Internet of Things 9 (2020) 100130

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

An IoT based device-type invariant fall detection system

Sheikh Nooruddin, Md. Milon Islam[*], Falguni Ahmed Sharna

Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna 9203, Bangladesh

ARTICLE INFO

Article history:
Received 28 April 2019
Revised 9 November 2019
Accepted 13 November 2019
Available online 20 November 2019

Keywords:
Fall detection
Device-type invariant
Large scale
Emergency response system
Elderly monitoring
Accelerometer
Machine learning

ABSTRACT

As the world elderly population is increasing rapidly, the use of technology for the development of accurate and fast automatic fall detection systems has become a necessity. Most of the fall detection systems are developed for specific devices which reduces the versatility of the fall detection system. This paper proposes a centralized unobtrusive IoT based device-type invariant fall detection and rescue system for monitoring of a large population in real-time. Any type of devices such as Smartphones, Raspberry Pi, Arduino, NodeMcu, and Custom Embedded Systems can be used to monitor a large population in the proposed system. The devices are placed into the users' left or right pant pocket. The accelerometer data from the devices are continuously sent to a multithreaded server which hosts a pre-trained machine learning model that analyzes the data to determine whether a fall has occurred or not. The server sends the classification results back to the corresponding devices. If a fall is detected, the server notifies the mediator of the user's location via an SMS. As a failsafe, the corresponding device alerts nearby individuals by sounding the buzzer and contacts emergency medical services and mediators via SMS for immediate medical assistance, thus saving the user's life. The proposed system achieved 99.7% accuracy, 96.3% sensitivity, and 99.6% specificity. Finally, the proposed system can be implemented on a variety of devices and used to reliably monitor a large population with low false alarm rate, without obstructing the users' daily living, as no external connections are required.

1. Introduction

The elderly make up a big part of our society. The percentage of the elderly in the world's population is increasing rapidly. According to the medium scenario of a prediction by the United Nations [1], in 2017, 962 million people (13% of the world population) were aged 60 or above. This is expected to more than double (2.1 billion) by 2050 and triple by 2100 (3.1 billion) [1]. One of the most prevalent causes of injury for the general population, especially the elderly is accidental falls. According to the World Health Organization (WHO) [2], Falling is the second leading cause of unintentional injury, death as an estimated 646,000 fatal falls occur over the world every year. Adults that are older than 65 years of age suffer the greatest number of fatal falls and fall-related death rates are highest among adults over 60 years of age [2]. Approximately, 28%–35% of adults that are over 65 years of age fall each year and the rate is much higher for 32%–42% of adults over 70 years of age [2].

In general, most of the fall happens at home as most living environments are filled with potential fall hazards [3]. Common hazards include clutter, poor lighting, obstructed ways, pets, slippery floors, and unstable furniture [4]. Older people suffering from neurological diseases such as epilepsy and dementia [5] are more prone to falls than the average elderly

[*] Corresponding author.
E-mail address: milonislam@cse.kuet.ac.bd (Md. Milon Islam).

https://doi.org/10.1016/j.iot.2019.100130
2542-6605/© 2019 Elsevier B.V. All rights reserved.

population. Living alone also increased the risk of falls. The falls themselves are not life-threatening in most cases. The most devastating effects resulting from falls are related to remaining on the floor for prolonged periods after a fall.

It is almost impossible to make the entire living environment fall-proof. However, modern technologies can be used to automatically detect falls in indoor environments and notify the emergency services and caregivers as soon as possible to minimize the response time. Almost every fall detection system normally has the following stages: data collection stage, feature extraction stage, learning stage or detection stage. In the data collection stage, data are collected from the users. In the feature extraction stage, more relevant and meaningful data are extracted from the collected data. In the systems employing machine learning algorithms [40], the extracted feature data is used to train a model that can classify the data later. In threshold-based detection systems [26], the extracted feature data is used to check whether it has passed the threshold or not to detect the fall.

Many types of fall detection systems have been developed over the years. The developed systems can be divided into two categories: Surveillance-based fall detection systems and Wearable sensors-based fall detection [5–7]. In Surveillance-based fall detection systems, cameras [8,9], depth cameras [10–12], range-Doppler radar [13,14], smart tiles [15,16], acoustic sensors [17], fiber optic sensors [18], infrared sensors [19,20], vibration detection sensors [21] etc. have been used. These devices are normally placed in a predefined space to monitor the activities of the elderly. In camera and radar-based systems, the cameras and radars are generally mounted in the ceiling or closer to the ceiling of the room. Due to their stationary attributes, they can only monitor certain predefined areas. Surveillance based systems are also comfortable for the elderly, as they don't get in the way of their daily life. But Surveillance based systems, especially camera-based systems do not protect the user's privacy, and thus cannot be used in areas such as toilets, washrooms, etc. Wearable sensor-based systems, on the other hand, employ various sensors such as tri-axial accelerometer [22–26], tri-axial accelerometer and gyroscope [27–30], etc. Some systems only employ single sensors [22–26], while others use a combination of sensors [27–30]. Due to advances in micro electromechanical systems, various sensors such as accelerometers and gyroscopes have become very compact, and thus can be easily integrated into embedded systems and other portable devices. Numerous smartphone-based fall detection systems and algorithms have been proposed in recent years [31–36]. Machine Learning is also being used extensively in fall detection systems [37–41]. Support Vector Machine (SVM) [38], K-Nearest Neighbors (K-NN) [39], Random Forest [40], Decision Trees [41] are the most used algorithms in fall detection. Wearable sensors are normally wrist-worn [31–33] or waist-mounted [22–25,36,37]. Most of the systems described thus far deals with the fall detection for a single person.

The goal of this paper is to introduce a novel IoT based device-type invariant wearable fall detection system using an accelerometer that can provide real-time monitoring of a large population in a large-scale environment such as nursing homes, hospitals, retirement homes, etc. The proposed system is based on client-server architecture. Any type of IoT device having internet connectivity and can interface with four modules, namely: an accelerometer, a buzzer, a GPS module, and a GSM module, can be used as a client device in the system. The developed device is placed into the user's left or right pant pocket. The client device continuously collects the accelerometer data from the surrounding environment and sends the data to the server. The server hosts a linear classifier model that analyzes the data and predicts whether a fall has occurred or not. The server sends the result back to the client. If a fall is detected, the client can immediately contact emergency services or mediators who can provide emergency response.

The rest of the paper is organized as follows: Section 2 provides a review of the related works. Section 3 provides an overview of the entire fall detection system. The methods and materials used to implement the system are outlined in Section 4. The experimental results of the proposed system are given in Section 5. Issues such as limitations and stability of the system are given in Section 6. Finally, a concluding remark is given in Section 7. Section 8 discusses the possible future works.

2. Related works

A wearable device using a single tri-axial accelerometer was developed by Wu et al. [22]. The wearable device is placed on the user's waist. The device detects falls based on acceleration analysis and sends a short message with the user's location data to caregivers. The system also gives the user a chance to withdraw an alarm. If the user withdraws the alarm, the user will receive an SMS notification of the safety of the user. No special mounting is required for this system as the algorithm does not need the axes of the accelerometer to be fixed strictly. The proposed system achieved 97.1% sensitivity and 98.3% specificity. The system may trigger false alarms as the normal activity of resting has a similar rotation as falling. Lim et al. [23] developed a system in which the fall-feature parameters are calculated from a single tri-axial accelerometer. The parameters are first applied to a simple threshold method and then the detected falls are applied to a Hidden Markov Model (HMM) to distinguish between falls and fall-like events. The system achieved 99.5% accuracy, 99.17% sensitivity, and 99.69% specificity when the ASVM $= 2.5$ g and $\theta = 55^\circ$ was the threshold values for the simple threshold method and the parameter θ was applied to the Hidden Markov Model. The system conserves computing effort and resources by applying only the detected fall events to the HMM.

An unobtrusive smartphone-based fall detection system was proposed by Aguiar et al. [24]. The system continuously screens the accelerometer data of the smartphone when the smartphone is carried in the user's belt or pocket. From the acceleration vector output of the accelerometer of the smartphone, a total of 14 different signal components (x, y, z projections, magnitude value and angles with x, y, z-axes of the phone) are computed and passed through a Butterworth digital filter by the system. The system then uses a decision tree to retrieve information regarding the most significant features

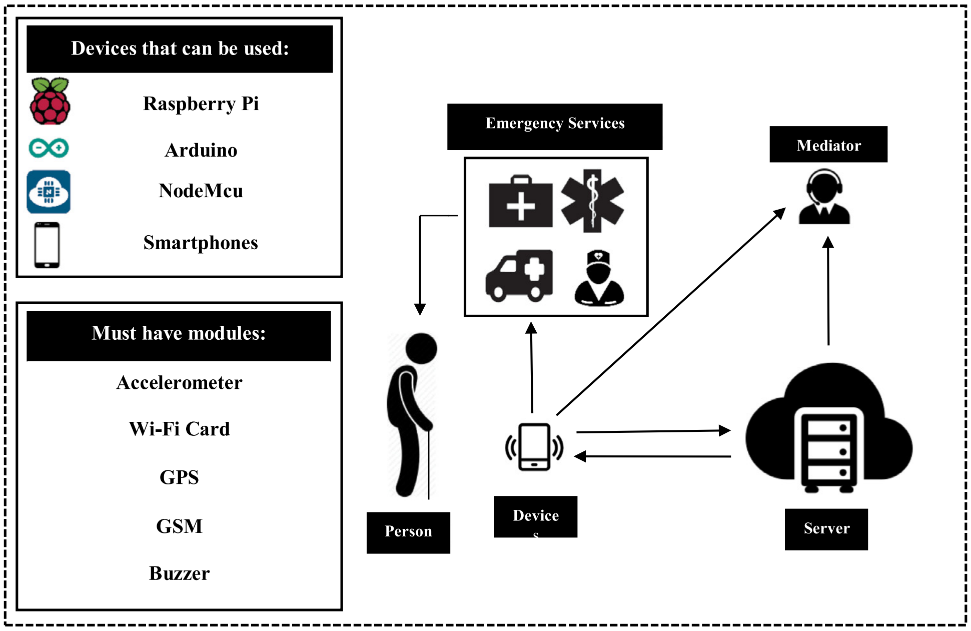


Fig. 1. System architecture of the proposed system.

and thresholds. This information is used in a state machine algorithm to detect the fall and send SMS and email notifications containing the user's location to the caregivers and emergency services. The sensitivity and specificity of the system are close to 97% and 99% for both usage positions (belt or pocket). The future works might incorporate the use of gyroscopes for detecting the rotation and barometer for calculating altitude in the system. As the system continuously screens the accelerometer data, battery usage of the smartphone is a concern. Li et al. [25] developed a waist-mounted device using TelosW mote with an accelerometer as the detector and the Neyman–Pearson detection framework as the classifier. The accelerometer in the systems periodically samples the acceleration of the person and compares the data with a predefined threshold. If the data exceeds the threshold, an alert message is delivered to the base station using an 802.15.4/Zigbee network. The base station can then forward the message to the emergency services and caregivers using either mobile communication networks or 802.11WLAN network. Laura et al. [26] developed a footwear-based fall detection system employing an accelerometer sensor and Force Sensing Resistors (FSR) situated in the sole of the footwear. The processing device, a Raspberry Pi, encased in a box, is connected to the sensors via long wires and worn at the waist like a belt. The threshold-based algorithm achieved a 97.1% maximum accuracy, and 90% accuracy with a FAR (False Alarm Rate) of 0.

