JOURNAL OF SOFTWARE ENGINEERING & INTELLIGENT SYSTEMS
ISSN 2518-8739
31st December 2018, Volume 3, Issue 3, JSEIS, CADMEI Copyright © 2016-2018
www.jseis.org

AUTOMATED TESTING OF MOTION-BASED EVENTS IN MOBILE APPLICATION

[1] SEYEDEH SEPIDEH EMAM, [2] JAMES MILLER

[1,2]*Electrical and Computer Engineering Department, University of Alberta, Edmonton, Canada*
Email: [1]emam@ualberta.ca, [2]jimm@ualberta.ca

OPEN ACCESS

Academic Editor: Hikmat Ullah Khan, COMSATS, Pakistan.

Published: 31st December, 2018

Funding: The author has no funds.

Article Type: Research Article

ABSTRACT

Automated test case generation is one of the main challenges in testing mobile applications. This challenge becomes more complicated when the application being tested supports motion-based events. In this paper, we proposed a novel, hidden Markov model (HMM)-based approach to automatically generate movement-based gestures in mobile applications. An HMM classifier is used to generate movements, which mimic a user's behaviour in interacting with the application's User Interface (UI). We evaluate the proposed technique on four different case studies; the evaluation indicates that the technique not only generates realistic test cases, but also achieves better code coverage when compared to randomly generated test cases.

Keywords : automated test case generation; motion-based events; hidden Markov model; mobile application; classification;

1. INTRODUCTION

In recent years, mobile devices have been produced in various types and shapes, offering a wide range of services and features. It is a very difficult task to develop mobile applications that are able to work appropriately on different mobile devices and operating systems (OSs) [1], [2]. On the other hand, releasing applications that are not fully functional, usable, and consistent can risk the developer's reputation in such a competitive environment. Testing the application's functionality and verifying its robustness are key factors in improving the application's quality.

Embedding new hardware devices, such as movement sensors (accelerometers and gyroscopes), in smartphones and tablets further complicates the testing procedure. Users are able to interact with the application by touching, tilting, shaking, and rotating the mobile devices. When a device is in motion or it's screen is continuously touched, the probability of unintentional inputs increases; in such circumstances, automatically generated test suites are needed to produce accurate test cases and accelerate the mobile application testing procedure.

Some tools and techniques have been developed to test the quality of the source code for mobile applications[3]–[5], but the number of approaches that focus on automated testing is still very limited. The majority of these automated testing tools offer capture-and-replay functionality to test the application's User Interface (UI) [6]–[8]. For instance, Choudhary et al. [5] have conducted a study on existing testing tools for Android applications. Although the study dealt with techniques for testing the mobile applications, it doesn't provide any insight or mechanism for generating test cases for motion-based mobile applications. In addition to this, the case studies considered in [5] do not have any motion-based facilities, and hence are not suitable to be utilized in this study.

Writing and continually improving motion-based test cases is a difficult task when testing mobile applications that use movement-sensor data. Therefore, considering existing mobile testing tools and approaches, two problems can be noticed: 1) no automated approach is provided (this problem is considered as a technical challenge in this study); and 2) generating test cases for motion-based mobile applications remains unconsidered (this problem is considered as a scientific challenge in this study). Thus, in this paper, we propose a new approach to address these limitations. It is argued that mimicking users' behaviours is one of the key factors in generating gesture-based test cases. It helps in executing realistic test scenarios and standard gestures [9][10].

In order to automatically generate test cases, mimicking human generated gestures in motion-based mobile applications, we propose a novel approach, which synthesizes the motions, and subsequently, simulates the test cases based upon the formalized gestures. Motion data is represented by the data captured, using the movement sensors and the objects' positions (2D coordinates) on the screen. An application can then use the sequences of motions to simulate the gestures and test the UI. To increase the chance of generating realistic movements, a set

Corresponding Author: Seyedeh Sepideh Emam
Email Address: emam@ualberta.ca

of training data is generated by human users and is used to train the hidden Markov model (HMM) classifiers. The models are iteratively used to generate new motion sequences for the application testing procedure. Gestures and animations are commonly considered to be the key components in modern mobile user interface design; hence this work directly targets the heart of the matter in this new and evolving application domain.

Our experiments on sample Android applications that support motion-based gestures reveal the effectiveness of the proposed approach as an automated testing process. Although, the focus of the empirical evaluation of the proposed approach is on Android applications, it is worth noting that the algorithm is also applicable to motion-based iOS applications and applications found on other mobile platforms.

In summary, the generated motions are used to automatically produce test cases, mimicking human-generated gestures with the technical goal of increasing code coverage. Therefore, the process is highly beneficial during regression testing, since the generated test cases can later be executed on newer versions of the application to uncover issues in the system.

This study contributes to the research in this area by:

- Proposing a new approach to synthesize motion data, and make it executable as a test input to the application being tested.
- Applying a HMM classifier on the training data to create a set of HMMs, and subsequently using them to generate motion sequences.
- Evaluating the effectiveness of the proposed approach in terms of, (1) mimicking the user's behaviour, and (2) increasing the code coverage of the software under test (SUT).

This paper is organized as follows. Section 2 provides related work and background information and definitions relating to mobile applications, particularly motion-based gesture testing. Section 3 describes an overview of the proposed approach, the gesture synthesis and simulation procedures, while Section 4 provides the design and implementation details of the proposed technique. Section 5 provides a running example of the proposed test case generation approach using real data. Section 6 discusses the evaluation phase, experimental setup, and results. Section 7 explains the experiment's run-time analysis. Section 8 examines the study's limitations and the threats to its validity. Finally, Section 9 presents the overall conclusions and some thoughts on potential future research.

2. RELATED WORK

2.1 Mobile application testing

Testing is a crucial activity in a software development procedure. Producing a defect-free application, addressing all of the requirements of the users, along with providing fully functional, consistent, and highly usable services are vital in highly competitive environments. Over the past few years, phenomenal progress in the mobile device market has led to an outstanding growth in the mobile application development industry. However, the growth in developing mobile testing procedures and techniques has been insufficient. Although many testing methods and tools exist for desktop and server/host software, most of them are not applicable for testing "mobile software" [11]. Moreover, most existing test generation techniques rely on a crawler to explore the dynamic states of the application under test. Such approaches are automated and systematic but lack the domain knowledge of system experts. Far et al. [12] propose a new technique to combine the human knowledge present in the form of input values with the inferred knowledge of automated crawling. Similar approaches have not been applied in testing mobile applications. Hence, in this study, we propose an approach, which relies on both human and automated exploration data. Ermuth et al. [13] also presents a UI-level test case generation technique that applies human-produced execution traces in order to automatically create complex sequences of events that are able to cover more pages, scenarios and code lines compared to a purely random test case generator. This approach relies on inferring sequences of low-level UI events (macro events) using data mining techniques and the inference of finite state machines (FSMs). However, Ermuth's technique [13] also has never used to test mobile applications.

Although many traditional testing tasks are common between mobile applications and the desktop/web-based applications, several key factors cause challenges in the mobile testing procedure. For example, the variety of mobile devices and diversity in OSs cause difficulties in testing device-specific factors [9]. Mobile devices are different in terms of screen sizes, platforms, input methods, and the quality of the sensor data. Such differences can easily multiply testing efforts. For testing an application, it needs to be exposed to a sufficient number of devices from different models, screen sizes, and OS versions. Covering an adequate number of factors leads to generating a large number of test cases that are required to be executed in an environment where short release cycles are common. This can easily affect the quality of the application, along with the time of the marketplace and the costs of construction. Integrating automation approaches with test case generation procedures is a key

factor in addressing these issues in the "mobile testing era", where many test cases need to be executed on a large selection of mobile devices and configurations to reproduce defects.

In this regard, [14] presents a framework to test the functionality of mobile applications when a device is moved to a new network. The framework uses an application-level emulator as a mobile agent to carry the application across networks to ease the testing process under different network technologies. Additionally, [15] suggests a quality assurance framework to define key patterns and metrics in mobile application testing. Although these research studies provide insights into the testing of the mobile applications, they still do not cover the test case generation phase. Several studies with a special focus on automated testing for mobile applications have also been conducted; [16]–[20] suggest different, automated, graphical user interface (GUI) testing approaches for Android applications. For example, [16] produces *AndroidRipper*; this tool seeks to explore the application's GUI and evaluate its effectiveness in terms of fault detection ability when compared to random approaches. Android Monkey generates purely random tests for Android mobile applications using a brute-force mechanism. Android Monkey usually achieves shallow code coverage compared to other state of the art approaches used for testing GUI in Android applications [21]–[23]. Mao et al. [24] also proposes another framework which combines random fuzzing, systematic and search-based exploration, exploiting seeding and multi-level instrumentation in order to automatically explore and optimise test sequences in Android applications. Moreover, [25] presents a new approach to automatically generate test oracles for testing user-interaction features found in mobile applications. Given a model of the mobile application's UI, this framework uses a library of oracles and generates a test suite to test the user-interaction features in the application. There also has been some work [26] that uses contextual information to randomly generate inputs to test mobile applications and automatically find crashes. Such approaches are more focused on discovering; reporting and reproducing crashes are not practically used to generate functional test cases covering the source code.

Although, some automated test case generation techniques are suggested for testing the UI of mobile applications, but their functionality and applicability in testing the new features of today's mobile phones are far from perfect. To test the UI, the mobile application needs to be executed with user interaction events. With technological advancement in smartphones and tablets, natural user interfaces (NUIs), which no longer use keyboards and keypads as human-machine interfaces, have become popular. Touch-sensitive screens, speech recognizers, and gesture detectors are the primary interaction channels in the new generation of mobile applications. This era of application testing is relatively new, and only a limited number of studies have been performed to address these testing challenges [9], [10].

