor, the accuracy of fall detection should be decreased due to the sensor captures less detail, making it more challenging to distinguish falls from other movements accurately.

Figure 4.4: Fall scenarios at different distances from the Kinect sensor
(a) 0.5m (b) 1.0m (c) 1.5m (d) 2.0m (e) 2.5m

4.2.2 Fall Detection Accuracy of Different Heights

Fall detection accuracy of different heights is observed and analysed. Several observed fall scenarios including fall to the left side, fall to the right side, fall to the front, fall to the back, and fall while sitting. Throughout the evaluation of all heights of Kinect sensor, the fall detection accuracy achieves 100% in all fall scenarios. The system identifies every fall scenario and accurately classifies without false positives or negatives. The real-world application of the system is taken into consideration, although the results obtain 100% detection accuracy. Factors such as furniture placement and background clutter may affect the performance and accuracy of the system.

Table 4.2: Fall detection accuracy of different heights.

Height of Kinect Sensor (m)	No. of simulated falls scenarios	No. of detected falls scenarios	No. of undetected falls scenarios	Accuracy (%)
1.0	5	5	0	100
1.5	5	5	0	100
2.0	5	5	0	100

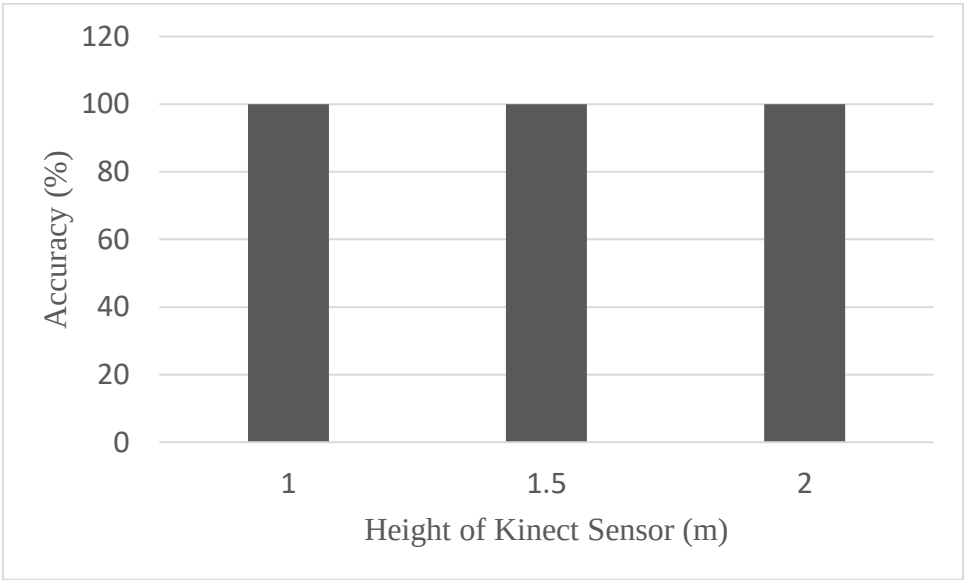


Figure 4.5: Fall detection accuracy of different heights.

Figure 4.6: Different fall scenarios at height of 1.0m. (a) fall to the left side (b) fall to the right side (c) fall to the front (d) fall to the back (e) fall while sitting

Figure 4.7: Different fall scenarios at height of 1.5m. (a) fall to the left side (b) fall to the right side (c) fall to the front (d) fall to the back (e) fall while sitting

Figure 4.8: Different fall scenarios at height of 2.0m. (a) fall to the left side (b) fall to the right side (c) fall to the front (d) fall to the back (e) fall while sitting

4.2.3 Fall Detection Accuracy of Different Lighting Conditions

The effectiveness of the Kinect sensor to detect fall incident was also conducted in different lighting conditions. The fall detection accuracy of 100% is obtained throughout the evaluation of all lighting conditions.

Although, this experiment was not necessary, as the sensor used to detect motion is an infrared sensor for depth-sensing which allows it to work in various lighting conditions, including in complete darkness.

Nevertheless, the experiment is deemed necessary to fortified what is already been known. This eliminates any questions in regards to the affect of lighting, i.e. rapid changes in lighting conditions, such as sudden transitions from light to dark or vice versa. These changes can temporarily disrupt the sensor to track movements accurately. Moreover, strong backlighting, where the primary light source is behind the person being tracked, can lead to silhouette distortion and reduced accuracy of sensor.

Table 4.3: Fall detection accuracy of different lighting conditions.

Lighting condition (lux)	No. of simulated falls scenarios	No. of detected falls scenarios	No. of undetected falls scenarios	Accuracy (%)
65	5	5	0	100
40	5	5	0	100
15	5	5	0	100
10	5	5	0	100
0	5	5	0	100

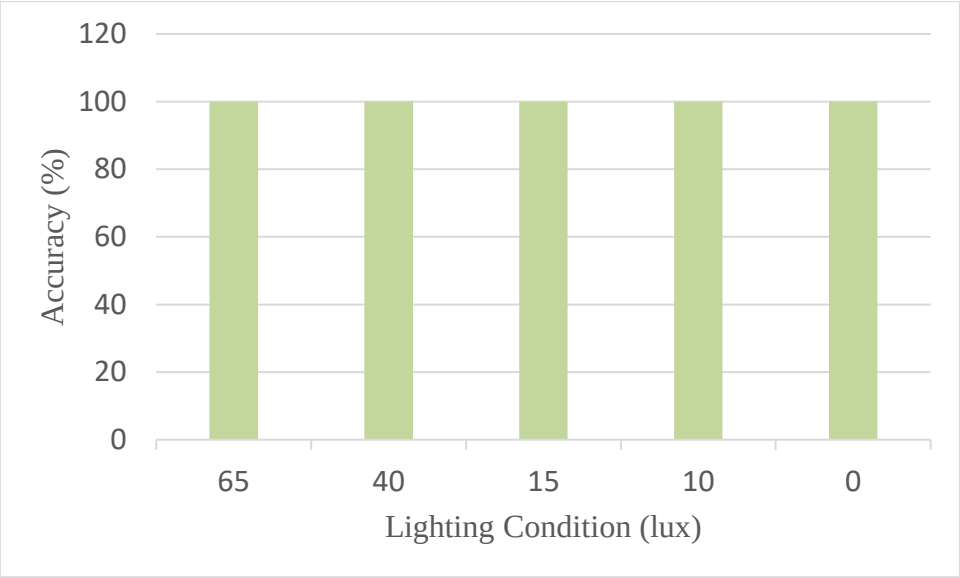
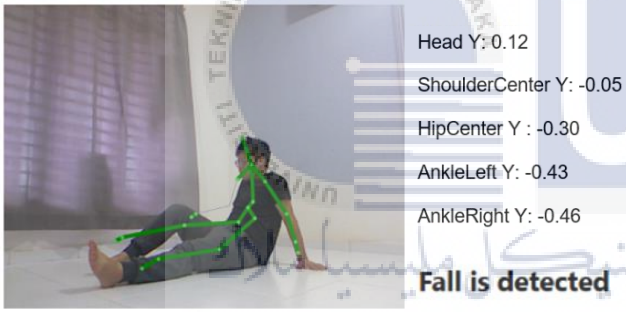
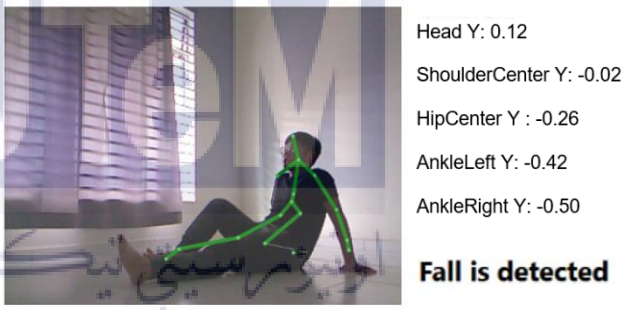