3. System overview

The proposed fall detection and response system architecture is illustrated in Fig. 1. On the left side of the illustration, the possible client devices and the necessary modules are shown. The proposed system is mainly composed of four blocks: The data collection device containing the accelerometer unit, the multithreaded server running the pre-trained model, the co-ordination center also called a mediator, and the emergency services consisting of the nearby hospitals and emergency first-aid stations.

A device used in both the data collection and co-ordination phase must contain or interface with the following modules: an accelerometer module, a Wi-Fi module, a GPS module, a GSM module, and a buzzer. All modern smartphones contain these modules. Single-board computers like Raspberry Pi and Single-board microcontrollers like Arduino and NodeMcu can also easily interface with these modules and thus can be used for these purposes. Raspberry Pi contains an on-board Wi-Fi module. NodeMcu is built upon the ESP8266 Wi-Fi module. An additional ADC module is required to interface the analog sensors with the Raspberry Pi. The device is kept on the user's left or right pant pocket. The device collects the user's movement data. The data for each second of movement is sent to the server.

The server used in the proposed system is multithreaded. Multithreading enables the server to concurrently provide service to multiple connected devices. The server keeps a record of the MAC addresses of the connected devices and their corresponding assigned room numbers. The server analyzes the data and sends the classification result back to the respective device. If a fall is detected, the server finds the corresponding location from the device's MAC address and relays that information to the mediator in charge of monitoring the users via an SMS. The actions of the server in a fall scenario are illustrated in Fig. 2.

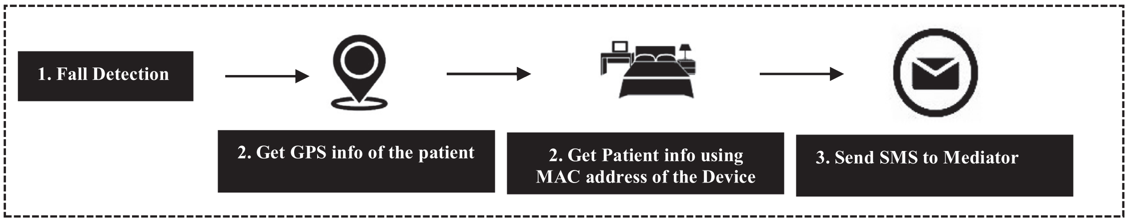

1. Fall Detection
2. Get GPS info of the patient
2. Get Patient info using MAC address of the Device
3. Send SMS to Mediator

Fig. 2. Actions of the server in a fall scenario.

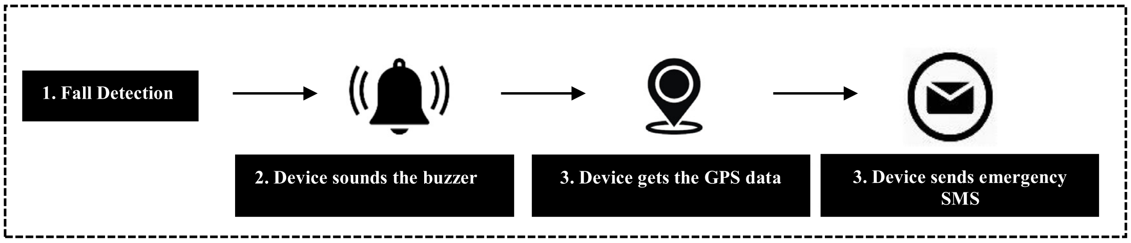

1. Fall Detection
2. Device sounds the buzzer
3. Device gets the GPS data
3. Device sends emergency SMS

Fig. 3. Actions of the client in a fall scenario.

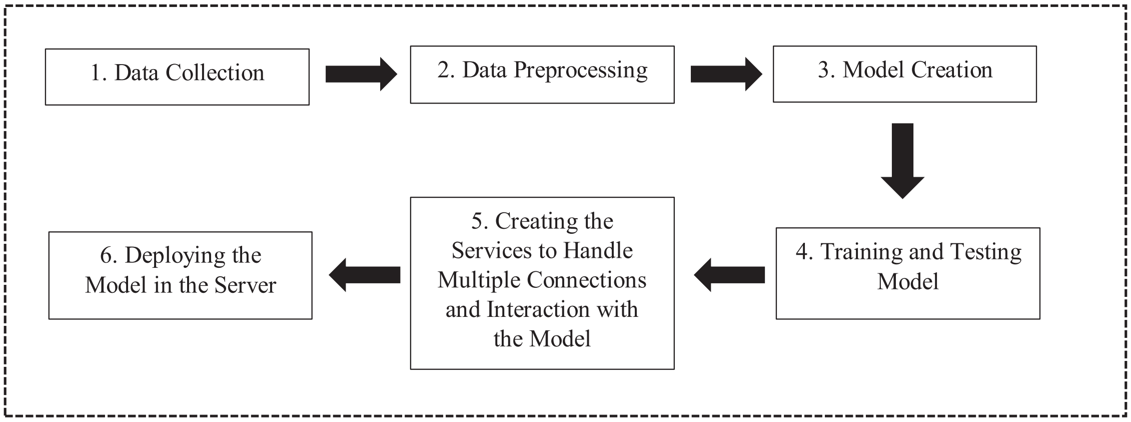

1. Data Collection
2. Data Preprocessing
3. Model Creation
4. Training and Testing Model
5. Creating the Services to Handle Multiple Connections and Interaction with the Model
6. Deploying the Model in the Server

Fig. 4. The workflow of the model development and deployment stage.

As a failsafe, in the case of a fall scenario, the device immediately sounds the buzzer. The failsafe actions of the client device in case of a fall scenario is depicted in Fig. 3. The device then collects the location data via the GPS module and contact the mediator via emergency SMS. The SMS contains the location of the user. The mediator can be a relative or an emergency call center operative or a completely automatic emergency response system. The device can also contact emergency service numbers via SMS or the mediator can contact them. As the emergency services can respond to the occurred fall in the shortest possible time, they can potentially save the user's life. As the sent emergency SMS contains geolocation info, the emergency responders can quickly pinpoint the location of the user. These failsafe actions are necessary to guarantee medical assistance to the distressed user. This system is cost-efficient if implemented in environments where monitoring of a large population is necessary, for example, in nursing homes, hospitals, retirement homes, etc.

4. Methods and materials

The workflow of the model development and deployment process i.e. data collection and preprocessing, model development, training and testing, services development, deployment are illustrated in Fig. 4.

4.1. Data collection and preprocessing

There are many public smartphone accelerometer-based datasets, such as: MobiFall [42], MobiAct [43], tFall [44], Gravity Project [45], Graz [46], UMAFall [47], UniMiB SHAR [48]. Among these datasets, only MobiFall, MobiAct, tFall, Graz, and UniMiB SHAR use a single accelerometer to record the falls and ADLs. Table 1 presents an overview of the considered datasets.

Smartphones have been used consistently to detect free falls for a long time now. Many smartphone vendors do not disclose the features of their built-in sensors in the technical specifications of the smartphones. All the datasets described in Table 1 used Samsung smartphones to build the dataset. The built-in accelerometers in these smartphones have a range of ±2 g (19.61 m/s^2). Mellone et al. [49] and Albert et al. [50] presented a comparison of the performance of built-in smartphone accelerometers and other dedicated or external sensing devices. Their study concluded that smartphones can effectively quantize and characterize human mobility. Vogt et al. [51] also used smartphones to characterize free fall-fall time with good degrees of accuracy. Smartphones with accelerometers having a range of ±2 g have also been consistently used to perform fall detection.

Among the presented datasets above, the tFall has the highest number of samples. The tFall dataset contains accelerometer sensor data of falls and ADL (Activities of Daily Life) of 10 volunteers. Out of 10 volunteers, 7 are male and 3 are female. The volunteers were aged between 20 and 42 years. Their mean age was 31.30 years with a standard deviation of 8.60 years. Their weight was between 54 and 98 kg, with the average weight being 69.20 kg and a standard deviation of 13.1 kg. The volunteer's height was determined to be between 161 and 181 cm with an average height of 173.00 cm and a standard deviation of 8.00 cm. The tFall dataset recorded accelerometer sensor values from putting a smartphone (Samsung Galaxy Mini) in the pant (left or right pocket) or Hand Bag (left or right side) of the volunteers. The range of the smartphone accelerometer used to build this dataset is presumed to be ±2 g. The accelerometer sensor values in this dataset are between −2.0303 g and 2.082 g. In the tFall dataset, the data were obtained by recording ADLs in natural conditions. The volunteers were tracked during their natural activities. No predetermined rules or patterns were placed on the volunteer's activities. Some compensation strategies were used to prevent falls. ADLs of one week were recorded. Although the users were constantly monitored, the entries in the dataset contain the sensor values for a limited time window of 6 s around the peak acceleration magnitude, 1.5 g in this case. The same 6 s interval was selected for the fall simulations. The tFall dataset contains 8 types of fall records: Backwards fall, Fall from sitting position in a chair, Forward fall, Forward falls with compensation strategies, Hitting an obstacle during the fall, Lateral left fall, Lateral right fall, Falls from passing out. The dataset has a total of 10,909 samples. 9883 samples represent ADLs and 1026 samples represent falls. The ADLs are not typified in the tFall dataset. As the tFall dataset represents motion data in everyday life scenarios, has the highest number of samples, and has a good ADL/fall data ratio, it was used in the proposed system to train the classifier model.