2.2 Testing Motion-based Gestures

Mobile applications, which allow users to control the applications' functionality through NUIs, normally recognize gestures by using the data provided by the embedded sensors in the mobile device [27]. Several smartphones and tablets contain accelerometers to control motion inputs. One of the most common applications of accelerometers is presenting the landscape and portrait views of the screen based on the way the device is being held [28]. The 3-axis model of the accelerometer is able to measure the magnitude and direction of the acceleration (gravitational force) as a vector $[ax_k, ay_k, az_k]$ for a motion k in a 3D space. Each acceleration parameter measures changes in velocity over time along a linear path. Combining all three accelerations, lets the application detect the device's movement in any direction and obtain the device's current orientation. Depending on the graphical capabilities of mobile applications, 2D or 3D versions of the acceleration vector are considered. Obviously, 2D applications do not use $az_k$to indicate a motion k. From the tester's perspective, testing applications that support motion-based events introduce a new complexity to the testing procedure; motion-based gestures should be accurately specified and reliably reproduced [9]. The lack of formal motion-gesture specification prevents testers from developing an automated test generation approach. To simulate the motions, atomic gestures should be formalized. The next section presents the simulation and synthesis procedures of motion-based events (gestures).

3. GESTURE SIMULATION

In the simplest test-case generation process, the test data-points can be provided to the application by using a random test generation approach, which randomly creates data frames within a defined range to move the object on the screen. It can be expected that the number of reasonable gestures, which are created randomly, are very limited. Therefore, even if these test cases are able to cover an acceptable number of branches in the source code, they may not be able to reveal faults a human user can discover simply because they cannot replicate standard gestures [10].

This study considers an automated test case generation procedure for mobile applications interacting with users using motion-based events. However, it is not limited to the applications only supporting motion-based events and can be applied on applications covering both types of inputs (motion-based and non-motion based). In

such applications, users normally interact with these types of applications by performing a sequence of gestures, e.g. by moving a flying or bouncing object on the screen or drawing geometrical shapes by touching the screen. In other words, user-generated gestures are transferred to the object or touched location to move the object toward the desired direction or to draw a geometrical shape (e.g. circle) around the touched point on the screen. It is noteworthy that motion-based events are not only used to move an object on the screen; sometimes, shaking a mobile phone in a specific direction or touching and dragging the screen leads to executing a function or opening another application [29]. This study focuses on the procedure to automatically generate test motions on both types of applications: (1) applications only supporting the data generated using accelerometer sensors; and (2) applications supporting both the data generated using the accelerometer sensors and the data generated using other types of events such as those produced by touching the sensitive screen. In such cases, several parameters can affect a single event (such as the object size, the size of the screen, an object's location, etc.).

Since users are free to touch, move and shake their mobile phones in any desirable direction and speed, a testing approach must be able to generate sets of standard gestures, which are not only executable on the application but also resemble the human-generated motions. Therefore, to automatically generate more reasonable gestures – mimicking human users – this research proposes a novel approach. It is hypothesized that this mimicking may also result in an increased level of code coverage of the SUT. The correctness of this assumption is examined in the empirical evaluation section (Section 6).

The proposed technique contains several steps and details, which are depicted in the framework provided in Figure 1. This figure shows the schematic overview of our proposed approach for a complex application containing acceleration parameters moving an object (bouncing ball) on the screen in different directions (as an example). This framework can easily be adjusted for any applications supporting motion-based events. The proposed approach consists of the following sequential steps:

1. Gathering training data: A human user is asked to interact with the application and generate motions to be used as a training set. (It is worth noting that the person is not trained or instructed to generate any specific types of motions from the applications and the generated motions are the result of a volunteer interacting with the application for the first time. This prevents the data-gathering phase from collecting biased data.)
2. Clustering motions: k-means clustering algorithm as a classic clustering technique is used to identify the relationship between data points (motions) generated by human users, and to determine the cluster of behaviours that they belong to. It is well known that data clustering is a successful approach in recognizing and categorizing human expressions, gestures and actions [10], [30], [31]. More specifically, in this study, the motion parameters are partitioned into k clusters, such that each motion is allocated to the cluster with the nearest mean. The clustered data later will be used to train an initial model of the gestures.
3. Training Initial HMM: In order to produce the first standard test gesture, an initial HMM is trained using the human-generated motions and their corresponding clusters. Hidden Markov models are well known for their application in pattern recognition such as speech, handwriting and gesture recognition. As we utilize time-varying motion sequences, HMMs can be used to model and learn human skills such as reasonable interactions with mobile applications [32], [33]. Basically, the initial HMM trains a model, where its hidden states indicate motions' clusters, generated in the first step. Training the model using the expectation-maximization (EM) algorithm, the probability of a gesture belonging to a specific cluster (state) is estimated and used to calculate the first motion acceleration parameters. The first motion's acceleration is calculated by computing the mean of the accelerations in each HMM state and by selecting one pair randomly. Using this approach, we can assure that the whole test generation procedure - including the initial motion - is produced through the models trained from the user-generated data, so they potentially mimic human generated gestures.
4. Generating the test data using HMM classifiers: In this step, we apply HMM classifiers on clustered data to generate test motions using the previously produced gestures. HMM classifiers are successfully used in several studies, considering the prediction of human activities and gestures [34]–[38]. For each cluster, the dynamics of each motion class is learned with one HMM. Thus, having m motion-clusters, m HMM classifiers need to be applied. HMM classifiers classify each motion as a function of a future time frame [39]. Thus, the probability of a test case belonging to each cluster is calculated using the well-known Forward algorithm [19]. The motion-cluster with highest Forward probability is selected and the mean of the acceleration of the motions belong to this cluster is considered as the next motion's acceleration.
5. Adding generated motions to the training set: In order to avoid over-fitting the model, the generated motions should be added to the training set. This helps the model to learn from the data rather than memorizing the trend. It is worth noting that adding more data to the training sets in the association with the cross-validation approach decrease the chance of overfitting. Additionally, since all motions which

are added to the training set are not derived from the statistical model (random motions, motions generated from physics equations,…) the model will not enforce the already modeled behaviors.

6. Storing and Executing test cases: Once, for example, the ball hits the vertical wall (or a terminal condition happens), sets of test motions generated since the last hit are stored as test cases and will be used to generate real motions in the mobile applications. Terminal conditions can be defined generally or per application. For example, a general terminal condition can happen once a specific number of test motions are generated, while a customized terminal condition happens when a flying object (if applicable) hits an edge. In this study, we considered the customized option for the stopping criteria.

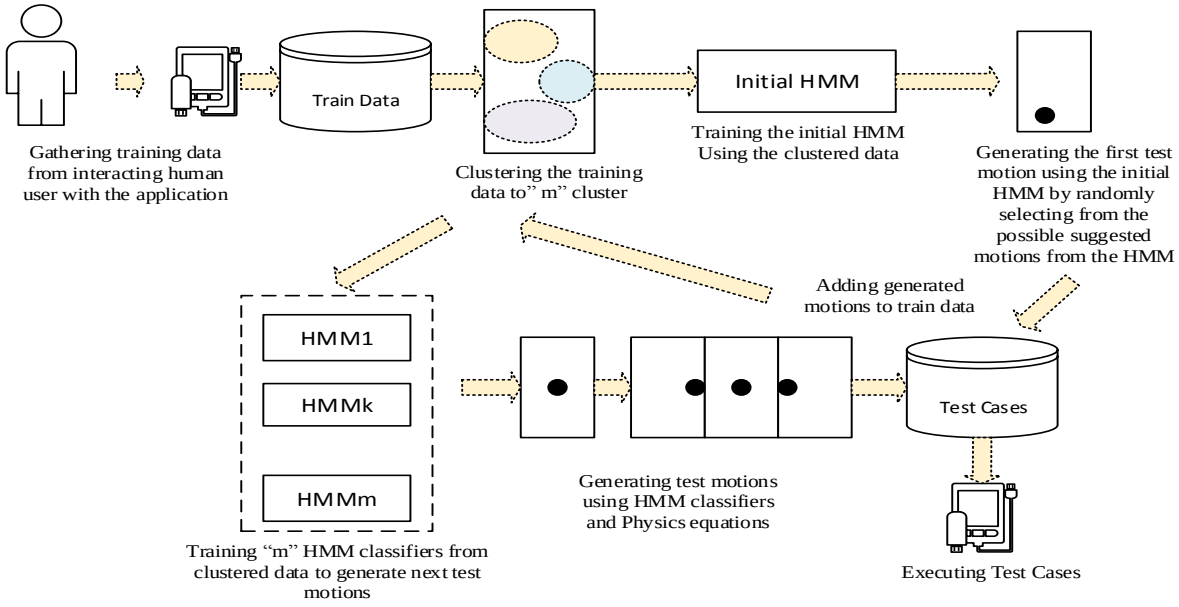

Figure. 1 1. An overview of applying the proposed approach on the application with flying object. It consists of both training the initial HMM (top) and test generation process using HMM classifiers (bottom)

It is worth noting that this framework provides an overview of the proposed approach. The implementation details of this framework are discussed in Section 4.

3.1 Synthesizing motion sequences

This section describes the method of instantiating the motion sequences for complicated motion-based applications, which transfer the users' gestures to a bouncing object. However, the application of this approach is not limited to events using sensor-generated data; it can be easily used to generate automated test cases for any type of motion-based events. Following the previous section, two sets of data (motion sequences) are considered in this study:

- The training data, which is captured during a real user's interaction with the application and is used to train the initial HMMs.
- The second set is the test data, which is generated by using the test generation algorithm and is presented to the application being tested to evaluate its functionality. To create meaningful test data, which is recognizable by the trained HMM and its corresponding classifier, we describe a single motion k by a 6-tuple $(lx_k, ly_k, vx_k, vy_k, ax_k, ay_k)$, where lx_k, ly_k indicates the object's location, vx_k, vy_k determine the velocity, and ax_k, ay_k describe the acceleration of the motion in 2D space at a specific time interval. Figure 2a shows the 3D acceleration axes on a smartphone, which also contains a z-axis. In order to simplify the explanation of the algorithm and cover more common applications, only 2D applications have been considered in this study. However, it is worth noting that it is possible to apply the same algorithm in 3D versions as well.
  An example of a single motion in a bouncing ball application is provided below:
  05-07 17:36:15.828: Vx(32065): -2.7148619
  05-07 17:36:15.828: Vy(32065): -2.7148619
  05-07 17:36:15.828: lBallX(32065): 549.0
  05-07 17:36:15.828: lBallY(32065): 20.0
  05-07 17:36:15.828: Ax(32065): 0.090979666
  05-07 17:36:15.828: Ay(32065): -0.12330139
  This can be presented in a 6-tuple format (the data is rounded for the sake of clarity): (549,20, -2.714, -2.714, 0.09, -0.123).