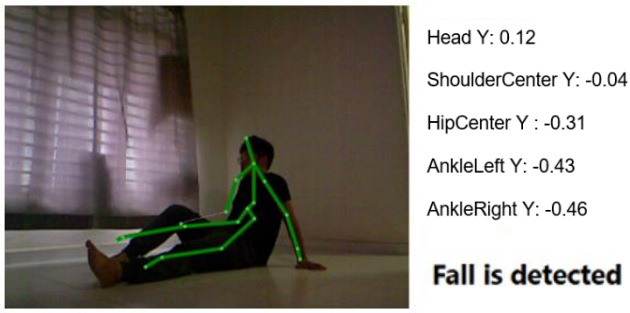
Figure 4.9: Fall detection accuracy of different lighting conditions.



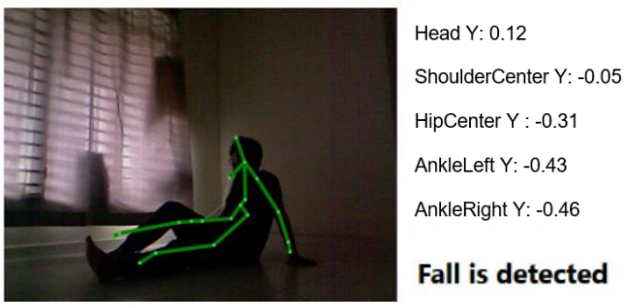
(a)



(b)



(c)



(d)

(e)

Figure 4.10: Fall scenarios at different lighting conditions.
(a) 65 lux (b) 40 lux (c) 15 lux (d) 10 lux (e) 0 lux

4.2.4 "HELP" Gesture Detection

The system includes "HELP" gesture detection. "Help Command Detected!! Check for fall" is displayed when one hand or both hands is raised above head. Gesture recognition is introduced to reduce the false negatives of fall detection.

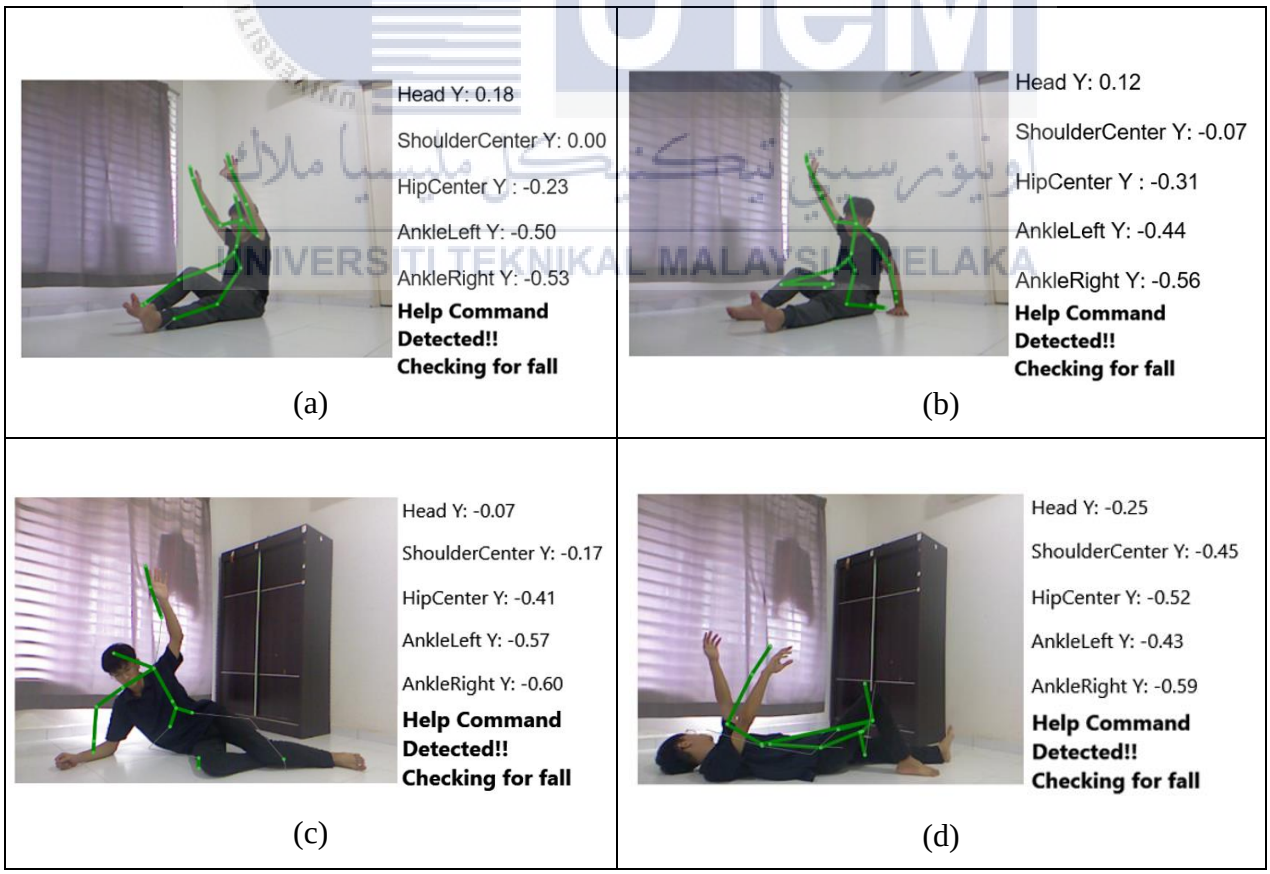


Figure 4.11: Gesture recognition. (a) Both hands (b) One hand

4.2.5 Validation of Non-Fall Detection Postures

Scenarios of standing, kneeling, and crawling were also tested to check for false-positive alarms. An upright standing posture is classified as “Safe” event and “Fall is not detected”. A true-positive fall event is detected in the kneeling and crawling postures.