The data had been interpolated at 50 Hz. The time between samples is 2 s. Every row in the dataset contains 301 accelerometer sensor values corresponding to a total of 6 s of data. Only the central 1 s are used in the data. Hence, only the 125th value to the 176th value was used. For the central 1 s, 51 values along the x-axis, 51 values along the y-axis, and 51 values along the z-axis were considered. This ADL and Fall data were collected by putting the device on the user's pant (left and right) pocket. The ADL and fall data collected by putting the accelerometer device in the volunteers' Hand Bag was excluded, as the intention was to develop a system in the future that would be put in the pant (left or right pocket) and used to test the system. An array was constructed by concatenating the data along the 3-axes for every second. Hence, for each second, a row containing a total of 153 values was generated. The values along the x-axes were labeled from x0 to x50. The values along the y-axes were labeled from y0 to y50. The values along z-axes were labeled from z0 to z50. An extra column was added with each row. The column represented the type of data. 0 means ADL data and 1 means fall data. After adding the label, the total size of the fall data array was (503, 154) and the total size of the ADL data array was (7816, 154). These two arrays were concatenated to create the array of the entire dataset of size (8319, 154). In real-life environments, fall data is harder to acquire than ADL data due to the rarity of fall events. A necessary step while training a model is to perform a train-test split. A train-test split is performed on the entire dataset to divide the dataset into a training dataset and test dataset. The training dataset is used to train the model. The test dataset is used to test the performance of the trained model. Various percentile amounts are generally used to perform the train-test split. For example, (70–30)% splits or (80–20)% splits are very common. Even (60–40)% splits are used in larger balanced datasets. A (70–30)% split means 70% instances of the total dataset are used for training a model and 30% instances of the total dataset are used for testing the model. A (70–30)% train-test split was performed on the tFall dataset. A total of 5823 instances were used to train the model. Among the 5823 train instances, 5455 instances were non-fall instances, and 368 instances were fall instances. A total of 2488 instances were used to test the performance of the model. Among the 2488 test instances, 2353 instances were non-fall instances, and 135 instances were fall instances. The dataset was imbalanced, meaning one class has significantly more instances than the other class. Imbalanced datasets bias the machine learning models. Various probabilistic complex resampling techniques such as SMOTE and ADASYN, NearMiss-1, NearMiss-2, etc. can be used to negate the effects of imbalance. Another simple solution to class imbalance is using class weights. If a and b are two classes in a binary classification problem and n_a, and n_b represents the number of a and number of b in training dataset, respectively, then, class weight for class a is calculated by

$$\frac{n_{samples}}{n_{classes} * n_a}$$

For class b, the class weight is calculated by:

$$\frac{n_{samples}}{n_{classes} * n_b}$$

Table 1
Overview of accelerometer based public motion datasets.

Name	Used Sensors	Positions of sensor	Types of ADL/Fall	Scenario	Number of Samples (ADL/Fall)	Sampling Rate	Range	Number of Subjects
MobiFall	A, G, O	Thigh (Trouser Pocket)	9/4	Gym Hall	630 (342/288)	87	±2 g	24
MobiAct	A, G, O	Thigh (Trouser Pocket)	9/4	NS	2326 (1879/647)	87	±2 g	57
tFall	A	Thigh (right or left pocket) Hand Bag (Left or Right Side)	NT/8	One week of Everyday Behavior	10,909 (9883/1026)	45(±12)	±2 g	10
Graz	A, O	Waist (belt bag)	10/4	Gym Hall	2460 (2240/220)	5	±2 g	5
UniMiB SHAR	A	Thigh (left or right trouser pocket)	9/8	NS	7013 (5314/1699)	50	±2 g	30

A = Accelerometer, *G* = Gyroscope, *O* = Orientation Measurements, NS = Not Specified, NT = Not Typified.

4.2. Model Development

TensorFlow™ was used [52] to develop the classifier model. TensorFlow™ is an open-source software library for high-performance numerical computation. TensorFlow™ provides great support for deep learning and machine learning based large-scale applications. TensorFlow™ enables fast model prototyping by providing higher-level functions that relieve the user from stressing about lower-level details. The developed model is a linear classifier model. A statistical classifier uses an object's characteristic to determine which class it belongs to. A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics. The input characteristics are called feature values and are normally provided in a vector called feature vector. For an input feature vector $\vec{r}$, the output value is determined by,

$$y = f(\vec{w} \cdot \vec{r}) = f\left(\sum_j w_j r_j\right) \tag{1}$$

Here, $\vec{w}$ is a weight vector and f is a function that converts the dot product of the weight vector and the feature vector into an output value. $\vec{w}$ is learned from the labeled training set for the model. Complex functions can also calculate the probability of an input belonging to a certain class.

A linear classifier works best for problems involving many variables and provides accuracy levels comparable to non-linear classifiers. A linear classifier takes less time than non-linear classifiers to train and use. Given training data $(y_i,\ x_i) \in \{-1,\ +1\} \times R^n,\ i = 1,\ \ldots,\ l.$ where, y_i is the label and x_i is the feature vector, the constructed generic decision function is as follows:

$$d(x) = w^T\ \emptyset(x) + b \tag{2}$$

Here, w is the weight vector and, b is the intercept, which is also known as the bias. A nonlinear classifier maps each instance, x to a higher dimensional vector $\emptyset(x)$, if data are not linearly separable. For a linear classifier, $\emptyset(x) = x$, meaning the data points are not mapped. For nonlinear classification, evaluating $w^T\emptyset(x)$ is expensive as $\emptyset(x)$ might be high dimensional. Kernel methods were introduced to handle such difficulties. If w is a linear combination of training data, i.e.,

$$w = \sum_{i=1}^{l} \alpha_i \phi(x_i) \text{ for some } \alpha\ \in R^l \tag{3}$$

And the following kernel function can be easily calculated:

$$k\left(x_i,\ x_j\right)\ \equiv\ \phi(x_i)^T \phi\left(x_j\right) \tag{4}$$

Then the decision function can be calculated by

$$d(x) = \sum_{i=1}^{l} \alpha_i K(x_i,\ x) + b \tag{5}$$

regardless of the dimensionality of $\emptyset(x)$.

For example,

$$k\left(x_i,\ x_j\right) \equiv \left(x_i^T x_j + 1\right)^2 \tag{6}$$

is the 2-degree polynomial kernel with

$$\emptyset(x) = \left[1,\ \sqrt{2}x_1,\ \ldots,\ \sqrt{2}x_n,\ \ldots,\ x_1^2,\ \ldots,\ x_n^2, \sqrt{2}x_1x_2,\ \sqrt{2}x_1x_3,\ \ldots,\ \sqrt{2}x_{n-1}x_n\right] \in R^{\frac{(n+2)(n+1)}{2}} \tag{7}$$

This kernel trick is used extensively in popular classifiers such as SVM or kernel Logistic Regression. However, training and testing processes are still time-consuming for large datasets. For example, the cost of predicting a testing instance via the decision function (5) for a kernel like (6), is O(ln).

Without using the kernel, w is available in an explicit form, so we can predict an instance by (1). With $\emptyset(x) = x$:

$$w^T\emptyset(x) = w^T x \tag{8}$$

Thus, for a linear classifier, without using kernels, the cost is O(n). Yuan et al. surveyed the performance of linear vs. non-linear classifiers in [53]. For every entry, the classifier has to work with 153 variables. A linear classifier is also chosen when classification speed is an issue. As a fall detection system needs to be very fast, a linear classifier model was the best choice. As the system has to figure out whether a fall has occurred or not, the developed classifier is a binary classifier. Thus, only two labels are used to describe the instances in the dataset: fall and non-fall.

4.3. Implementation of the multithreaded server

A server is a computer program or device that provides functionality, also named "service" to other computer programs and devices, collectively referred to as "clients". A server also manages its network resources. Servers are generally dedicated,

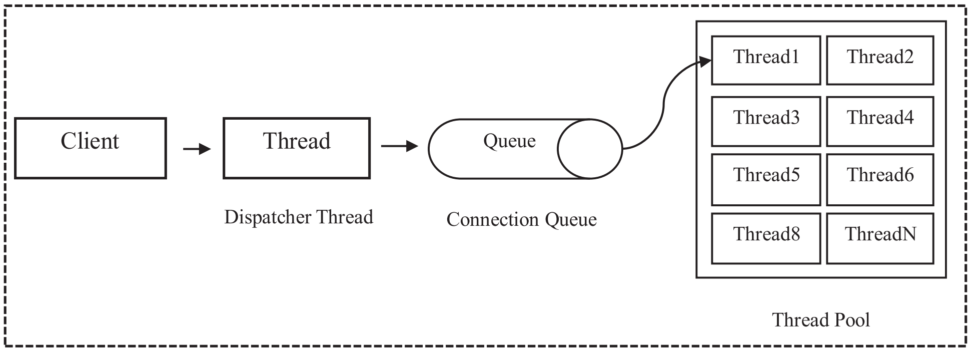


Fig. 5. Multithreaded server architecture.

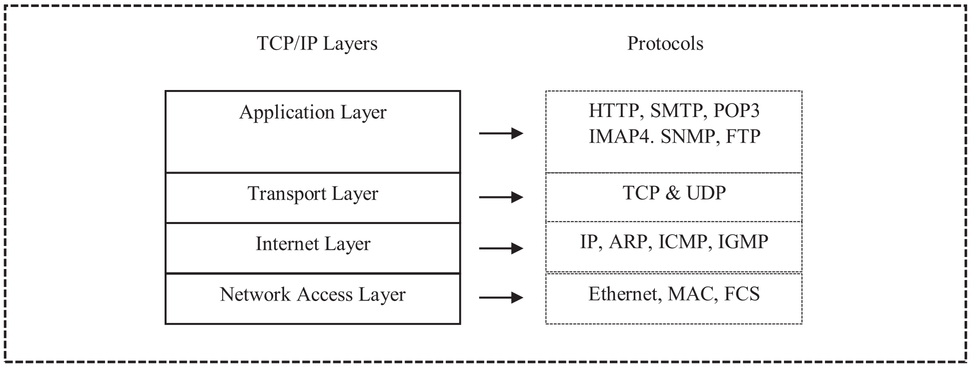


Fig. 6. TCP/IP model and protocols.

meaning they do not perform any task other than managing clients and providing services. A multithreaded server uses multiple threads to service requests concurrently. As the goal is to develop a fall detection system that can be implemented in large scale environments, for example, nursing homes, and hospitals or even in enterprise scenarios, a multithreaded server is a perfect solution. Multithreaded servers generally employ a thread-per-connection model. In multithreaded server architecture, generally, a dispatcher thread handles the connection requests and maintains a pool of worker threads. The dispatcher thread blocks on the socket for new connections. All new connections are put in a connection queue. Threads take the connections from the queue and perform their tasks and wait for new connections in the queue. In a multithreaded server, all the threads share the same address space. Thus, sharing global variables and states is possible among multiple threads. A thread also has a lower memory footprint than processes. The threads also require fewer resources to create or terminate than processes. A thread pool helps to restrict the maximum number of threads the server can have simultaneously, thus ensuring predictable latencies and preventing overload. The overall architecture of a multithreaded server is illustrated in Fig. 5.