This study also considers two time intervals during the test generation procedure:

- The first time interval constantly happens every φ milliseconds [7] to capture the information regarding the current motion and position of the object on the screen and to calculate the next motion using the well-known SUVAT equations [29], [41].
- The second time interval happens every θ milliseconds, which is estimated by selecting the minimum possible time between two gestures, generated by human users. (This time can vary with the complexity of the gestures in different applications). Hence, the estimation of θ assists the algorithm to generate more realistic (complex) gestures as it accounts for the limitations of kinematics.
- It is worth noting that these time intervals can overlap in the sense that in the time window between two θ intervals, the φ interval may happen when $\theta > \varphi$.

Figure 2b shows an atomic gesture consisting of a sequence of motions happening within these two intervals. Each sequence of motions is terminated by the occurrence of a specific condition in the application being tested, depending on the application's objectives and functionalities. For example, a simple terminal condition can happen when the flying object hits another object (such as the edges of the screen or another flying object) on the screen.

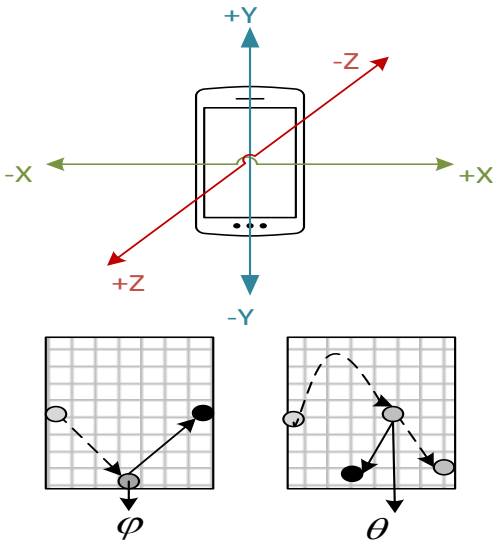

Figure. 2. (a) 3D acceleration axes on smartphones; and (b) an atomic gesture containing a sequence of motions happening within two intervals: (left) a bouncing object keeps moving in the screen after hitting the edge in first time-interval φ; (right) the proposed approach calculates the next movement after the second time-interval θ happens

Additionally, in the following paragraph, some of the SUVAT equations (equation of motions), which are useful in calculating the coordinates of the motions are provided:

- $v = at + v_0$
- $l = l_0 + v_0 t + \frac{1}{2} a t^2$
- $l = l_0 + \frac{1}{2}(v_0 + v)t$
- $l = l_0 + vt - \frac{1}{2} a t^2$
- $v^2 = v_0^2 + 2a(l - l_0)$

where

- l_0 is the object's initial position
- l is the object's final position
- v_0 is the object's initial velocity
- v is the object's final velocity
- a is the object's acceleration
- t is the time

Definition1: A test case (TC) consists of a set of motions ($M = \{m_1, \dots, m_n\}$),where $m_{k \le n}$ is a 6-tuple ($lx_k, ly_k, vx_k, vy_k, ax_k, ay_k$). The number of tuples (motions) in each TC depends on the number of detectable motions before the termination situation happens.

4. HMM- BASED TEST CASE GENERATION

Human activity recognition and classification has been studied using several different machine learning approaches such as multi-class support vector machines (SVM) [42], k-Nearest Neighbor (k-NN) [43], Neural Networks (NN) [5] and HMM-based approaches, but in cases that the activity sequences are time-varying, HMM based approaches have produced better performance and results [32], [44], [45]. In addition, the Markovian process had been used in several motion detection-related studies to create statistical models from clustered data [10]. Following these studies, we also cluster our training data by using the k-means algorithm [21] to identify the data points (motions) containing related gestures and to assign them to the same clusters (the number of clusters (k) is selected by using the silhouette score [22]).

In other words, the clustering algorithm is applied to groups of motions with similar behaviour and allocates them into a single cluster. These clusters will be used as the class labels for the HMM classifiers. This means that each class indicates a set of similar motions in the corresponding cluster. Therefore, a motion, which belongs to a class during the classification process, also shares similar characteristics with the motions in their corresponding cluster. It is also worth noting that since the motions' clusters, detected by the clustering algorithm, play the role of class labels in the proposed HMM classification procedure; we use the term of class label instead of cluster to avoid unwanted ambiguities.

Consequently, the clustered data will be used to train an initial Hidden Markov Model. Since an HMM is a Markovian process that contains two sets of states (the observable and the hidden [latent] states), only the motion sequences, depending on the latent states (motions' clusters or classes), are visible in such a model. Therefore, as the classes are invisible from an *observer's view*, only the motions in this model are completely observable, an HMM can create a more powerful model compared to regular Markov models or partially observable Markov decision processes (POMDP) [26]. The HMM in this study is characterized by the following elements [19]:

- a set of latent states $S = \{s_1, s_2, \dots, s_L\}$, which are hidden from the external observer and indicates the class of motion sequences;
- a set of observable states $V = \{v_1, v_2, \dots, v_N\}$, where each is mapped to a corresponding motion sequence (m_k);
- a transition probability $[A]_{ij} = \{a_{ij}\}$,
  $a_{ij} = P(Q_{t+1} = s_j | Q_t = s_j), 1 \le \text{i,j} \le \text{L}$, which determines the transition probability between different classes. For the initial modelling process, because human users generate the motions, the initial transition probabilities between different classes of motions can be extracted directly from the training data;
- an emission probability $[B]_{jk} = \{b_{j\,(vk)}\}$,
  $b_j\,(v_k) = P(M_t = v_k | Q_t = s_j), 1 \le \text{j} \le \text{L}, 1 \le \text{k} \le \text{N}$, which indicates the probability of a motion sequence belonging to a specific class (estimated by frequency counting on the clustered training corpus); and
- initial state distribution, $\Pi = \{\pi_i\}$,
  $\pi_i = P(Q_1 = s_i), 1 \le \text{i} \le \text{L}$. Each and every state can be an initial state in this study.

Using the values of A, B, and Π, an HMM can be used as a generator to create an observation sequence (where T is the number of motions in the test case): $M = \{M_1, M_2, M_3, \dots, M_T\}$. We use the notation $\Lambda = (\text{A, B, } \Pi)$ to simply indicate the complete parameter set of the HMM with respect to the Markovian process, which illustrates that the probability of a motion's occurrence only depends on the previous motion:

$$P(s_{t+1} | s_t, s_{t-1}, s_{t-2}, \dots) = P(s_{t+1} | s_t)$$

This initial HMM model is used as an input to an expectation-maximization (EM) algorithm; specifically, we utilize the Baum-Welch algorithm in this study [27]. This algorithm estimates the optimal model with the highest likelihood of the estimated parameters. In algorithm 1, this procedure is done by running the HMM function in the second line. Then, the initialAccel function initializes, the acceleration parameters of the first test motion by calculating the mean of the acceleration pairs (i.e. (ax, ay) in 2D space) in each HMM state and by selecting one pair randomly. Then, in lines three and four of this algorithm, the CreateMotion function is generating a motion sequence using the SUVAT equations and the Update function is storing the newly created motion sequence as the current motion. After generating the initial motion, the CreateMotion and Update functions are called again but this time within the time interval φ, until a termination condition happens (line 5-10). This procedure generates a simple gesture based upon the previous motion, using appropriate physics equations. In order to generate more

realistic and complicated gestures, we propose using the HMM classifier to detect the sequence class label at each interval θ [18], [28], [29].

The HMMClassifier function in line 12 of the algorithm classifies the current motion sequence into an appropriate class of gestures. This function combines a set of sequences of motions and a list of class labels to train one HMM per class label (where L is the number of class labels). Subsequently, the trained models are used to calculate the forward probability of a motion sequence M per model $\Lambda_{i\leq L}$ ($P(M|\Lambda_i)$). $P(M|\Lambda_i)$ is calculated using the Forward algorithm, which is internally called during the execution of the HMMClassifier function. The forward algorithm computes the forward probability, $\alpha_k(t)$, as the joint probability of observing the first t vectors $m_t, T = 1, \dots, t$ while in state k at time t. Another way to state this would be that $\alpha_k(t) = P(m_1, m_2, \dots, m_t, s_t = k|\Lambda)$ which is the probability of observing $(m_1, m_2, \dots, m_t)$, assuming that the system is in state k at time t. Given a list of forward probabilities for a motion sequence M, we are able to easily detect a model with the maximum probability and assign its corresponding class label as the motion's class label [19]. Determining the class label of a motion sequence allows us to easily detect the motion sequences belonging to the same class from the training data set, and estimate the next motions values by calculating the mean of the accelerations of the motions (the Accel function in line 12). Moreover, the generated motion is added to the training set to avoid over-fitting. This helps the model to learn from the data rather than memorizing the trend (lines 10 and 16). It is worth noting that in this study, we also use the term of "occurrence likelihood" to refer to the forward probability.

Putting it all together, lines five to seventeen of Algorithm 1 create a set of motion sequences within two different intervals. Simple gestures are generated based on physics equations once the first time-interval happens. But, the more complicated motions (e.g. gestures with variable accelerations) that may require a longer time period to be created by a human user are generated within the second time interval. This process provides sufficient duration to allow the method to generate more complex gestures. An example of a simple motion is the one calculated by the SUVAT equations after the bouncing ball hitting the horizontal wall. While the complex one is a motion calculated by HMM classifiers for a ball slowly bouncing in the middle of the screen. In reality, when the ball is slowly moving in the screen, the human user can change the direction of movement by shaking the phone in several different directions. Therefore, in an automated test generation process, a trained model is needed to predict the most probable acceleration of the gesture from the last motion's parameters. In this study, the motion generation is stopped and a test case is generated once a terminal condition (e.g. hitting the vertical wall) happens.

When the application under test is motion-based application with no acceleration parameter involved, lines 18-30 of this algorithm will be applicable. In such situations, the first motion can be created by randomly selecting a touched-point in the screen. In a 2D space, the motion sequence only contains the coordinates of the touched-point(x, y). Similar to Algorithm1, within different intervals φ and θ, random touch points are generated, or the HMMClassifier function predicts the next motion class-label and the Position function returns the position of a touched point by calculating the mean of the position pairs in the predicted class. Depending on the application design's objectives, the position of a touched-point can be used to draw a shape or render functionality such as vibrating the phone or opening a dialogue box. In this algorithm, in order to focus more on the second case study, we consider creating a geometrical shape (e.g. circle) with the touched-point coordinates as its center (we call it MakeAnAction function). Additionally, when an application covers non-gyroscopic events such as clicking a button or choosing an item from the menu, the RandomEvent function randomly generates an event, executable within the current state of the application.