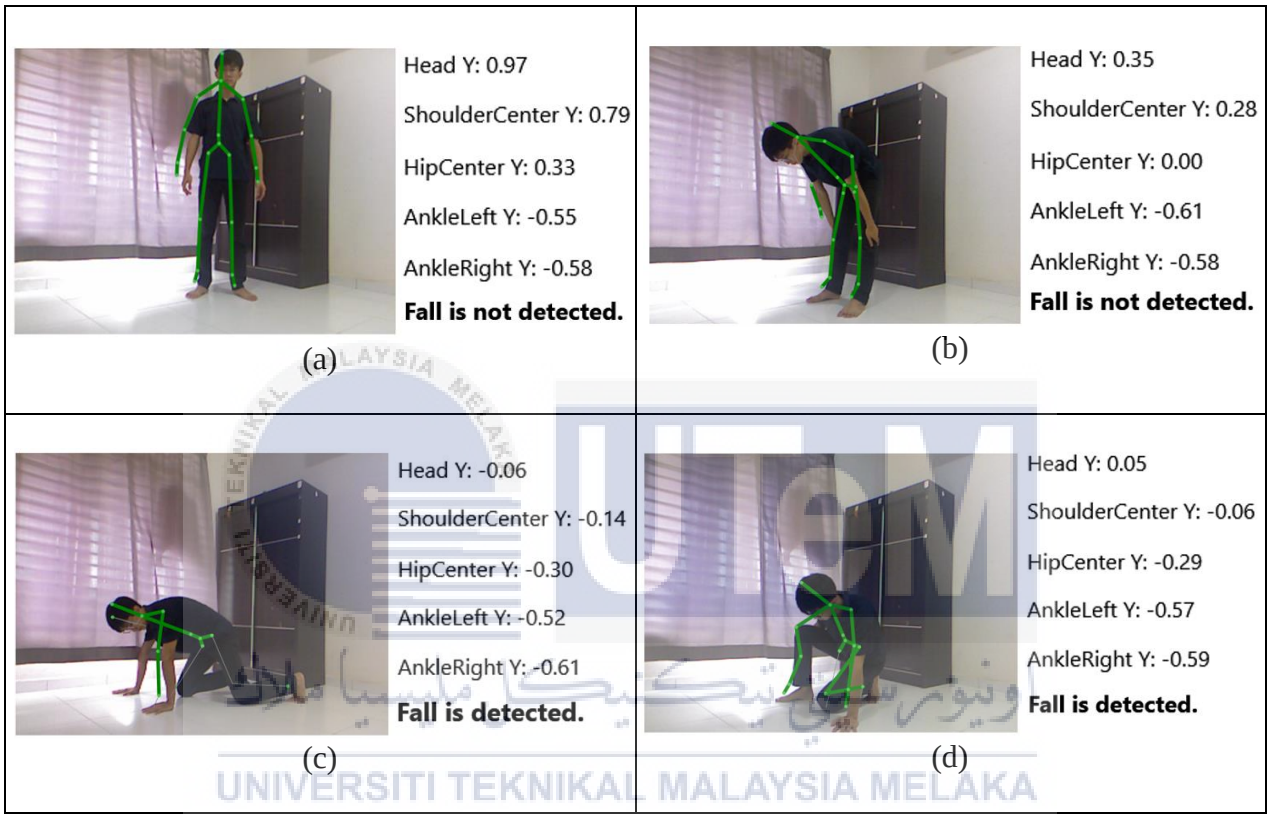


Figure 4.12: Testing scenarios. (a) standing (b) bending over (c) crawling (d) kneeling

4.2.6 Fall Detection when There Is More Than One Person

This validates the fall detection when there are more than one person within the field of view. In figure 4.11(a), when one person is in a fall posture while the another person is still standing, y-coordinates of joints of falling person is shown and “Fall is detected” is displayed. Kinect sensor can track up to two individuals simultaneously in the field of view. The accuracy of skeleton tracking starts to degrade when there is more than two person. The

system have difficulty to distinguish between the movements of multiple individuals in crowded or complex scenes.

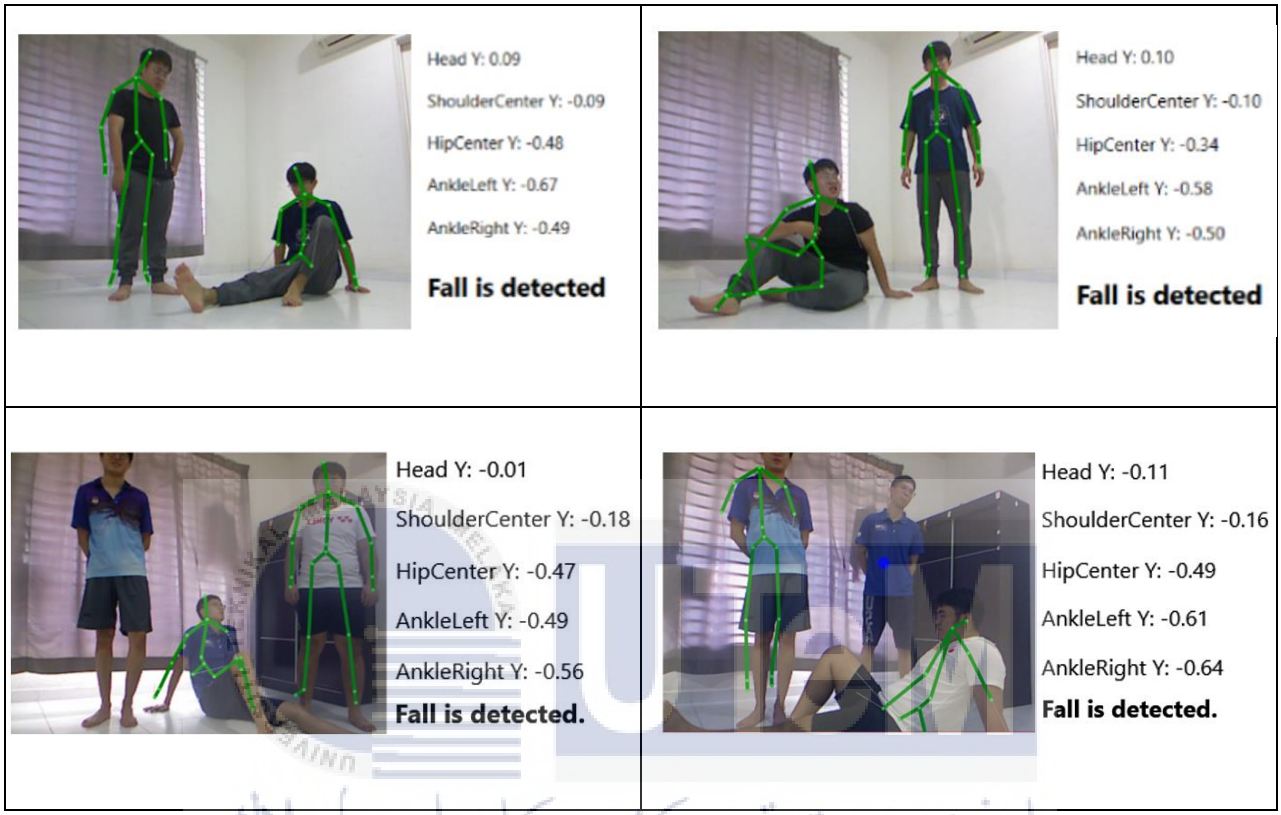


Figure 4.13: Fall scenarios when there are more than one person.

4.3 Summary

This chapter presented the result and analysis of differenet fall scenarios in the aspects of distance, height and lighting condition. This chapter also presented the use of gesture recognition in the detection system.

CHAPTER 5

CONCLUSION

5.1 Conclusion

In conclusion, using a Kinect sensor, a fall detection system is an innovative and effective approach to detect and monitor falls in various situations. The Kinect sensor's depth sensing and skeletal tracking are used to analyze and determine fall events accurately. In addition, the system can capture and analyze real-time human movement data. The Kinect Developer Toolkit v1.8 provides a comprehensive set of tools and libraries to seamlessly access and utilize the Kinect sensor's features within Visual Studio 2019.

The system extracts the position of joints such as the head, shoulders, hips, and ankles by analysing the skeleton data. The system computes the change in height of the tracked joints by calculating the head-to-ankle distance. If the initial head-to-ankle distance is zero, y-coordinates value from head to ankle is calculated. It means that this is the first frame where the person is detected. The calculated value is then multiplied with height threshold factor value of 0.7 to obtain distance threshold value.

If the initial head-to-ankle distance is not zero, current head-to-ankle distance is calculated. It means that the person has been detected in previous frames, so the initial head-to-ankle distance and the distance threshold have already been calculated. Absolute difference between initial and current head-to-ankle distance is calculated and compared with distance threshold value. Another position threshold value of 0.3 is compared with y-coordinates value of each joints.