The communication protocol used in this system is TCP/IP (Transmission Control Protocol/Internet Protocol). TCP/IP was chosen over UDP (User Datagram Protocol) because TCP/IP is more reliable and provides more functionalities. TCP/IP has built-in flow control and congestion control capabilities. Flow control ensures that the sender will not overwhelm the receiver if the receiver is busy for some reason. Congestion control throttles the sender when the network is overloaded. As TCP/IP uses a three-way handshaking protocol, while creating the connection between host/client and server, the connection is more reliable. In this handshaking protocol also known as TCP handshake, the server and the host/client both need to exchange SYN and ACK packets before starting the actual data communication. The protocols used within the layers of TCP/IP are depicted in Fig. 6. Fig. 7 shows the three-way handshaking protocol used in TCP/IP.

Existing advanced services such as the IBM Watson were not used, as the servers in this system need to be lightweight to accommodate a large number of clients. Also, advanced services provide additional advanced features that are not needed in the system. The main functionality of the server is to perform the prediction task and provide concurrency. If an error occurs at any thread, the thread flushes its local data, creates a new thread, joins the thread and terminates itself. The flowchart in Fig. 8(a) represents the workflow of a single thread in the server. When a thread is started, it configures its hardcoded buffer size, gets an IP address and port number from the dispatcher thread. The thread then loads the machine learning model. For loading the machine learning model, it takes a significant amount of time. The thread then loads the custom services and starts listening for client device connections. If a client device connects to the thread via the dispatcher thread, it starts waiting for messages from the client device. Every message from the client device contains the MAC address of the

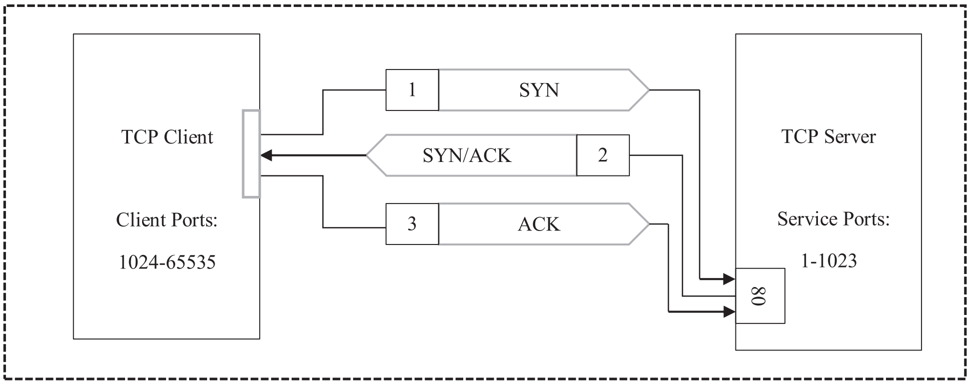


Fig. 7. TCP/IP three-way handshake.

device, the location info of the device, and the motion data of 1 s of the client. The thread then parses the received message and classifies the motion data. If a fall is detected the server searches the stored record using the device's MAC address and collects relevant information about the client such as the client's name, his assigned room number in the case of a nursing home or hospital scenario. The thread then generates a message with the patient info, location info, and emergency help message and pushes the message as an HTTP request through the SMS gateway. The recipient of the message is the person in charge. The mediator's contact information is hardcoded into the server. The flowchart in Fig. 8(b) represents the workflow of the entire server. At initialization, the server needs to know the maximum number of threads it can create. This is required, as making the server capable of creating infinite threads makes the system less secure, will put a heavy burden on the system hosting the server, and can be exploited by hackers. The required maximum number of threads is essentially the maximum number of patients that can be monitored concurrently. Once the server gets this information, it starts creating the threads and joining the threads. An open-source Wireless Application Protocol (WAP) and SMS gateway "kannel" was used as the SMS gateway for the server. The workflow of an SMS gateway is depicted in Fig. 9. The server sends an HTTP request containing the SMS and the recipient number to the gateway. The gateway then sends the SMS to the provided contact number.

4.4. *Implementation of the clients*

Fig. 10 illustrates the necessary modules that are interfaced with the client device. The tri-axial accelerometer is used to screen the motion data of the users. The buzzer is used to alert nearby individuals of the fall. The GPS is used to get the location data of the user and the GSM module is used to send SMS to emergency services. Fig. 11 illustrates some of the devices that can be used as the client device. The proposed IoT based device-type invariant system can be implemented on various types of devices that can interface with the necessary modules. Smartphones, Raspberry Pi, Arduino, NodeMcu, etc. can be used as the client device.

Android is an operating system developed by Google mostly for touchscreen mobile devices such as smartphones and tablets and has been recently extended to support smart TV, watches, automatic cars, etc. A modified Linux kernel and other open-source software are the basis for the android operating system. Modern smartphones contain many sensors. Sensors can be both hardware and software-based. Android provides a simple sensor framework for easy control of sensors and easy acquisition of sensor data. Almost all modern smartphones contain a tri-axial accelerometer sensor. Most smartphones employ a software or hardware buffer to constantly get the sensor data and provide data to the user at the rate specified in the running program. The Android sensor framework known as *SensorManager* has some constant values for getting the sensor reading at a specified rate. For example, *SENSOR_DELAY_FASTEST, SENSOR_DELAY_GAME, SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI*. These are preset for performing various tasks such as playing games, detecting screen orientation, etc. Batching refers to buffering sensor events in a sensor hub and/or hardware FIFO (First In First Out) before reporting the events through the *SensorManager. SensorManager* provides opportunities to control batching using some parameters. In the android operating system, the arguments to the batching function: *sampling_period_ns*, max*_report_latency_ns* can be used to control the sampling or interpolation rate of the sensor data retrieval. The *registerListener* method of *SensorManager* is used to get the related sensor readings. This method has a parameter "*rate*" that can be used to specify the sampling or interpolation rate of the sensor data. Custom delays between events can also be set. However, the rate isn't guaranteed as this depends on the hardware. But normally events are received faster than the specified rate. The app will perfectly work

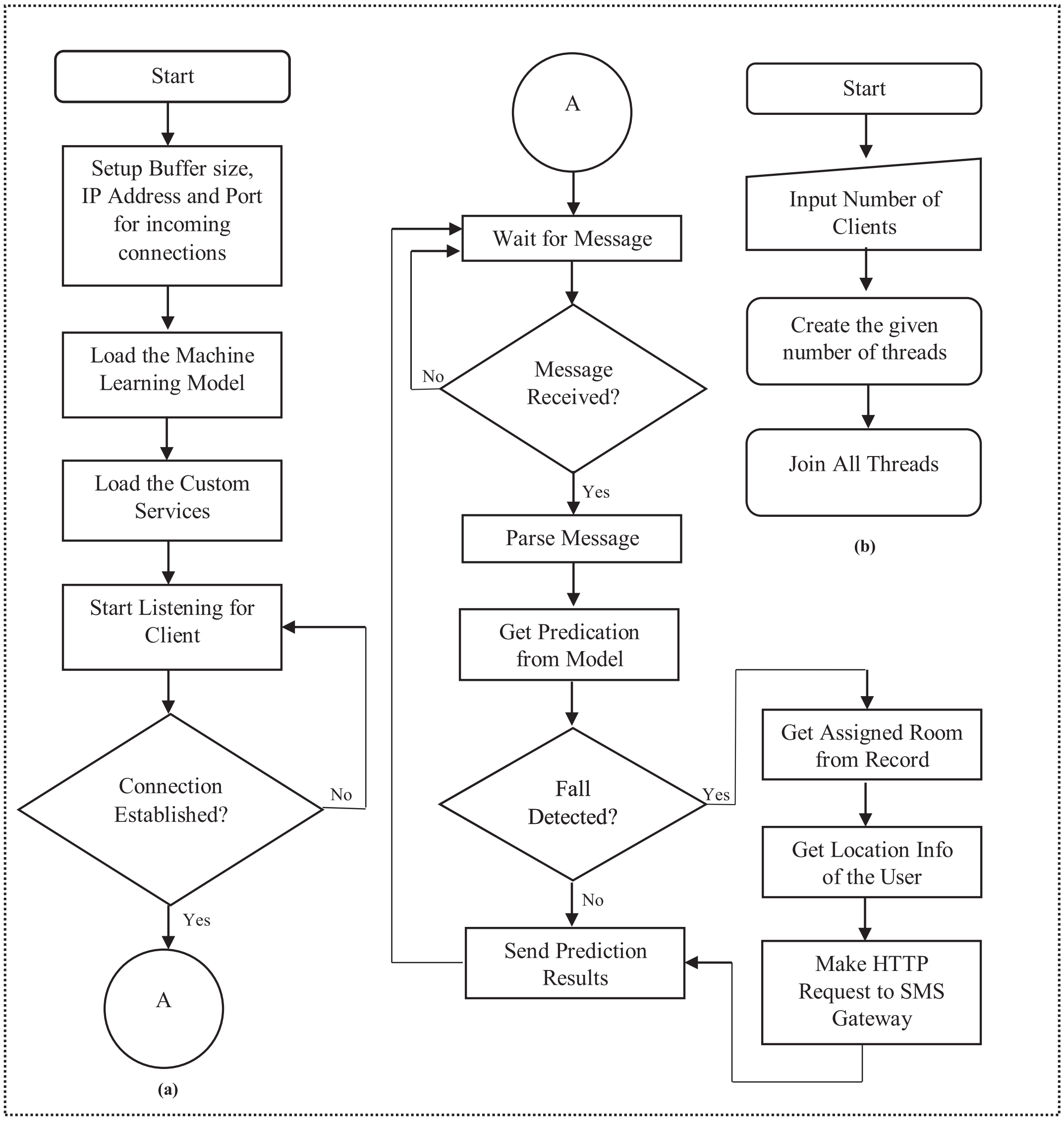

Start
Setup Buffer size, IP Address and Port for incoming connections
Load the Machine Learning Model
Load the Custom Services
Start Listening for Client
Connection Established?
No
Yes
A
(a)
A
Wait for Message
Message Received?
No
Yes
Parse Message
Get Predication from Model
Fall Detected?
Yes
No
Send Prediction Results
Get Assigned Room from Record
Get Location Info of the User
Make HTTP Request to SMS Gateway
Start
Input Number of Clients
Create the given number of threads
Join All Threads
(b)

Fig. 8. (a) Flowchart of a thread (b) Flowchart of the multithreaded server.

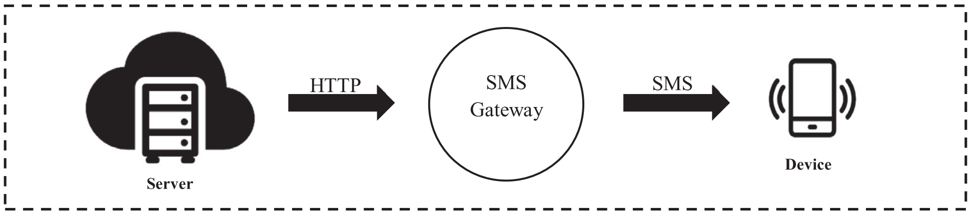


Fig. 9. SMS gateway to send SMS to devices.

on devices running Android 4.0 (Ice Cream Sandwich), API level 14 or higher. JAVA is the primary programming language used for android app development. Although other languages such as Kotlin, C++, etc. are used. Third-party frameworks, languages, and tools are also supported. As android natively supports java, various java libraries are readily available to use in android. JAVA has native support for connection over TCP/IP and UDP via sockets. JAVA automatically takes care of the low-level networking details.