Algorithm. 1 Test case generation procedure for cases with/without acceleration involved

Input: *Initial position of the object (x,y), training data set (S), set of class labels (C);* $i = 2$*;*
Output: *Test case (TC)*

1. ***if*** $(Accel\_MotionEvent)${
2. (ax,ay)⟵ initialAccel(*HMM(S,C)*)
3. m_1 ⟵CreateMotion(ax,ay,x,y)
4. Update(ax,ay,x,y)
5. **While** *(!terminalCondition)*
6. ***if*** $(curTime - lastUpdate1) \geq \varphi)$
7. $i \leftarrow i + 1$
8. m_i ⟵CreateMotion(ax,ay,x,y)
9. Update(ax,ay,x,y)
10. $S \leftarrow S \cup \{m_i\}$
11. ***if*** $(curTime - lastUpdate2) \geq \theta)$
12. (ax,ay)⟵Accel(*HMMClassifier(*m_i*,S,C)*)
13. $i \leftarrow i + 1$
14. $m_i \leftarrow$ CreateMotion(ax,ay,x,y)
15. Update(ax,ay,x,y)
16. $S \leftarrow S \cup \{m_i\}$

```
End while
if (NonAccel_MotionEvent){
    While (!terminalCondition)
     If (curTime – lastUpdate1) ≥ φ)
         i ← i + 1
         m_i ←RandomPosition (x,y)
         MakeAnAction(x,y)      (e.g Create a shape)
     If (curTime – lastUpdate2) ≥ θ)
         i ← i + 1
         m_i ←Position (HMMClassifier(m_{i−1},S,C))
         S ← S ∪ {m_i}
         MakeAnAction(x,y)
    End while
}
else{
    t_j ←RandomEvent();
    j ← j + 1
}
Return TC ← {m_1, … , m_i} + {t_1, … , t_j}
```

**lastUpdate1 indicates the last update that happened at interval φ while lastUpdate2 indicates the last update that happened at interval θ*

5. RUNNING EXAMPLE

In order to clarify the proposed test case generation procedure, we considered a very small portion of the training data generated by a human user in the bouncing ball application. In this running example, we follow the test generation framework (Figure 1) provided in Section 3 step by step to generate test cases:

1. Gathering training data: 30 motions in the format of 6-tuple $(lx_k, ly_k, vx_k, vy_k, ax_k, ay_k)$ are gathered as the result of user interaction with the application.
2. Clustering motions: the training data is clustered into 2 distinct clusters (classes) using the k-means algorithm. Due the space limitations a partial view of the clusters are provided in Table I.

Table. 1 Simplest supported actions and gestures in both types of applications

Cluster 1	***Cluster 2***
(211.362, 502, 9.787, 9.787, -0.306, 7.948)	(20.0, 344.450, 24.511, 24.511, -4.563, -3.260)
(220.376, 502, 9.259, 9.259, 0.550, 7.753)	(20.0, 45.517, 8.898, 8.898, -6.545, 6.408)
(229.642, 502, 8.681, 8.681, -0.550, 7.753)	(26.236, 20.0, 3.899, 3.899, 11.504, 5.465)
(245.160, 502, 7.443, 7.443, -0.835, 7.907)	(20.0, 182.771, 23.407, 23.407, 8.195, -7.834)
(252.285, 502, 6.566, 6.566, -0.835, 7.907)	(20.0, 235.211, 19.966, 19.966, -8.742, 0.182)
(270.067, 502, 2.694, 2.694, -1.039, 7.953)	(300.0, 118.057, -28.868, -28.868, -8.030, -4.404)
(272.012, 502, 1.572, 1.572, -1.067, 7.899)	(300.0, 367.611, -36.233, -36.233, 8.330, 1.120)
(271.131, 502, -1.667, -1.667, -1.170, 7.818)	(20.0, 378.444, 28.222, 28.222, -2.802, -0.751)
…	…

3. Initial HMM training: the clustered data is then used to train the initial HMM using Baum Welch algorithm. In this case, the HMM model contains 30 observable states and 2 hidden states (since there are only two clusters). Then, the acceleration parameter of the first test data motion is generated by calculating the mean of the acceleration pairs of the motions belonging to each hidden state of the initial HMM and subsequently selecting one pair randomly. After determining the initial acceleration parameter, the first motion is created using this parameter and the SUVAT equations. In this case, given:

   the (1) initial acceleration parameter:

   $$(ax, ay) = (0.598, -0.915)$$

   (2) the initial location of the ball in the screen:

   $$(lx_0, ly_0) = (309{,}253)$$

   and (3) knowing that the initial velocity is equal to zero (ball is not moving at the beginning):

   $$(vx_0, vy_0) = (0{,}0)$$

   motion $m_1(309.080{,}252.876{,}0.175, -0.274{,}0.060, -0.0916)$ is generated using physics equations: $v = at + v_0$ and $l = l_0 + v_0 t + \frac{1}{2}at^2$.

Then within the time interval $\varphi = 300ms$ other motions are also generated through the same process with the difference that the acceleration of the current motion is used as the initial acceleration for the next ones. These motions will be added to the training set to avoid over-fitting. (Figure 3 depicts a schema of the trained initial HMM).

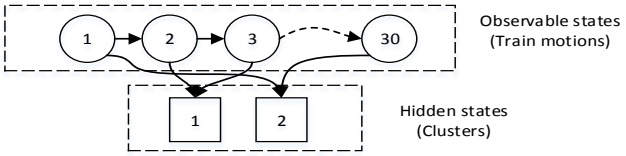

Figure. 3 3. An overview of trained HMM in running example

4. Test data generation using HMM classifiers: Now, in order to generate more complex motions (within time interval $\theta = 500\ ms$), covering unexpected human-generated gestures, two (number of classes) HMM classifiers are trained and the forward probability of the current motion is calculated to reveal the class of motions it belongs to. Then, the mean of the accelerations of the motions belonging to this class is calculated; and again, are used as input of the motion equations to calculate the velocity and location parameters. For example, if the occurrence likelihood (forward probability) of the current motion $m_i(20, 492.070, 2.162, 2.162, -0.007, 0.246)$ in class c_2 reaches the maximum amount compared to the other class (c_1), the mean of the acceleration of the motions in class c_2 is calculated and will be used as the new current motion's acceleration. In this case, the mean of the accelerations in $c_2$is equal to (0.347,1.116). Therefor, using physics equations, the next motion would be:
$$m_{i+1}(21.124, 493.290, 2.336, 2.720, 0.347, 1.116),$$
This motion also will be added to the training set.

Once, the ball hits the vertical wall, the motions generated since the last hit, are saved in the form of a test case and will be executed to move the ball toward different directions on the screen.

6. EMPIRICAL EVALUATION

To study the proposed approach, we performed an experiment on four case studies from three different mobile applications that could detect and execute motion-based gestures. Unfortunately, finding case studies in this area is far from straightforward. Firstly, the volume of open-source games is limited and many of them are ports of existing games from traditional platforms. For example, Wikipedia, AOpenSource.com, F-Droid, and Prism-break, provide lists of notable open-source applications for the Android platform. However, upon review, the reader soon discovers that nearly all of the applications are ports of desktop or laptop applications. Hence, none of the applications feature user-interaction via gyroscopic input devices. This obviously renders these applications useless as case studies.

Additionally, it is worth noting that even though our approach is able to generate test cases for mobile applications covering both gyroscopic and non-gyroscopic events, the focus of the study is on providing a practical approach for generating motion-based events. This means that the portion of our algorithm, which is producing the test cases for non-gyroscopic events, can be easily replaced with other effective GUI-based test case generation techniques such as [20], [21],

This allows us to comprehensively examine these applications and produce a set of unbiased results from experimenting with these applications.

In the evaluation section, we attempt to answer the following research questions:

1. Can the test-generated motions mimic actual user behaviour?
2. Does the proposed method improve the code coverage of the SUT when compared to existing automated techniques (random testing)?
3. (a) How does the proposed approach compare with random algorithms in terms of the computational complexity? (b) Can random algorithms produce better test cases (in terms of the code coverage) than our proposed approach, if the same volume of computational resources, as given to the HMM-based approach, is assigned to them? The answers to these questions are provided in the separate section (Section 7: Run-time analysis)

6.1 Case study 1: bouncing ball

The first case study is an Android application, a bouncing ball application, which is designed to record a data set of coordinates from shake and tilt gestures performed by human users *(LOC=716).* This application only

contains one flying object (round ball), which bounces on the screen; the ball moves by processing the information it captures from a phone's accelerometer.

The dynamics of a bouncing ball follows a set of well-studied physics laws and equations [30], which are used in this study. Since covering the details of such equations is beyond the scope of this research, we only discuss some of the case-specific motions and equations:

- When the application starts, the ball position is stable in a corner of the screen, waiting for a motivation. Depending on the power of the first motion, the ball starts moving toward the motion's direction. In this study, the time interval φ is fixed at 300 milliseconds to capture the information regarding the current motion and position of the ball on the screen and to calculate its next position. The time interval is set to 300 milliseconds to follow the approaches proposed in [9], [10]. In other words: $300 < the\ time\ periods\ between\ user-created\ motions$
- The second time interval θ is equal to 500 milliseconds in this study because the time windows between gestures created by users vary from 500 milliseconds to one second, we select the lower bound to create more standard motions.
- While the ball is moving on the screen, the motion data is re-ordered in the 6-tuple format, used to express test motions (Section 3.1). Each sequence is terminated whenever the ball hits the vertical edges of the screen.

Table 2 (First two columns) indicates the simplest possible actions that can be performed through this application, along with their corresponding gestures. It is noteworthy that in designing this table, it is assumed that the ball has enough space to move toward each direction. Obviously, it cannot for example move to the left when it has already hit the right-side edge. Any combinations of these actions (e.g. curving), which may be produced by rotating, tilting the device, and so on is also considered in this case study. For example, when the user rotates or tilts the mobile phone toward the right, the ball can move in a curve instead of moving in a straight line to the right.