Fall event is considered when absolute difference between initial and current head-to-ankle distance is greater than distance threshold value, and y-coordinates values of each

joints are lower than position threshold. Instead, safe event is considered. “Help Command Detected!! Check for fall” is displayed when one hand or both hands is raised above head.

Although Visual Studio 2019 and Kinect Developer Toolkit v1.8 are older versions of the development tools, they can still provide a solid foundation to implement a functional fall detection system. Kinect-based fall detection system offers a non-intrusive method of monitoring individuals, without the need of wearing any devices. This can be beneficial for seniors who might forget or be unwilling to wear monitoring devices. Moreover, fall detection system can be developed in Visual Studio 2019 to support advanced programming capabilities and facilitate the integration of Kinect SDK, and hence enabling developers to create more efficient and reliable applications.

Lastly, there are challenges of the fall detection system such as the limited field of view of the Kinect sensor and the potential for false positives and negatives. The effectiveness of fall detection system can be influenced by environmental factors like space layout. With advancements in AI and machine learning, the accuracy and efficiency of fall detection system can be significantly improved.

5.2 Recommendations for Future Study

The fall detection system has the potential to enhance the safety of older people in various environments. There are several recommendations to enhance the efficiency and adaptability of the system. Firstly, investigation of advanced algorithms for more accurate gesture and fall recognition. Machine learning and deep learning techniques can be incorporated to significantly improve the system to differentiate between normal activities and falls, hence reducing false positives and negatives.

Furthermore, integration of Kinect system with Internet of Things (IoT) devices for a more comprehensive monitoring approach. For example, connection of fall detection

system with smart home system to automate alerts and responses, such as locking doors, turning off appliances, or adjusting the lighting during a fall event.

Moreover, combination of Kinect with wearable devices or environmental sensors to enhance sensor capabilities. It could provide a more holistic solution to fall detection and activity monitoring. In addition, study of data privacy and security concerns when the system is connected to the internet or other devices.

Other than that, development of user-friendly interfaces for users and caregivers. Research can be conducted into the design of interfaces that are intuitive and easy to use for people. With that, engagement in cross-disciplinary research involving healthcare professionals, engineers, and user experience designers to ensure that the system is medically relevant.

On the other hand, one of the recommendations is conduct of extensive clinical trials and real-world testing to validate the effectiveness, reliability, and practicality of the system in various environments such as homes, hospitals, and eldercare facilities. Additionally, examination of the potential for integrating Kinect-based system with telehealth services. This could facilitate remote monitoring and consultation to be more accessible.

Moreover, integration of the Kinect system to be personalized for individual users. This includes adjusting sensitivity, customizing alerts and the system to recognize specific health conditions or mobility issues. Lastly, investigation of accessibility and affordability in different regions including low-income countries to enhance the global applicability and impact.

In short, future studies can significantly contribute to the development of more advanced, reliable, and user-friendly Kinect-based fall detection system, improving the quality of life and safety for older people at risk of falls.

REFERENCES

[1] R. Alazrai, A. Zmily, and Y. Mowafi, 'Fall Detection for Elderly Using Anatomical-Plane-Based Representation', *36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2014.

[2] T. T. H. Tran and J. Morel, 'An Analysis on Human Fall Detection Using Skeleton from Microsoft Kinect', *IEEE Fifth International Conference on Communications and Electronics (ICCE)*, Oct. 2014.

[3] V. Bevilacqua, N. Nuzzolese, and D. Barone, 'Fall Detection in Indoor Environment with Kinect sensor', *IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA) Proceedings*, 2014.

[4] Z. A. Mundher and J. F. Zhong, 'A Real-Time Fall Detection System in Elderly Care Using Mobile Robot and Kinect Sensor', *International Journal of Materials, Mechanics and Manufacturing*, vol. 2, no. 2, pp. 133–138, May 2014.

[5] R. Igual, C. Medrano, and I. Plaza, 'Challenges, Issues and Trends in Fall Detection Systems', BioMedical Engineering Online, Jul. 06, 2013.

[6] G. Koshmak, A. Loutfi, and M. Linden, 'Challenges and Issues in Multisensor Fusion Approach for Fall Detection', *Journal of Sensors*, vol. 2016. Hindawi Publishing Corporation, 2016.

[7] F. Wu, H. Zhao, Y. Zhao, and H. Zhong, 'Development of a Wearable-Sensor-Based Fall Detection System', *International Journal of Telemedicine and Applications*, 2015.

[8] D. Yacchirema, J. S. D. Puga, C. Palau, and M. Esteve, 'Fall Detection System for Elderly People using IoT and Big Data', *9th International Conference on Ambient Systems, Networks and Technologies*, pp. 603–610, 2018.

[9] P. Pierleoni *et al.*, 'A Wearable Fall Detector for Elderly People Based on AHRS and Barometric Sensor', *IEEE Sensors Journal*, vol. 16, no. 17, pp. 6733–6744, Sep. 2016.

[10] P. Tsinganos and A. Skodras, 'A Smartphone-Based Fall Detection System for the Elderly', *10th International Symposium on Image and Signal Processing and Analysis Proceedings*, IEEE, 2017.

[11] T. R. Mauldin, M. E. Canby, V. Metsis, A. H. H. Ngu, and C. C. Rivera, 'Smartfall: A Smartwatch-Based Fall Detection System using Deep Learning', *Sensors (Switzerland)*, vol. 18, no. 10, Oct. 2018.

[12] Y. Harari, N. Shawen, C. K. Mummidisetty, M. V. Albert, K. P. Kording, and A. Jayaraman, 'A Smartphone-Based Online System for Fall Detection with Alert Notifications and Contextual Information of Real-Life Falls', *Journal of Neuroengineering and Rehabilitation*, vol. 18, no. 1, Dec. 2021.

[13] C. Kawatsu, J. X. Li, and C. J. Chung, 'Development of a Fall Detection System with Microsoft Kinect', *Robot Intelligence Technology and Applications*, Springer Verlag, pp. 623–630, 2012.

[14] Y. F. Peng, J. J. Peng, J. P. Li, P. T. Yan, and B. Hu, 'Design and Development of the Fall Detection System based on Point Cloud', *Procedia Computer Science*, pp. 271–275, 2019.

[15] J. Barabas, T. Bednar, and M. Vychlopen, 'Kinect-Based Platform for Movement Monitoring and Fall-Detection of Elderly People', *12th International Conference on Measurement, Smolenice, Slovakia*, pp. 199–202, 2019.

[16] T. H. Tsai and C. W. Hsu, 'Implementation of Fall Detection System Based on 3D Skeleton for Deep Learning Technique', *IEEE Access*, vol. 7, pp. 153049–153059, 2019.

[17] Y. Xu, J. Chen, Q. Yang, and Q. Guo, 'Human Posture Recognition and Fall Detection Using Kinect V2 Camera', in *Chinese Control Conference (CCC)*, 2019.

[18] T. Kalinga, C. Sirithunge, A. G. Buddhika, P. Jayasekara, and I. Perera, 'A Fall Detection and Emergency Notification System for Elderly', *6th International Conference on Control, Automation and Robotics*, Institute of Electrical and Electronics Engineers Inc., pp. 706–712, Apr. 2020.

[19] M. Bundele, H. Sharma, M. Gupta, and P. S. Sisodia, 'An Elderly Fall Detection System using Depth Images', *5th IEEE International Conference on Recent Advances and Innovations in Engineering*, Institute of Electrical and Electronics Engineers Inc., Dec. 2020.