Raspberry Pi is a series of single-board computers. Raspbian, a Debian based operating system is the primary operating system of the Raspberry Pi. Raspbian supports programming in the Python language. Python provides great support for

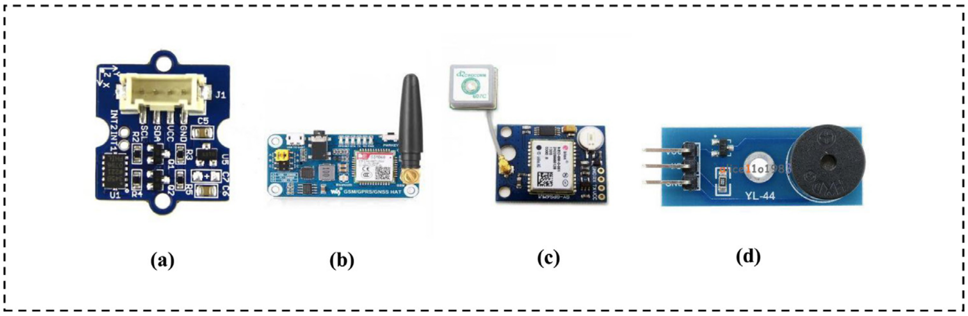


Fig. 10. Necessary Modules. (a) Tri-axial accelerometer (b) GSM (c) GPS (d) Buzzer.

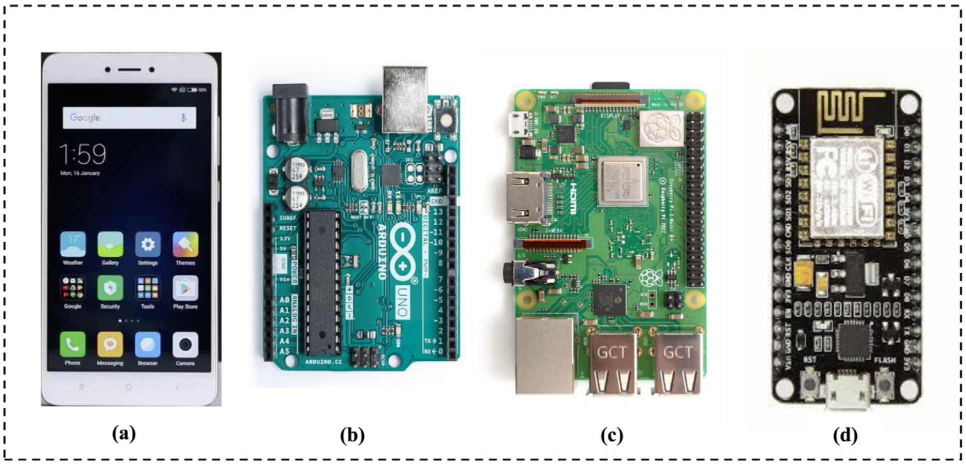


Fig. 11. Possible client devices. (a) Smartphone (b) Arduino Uno (c) Raspberry Pi Model B3+ (d) NodeMcu.

handling connections with the server and abstract lower-level connection details. Python also has great support for socket programming. Raspberry Pi computers do not contain any analog pins. Hence, an additional ADC is required to interface the raspberry pi to analog modules. Arduino is a single-board microcontroller. An Integrated Development Environment (IDE) named Arduino IDE can be used to code and control the Arduino board. Arduino board does not have on-board Wi-Fi connectivity. To get internet connectivity, an additional Wi-Fi shield or another low-cost Wi-Fi microchip, such as ESP8266 needs to be used. The Arduino IDE supports the languages C and C++ but uses special code structuring. C and C++ also provide great support for socket programming. NodeMcu is an open-source IOT platform that runs on the ESP8266 Wi-Fi SoC (System on Chip). The NodeMcu system is based on the ESP-12 module. The firmware on the NodeMcu uses the Lua scripting language. Lua also provides extensive support for socket programming and abstracts lower-level networking details. A custom embedded system can also be designed to use as client devices in the proposed system.

Fig. 12 represents the workflow of the client application. In initialization, the client device sets the SEND flag to TRUE, and stores the device's MAC information. The client application uses two threads. One thread is used to continuously collect the motion data of the user. The second thread is used to send the collected accelerometer data along with the device MAC address, and the location info to the server, receive a response, parse the response, and take actions based on the response. The client device connects to the server using the given IP address and port number. If a fall is detected, the client device at first sounds the buzzer. In the case of a smartphone, the client application sounds an alarm. The client device then collects the location data of the user. The client application then generates a message containing the location data and sends the message to saved emergency contacts, mediator, and emergency medical services. Thus, immediate medical attention is guaranteed by the system.

5. Experimental results analysis

5.1. Experimental Setup

The Grove - 3-Axis Digital Accelerometer (±1.5 g) was used as the accelerometer module. The SIM868 GSM module and the Ublox NEO-6 M GPS module was also used for providing connectivity. An active buzzer was chosen for generating warning noises because an active buzzer does not need external oscillators or timing circuits, unlike a passive buzzer.

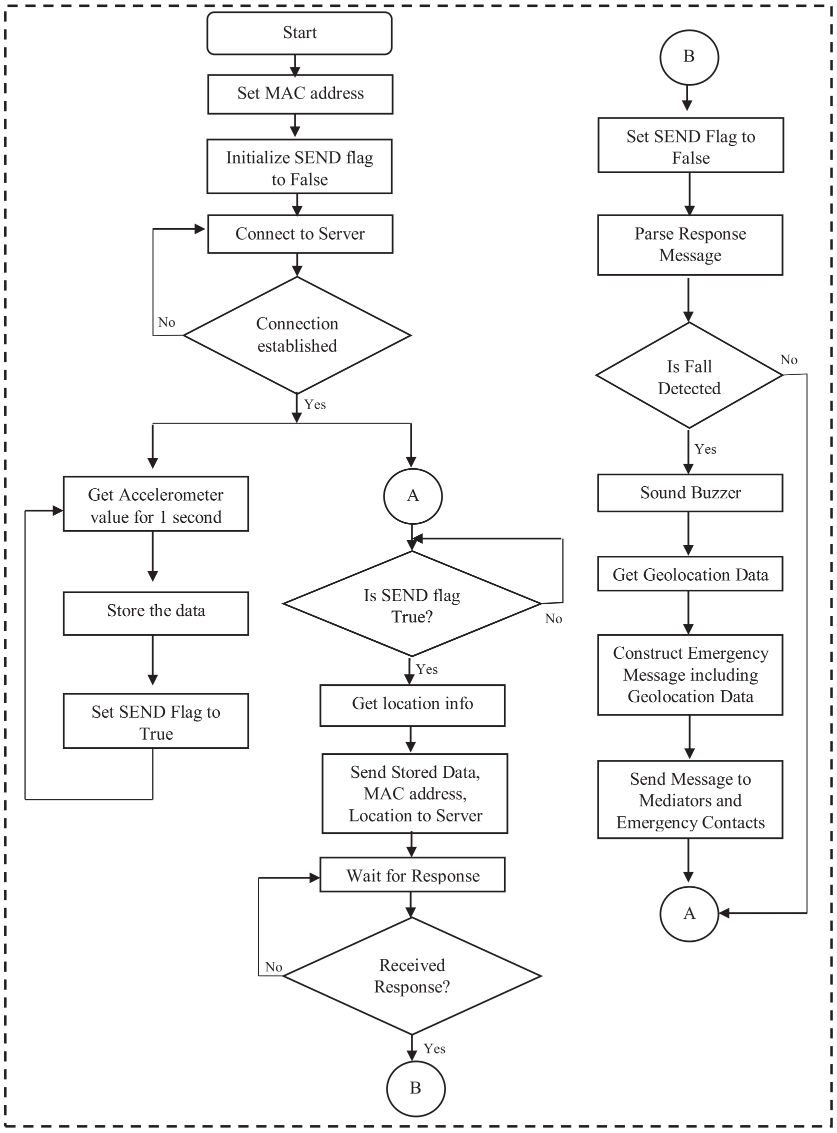


Fig. 12. Flowchart of the client application.

Two smartphones: a Xiaomi Redmi Note 4 and a Xiaomi Redmi 4A were extensively tested as client devices for the proposed system. The Xiaomi Redmi Note 4 has Octa-core 2.0 GHz Cortex-A53 CPU, Adreno 506 GPU, Qualcomm MSM8953 Snapdragon 625 (14 nm) chipset, running on Android 7.0(Nougat), with 3 GB onboard RAM. The Wi-Fi chipset used in the Xiaomi Redmi Note 4 supports Wi-Fi 802.11 a/b/g/n standard. The Xiaomi Redmi 4A has Quad-core 1.4 GHz Cortex-A53 CPU, Adreno 308 GPU, Snapdragon 425 chipset, running on Android 6.0.1 (Marshmallow). The onboard Wi-Fi chip in Redmi 4A supports 802.11 a/b/g/n Wi-Fi standard. The onboard accelerometer specifications are not known as they are usually kept secret. Smartphones are built-in with the necessary modules for the proposed system.

A Raspberry Pi 3 Model B+ was also tested as a client device. The Raspberry Pi 3 Model B+ runs on a Broadcom BCM2837B0 SoC with a 1.4 GHz, 64-bit quad-core ARM Cortex-A53 processor, with 512 KB shared L2 cache. This model of Raspberry Pi has 1 GB of RAM. The Raspberry Pi 3 Model B+ features dual-band IEEE 802.11b/g/n/ac Wi-Fi standard, and thus can easily connect and interact with the developed server. Raspberry Pi 3 Model B+ has a 40-pin layout. But all of these are digital I/O pins and do not support analog modules. A 10-bit ADC was used to interface with the analog modules with the Raspberry Pi. An Arduino Uno rev3 was also tested as a client device for the proposed system. The Arduino Uno rev3 is based on the ATmega328P microcontroller. This board contains 6 analog input pins and 14 digital I/O pins including 6 PWM output pins. But as the board does not contain any built-in Wi-Fi connectivity, we used a Wi-Fi shield. The multi-threaded server was developed and ran on a late 2013 MacBook Pro with 2.8 GHz dual-core Intel Core i7 (Turbo Boost up to 3.3 GHz) with 4 MB shared L3 cache, 16GB 1600 MHz DDR3L onboard memory, and Intel Iris Graphics running Ubuntu

14.04.5 LTS. The server was written in Python 3.6. All of these devices are compact enough to be placed into the left or right pant pocket. But special casings are necessary to protect the users' from the sharp edges in the devices like Raspberry Pi, Arduino, NodeMcu, etc.