Table. 2 Simplest supported actions and gestures in both types of applications

Bouncing Ball / Extended Bouncing Ball		***Bubbles***		***Diamond***	
Action	***Gesture***	***Action***	***Gesture***	***Action***	***Gesture***
Tilt the device toward left	The ball bounces to the left side of the screen	Touch/Push the screen.	The circle is drawn around the touched-point	Tilt the device toward left	The object bounces to the left side of the screen
Tilt the device toward right	The ball bounces to the right side of the screen			Tilt the device toward right	The object bounces to the right side of the screen
Tilt the device to the front	The ball bounces down.			Tilt the device to the front	The object bounces down
Tilt the device to the back	The ball bounces up			Touch/Push the buttons	The action recorded in the button will be triggered

6.2 Case study 2: bubbles

The second case study is another android application called Bubbles, which is able to draw circles around the touched points on the screen (LOC=423). In order to generate circles (bubbles), the user touches or pushes the screen resulting in a circle being gradually grown from the touched point. The maximum length of the circle's radius is predefined and fixed, so the circle keeps growing until it's radius is equal to the maximum number or the user touches another point in the screen. Table II (Second two columns) shows the action (motion event) and its corresponding gesture. In this case, the motions containing the coordinates of the touched points are captured within the same time intervals as the first case study to generate a set of motions. The sequences of motions are continuously generated until a border is touched. Then, the generated set is considered as a test case.

6.3 Case study 3: extended bouncing ball

We also modified the Bouncing ball application by adding one more flying object in the screen. The second ball behaves the same as the first one (Table II – First two columns), except for the difference that its initial location in the screen is in the bottom right-hand corner (the original ball is located in the left side), thus depending to the amount of acceleration received from the sensors, they can move in diverse directions. The same test

generation algorithm is used to produce test cases for the extended Bouncing ball application (LOC= 1054) as the simple version and test motions are stored in two separate sets of test suites for each ball.

6.4 Case study 4: diamond

In order to evaluate the performance of our proposed test case generation approach in a more complex framework we also applied our technique to generate test cases for another real-world mobile game. This game. This game is called Diamond (LOC= 4311); it is a classic game implemented in a modern way using accelerometer sensors. In this application, the user controls an object in the screen and tries to collect as many diamonds as possible by moving the mobile phone toward the correct direction. The player also has to avoid hitting enemy objects and reach to the end point safely. The moving object follows the common behaviour of a bouncing object (Table II – third two columns) and the terminal condition happens when the game is over (the game is over, when the player hits an enemy object or reaches the end point). Moreover, in order to enter the game, change the settings or quit the game, the player needs to select items from the menu by pushing some buttons. Therefore, the application is able to handle more than one input type (both gyroscopic and non-gyroscopic data).

6.5 Comparison criteria

In order to address the first research question and analyze the performance differences between the proposed approach and other test case generation methods, Non-parametric Statistical Hypothesis Tests and Effect Size (cliff's delta) Measures are applied:

Non-parametric Statistical Hypothesis Test: In this case, we established a null hypothesis and an alternative hypothesis to be evaluated. The null hypothesis (H0) states the two test case generation techniques provide the same performance in covering the source code. On the other hand, the alternative hypothesis (H1) states that if the difference between the medians of the coverage percentages, is not zero then they will be considered as different. Therefore, by considering a significance level $\alpha = 0.05$, we would be able to reject null hypothesis if $p - value < 0.05$ for each independent situation.

Effect Size: In order to add a "magnitude of a treatment effect" to our comparison criteria, Cliff's Delta measure is calculated. Cliff's Delta statistic [53] is a nonparametric effect size measure that quantifies the difference between two groups of observations by testing the equivalence of probabilities of scores. In this study, the magnitude of differences between test generation techniques is assessed using the following thresholds: $|d| < 0.147$ "negligible", $|d| < 0.33$ "small", $|d| < 0.474$ "medium", otherwise "large" [54]. In addition, Cliff's Delta is a bounded measure [-1, +1] where the limiting values indicate that the two populations have no overlap.

6.6 Experimental results

To answer the research questions and evaluate the efficiency of the proposed test generation approach, a volunteer interacted with all the applications and produced motion sequences which are then used as training sets. For instance, in the Bouncing ball application, a set of training data was obtained by recording the motion coordinates for three minutes from a total of 317 gestures performed on two different Android devices. Applying the silhouette score, we grouped the motions into 95 clusters. For the extended version of this application, 600 motions and 105 clusters were considered. This data is recorded in 6 minutes. For the Bubble application, these numbers were 481 and 95 respectively (motions are stored for 2 minutes). Training motions are also stored in the 6-tuple format used to express test motions (Section 3.1). The amount of time allocated to each training process is estimated based upon the time a new user needs to become visually familiar with the application and to generate a set of motions. In this study, this time is estimated by calculating the mean of the time that new users require to generate a reasonable set of motions for the considered applications.

To evaluate the quality of the generated test cases in all case studies, 20 sets of 200 motion sequences were generated using the proposed technique. In addition, for the Bouncing ball application, the same number of motion sequences (20 sets of 200 motions) was created by two random test generator procedures:

- Physics-based: takes a human-user motion to initialize the acceleration or position parameters then creates the next motions based on the current one by randomly selecting a physics equation (Algorithm 2).
- Simple Random Algorithm: Creates test cases by simply generating random motion sequences within the data ranges supported by the hardware. The well-known Mersenne Twister (MT) approach, a pseudo random number generator (PRNG) is used in this study, which generates random numbers based on Mersenne prime $2^{19937} - 1$, using a 32-bit word length [55]. In this study, a human user also generates the initial motion. Since, the HMM-based technique is using human-data to train the initial model and generate the first motion, the simple random test case generation process also get initialized by human-generated data.

- Hybrid approach: In order to conduct a fair comparison between approaches, some experiments have been designed to execute combinations of human and randomly generated test cases (e.g. "Human + Simple random" and "Human + Physics-based"). This means that using human data is not limited to the initialization phase and user-generated data forms half of the test cases. Therefore, a hybrid test case consists of a combination of human generated motions and random motions.
- Random-Human: we also created another random-based approach by randomly selecting motion events from the training data. In this approach, we generated test cases by picking random motions from the human generated training set.
- Monkey [13]: In order to compare the performance of proposed approach with other well-known existing tools. We also generated test uses for the considered case studies using the Monkey tool. This tool is able to send a pseudo-random stream of user events (such as clicks, touches, or gestures, as well as a number of system-level events) to the system. Such streams act as a set of test cases for the application under test. We used the vanilla Monkey from the Android distribution for our experimentation. The monkey tool was configured to ignore any crash, system timeout or security exceptions during the experiment and to continue till the experiment timeout was reached. In addition, we configured it to have the same wait period between actions, as this same delay value was also used in other tools. Configuring Monkey for our infrastructure required no extra effort. So we did nothing extra and no specialized or unique information exists for this approach. Basically, we just maximize the performance of Monkey.
- Sapienz [24]: Another well-known testing approach for Android mobile applications is called Sapienz. This technique uses multi-objective search-based testing to automatically generate test cases. In another words, Sapienz combines random fuzzing, systematic and search-based exploration, exploiting seeding and multi-level instrumentation together to generate automated tests cases for Android applications. In this study, we applied the white-box manner, which uses fine-grained instrumentation at the statement level. Since the original version of Sapienz is using a time variable, we manually stopped the test generation after the required number of events are gathered; basically, we just maximize the performance of Sapienz.

Algorithm. 2 Random test case generation procedure for cases with acceleration involved (Physics-based)

Input: *Initial position of the bouncing object (x,y);* $i = 2$*;*
Output: *Random Test case (TC)*

1. (ax,ay)⟵ getHumanMotion()
2. **While** *(!terminalCondition*)*
3. e ⟵Select RandomEquation()
4. m_i ⟵CreateMotion(ax,ay,e)
5. $i \leftarrow i + 1$
6. **End while**
7. **Return** $TC \leftarrow \{m_1, \dots, m_i\}$

**In this case we terminated the process after generating 200 test cases*

It is worth noting that in cases (e.g. the second case study), where the acceleration parameter is playing a significant role in defining a motion, the generated accelerations in random test cases are limited to the acceleration range supported by the hardware. In addition, since the acceleration parameter and its corresponding physics equations are not considered in the second case study, only the simple random algorithm is implemented to generate the random touched-points.

To answer the first research question, we classified two sets of test cases (derived from Bouncing ball and Bubbles applications) by using the HMM classifier into 95 classes which are defined based upon the data generated by the human users. The same procedure is applied on the test data generated for the extended Bouncing ball and Diamond applications and they are classified into 105 clusters. Then the occurrence likelihood (LC) of each sequence of motions for each class label are calculated where $\{LC = P(M|\Lambda_i), \Lambda_{i\leq L} and\ M \in TC\}$, where L is the number of classes. In this case, when $\max_{L} P(M|\Lambda_i)$ is a small quantity, it can be concluded that the test case TC is not behaving similar to the test cases that were used to create the classes. Additionally, since these classes are created using human-generated motions, it can be implied that the probability of the test case TC being generated by a human user is low.

The results show that the motions generated using the HMM-related technique have a higher forward probability (occurrence likelihood) compared to both Simple Random and Physics-based approaches. Accordingly, it can be concluded that the test cases generated using the proposed technique are more likely to be generated by a human user. The reason is that each class label describes a set of human-generated motions; therefore, once a motion has high occurrence likelihood in one of these classes, it can be concluded that the probability of being generated by a human user for this motion is high.