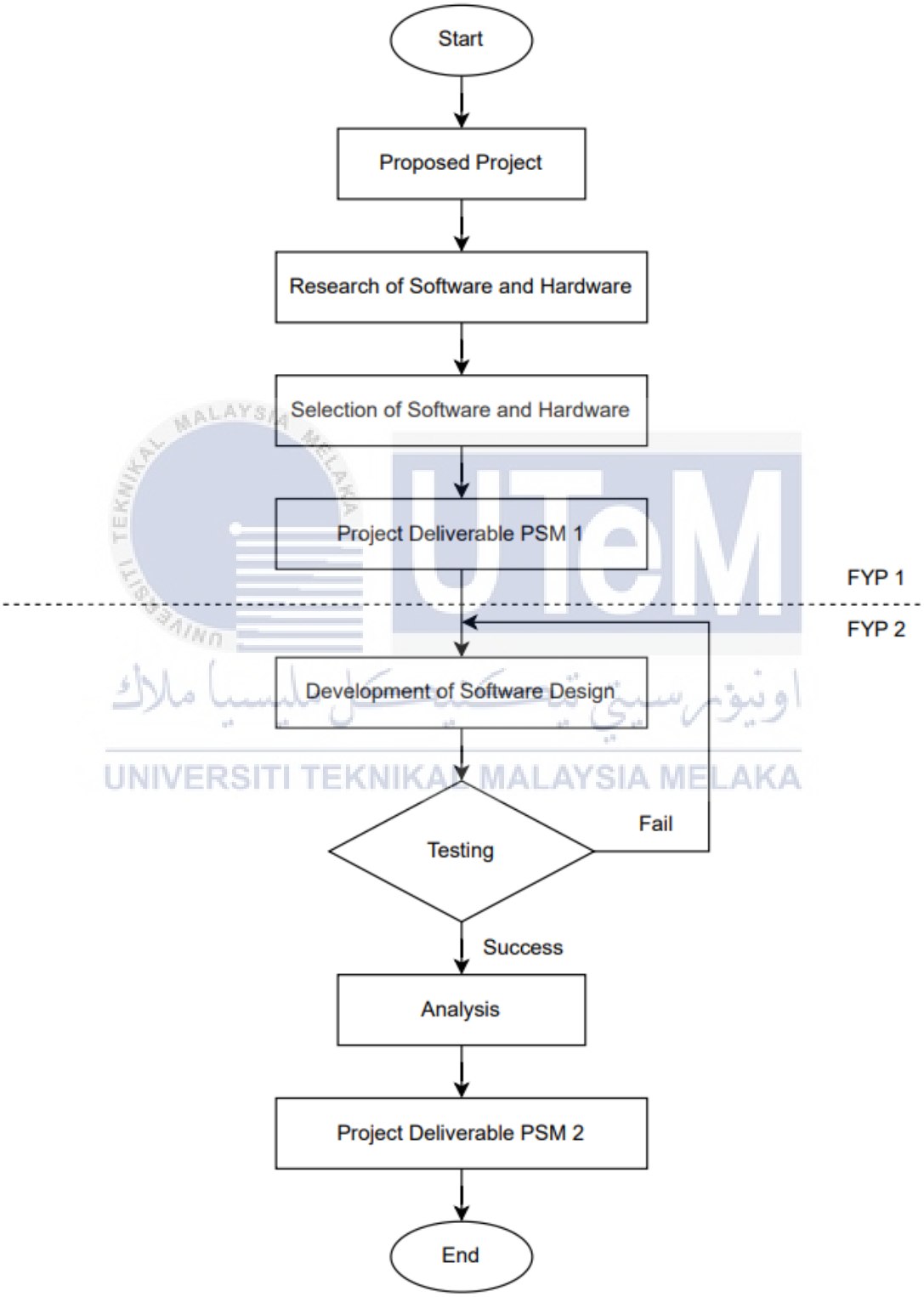
[20] N. A. Saidin and S. A. A. Shukor, 'An Analysis of Kinect-Based Human Fall Detection System', *IEEE 8th Conference on Systems, Process and Control*, Institute of Electrical and Electronics Engineers Inc., pp. 220–224, Dec. 2020.

[21] M. Praveen Kumar, M. Sudhakaranr, and Dr. R. Seyezhai, 'Kinect Sensor Based Human Fall Detection System Using Skeleton Detection Algorithm', *International Conference on Engineering Innovations and Solutions*, 2021.

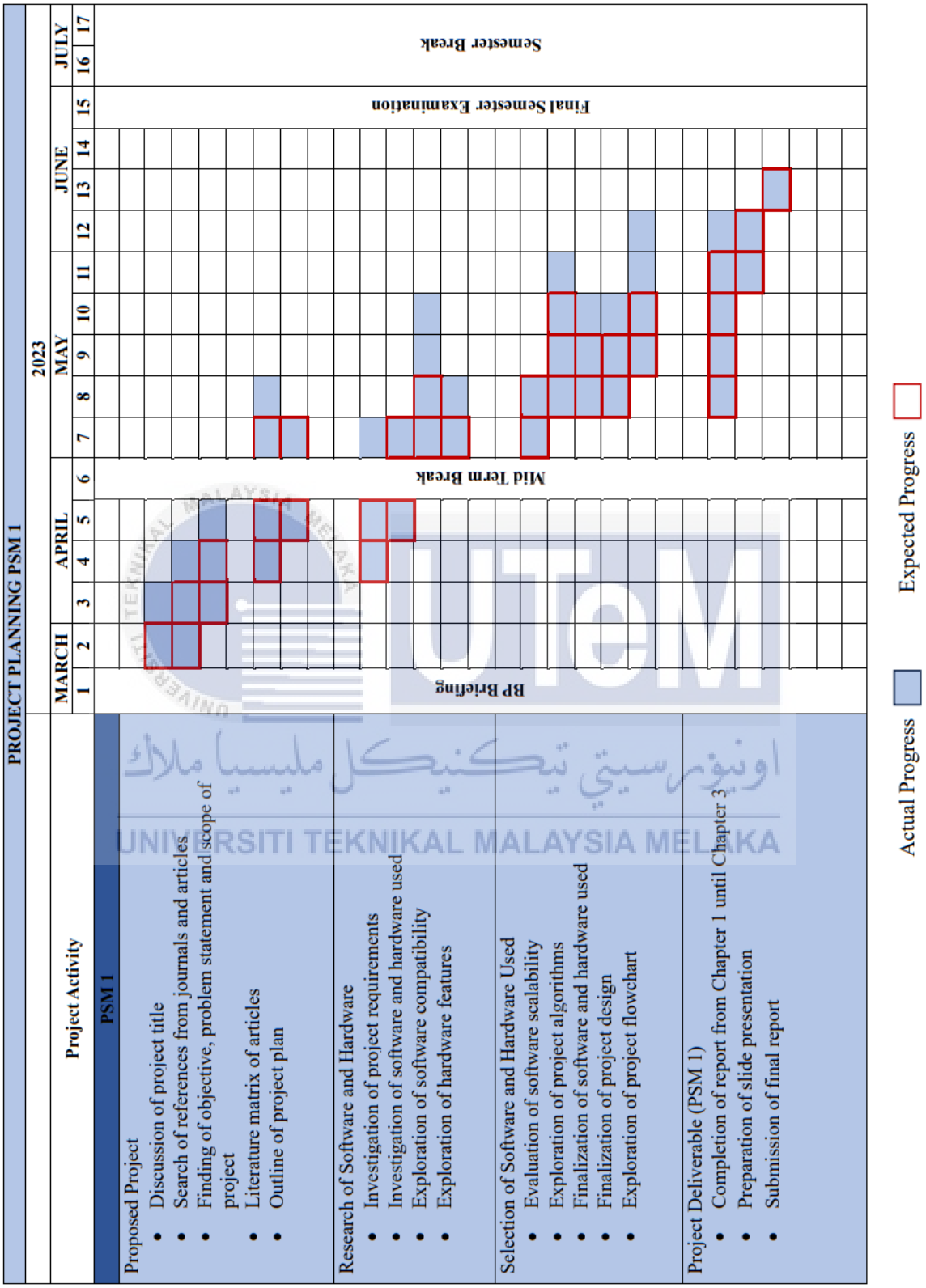
[22] T. J. Shi, X. M. Sun, Z. H. Xia, L. Y. Chen, and J. X. Liu, 'Fall Detection Algorithm Based on Triaxial Accelerometer and Magnetometer', May 2016.