5.2. Results analysis

To measure the performance of the developed system, several measurements are used. Specificity and sensitivity are the two statistical measures used to measure the performance of binary classifiers. Accuracy is another term used to measure the overall performance of a classifier model. Sensitivity is the proportion of actual positives that have been correctly identified as positives by the classifier. Specificity is the proportion of actual negatives that have been correctly identified as negatives by the classifier. Precision or positive predictive value is defined as the fraction of relevant instances among the retrieved instances. Both Precision and Recall describe the measurement of relevance. Precision can be seen as a measure of exactness or quality, whereas recall is a measure of completeness or quantity. In statistical analysis of binary classifiers, F_1 score is the measure of a test's accuracy. It considers both the value of the test's precision and recall to determine the score. The F_1 score is the harmonic mean of precision and recall. The maximum F_1 score is 1 (perfect precision and recall) and the minimum score is 0.

To evaluate the performance of the trained model, a confusion matrix is generated in which, True Positives (TP) are the fall data that has been correctly classified as "fall" by the classifier. True Negatives (TN) are non-fall data that has been correctly classified as non-fall by the classifier. False Positives (FP) are non-fall data that have been incorrectly classified as falls by the classifier and False Negatives (FN) are fall data that has been incorrectly classified as non-fall by the classifier. A reliable machine-learning model should have a low False Positive and Fall Negative rate.

Precision is the number of correct positive results divided by the number of all positive results returned by the classifier, and Recall is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

F1 score is calculated using the following formula.

$$F_1 = 2 * \frac{Precision\ *\ Recall}{Precision\ +\ Recall} \tag{10}$$

Sensitivity also known as True Positive Rate (TPR) or Recall is calculated using the following formula.

$$Sensitivity = \frac{TP}{TP + FN} \tag{11}$$

Specificity also known as True Negative Rate (TNR) or Selectivity is calculated using the following formula.

$$Specificity = \frac{TN}{TN + FP} \tag{12}$$

Accuracy is calculated using the following formula.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

The confusion matrix used to represent the performance of the classifier model is illustrated in Fig. 13. Out of a total of 2488 test cases, 2352 cases were True Negatives (TN), 1 case was False Negative (FN), 5 cases were False Negatives (FN), and 130 cases were True Positives (TP). The performance measures were calculated from these test values. The developed model achieved 99.7% accuracy, 96.3% sensitivity, and 99.6% specificity. The precision of the model is 99.2% and F_1 score is 97.7%. Medrano et al. [44] tested various machine learning and deep learning models such as 1NN, SVM, kNN, kNN-sum, and Kmeans+NN on the tFall dataset. Among all the models, the authors expected 1NN to perform the best. However, SVM outperformed all other tested models. The authors evaluated the models based on their AUC scores. The AUC score of the top two models, SVM and 1NN were 0.977 and 0.956, respectively. The measured AUC score of the classifier model used in the proposed system is 0.986. However, one thing to consider is that, the models in [44] were trained on the entire dataset, whereas, the model used in the proposed system was only trained on a subset of the entire dataset, as the classifier was trained to detect fall events when the client devices were put in the pant pocket (right or left) of the users. Training the classifier model on the entire dataset would reduce the AUC score. The overall performance of the developed model is illustrated in Fig. 14.

Fig. 15(a) and (b) shows an emergency message sent to emergency services by the client device and server respectively. From Fig. 15(a), it can be shown that a fall has happened in the location of the Longitude: 22.9005 N and Latitude: 89.5024E. In the case of the server, shown in Fig. 15(b), the user name John, Room No. 305 W along with Longitude and Latitude are sent to the mediator.

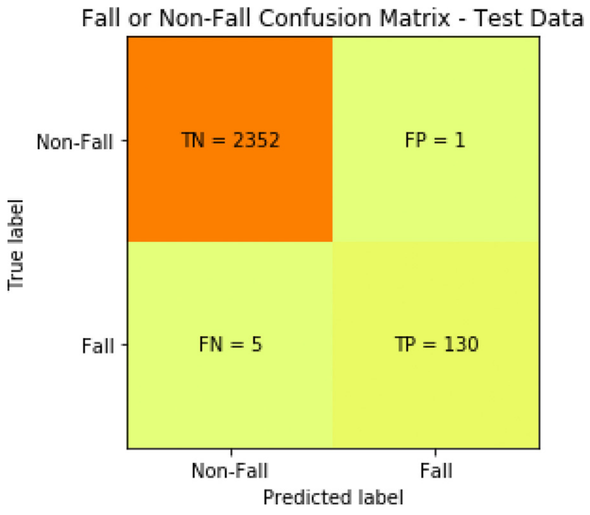
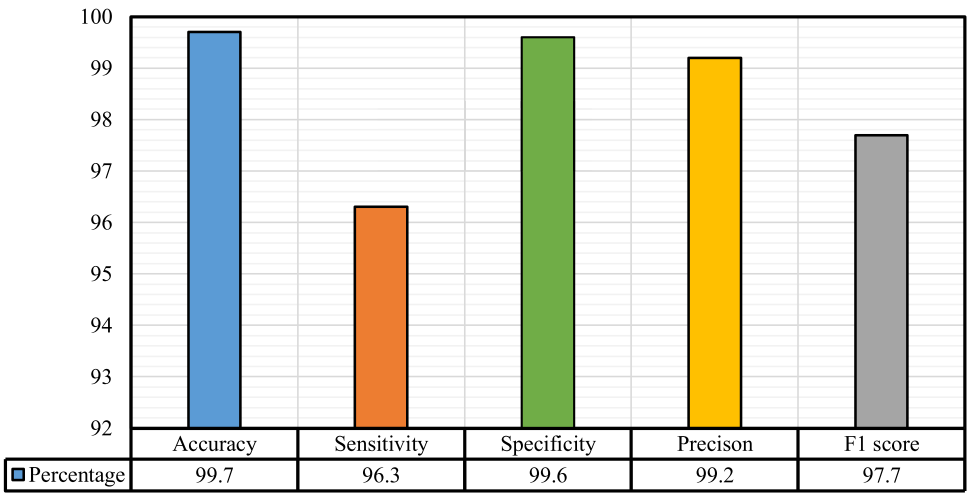
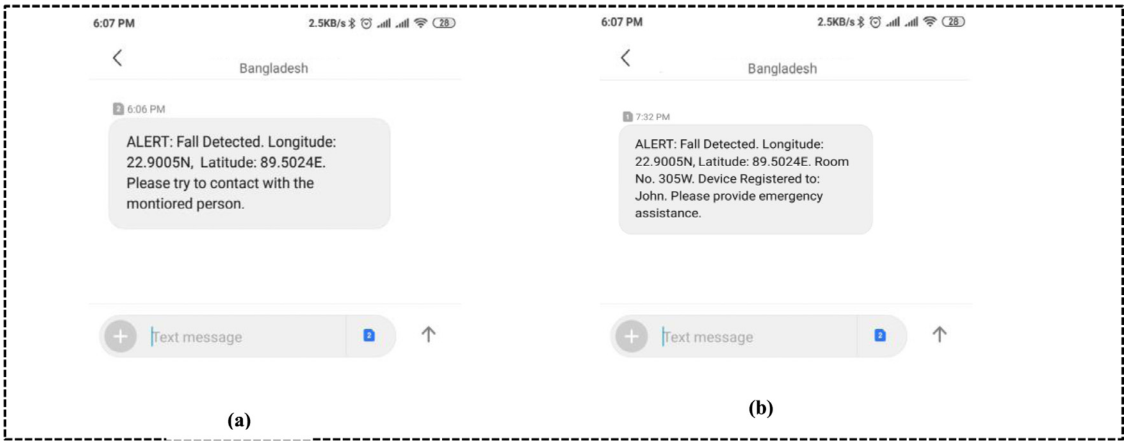


Fig. 13. Confusion matrix of the developed model.



	Accuracy	Sensitivity	Specificity	Precison	F1 score
🟦 Percentage	99.7	96.3	99.6	99.2	97.7

Fig. 14. Overall performance of the deployed model.



6:06 PM

ALERT: Fall Detected. Longitude: 22.9005N, Latitude: 89.5024E. Please try to contact with the montiored person.

7:32 PM

ALERT: Fall Detected. Longitude: 22.9005N, Latitude: 89.5024E. Room No. 305W. Device Registered to: John. Please provide emergency assistance.

Fig. 15. (a) Emergency message to emergency services by the client. (b) Emergency messages to mediator via server.

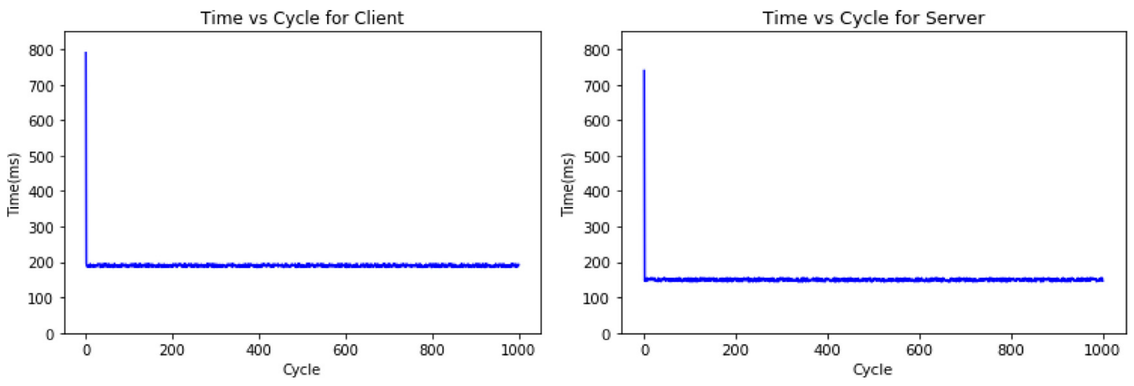


Fig. 16. Response time. (a) Client (b) Server.

5.3. *Responsiveness of clients and the server*

In the case of the client, a cycle is defined as the overall time required for a client to send the data message, getting the response message from the server and parsing the response message. The 1st cycle takes a significant amount of time, approximately 790 ms. This 2nd cycle and all other consecutive cycles take a constant amount of time, approximately 190 ms to complete. This data is consistent for the two developed clients: the Android Client, and the Raspberry Pi client. This rate depends on various variables such as the connection speed, type of network, weather condition, etc. The response time both for the client and server is shown in Fig. 16. In the case of the server, a cycle is defined as the overall time required for a thread in the server to receive the data message from a client, parse the message, make a prediction on the data, and send the data to the respective client. Like the 1st cycle of the client, the 1st cycle of the server takes a significant amount of time, approximately 740 ms. The 2nd cycle and all other consecutive cycles take approximately 150 ms to complete.