Table. 3 Results of calculating effect size measure and the mean of code coverage for test case generation methods in all case studies

	Approach	*Mean of Code Coverage (%)*	*Approach*	*Delta Estimate*	*p-value*
Bouncing ball	HMM-based	79.26	HMM-based Vs. Physics-based	-0.965	4E-05
	Physics-based	55.95	HMM-based Vs. Simple Random	-1	4E-05
	Simple Random	33.05	HMM-based Vs. Human + Physics-based	-0.7357	0.00019
	Human + Physics-based	63.63	HMM-based Vs. Human + Simple Random	-0.6761	0.00034
	Human + Simple Random	62.73	HMM-based Vs. Human	-0.7225	0.00017
	Human	60.2	HMM-based Vs. Random-Human	-0.8575	7.6E-06
	Random-Human	59.05	HMM-based Vs. Monkey	-0.95	1.9E-06
	Monkey	37	HMM-based Vs. Sapienz	-0.95	1.9E-06
	Sapienz	41.15			
Extended Bouncing ball	HMM-based	81.3	HMM-based Vs. Physics-based	-0.95	1.9E-06
	Physics-based	52.75	HMM-based Vs. Simple Random	-0.95	1.9E-06
	Simple Random	31.77	HMM-based Vs. Human + Physics-based	-0.71	3.4E-05
	Human + Physics-based	62.78	HMM-based Vs. Human + Simple Random	-0.575	0.0028
	Human + Simple Random	62.98	HMM-based Vs. Human	-0.62	0.0002
	Human	62.05	HMM-based Vs. Random-Human	-0.87	1.3E-05
	Random-Human	58.7	HMM-based Vs. Monkey	-0.95	1.9E-06
	Monkey	37.75	HMM-based Vs. Sapienz	-0.95	1.9E-06
	Sapienz	42.3			
Bubbles	HMM-based	92.06	HMM-based Vs. Human + Simple Random	-0.9325	5E-05
	Simple Random	74.28	HMM-based Vs. Human	-0.9325	4E-05
	Human + Simple Random	79.97	HMM-based Vs. Simple Random	-1	4E-05
	Human	78.94	HMM-based Vs. Random-Human	-0.9425	1.9E-06
	Random-Human	76.8	HMM-based Vs. Monkey	-0.95	1.9E-06
	Monkey	74.85	HMM-based Vs. Sapienz	-0.985	1.9E-06
	Sapienz	75.65			
Diamond	HMM-based	77.76	HMM-based Vs. Simple Random	-0.95	1.9E-06
	Simple Random	40.82	HMM-based Vs. Human + Physics-based	-0.705	7.6E-06
	Human + Physics-based	63.02	HMM-based Vs. Human + Simple Random	-0.7025	0.0003
	Human + Simple Random	61.39	HMM-based Vs. Human	-0.5525	0.0022
	Human	63.85	HMM-based Vs. Random-Human	-0.575	0.0016
	Random-Human	62.99	HMM-based Vs. Monkey	-0.95	1.9E-06
	Monkey	42.25	HMM-based Vs. Sapienz	-0.95	1.9E-06
	Sapienz	50.91			

Figures 4a, 4b, 4c and 4d depict boxplots showing the distribution of the occurrence likelihoods of the motions produced by the HMM classifier model and the simple random approach for all considered applications. According to these figures, it also can be concluded that the generated random tests in the Bubble application are "behaving better" than the random tests in the rest of applications. The reason is that the gestures in the Diamond and Bouncing ball applications are more complex than the gestures in the Bubble application in terms of motion sequences. This makes it more difficult to generate gestures resembling human behaviour using the random approach in the Bouncing ball and Diamond applications when compared to the Bubble.

To address the second research question, the JaCoCo code coverage library was used. Using this toolkit, bytecode instrumentation is applied, and the branch coverage value is measured. Since we generated 20 sets of 200 test cases using each approach, the means of the coverage percentages on all sets, are calculated to achieve more accurate results (In total, 132000 motion sequences are generated during the experiments). Table III reports the means of the branch-coverage percentages calculated by running each of the test case generation approaches in all applications.

Figures 5a, 5b, 5c and 5d also show the distribution of the code coverage using box plots. According to these results, HMM-based test cases achieve better coverage compared to the random and human-generated test cases. In addition, the results of applying the Wilcoxon signed rank test indicates the HMM-based approach is significantly different from the other techniques in terms of code coverage. Table III also reports the p-values and delta estimates at the 95% confidence interval. The Cliff's Delta measure provides more detailed information to this picture by showing that a "large" effect size exists (in favour of HMM-based approach) for all of the comparisons.

The achieved results confirm that the HMM-based test case generation approach not only automates the test generation and execution procedure for motion-based events, but also (1) creates better test cases in terms of mimicking actual user gestures; and (2) improves the (branch) code coverage for the SUT.

It is also worth noting that the performance of our approach in terms of the code coverage is significantly better than both Sapeinz and Monkey approaches (e.g. for Diamond application: $Delta\ Estimate = -0.95$ and $p - value = 1.9E - 06$). In the next section, we compare our proposed approach with random generation in terms of the time complexity.

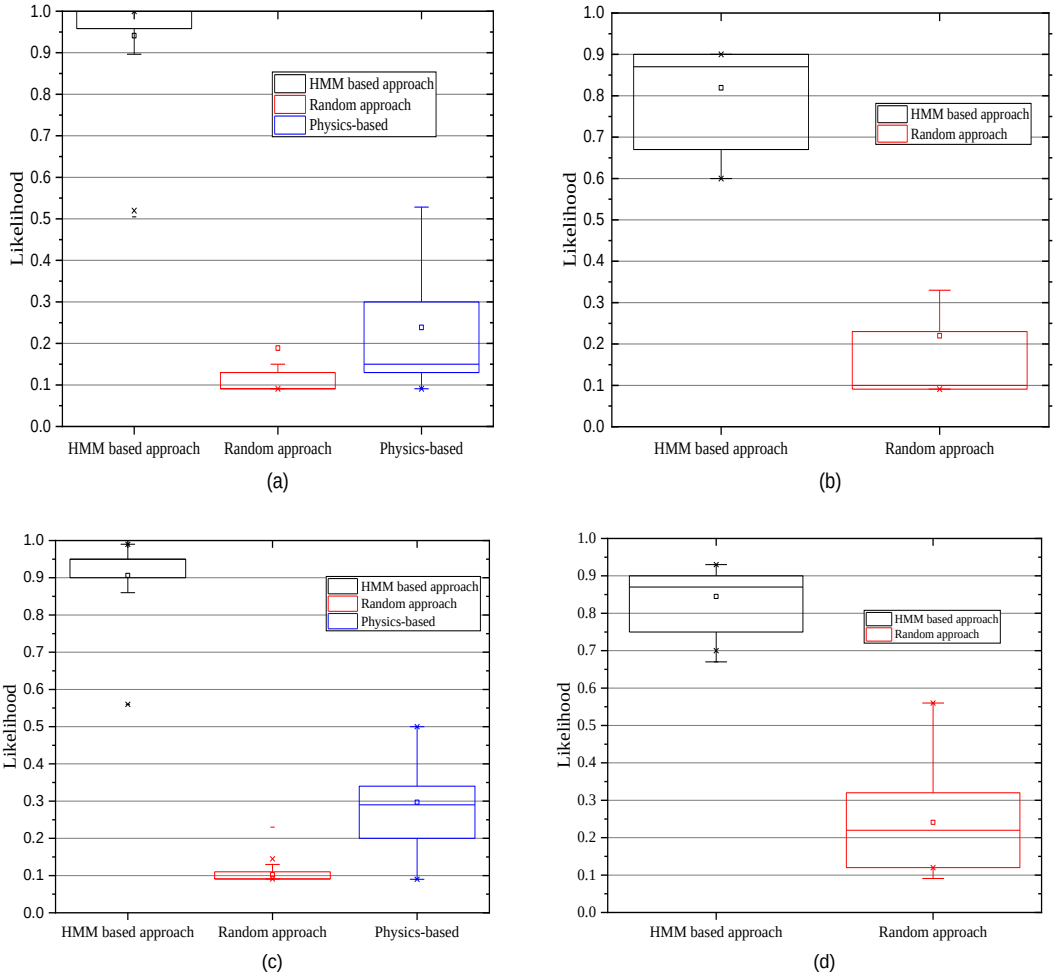

Figure. 4. (a) Boxplot summarizing the achieved likelihoods for each considered approach in the Bouncing ball application (b) Boxplot summarizing the achieved likelihoods for each considered approach in Bubbles application (c) Boxplot summarizing the achieved likelihoods for each considered approach in Extended Bouncing ball application (d) Boxplot summarizing the achieved likelihoods for each considered approach in Diamond application

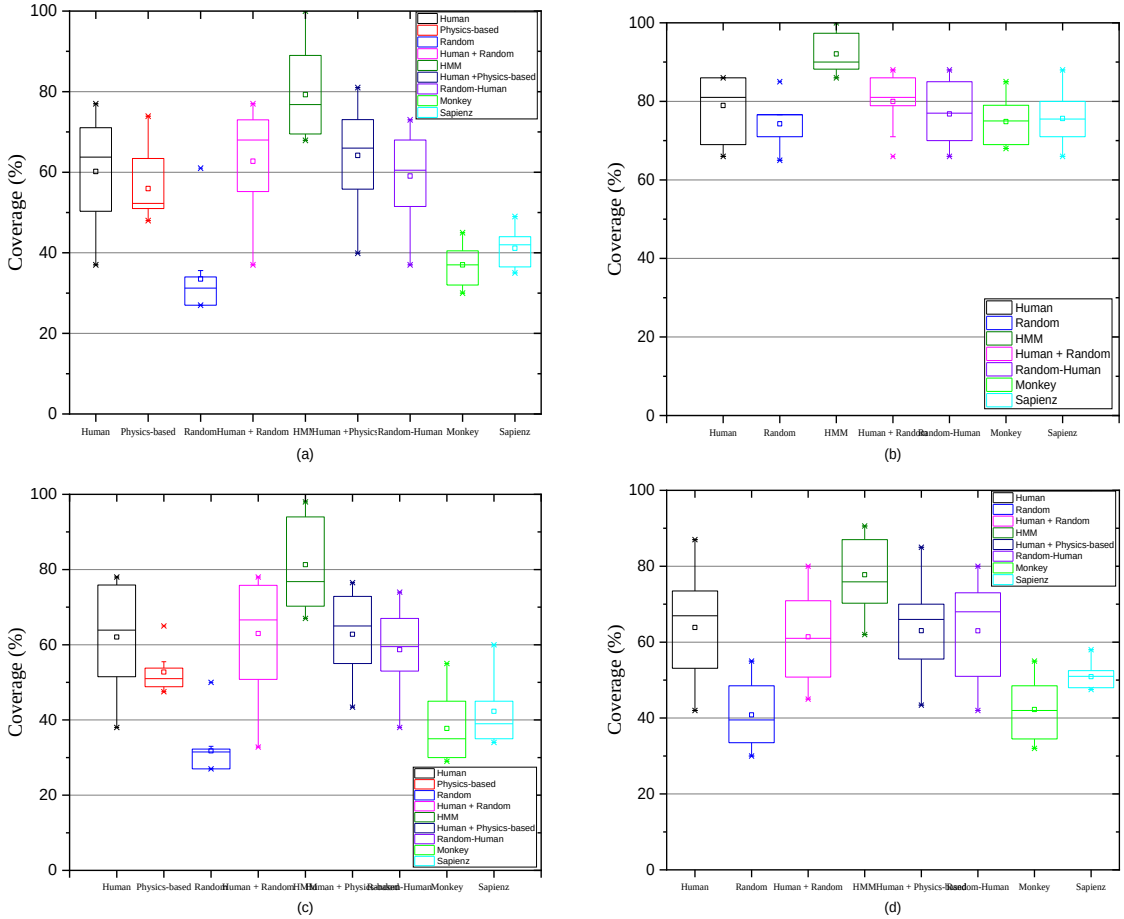

Figure. 5. (a) Boxplot summarizing the results of calculating the code coverage for each approach in the Bouncing ball application (b) Boxplot summarizing the achieved results of calculating the coverage for each approach in the Bubbles application (c) Boxplot summarizing the results of calculating the code coverage for each approach in the Extended Bouncing ball application (d) Boxplot summarizing the results of calculating the code coverage for each approach in the Diamond application

7. RUN-TIME ANALYSIS

In order to answer the part (a) of the third research question, we considered the computational complexity of the proposed test generation approach by running a single instance of this technique on a hardware and software platform consisting of a 2x2.4 GHz Quad-Core CPU, 32 GB RAM on a Mac Pro, Eclipse Indigo and a Samsung Galaxy S5.