APPENDICES

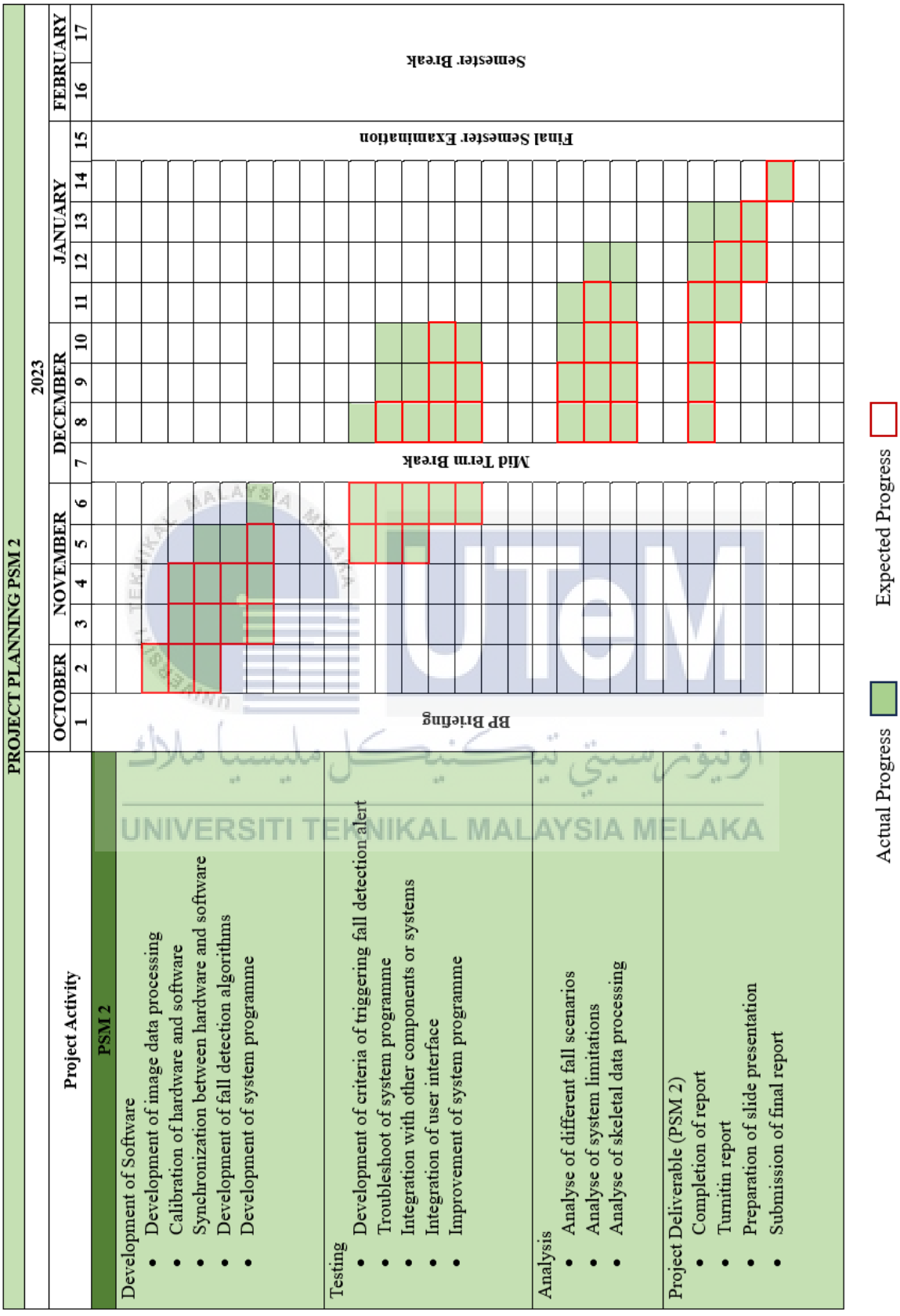
Project Flowchart



Gantt Chart FYP 1



Gantt Chart FYP 2



C# Coding of WPF Application

```
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using Microsoft.Kinect;
using System.IO;

namespace FallDetection
{
    public partial class MainWindow : Window
    {
        private KinectSensor kinectSensor;
        private const float RenderWidth = 640.0f;
        private const float RenderHeight = 480.0f;
        private const double JointThickness = 3;
        private const double BodyCenterThickness = 10;
        private const double ClipBoundsThickness = 10;
        private readonly Brush centerPointBrush = Brushes.Blue;
        private readonly Brush trackedJointBrush = new SolidColorBrush(Color.FromArgb(255, 68, 192, 68));
        private readonly Brush inferredJointBrush = Brushes.Yellow;
        private readonly Pen trackedBonePen = new Pen(Brushes.Green, 6);
        private readonly Pen inferredBonePen = new Pen(Brushes.Gray, 1);
        private DrawingGroup drawingGroup;
        private DrawingImage imageSource;

        private WriteableBitmap colorBitmap;

        private double initialHeadToAnkleDistance = 0;
        private const double positionThreshold = 0.3;

        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            this.imageSource = new DrawingImage(this.drawingGroup);
            SkeletonImage.Source = this.imageSource;

            if (KinectSensor.KinectSensors.Count > 0)
            {
                kinectSensor = KinectSensor.KinectSensors[0];

                if (kinectSensor != null)
                {

                    kinectSensor.Start();

                    // Enable color stream

                    kinectSensor.ColorStream.Enable();

                    // Allocate space for the color pixel data
                    colorBitmap = new WriteableBitmap(kinectSensor.ColorStream.FrameWidth, kinectSensor.ColorStream.FrameHeight, 96.0, 96.0, PixelFormats.Bgr32, null);

                    // Set the image source of the ColorImage control to the colorBitmap
                    ColorImage.Source = colorBitmap;

                    kinectSensor.ColorFrameReady += KinectSensor_ColorFrameReady;

                    // Enable skeleton stream

                    kinectSensor.SkeletonStream.Enable();


                    kinectSensor.SkeletonFrameReady += KinectSensor_SkeletonFrameReady;



                    StatusTextBlock.Text = "Kinect sensor connected";
                }
                else
                {

                    StatusTextBlock.Text = "No Kinect sensor found";
```

```
            this.drawingGroup = new
DrawingGroup();


            }
            else
            {
                StatusTextBlock.Text =
"No Kinect sensor found";
            }
        }

private void Window_Closing(object
sender,
System.ComponentModel.CancelEventArgs e)
        {

            if (kinectSensor != null)
            {
                kinectSensor.Stop();
                kinectSensor = null;
            }
        }

        private void
KinectSensor_ColorFrameReady(object
sender, ColorImageFrameReadyEventArgs e)
        {
            using (ColorImageFrame
colorFrame = e.OpenColorImageFrame())
            {
                if (colorFrame != null)
                {
                    byte[] colorData =
new byte[colorFrame.PixelDataLength];

colorFrame.CopyPixelDataTo(colorData);


colorBitmap.WritePixels(new Int32Rect(0,
0, colorFrame.Width, colorFrame.Height),
colorData, colorFrame.Width *
colorFrame.BytesPerPixel, 0);
                }
            }
        }

private void
KinectSensor_SkeletonFrameReady(object
sender, SkeletonFrameReadyEventArgs e)
        {

            Skeleton[] skeletons = new
Skeleton[0];

            using (SkeletonFrame
skeletonFrame = e.OpenSkeletonFrame())
            {
                if (skeletonFrame !=
null)
                {
                }

skeletonFrame.CopySkeletonDataTo(skele
tons);

                    foreach (Skeleton
skeleton in skeletons)
                    {

if (skeleton.TrackingState ==
SkeletonTrackingState.Tracked)
                        {

                            double
headY =
skeleton.Joints[JointType.Head].Positi
on.Y;
                            double
shouldercenterY =
skeleton.Joints[JointType.ShoulderCent
er].Position.Y;
                            double
hipcenterY =
skeleton.Joints[JointType.HipCenter].P
osition.Y;
                            double
ankleleftY =
skeleton.Joints[JointType.AnkleLeft].P
osition.Y;
                            double
anklerightY =
skeleton.Joints[JointType.AnkleRight].
Position.Y;


UpdateJointPositionsUI(headY,
shouldercenterY, hipcenterY,
ankleleftY, anklerightY);

                            if
(IsPersonFallingDown(skeleton))
                            {

FallEvent();
                            }
                            else
                            {

SafeEvent();
                            }

                            if
(IsHandRaisedAboveHead(skeleton))
                            {

NotifyFallEvent();
                            }
                        }
                    }
                }
```

```
using (DrawingContext dc = this.drawingGroup.Open())
            {

dc.DrawRectangle(Brushes.Transparent, null, new Rect(0.0, 0.0, RenderWidth, RenderHeight));

                if (skeletons.Length != 0)
                {
                    foreach (Skeleton skel in skeletons)
                    {

RenderClippedEdges(skel, dc);

                        if (skel.TrackingState == SkeletonTrackingState.Tracked)
                        {

this.DrawBonesAndJoints(skel, dc);
                        }
                        else if (skel.TrackingState == SkeletonTrackingState.PositionOnly)
                        {

dc.DrawEllipse(

this.centerPointBrush,
                                null,

this.SkeletonPointToScreen(skel.Position),

BodyCenterThickness,

BodyCenterThickness);

}
                    }
            }

         this.drawingGroup.ClipGeometry = new RectangleGeometry(new Rect(0.0, 0.0, RenderWidth, RenderHeight));

            }
        }
private bool IsPersonFallingDown(Skeleton skeleton)
        {

            double yHead = skeleton.Joints[JointType.Head].Position.Y;
            double yShoulderCenter = skeleton.Joints[JointType.ShoulderCenter].Position.Y;
double yHipCenter = skeleton.Joints[JointType.HipCenter].Position.Y;
            double yAnkleLeft = skeleton.Joints[JointType.AnkleLeft].Position.Y;
            double yAnkleRight = skeleton.Joints[JointType.AnkleRight].Position.Y;


            double HeightThresholdFactor = 0.8;

            double distanceThreshold = 0;

            if (initialHeadToAnkleDistance == 0)
            {

initialHeadToAnkleDistance = yHead - yAnkleRight;
                distanceThreshold = initialHeadToAnkleDistance * HeightThresholdFactor;
            }


            double currentHeadToAnkleDistance = yHead - yAnkleRight;


            if (Math.Abs(initialHeadToAnkleDistance - currentHeadToAnkleDistance) > distanceThreshold &&
                yHead < positionThreshold &&
                yShoulderCenter < positionThreshold &&
                yHipCenter < positionThreshold &&
                yAnkleLeft < positionThreshold &&
                yAnkleRight < positionThreshold)
            {
                return true; // Fall detected
            }

            return false; // Safe
        }