5.4. *Power consumption*

The Arduino Uno board draws about 42 mA in the idle state when no other components are connected with it. With a minimum supply voltage of 7 volts, the power consumption of the board is 0.29 Watts. The Raspberry Pi 3 Model B+ consumes about 400 mA of current at 5.0 V (which is about 1.9–2.1 Watt) when it is in idle state. The Groove 3-Axis Digital Accelerometer($\pm$1.5 g) is based on Freescale's low power consumption module, MMA7660FC. The working voltage of this module is 3.0–5.5 V. The module also has the following properties: Off Mode Current: 0.4μA, Standby Mode Current: 2μA, Active Mode Current: 47μA at 1 ODR. The GSM module SIM868 is a quad-band GSM/GPRS module that works on frequencies GSM 850 MHz, EGSM 900 MHz, DCS 1800 MHz and PCS 1900 MHz. This module is designed to run on either 3.3 V or 5 V power supply. SIM868 is designed with a power-saving technique so that the current consumption is as low as 0.65 mA in sleep mode (with GNSS engine powered down). The average current consumption in active state is 55 mA. This module consumes more power as it provides both GSM and GPRS services. The worst-case power consumption by the GPS module Ublox NEO-6 M is 67 mA. The average power consumption of the GPS module is 45 mA. The standalone GPS module is more power consuming than the other components, as the correlators inside a GPS module are constantly used to get position information. The current consumption of an active buzzer is on average 20 mA to 30 mA regardless of the supply voltage. The supply voltage can be either 3.3 V or 5 V. The supply voltage determines the loudness of the buzzer warning.

The average power consumption of the Arduino Uno when all the components are connected and active are 230–250 mA. This power draw is due to consistent network connectivity, and constant usage of power-intensive modules such as the GSM and GPS modules. When a 5000mAh power bank was used, the client device depleted the power source in about 21 h. By using very common larger capacity power banks such as 10000mAh or 20000mAh power banks, the Arduino clients can effectively run for 40~45 consecutive hours, 80~87 consecutive hours, respectively. It should also be noted that, while calculating the power consumption of Arduino Uno, the active buzzer was always active. However, in real-life scenarios, the active buzzer only becomes active when a fall event is detected. With the buzzer deactivated, the Arduino Uno runs for 23 consecutive hours on a 5000mAh power source.

The Raspberry Pi 3 Model B+ consumes more power than the Arduino Uno as it has an actual processor, RAM, etc. and can be considered as a mini-computer. The average power consumption of the Raspberry Pi when all the components are connected and active are 580–600 mA. When a 5000mAh power bank was used, the Raspberry Pi based client device depleted the power source in about 8.5 h. If larger capacity power banks such as 10000mAh or 20000mAh are used, the Raspberry Pi clients can effectively run for 16.5 h and 33.5 h, respectively. These metrics were measured while the buzzer was always active. With the buzzer deactivated, the Raspberry Pi client runs for 9 consecutive hours on a 5000mAh power source. However, the client devices become noticeably hot when operated for long hours at a time.

The Xiaomi Redmi Note 4 has a built-in Li-ion 4100mAh battery. Android devices have higher power consumption than other microprocessor-based systems. The majority of this power goes into display services, background services, etc. For calculating the power consumption, all the visible background services except the system services were terminated. Only the client application and the server were connected. The smartphone depleted the entire battery in about 14 consecutive hours. The Xiaomi Redmi 4A has a built-in Li-ion 3120mAh battery. Like the Xiaomi Redmi Note 4, all other background services except the system services were terminated. Only the client application and the server were connected. The smartphone ran the client application for 12 consecutive hours before shutting down due to completely depleted battery. The smartphones were noticeably colder than microprocessor-based clients. This is possibly due to better cooling and throttling systems in smartphones.

6. Discussion

6.1. Implementation challenges

To properly integrate the system into a large-scale environment, various network infrastructure equipment such as routers, access points, servers, etc. would have to be set up. With the advent of faster internet in recent years, internet speed would not be a problem. Another concern is the battery limitations of the client devices. As all of the client devices rely on battery power, they would regularly need recharging. While smartphones would be powered by their batteries, other client devices employing Raspberry Pi, Arduino, NodeMCU, etc. can use portable power sources like power banks. Smartphones can be recharged when elderly people are safely taking rest. The depleted power banks can be swapped with charged power banks in a matter of minutes for Raspberry Pi, Arduino, NodeMCU, etc. powered client devices. Power banks would also provide more service time as they generally hold 3x-5x more power than traditional smartphone batteries, and devices powered by microprocessors waste less power in background processes, OS related tasks, scheduling, etc. Thus, microprocessor-based clients with portable power banks would have a clear advantage over smartphone-based clients in terms of power usage. But smartphones have other utilities such as easier communication, relaxation, etc. A single generic server in an establishment can easily take care of multiple thousand clients. Many online services lend servers to customers. But using a rented server might add propagation delay and other security issues to the system.

6.2. Limitations of the system

One limitation of the proposed system is that it requires consistent network connectivity between the clients and the server. Hence, if network connectivity is unavailable for some reason, the system becomes unable to provide the necessary detection and rescue services. To remedy this, a simple manual button in case of microprocessor-based clients and a virtual button in case of smartphone clients can be added. This button will act like an “SOS” button, and when pressed, will provide the necessary rescue services such as contacting the emergency services, sounding the buzzer to warn nearby people, etc. A simple threshold-based detection algorithm can also be developed which will provide detection and rescue services only during the unavailability of network services. Another limitation of the developed system is that the client devices can only detect the falls if the devices are put in the pant pocket (left or right) of the patients. The underlying classifier was trained on accelerometer data recorded during ADL and fall activities while the recording device was put on the pant pockets. tFall dataset also contains accelerometer readings while the recording device was put in other locations such as the handbag, chest, etc. A simple solution to this problem would be to train the model on the entire dataset. The system would then be capable of detecting fall events regardless of the clients’ location in the user’s body. Another limitation of the system might be that the client devices are not waterproof all the time. Although some current smartphones are waterproof, microprocessor-based devices and general smartphones are not. Thus, the respective protective casing of the client devices should be provided.

6.3. Stability of the system

One of the main complexities generally faced by fall detection systems is the failure to distinguish between normal lying down ADLs and fall events. The proposed system is perfectly capable of distinguishing between these events as evident from the trained model’s evaluation metrics. The high levels of precision and specificity of the model, respectively, 99.2% and 99.6% indicate that the model has good separation between the two classes and can confidently detect any type of fall events separately from ADLs. Another complexity faced by fall detection systems is difficulties in determining falls that happen suddenly, or fall events that happen over a long time. The tFall dataset was created by collecting the accelerometer readings from real-time ADL and fall events. The ADL and fall events were not artificial in any way and no post-processing was done on the collected data. Thus, the dataset contains fall data from real-life scenarios. In real-life scenarios, falls happen quickly or over a long period. As the model used on the proposed system was trained on the tFall dataset, and the model shows high levels of accuracy, the system is perfectly capable of detecting fall events that happen very fast or fall events that occur over a long period. The developed system requires consistent network connectivity between the clients and the server. If the connection is dropped, the clients continuously try to connect with the server. In the case of network connectivity failure, fall detection and rescue services become unavailable. To remedy this issue, threshold-based local fall detection algorithms can be developed in the future that will provide detection and rescue services when network connectivity becomes unavailable for some reason.

7. Conclusion

In this paper, an IoT based fall detection system is proposed which is device-type invariant, meaning, all types of devices such as smartphones, Raspberry Pi, Arduino, NodeMcu, etc. can be used as long as they interface with the necessary modules: a tri-axial accelerometer, a buzzer, GSM, GPS, and Wi-Fi. The system is based on a client-server architecture. The multi-threaded server hosts a pre-trained linear classifier model. The devices connect with threads of the server and continuously send motion data from the accelerometer to the server. The server predicts if a fall has occurred or not and responds accordingly. The devices receive the response message. If a fall is detected, the server sends an emergency message containing the distressed client's location and other relevant information to the mediator. As a failsafe, the device sounds the buzzer and contacts emergency services and mediators via SMS containing location data of the device. The monitored person can thus get immediate medical assistance. This system can be implemented in large-scale environments where real-time monitoring of a lot of persons is necessary, such as hospitals, nursing homes, etc. The developed linear classifier model used in this system achieved 99.7% accuracy, 96.3% sensitivity, and 99.6% specificity. The developed system is very fast as the response time is very fast, approximately, 190 ms. This means, within approximately 190 ms of sending the motion data to the server, the server and client's device knows whether a fall has occurred or not.

8. Future works

The system presented in this paper requires constant network connectivity. In the future, a threshold-based algorithm can be developed that will only work as a failsafe in case of network connectivity failure. If the client device detects falls via the threshold-based algorithm, it will perform the necessary tasks such as notifying the mediators and emergency contacts, warning nearby personnel by sounding the buzzer, etc. until network connectivity is restored. A manual button in the case of microprocessor-based clients and a virtual button in the app in the case of smartphones can be developed. This button will act as an "SOS" button for the clients. When this button is pressed, the client devices will provide rescue services such as contacting emergency services and mediators, sounding the buzzer to warn nearby people about the fall event, etc. A "Cancel Alert" button can also be added which will let the mediators and emergency contacts know that the user is fine, in case of a false detection by the client device. UniMiB SHAR and tFall datasets both used smartphones and accelerometer to record the ADL and fall data and have similar sampling frequency. These two datasets can be merged in the future to create a single large dataset. This hybrid dataset can be then used to train more robust and accurate models. tFall dataset also contains accelerometer readings of fall and ADL events when the recording device was put in the handbook, pocket, etc. location. If all the types of falls and ADL's from the tFall dataset are used to train the model, the system will also be location independent, as the client device would be able to detect falls and take necessary actions regardless of its position in the patient's body, i.e. neck, waist, handbag, arm, etc. The form factor of the microprocessor-based client devices can also be improved in the future.

Declaration of Competing Interest

The authors have no conflict of interest.