To investigate the time complexity of HMM-based approach, we analyzed the complexity of the involved algorithms. Based on the Baum-Welch and forward algorithms' time complexities, the computation order of our approach is polynomial $O(T^2 n)$, where T represents the number of hidden states, and n indicates the number of observations. Hence, each algorithm's time complexity will not grow exponentially by increasing the number of motions.

Moreover, recent literature has considered the question of should algorithms be compared against their speed of performance – if an algorithm is twice as slow as the other algorithm should the quick algorithm get twice as many tries at getting it correct? The answers to these questions should also address part (b) of the third research question.

To provide an accurate answer to this question it should be noted that, while it is easy to have sympathy for this viewpoint, it is very difficult to construct an unbiased examination of algorithms from this perspective. Consider, testing and test case generation, the topic of this article, the first problem encountered is that test generation is only a sub-process. Following [56] an automated testing system has three components; test generation, test execution, and examination of the test results. So, the total time (t_t) is combination of all; $t_t = t_g + t_e + t_o$ where t_g, t_e, and $t_o$stand for generation time, execution time, and result examination time, respectively. Test generation and execution can be automated easier than test result examination – the production

of meaningful test oracles is still at a very early stage in research. With respect to examination of the test results, two options are normally used:

- A test oracle is constructed to automate the test examination. The test oracle usually has a simplified definition of a defect. Does the system crash or not, is an example of such a description. Here each crash is considered a "defect".
- The test results are investigated manually by the tester.

When t_e is small (very small programs) and the test result examination is fully automated (small t_o), one would be better off running more test cases instead of generating more efficient test cases [57]. In such a situation, methods that have high runtime compared to random generation are not cost effective. However, industrial software's execution runtime is usually large enough to have adequate time for test generation. Yoo et al. [58] have considered using parallelised search based optimisation algorithms to find optimal sets of test cases or to prioritize test cases for regression testing, since executing all test cases for large-scale applications is a very time consuming task. The total test execution time in their study is equal to the times for initializing test cases, evaluating the fitness values of different generations and the remaining parts of the execution time.

Further, test result examination is not typically fully automated, unless a simplistic test oracle (e.g. finding system crashes) is utilized. Hence, test result examination normally requires manual work by the tester. Hence, generating more effective test cases, which normally have higher runtime than random test cases is believed to improve failure detection in most cases. Hence, in many situations running test case generation algorithms for equal amounts of time may actually be a rather poor objective.

Additionally, fast algorithms producing large numbers of poor test cases have a significantly detrimental effect on the effectiveness of the entire testing process. Previous research shows that individual aspects such as testers' skills have as strong an effect on the results of testing, as do the test case generation techniques. Other components, including test case execution and especially manual test oracle processes are far from straightforward. Several empirically-based findings [59]–[63] have emphasized the role of experience and skills in these software testing activities. Hence, in general, making the execution and manual test oracles components more complex by running test case generation algorithms that produce large volumes of poor test cases is normally a bad idea. Other studies explicitly warn against producing large numbers of unproductive test cases. For example, in Williams et al. [64] based upon interviews with actual practitioners at Microsoft, state: "Unit testing coverage needs to be measured. The quantity of test cases is a bad measurement. More reliable is some form of code coverage (class, function, block etc.). Those measurements need to be communicated to the team and be very visible."

This implies that the practitioners are looking for techniques which assist in producing test sets which have good characteristics (such as coverage) while avoiding bad characteristics (such as large volume).

Much research exists which suggests that automatic test case generation algorithms should be principally concerned with producing high-quality test cases rather than worrying about execution times except in extreme situations.

Even if we ignore this, comparing execution times are still a highly problematic undertaking. Often the algorithms will be produced by different authors, be at different stages of development, and utilize different technologies. For example:

1. The current algorithm is produced by a student programmer, whereas a random library Mersenne Twister has been actively produced and evolved over a substantive period by a large pool of professions. Who actively ensure that the code is efficiency whereas the current algorithm is simply a first-cut prototype with no real interest in efficiency?
2. Mersenne Twister has seen decades on development with many proposals on producing more and more efficient versions whereas the current algorithm has seen none.
3. Most random libraries are written in C; whereas the current algorithm is written in R. Anecdotal comparisons often state that algorithms written in R run 1000 times slower than equivalent algorithms in C. Hence, any attempt to compare two such algorithms via execution time would be highly biased rendering any such results next to useless.

Perhaps, a better viewpoint is to consider the algorithms via their algorithmic complexity statements. However, even here the volume of work on developing an algorithm creates a significant bias. [65] noted that adaptive testing algorithms (ART) such as [66] were not effective because of their $O(n^2)$ time complexity, where n is the number of test cases. However, the field had previously made no real attempt at producing more efficient algorithms. Recently Shahbazi et al. [56] has produced a new ART algorithm, which produces more effective test cases than previous ART algorithms. In addition, the paper also looked at time complexity and produces test cases

with a time complexity of $O(n)$. Following up from this work, Singh et al. [67] have recently produced a concurrent version of this algorithm with time complexity of $O(n/p)$, where p is the number of processors available to the algorithm. Hence, even the algorithmic complex of an algorithmic tends to reduce over time as more effort is spent upon a topic. Implying that for any algorithm with a known polynomial-time solution, that even algorithmic complexity is a non-stable indicator of performance.

Having said all of this, we still provide some guidance on the effectiveness of the algorithms with regard to computational complexity. Therefore, in this study, which the computational complexity of the proposed approach is $O(T^2n)$, assuming that the method generates 200 motions using train data clustered into 13 different classes, the asymptotic complexity of the test generation process would be $13 \times 13 \times 200 = 33800$. Thus, if we allocate the same asymptotic complexity to random test generation, with computational complexity $O(n)$, random test generation approach will be able to generate 33800 motions in the provided time. Obviously, it would be more expensive to run 33800 random motions compared to 200 motions generated by HMM-based approach.

As illustrated in Table IV, it has been noticed that running all of these motions (33800) in the Bouncing ball application improves the average code coverage up to 42% for the random approach, which is sill lower than coverage, reached by HMM-based technique (75%), running 200 motions. In addition, running the 33800 test motions in Bubbles increases the coverage to 78% for random, while the percentage of code coverage is 92% for the HMM-based test case generation technique using 200 test motions. Therefore, it can be concluded that providing the same resources as HMM-based approach to random does not necessarily lead to significant improvement in the code coverage. Additionally, the time it takes to generate 200 motions using the HMM-based technique (t_g) is less than a minute, for the bouncing ball application, while it takes 3 minutes to execute them; therefore $t_g < t_e$. While, the time is required to execute the 33800 test cases generated by random approach is 23 minutes. This result confirms the statement provided at the beginning of this section, illustrating that generating too many test cases using random techniques is not always a good option for improving code coverage. Specifically, high test-execution time in industrial case studies with large test suites provides sufficient time to generate more efficient test cases, using well-designed test case generation approaches.

Table. 4 Results of providing same resources as HMM-bases to random

	Approach	*Code Coverage (%)*	t_g*(min)*	t_e*(min)*
Bouncing ball	HMM-based (200 motions)	75%	0.5	3
	Random (33800 motions)	42%	0.17	23
Extended Bouncing ball	HMM-based (200 motions)	75%	0.5	3.2
	Random (33800 motions)	40%	0.17	24
Bubbles	HMM-based (200 motions)	92%	0.2	1.2
	Random (33800 motions)	78%	0.08	15.6
Diamond	HMM-based (200 motions)	71%	0.7	2.8
	Random (33800 motions)	46%	0.25	20

8. THREATS TO VALIDITY

In this section, we consider the potential threats to the validity of our research and discuss the methods used to address them. In this study, we are principally concerned with three types of threats: internal validity, external validity, and the power of the experiment.

Threats to the internal validity might come from the method of assigning the time intervals in the empirical study. If the time intervals are estimated to be too short (long), then more (less) motions will be generated compared to when a human user is interacting with the application. To address this issue, we estimated the minimum and maximum numbers of generated movements via several users' experiences and considered their average as a type-one interval (φ).

On the other hand, the threats to the external validity of our research are centred on the generalization of the results to other SUT motion-based applications. In this study, we consider four applications and two types of motion-based applications (both with and without gyroscopic inputs). However, we also point out that the proposed technique should be applied to more and different case studies (e.g. 3D applications) in future work.

The third threat represents the *power* issue. This can lead to type-two errors in studies with insufficient numbers of samples. To address this issue, we recorded three sets of 317, 481 and 600 motion sequences to design the training sets. The data was also grouped into 95 and 105 classes, which led to training two sets of 95 HMMs and a set of 105 HMMs.

Finally, at the meta-level an obvious risk exists: Are the three research questions good proxies for defect finding capabilities? Ideally, any paper would wish to consider this research question directly. However, given the relative infancy of these types of systems, insufficient data (with regard to defects) is believed to exist to allow such an experiment to be adequately constructed. Hence, the adoption of the proxies for the exploration is required.