  private void UpdateJointPositionsUI(double headY, double shouldercenterY, double hipcenterY, double ankleleftY, double anklerightY)
        {
```

```
HeadYText.Text = $"Head Y: {headY:F2}";
            ShoulderCenterYText.Text =
$"ShoulderCenter Y:
{shouldercenterY:F2}";
            HipCenterYText.Text =
$"HipCenter Y: {hipcenterY:F2}";
            AnkleLeftYText.Text =
$"AnkleLeft Y: {ankleleftY:F2}";
            AnkleRightYText.Text =
$"AnkleRight Y: {anklerightY:F2}";
        }



        private bool
IsHandRaisedAboveHead(Skeleton skeleton)
        {
            Joint head =
skeleton.Joints[JointType.Head];
            Joint handLeft =
skeleton.Joints[JointType.HandLeft];
            Joint handRight =
skeleton.Joints[JointType.HandRight];

            // Check if either hand is
raised above the head
            return
(handLeft.TrackingState ==
JointTrackingState.Tracked &&
handLeft.Position.Y > head.Position.Y) ||

(handRight.TrackingState ==
JointTrackingState.Tracked &&
handRight.Position.Y > head.Position.Y);
        }

private void SafeEvent()
        {
            StatusTextBlock.Text = "Fall
is not detected.";
        }



        private void FallEvent()
        {
            StatusTextBlock.Text = "Fall
is detected.";
        }



        private void NotifyFallEvent()
        {
            StatusTextBlock.Text = "Help
Command Detected!! \fChecking for fall";
        }

private void DrawBonesAndJoints(Skeleton
skeleton, DrawingContext drawingContext)
        {
```

```
// Render Torso
            DrawBone(skeleton,
drawingContext, JointType.Head,
JointType.ShoulderCenter);
            DrawBone(skeleton,
drawingContext,
JointType.ShoulderCenter,
JointType.Spine);
            DrawBone(skeleton,
drawingContext, JointType.Spine,
JointType.HipCenter);
            DrawBone(skeleton,
drawingContext,
JointType.ShoulderCenter,
JointType.ShoulderLeft);
            DrawBone(skeleton,
drawingContext,
JointType.ShoulderCenter,
JointType.ShoulderRight);
            DrawBone(skeleton,
drawingContext, JointType.HipCenter,
JointType.HipLeft);
            DrawBone(skeleton,
drawingContext, JointType.HipCenter,
JointType.HipRight);

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

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

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

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

foreach (Joint joint in skeleton.Joints)
            {
                if (joint.TrackingState
== JointTrackingState.Tracked)
                {
                    Brush drawBrush =
null;

                    if
(joint.TrackingState ==
JointTrackingState.Tracked)
                    {
                        drawBrush =
trackedJointBrush;
                    }
                    else if
(joint.TrackingState ==
JointTrackingState.Inferred)
                    {
                        drawBrush =
inferredJointBrush;
                    }
                    if (drawBrush !=
null)
                    {

drawingContext.DrawEllipse(drawBrush,
null,
SkeletonPointToScreen(joint.Position),
JointThickness, JointThickness);
                    }
                }
            }
        }

private Point
SkeletonPointToScreen(SkeletonPoint
skelpoint)
        {

            DepthImagePoint depthPoint =
kinectSensor.CoordinateMapper.MapSkeleton
PointToDepthPoint(skelpoint,
DepthImageFormat.Resolution640x480Fps30);
            return new
Point(depthPoint.X, depthPoint.Y);
        }
```

```
private void DrawBone(Skeleton
skeleton, DrawingContext
drawingContext, JointType jointType0,
JointType jointType1)
        {
            Joint joint0 =
skeleton.Joints[jointType0];
            Joint joint1 =
skeleton.Joints[jointType1];

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

            // Assume all drawn bones
are inferred unless BOTH joints are
tracked
            if (joint0.TrackingState
== JointTrackingState.Inferred &&
                joint1.TrackingState
== JointTrackingState.Inferred)
            {
                return;
            }

            Pen drawPen =
this.inferredBonePen;
            if (joint0.TrackingState
== JointTrackingState.Tracked &&
joint1.TrackingState ==
JointTrackingState.Tracked)
            {
                drawPen =
trackedBonePen;
            }


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

private static void
RenderClippedEdges(Skeleton skeleton,
DrawingContext drawingContext,
WriteableBitmap colorBitmap)
        {
            double actualRenderWidth =
colorBitmap.PixelWidth;
```

```
if
(skeleton.ClippedEdges.HasFlag(FrameEdges
.Bottom))
            {
drawingContext.DrawRectangle(
                    Brushes. Red,
                    null,
                    new Rect(0,
RenderHeight - ClipBoundsThickness,
actualRenderWidth, ClipBoundsThickness));
            }

            if
(skeleton.ClippedEdges.HasFlag(FrameEdges
.Top))
            {

drawingContext.DrawRectangle(
                    Brushes.Red,
                    null,
                    new Rect(0, 0,
actualRenderWidth, ClipBoundsThickness));
            }

            if
(skeleton.ClippedEdges.HasFlag(FrameEdges
.Left))
            {

drawingContext.DrawRectangle(
                    Brushes.Red,
                    null,
                    new Rect(0, 0,
ClipBoundsThickness, RenderHeight));
            }

            if
(skeleton.ClippedEdges.HasFlag(FrameEdges
.Right))
            {

drawingContext.DrawRectangle(
                    Brushes.Red,
                    null,
                    new
Rect(actualRenderWidth -
ClipBoundsThickness, 0,
ClipBoundsThickness, RenderHeight));
            }
        }

private void Button_Click(object sender,
RoutedEventArgs e)
        {
            RenderTargetBitmap
renderTargetBitmap = new
RenderTargetBitmap((int)this.ActualWidth,
(int)this.ActualHeight, 96d, 96d,
PixelFormats.Pbgra32);

renderTargetBitmap.Render(this);

BitmapEncoder encoder = new
PngBitmapEncoder();

encoder.Frames.Add(BitmapFrame.Create(
renderTargetBitmap));

            string time =
DateTime.Now.ToString("hh'-'mm'-'ss",
CultureInfo.CurrentUICulture.DateTimeF
ormat);
            string myPhotos =
Environment.GetFolderPath(Environment.
SpecialFolder.MyPictures);
            string path =
Path.Combine(myPhotos,
"KinectWindowSnapshot-" + time +
".png");

            try
            {

Directory.CreateDirectory(Path.GetDire
ctoryName(path));
                using (FileStream fs =
new FileStream(path, FileMode.Create))
                {
                    encoder.Save(fs);
                }
                StatusTextBlock.Text =
"Screenshot saved.";
            }
            catch (IOException)
            {
                StatusTextBlock.Text =
"Error saving screenshot.";
            }
        }


        private void
ApplyTiltButton_Click(object sender,
RoutedEventArgs e)
        {
            int selectedAngle =
(int)TiltAngleSlider.Value;

AdjustKinectTiltAngle(selectedAngle);
        }


        private void
AdjustKinectTiltAngle(int angle)
        {
            if (kinectSensor != null
&& kinectSensor.IsRunning)
            {
                angle = Math.Max(-27,
Math.Min(27, angle));

                try
```

```
                {

kinectSensor.ElevationAngle = angle;
                    StatusTextBlock.Text
= $"Tilt angle set to {angle} degrees.";
                }
                catch
(InvalidOperationException ex)
                {
                    StatusTextBlock.Text
= "Error adjusting tilt angle.";
                }
            }
        }
    }
}
```