References

[1] United Nations, Department of economic and social affairs, population division. World Population Prospects: The 2017 Revision, Key Findings and Advance Tables. Working Paper No. ESA/P/WP/248, 2017.
[2] World Health Organization, Falls [Online]. Available http://www.who.int/mediacentre/factsheets/fs344/en/,accessed on: Sep. 20, 2018.
[3] Nihseniorhealth: Causes and Risk Factors [Online]. Available: http://nihseniorhealth.gov/falls/causesandriskfactors/01.html/ ,accessed on: Sep. 20, 2018.
[4] S.R. Lord, H.B. Menz, C. Sherrington, Home environment risk factors for falls in older people and the efficacy of home modifications, Age Ageing 35 (Suppl 2) (Sep. 2006) ii55–ii59.
[5] M. Mubashir, L. Shao, L. Seed, A survey on fall detection: principles and approaches, Neurocomputing 100 (2013) 144–152.
[6] S.C. Mukhopadhyay, Wearable sensors for human activity monitoring: a review, IEEE Sens. J. 15 (3) (2015) 1321–1330.
[7] S. Chaudhuri, H. Thompson, G. Demiris, Fall detection devices and their use with older adults: a systematic review, J. Geriatr. Phys. Ther. 37 (4) (2014) 178 (*2001*).
[8] B. Senouci, I. Charfi, B. Heyrman, J. Dubois, J. Miteran, Fast prototyping of a SoC-based smart-camera: a real-time fall detection case study, J. Real Time Image Process. 12 (4) (2016) 649–662.
[9] S. Wang, L. Chen, Z. Zhou, X. Sun, J. Dong, Human fall detection in surveillance video based on PCANet, Multimed. Tools Appl. 19 (2016) 11603–11613.
[10] Z.P. Bian, J. Hou, L.P. Chau, N. Magnenat-Thalmann, Fall detection based on body part tracking using a depth camera, IEEE J. Biomed. Health Inform. 2 (2015) 430–439.
[11] E.E. Stone, M. Skubic, Fall detection in homes of older adults using the Microsoft Kinect, IEEE J. Biomed. Health Inform. 19 (1) (2015) 290–301.
[12] G. Mastorakis, D. Makris, Fall detection system using Kinect's infrared sensor, J. Real Time Image Process. 9 (4) (2014) 635–646.
[13] B.Y Su, K.C. Ho, M.J. Rantz, M. Skubic, Doppler radar fall activity detection using the wavelet transform, IEEE Trans. Biomed. Eng. 62 (3) (2015) 865–875.
[14] B. Jokanović, M. Amin, Fall detection using deep learning in range-Doppler radars, IEEE Trans. Aerosp. Electron. Syst. 54 (1) (2018) 180–189.
[15] L. Klack, C. Möllering, M. Ziefle, T. Schmitz-Rode, Future care floor: a sensitive floor for movement monitoring and fall detection in home environments, in: Proceedings of the International Conference on Wireless Mobile Communication and Healthcare, Berlin, Heidelberg, Springer, 2010, pp. 211–218.
[16] M. Daher, A. Diab, M.E.B.E. Najjar, M.A. Khalil, F. Charpillet, Elder tracking and fall detection system using smart tiles, IEEE Sens. J. 17 (2) (2017) 469–479.

[17] Y. Li, K.C. Ho, M. Popescu, A microphone array system for automatic fall detection, IEEE Trans. Biomed. Eng. 59 (5) (May 2012) 1291–1301.
[18] G. Feng, J. Mai, Z. Ban, X. Guo, G. Wang, Floor pressure imaging for fall detection with fiber-optic sensors, IEEE Pervasive Comput. 15 (2) (2016) 40–47.
[19] Q. Guan, C. Li, X. Guo, B. Shen, Infrared signal based elderly fall detection for in-home monitoring, in: Proceedings of the Ninth International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), 1, IEEE, 2017, pp. 373–376.
[20] W.H. Chen, H.P. Ma, A fall detection system based on infrared array sensors with tracking capability for the elderly at home, in: Proceedings of the Seventeenth International Conference on E-health Networking, Application & Services (HealthCom), IEEE, 2015, pp. 428–434.
[21] Y. Zigel, D. Litvak, I. Gannot, A method for automatic fall detection of elderly people using floor vibrations and sound-proof of concept on human mimicking doll falls, IEEE Trans. Biomed. Eng. 56 (12) (Dec. 2009) 2858–2867.
[22] F. Wu, H. Zhao, Y. Zhao, H. Zhong, Development of a wearable-sensor-based fall detection system, Int. J. Telemed. Appl. 2015 (2015) 2.
[23] D. Lim, C. Park, N.H. Kim, S.H. Kim, Y.S. Yu, Fall-detection algorithm using 3-axis acceleration: combination with simple threshold and hidden Markov model, J. Appl. Math. (2014) 2014.
[24] B. Aguiar, T. Rocha, J. Silva, I. Sousa, Accelerometer-based fall detection for smartphones, in: Proceedings of the IEEE International Symposium on Medical Measurements and Applications (MeMeA), IEEE, 2014, pp. 1–6.
[25] Y. Li, G. Chen, Y. Shen, Y. Zhu, Z. Cheng, Accelerometer-based fall detection sensor system for the elderly, in: Proceedings of the IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, 3, IEEE, 2012, pp. 1216–1220.
[26] L. Montanini, A.D. Campo, D. Perla, S. Spinsante, E. Gambi, A footwear-based methodology for fall detection, IEEE Sens. J. 18 (3) (2017) 1233–1242.
[27] P. Pierleoni, A. Belli, L. Palma, M. Pellegrini, L. Pernini, S. Valenti, A high reliability wearable device for elderly fall detection, IEEE Sens. J. 15 (8) (2015) 4544–4553.
[28] J.K. Lee, N.R. Stephen, E.J. Park, Inertial sensing-based pre-impact detection of falls involving near-fall scenarios, IEEE Trans. Neural Syst. Rehabil. Eng. 23 (2) (2015) 258–266.
[29] A.M. Sabatini, G. Ligorio, A. Mannini, V. Genovese, L. Pinna, Prior-to-and post-impact fall detection using inertial and barometric altimeter measurements, IEEE Trans. Neural Syst. Rehabil. Eng. 24 (7) (2016) 774–783.
[30] P. Pierleoni, A. Belli, L. Maurizi, L. Palma, L. Pernini, M. Paniccia, S. Valenti, A wearable fall detector for elderly people based on AHRS and barometric sensor, IEEE Sens. J. 16 (17) (2016) 6733–6744.
[31] E. Casilari, M.A. Oviedo-Jiménez, Automatic fall detection system based on the combined use of a smartphone and a smartwatch, PloS One 10 (11) (2015) e0140929.
[32] T. Vilarinho, B. Farshchian, D.G. Bajer, O.H. Dahl, I. Egge, S.S. Hegdal, A. Lønes, J.N. Slettevold, S.M. Weggersen, A combined smartphone and smartwatch fall detection system, in: Proceedings of the IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, IEEE, 2015, pp. 1443–1448.
[33] L. Chen, R. Li, H. Zhang, L. Tian, N. Chen, Intelligent fall detection method based on accelerometer data from a wrist-worn smart watch, Measurement 140 (July 2019) (2019) 215–226.
[34] J. Yuan, K.K. Tan, T.H. Lee, G.C.H. Koh, Power-efficient interrupt-driven algorithms for fall detection and classification of activities of daily living, IEEE Sens. J. 15 (3) (2015) 1377–1387.
[35] L.J. Kau, C.S. Chen, A smart phone-based pocket fall accident detection, positioning, and rescue system, IEEE J. Biomed. Health Inform. 19 (1) (2015) 44–56.
[36] A.L.S. De Lima, LJW Evers, T. Hahn, L. Bataille, J.L. Hamilton, M.A. Little, Y. Okuma, B.R. Bloem, M.J. Faber, Freezing of gait and fall detection in Parkinson's disease using wearable sensors: a systematic review, J. Neurol. 264 (8) (2017) 1642–1654.
[37] T.B. Rodrigues, D.P. Salgado, M.C. Cordeiro, K.M. Osterwald, F.B. Teodiano Filho, V.F. de Lucena Jr, E.LM Naves, N. Murray, Fall detection system by machine learning framework for public health, Proc. Comput. Sci. 141 (2018) 358–365.
[38] O. Aziz, J. Klenk, L. Schwickert, L. Chiari, C. Becker, E.J. Park, G. Mori, S.N. Robinovitch, Validation of accuracy of SVM-based fall detection system using real-world fall and non-fall datasets, PLoS One 12 (7) (2017) e0180318.
[39] Bogdan Kwolek, Michal Kepski, Improving fall detection by the use of depth sensor and accelerometer, Neurocomputing 168 (2015) 637–645.
[40] A. Abobakr, M. Hossny, S. Nahavandi, A skeleton-free fall detection system from depth images using random decision forest, IEEE Syst. J. 12 (3) (2018) 2994–3005.
[41] P. Bilski, P. Mazurek, J. Wagner, W. Winiecki, Application of decision trees to the fall detection of elderly people using depth-based sensors, in: Proceedings of the IEEE International Conference on Intelligent Data Acqut and Advanced Computing Systems (IDAACS 2015), 2015.
[42] P. Vavoulas, T. Spanakis, The mobifall dataset: an initial evaluation of fall detection algorithms using smartphones, in: Proceedings of the IEEE 13th International Conference on Bioinformatics and Bioengineering (BIBE 2013), 2013, pp. 1–4.
[43] C. Vavoulas, P. Malliotakis, The mobiact dataset: recognition of activities of daily living using smartphones, in: Proceedings of the International Conference on Information and Communication Technologies for Ageing Well and e-Health (ICT4AWE), 2016, pp. 143–151.
[44] C. Medrano, R. Igual, I. Plaza, M. Castro, Detecting falls as novelties in acceleration patterns acquired with smartphones, PLoS One 9 (4) (2014) e94811.
[45] F. Vilarinho, D. Bajer, H. Egge, S. Lones, Weggersen, A combined smartphone and smartwatch fall detection system, in: Proceedings of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015, pp. 1443–1448.
[46] C. Wertner, P. Schindler, An open labelled dataset for mobile phone sensing based fall detection, in: Proceedings of the Twelfth EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS 2015), 2015, pp. 277–278.
[47] Santoyo-Ramón Casilari, Cano-García, Analysis of a smartphone-based architecture with multiple mobility sensors for fall detection, PLoS One 11 (12) (2016) e0168069.
[48] M. Micucci, Napoletano, UniMiB SHAR: a new dataset for human activity recognition using acceleration data from smartphones, IEEE Sens. Lett. (2) (2016) 15–18.
[49] S. Mellone, C. Tacconi, L. Chiari, Validity of a smartphone-based instrumented timed up and go, Gait Posture (36) (2012) 163–165.
[50] M.V. Albert, K. Kording, M. Herrmann, A. Jayaraman, Fall classification by machine learning using smobile phones, PLoS One 7 (5) (2012) e36556.
[51] P. Vogt, J. Kuhn, Analyzing free fall with a smartphone acceleration sensor, Phys. Teach 50 (2012) 182–183.
[52] C. Chan, I. Verdiesen, J. Carvajal-Godínez, P.S. Mani, TensorFlow^{TM} - Open Source library for machine learning applications, TU Delft Students On Software Architecture, 2016 Edition, GitBook, May 2016.
[53] G.X. Yuan, C.H. Ho, C.J Lin, Recent advances of large-scale linear classification, Proc. IEEE 100 (9) (2012) 2584–2603.