**CHAPTER 2**

**LITREATURE SURVEY**

**2.1 Fall Detection using Smartphones**

An SP-based (Smartphone)fall detector that uses a combination of TBM and ANN . In spite of the reported 100% classification performance in offline analysis, the data set used for training and testing the ANN was very small (86samples in total). Also, the performance of the application(false alarms rate and battery consumption) in real-life scenarios, commonly known as online analysis, was not presented. More recently, Kerdegari et al. developed an Android application, SFD, built on the notion of multi-layer perceptron(MLP) neural network for fall event detection [17]. During offline analysis, their algorithm achieves 92:03% sensitivity,91:07% specificity, and 91:06% accuracy on data recorded around the waist. However, when applied for online analysis, the system performance slightly degrades in terms of specificity (93:18% sensitivity, 88:88% specificity, and 91:25% accuracy). Additional to the long system decision time (at least 30 s for algorithm decision + 60 s default time for alert cancellation),no specific information on the false alarm rate with respect to the placement of the SP is provided. Furthermore, more battery power is drained due to the continuous use of

ANN. Successful deployment of a fall detection system among elderly population depends on various factors: usability, battery lifetime, privacy issues, cost, and reliability. System usability hugely depends on the number of sensors used, their prefix orientation, locations, battery life, and the like. The system should be reliable, i.e., it should detect almost all the fall events with 100% sensitivity while keeping the false alarm rate at its minimum. High false alarm rates can be very annoying and ultimately reduce user compliance with the system. To decrease false alarms, some researchers utilize postural information by either employing multiple sensors or by using a single tri-axial sensor with prefix orientation.

As older people tend to easily forget, neither of the two options serve convenient in terms of usability. As the existing literature relating to these issues is lacking, there is no widely accepted system among elderly until now. Hence, it is of great significance to develop a reliable fall detection system to fill this gap while accounting for the existing practical issues.

Accordingly, we present an automated high performance SP-based fall detection system focusing on practical issues such as user convenience and power consumption. The proposed

standalone fall detector is developed as an Android app, namely FallDroid, which uses the accelerometer sensor embedded in SPs. The designed application provides an elder friendly GUI and supports the two most convenient SP carrying locations: waist (belt/pouch) and thigh (pant pocket). In comparison with ML techniques, the proposed two-step algorithm is shown to be more power-efficient. In the first step, a low computational cost approach based on TBM is used, followed by the pattern recognition technique, multiple kernel learning support vector machine (MKL-SVM) in the second step which is rarely invoked. The battery consumption was analysed and reported for different scenarios. The recorded datasets were acquired from human trials conducted systematically in both, laboratory and free living environments

**2.2 Malware Detection in Android**

Android platform includes a multi-user operating system based on a Linux kernel, middleware, and a set of applications(apps). Users install apps acquired from app markets, e.g.,official Google’s play or alterative app markets. Android implements a number of security mechanisms of which the most prominent includes app sandbox and a permission framework that enforces access control to core functionalities. App sandbox is set up in a kernel lever. It enforces security between apps and the system through identifying and isolating app resources. Each Android app is assigned a unique User ID (UID) and run as the user in a separate process. Under the app sandbox mechanism, apps cannot interact with each other and an app has limited access to the operating systems. While Android apps are mainly programmed in Java, native codes can also be integrated with Java apps. All types of apps, including Java, native or the hybrid are sandboxed in the same way and thus have the same degree of security. One of the central design points of the Android security mechanism is permission control. As Android sandboxes apps from each other, apps must explicitly declare the permissions they need for additional capacities. Without a permission, an application by default is not able to do anything that could adversely impact the user experience, the network, or data on the device. The Android app developer statically declares the permissions the app requests in a manifest file (AndroidManifest.xml). When a user installs an app, a dialog will be displayed to indicate a permission list the app requests and asks the user whether to continue the installation. This is an all-or-nothing decision. If the user decides to install the app, all the requested permissions will be granted. The user is not able to grant or deny individual permission. The permissions are applied once the app is installed. The user will no longer be notified of the permissions granted in the running the app.

While there exist some third-party apps (or rooting the device) that can help to manage the permissions on a per-app basis, normally if the user wants to block the permissions granted at

the install time, the user needs to remove the app. Android permissions provide a mechanism of access control to core facilities of the system. However, it imparts a significant responsibility to both the app developers and the app users. The developers need to accurately specify the requested permissions and the users need to understand the risk involved and thus make a rational decision regarding whether install the app or not. Ideally, the developer should follow the least-privileged set of permission requests and the user should understand the risk of granting certain combinations of permissions. Android itself provides an attribute called “protection Level” that characterizes the potential risk implied in the permissions. The permission attributes can be categorized as “normal”, “dangerous”, “signature” and “signature Or System”. The last two attributes are system granted only. The permission with the normal attributes is lower-risk (e.g., SET\_WALLPAPER) and will be automatically granted, without asking for the user’s explicit approval. The permission tagged as dangerous is higher-risk (e.g.,READ\_SMS) that would access to private user data or control over the device. However, the two permission categorizations provided by Android are very coarse for both the developers and the users. In this work, our goal is to systematically rank the permissions w.r.t. their risk for the users to have a better understanding of permissions, and to identify a subset of risky permissions that are most relevant to malapps for the developers to accordingly decide how to declare the permission requests. In addition, we are motivated to analyse and detect malapps with permission matrix and construct a decision rule set to universally detect unknown malapps. Our work would provide a whole picture of the relationships between the permission usage and their risk in Android apps, and a vision regarding the use of only permissions for the analysis and detection of malapps.