XAML Coding

```
<Window x:Class="FallDetection.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Fall Detection App" Height="600" Width="1000"
        Loaded="Window_Loaded" Closing="Window_Closing" >


    <Grid>
        <Grid>

            <Image x:Name="ColorImage" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Margin="113,10,285.2,52" />

            <Image x:Name="SkeletonImage" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Margin="112,10,285.2,52" />

            <TextBlock x:Name="StatusTextBlock" HorizontalAlignment="Left"
Margin="734,255,0,0" TextWrapping="Wrap" Text="Status" VerticalAlignment="Top"
FontSize="24" FontWeight="Bold" RenderTransformOrigin="1.12,-0.242" />

            <TextBlock x:Name="HeadYText" HorizontalAlignment="Left" TextWrapping="Wrap"
VerticalAlignment="Top" Text="Head" Margin="734,41,0,0" FontSize="16"
RenderTransformOrigin="0.6,-0.612"/>

            <TextBlock x:Name="ShoulderCenterYText" HorizontalAlignment="Left"
Margin="734,76,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Text="Shoulder"
FontSize="16" RenderTransformOrigin="-0.165,-1.862"/>

            <TextBlock x:Name="HipCenterYText" HorizontalAlignment="Left"
Margin="734,113,0,0" TextWrapping="Wrap"  VerticalAlignment="Top" Text="Hip"
FontSize="16" RenderTransformOrigin="0.522,-1.712"/>

            <TextBlock x:Name="AnkleLeftYText" HorizontalAlignment="Left"
TextWrapping="Wrap" VerticalAlignment="Top" Text="AnkleLeft" Margin="734,148,0,0"
FontSize="16" RenderTransformOrigin="0.502,-0.677"/>

            <TextBlock x:Name="AnkleRightYText" HorizontalAlignment="Left"
Margin="734,184,0,0" TextWrapping="Wrap" Text="AnkleRight" VerticalAlignment="Top"
FontSize="16" RenderTransformOrigin="-0.139,0.727"/>

            <Button x:Name="ScreenshotBtn" Content="Screenshot"
HorizontalAlignment="Left" Margin="845,457,0,0" VerticalAlignment="Top" Width="85"
Click="Button_Click" Height="26" FontSize="14" RenderTransformOrigin="1.308,-0.321"/>

            <Slider x:Name="TiltAngleSlider" Minimum="-27" Maximum="27"
TickFrequency="1" SmallChange="1" LargeChange="5" Value="0"  Margin="734,375,54.6,160.4"
/>

            <TextBlock x:Name="TiltAngleValueText" Text="{Binding
ElementName=TiltAngleSlider, Path=Value, StringFormat='Tilt Angle: {0:F0}°'}"
HorizontalAlignment="Left" Margin="798,410,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" FontSize="14" RenderTransformOrigin="0.367,0.476"/>

            <Button x:Name="ApplyTiltButton" Content="Apply Tilt"
HorizontalAlignment="Left" Click="ApplyTiltButton_Click" VerticalAlignment="Top"
Margin="734,457,0,0" Height="26" Width="85" FontSize="14" RenderTransformOrigin="3.968,-
3.336"/>

        </Grid>
    </Grid>
</Window>
```