9. CONCLUSION AND FUTURE RESEARCH

Testing mobile applications that use motion-based gestures to interact with users poses a new challenge. Test inputs should be realistic motion sequences, which are able to simulate the user's behaviour in interacting with the application. This helps in revealing defects, which remain unknown in applications because they do not conform to expected human-generated motions. Since, Markovian models have been successfully used in software testing studies to generate models representing common user behaviour in UI testing [10], [68], [69].

In this paper, we have proposed a new HMM-based approach, which presents a solution for automating the testing process for applications supporting motion-based events. Using this method, gestures can be formally specified as sequences of motions, which are easy to re-execute in the application. Therefore, an HMM classification approach is used to classify the current movement into a class of motions providing the best description of the gesture's characteristics. Then, according to the results provided by the classification approach and using standard movement equations, a realistic proxy for the likely next movement coordinates can be estimated.

We evaluated our approach by generating a set of test inputs for four Android applications with a gaming theme. The empirical results show that the generated test cases using HMM-based approach not only cover a higher number of branches in the source code compared to randomly generated test cases, but the occurrence likelihood of the corresponding motion sequences in model trained by user generated data is also higher in HMM-based approach. This indicates that the new approach outperformed the random methods (including Monkey, Sapienz and Random-Human) in generating test cases that mimic human-user behaviour.

It should be noted that challenge in extending the number of case studies in this research is not because of the limitation of our approach in type of the motion-based applications it can process, but the issue is in finding open-source applications. Our proposed approach is a white box testing technique requires the actual source code to generate appropriate test cases.

Although these are promising results, we believe that our experiments only cover an initial exploration of this area, and several issues remain to be addressed in further studies:

- **Different types of motion-based applications.** This study only considers two types of mobile applications that use motion events to interact with users. Future studies should be performed on the proposed technique in more complicated applications with 3D graphical design.
- **Different time intervals.** Using more empirical experiences with different time intervals should also be considered in future work.
- **Influence of training data on efficiency of generated test cases.** Our evidence is based on estimating HMMs on a single set of training data. There is the potential to use different methods of sampling on training data, and evaluate their impact on the efficiency of test cases. For example, different time intervals and terminating conditions can be used during the training data capturing process.
- **Fault detection capability.** The ability of generated test cases to detect faults should also be investigated. Unfortunately, we are currently unaware of any suitable application with a published list of real-life defects.

REFRENCES

1. A. Valdi, E. Lever, S. Benefico, D. Quarta, S. Zanero, and F. Maggi, "Scalable Testing of Mobile Antivirus Applications," *Computer (Long. Beach. Calif).*, vol. 48, no. 11, pp. 60–68, 2015.
2. J. Gao, X. Bai, W. T. Tsai, and T. Uehara, "Mobile Application Testing: A Tutorial," *Computer (Long. Beach. Calif).*, vol. 47, no. 2, pp. 46–55, 2014.
3. B. Jiang, X. Long, and X. Gao, "MobileTest: A tool supporting automatic black box test for software on smart mobile devices," in *29th International Conference on Software Engineering, ICSE'07*, 2007, pp. 8–14.
4. A. I. Wasserman, "Software Engineering Issues for Mobile Application Development," *ACM Trans. Inf. Syst.*, pp. 1–4, 2010.
5. S. R. Choudhary, A. Gorla, and A. Orso, "Automated Test Input Generation for Android: Are We There Yet?," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015.

6. M. Utting, B. Legeard, and M. Utting, *Practical model-based testing: A tools approach*. Morgan-Kaufmann, 2006.
7. D. Amalfitano, A. R. Fasolino, and P. Tramontana, "A GUI crawling-based technique for android mobile application testing," in *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, 2011, pp. 252–261.
8. L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "RERAN: Timing- and touch-sensitive record and replay for Android," *Proc. - Int. Conf. Softw. Eng.*, pp. 72–81, 2013.
9. M. Hesenius, T. Griebe, S. Gries, and V. Gruhn, "Automating UI Tests for Mobile Applications with Formal Gesture Descriptions," in *Proceedings of MobileHCI'14*, 2014, pp. 213–222.
10. C. J. Hunt, G. Brown, and G. Fraser, "Automatic testing of natural user interfaces," *IEEE 7th Int. Conf. Softw. Testing, Verif. Valid.*, pp. 123–132, 2014.
11. B. Kirubakaran and V. Karthikeyani, "Mobile application testing — Challenges and solution approach through automation," in *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, 2013, pp. 79–84.
12. A. M. Fard, M. Mirzaaghaei, and A. Mesbah, "Leveraging Existing Tests in Automated Test Generation for Web Applications," in *29th ACM/IEEE international conference on Automated software engineering (ASE)*, 2014.
13. M. Ermuth and M. Pradel, "Monkey See , Monkey Do : Effective Generation of GUI Tests with Inferred Macro Events," in *International Symposium on Software Testing and Analysis (ISSTA)*, 2016, pp. 82–93.
14. I. Satoh, "A testing framework for mobile computing software," *IEEE Trans. Softw. Eng.*, vol. 29, no. 12, pp. 1112–1121, 2003.
15. D. Franke and C. Weise, "Providing a software quality framework for testing of mobile applications," in *4th IEEE International Conference on Software Testing, Verification, and Validation*, 2011, pp. 431–434.
16. D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De, U. Federico, and I. I. Napoli, "Using GUI Ripping for Automated Testing of Android Applications," in *Proceedings of the 27th IEEE international conference on Automated Software Engineering*, 2012, pp. 258–261.
17. C. Hu and I. Neamtiu, "Automating gui testing for android applications," in *Proceeding of the 6th international workshop on Automation of software test - AST '11*, 2011, no. Section 4, p. 77.
18. C. D. Nguyen, A. Marchetto, and P. Tonella, "Combining model-based and combinatorial testing for effective test case generation," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA)*, 2012, p. 100.
19. C. M. Prathibhan, A. Maliani, N. Venkatesh, and K. Sundarakantham, "An automated testing framework for testing android mobile applications in the cloud," in *IEEE International Conference on Advanced Communication Control and Computing Teclmologies (ICACCCT)*, 2014, no. 978, pp. 1216–1219.
20. T. Azim and I. Neamtiu, "Targeted and Depth-first Exploration for Systematic Testing of Android Apps," in *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, 2013.
21. S. Malek, "EvoDroid : Segmented Evolutionary Testing of Android Apps," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2014, pp. 599–609.
22. M. Linares-v, M. White, C. Bernal-c, K. Moran, and D. Poshyvanyk, "Mining Android App Usages for Generating Actionable GUI-based Execution Scenarios," in *roceedings of the 12th Working Conference on Mining Software Repositories (MSR)*, 2015.
23. X. Zeng, D. Li, W. Zheng, F. Xia, Y. Deng, W. Lam, W. Yang, and T. Xie, "Automated Test Input Generation for Android : Are We Really There Yet in an Industrial Case ?," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 3–8.
24. K. Mao, M. Harman, and Y. Jia, "Sapienz : Multi-objective Automated Testing for Android Applications," in *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA)*, 2016, pp. 94–105.
25. R. N. Zaeem, M. R. Prasad, and S. Khurshid, "Automated generation of oracles for testing user-interaction features of mobile apps," *IEEE 7th Int. Conf. Softw. Testing, Verif. Valid.*, pp. 183–192, 2014.

26. K. Moran, M. Linares-v, C. Bernal-c, C. Vendome, D. Poshyvanyk, and C. William, "Automatically Discovering , Reporting and Reproducing Android Application Crashes," in *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2016.
27. J. An and K.-S. Hong, "Finger gesture-based mobile user interface using a rear-facing camera," in *2011 IEEE International Conference on Consumer Electronics (ICCE)*, 2011, pp. 303–304.
28. E. S. Choi, W. C. Bang, S. J. Cho, J. Yang, D. Y. Kim, and S. R. Kim, "Beatbox music phone: Gesture-based interactive mobile phone using a tri-axis accelerometer," in *Proceedings of the IEEE International Conference on Industrial Technology*, 2005, vol. 2005, pp. 97–102.
29. C. B. Park and S. W. Lee, "Real-time 3D pointing gesture recognition for mobile robots with cascade HMM and particle filter," *Image Vis. Comput.*, vol. 29, no. 1, pp. 51–63, 2011.
30. S. O. Hara, Y. M. Lui, and B. A. Draper, "Unsupervised Learning of Human Expressions , Gestures , and Actions," in *IEEE International Conference on Automatic Face & Gesture Recognition and Workshops*, 2011, pp. 1–8.
31. S. Gibet, N. Country, and J.-F. Kamp, *Lecture Notes in Artificial Intelligence*. 2005.
32. D. Trabelsi, S. Mohammed, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "Supervised and unsupervised classification approaches for human activity recognition using body-mounted sensors," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2012, no. April, pp. 25–27.
33. W. J. Li, "A hybrid HMM / SVM classifier for motion recognition using μ IMU data A Hybrid HMM / SVM Classifier for Motion Recognition Using ȝIMU Data *," in *IEEE International Conference on Robotics and Bioimetics*, 2008, no. January, pp. 115–120.
34. D. Trabelsi, S. Mohammed, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "An Unsupervised Approach for Automatic Activity Recognition based on Hidden Markov Model Regression," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 3, pp. 1–7, 2013.
35. M. S. K. Gaikwad, "HMM Classifier for Human Activity Recognition," *Comput. Sci. Eng. AN Int. J.*, vol. 2, no. 4, pp. 27–36, 2012.
36. A. Mannini and A. M. Sabatini, "Accelerometry-Based Classification of Human Activities Using Markov Modeling," *Comput. Intell. Neurosci.*, 2011.
37. T. Yang and Y. Xu, "Hidden Markov Model for Gesture Recognition," Carnegie Mellon Universioty, 1994.
38. R. Cilla, M. A. Patricio, A. Berlanga, and J. M. Molina, "Recognizing Human Activities from Sensors Using Hidden Markov Models Constructed by Feature Selection Techniques," *J. Algorithms*, vol. 2, pp. 282–300, 2009.
39. O. Perez, M. Piccardi, G. Jesus, and J. M. Molina, "Comparison of Classifiers for Human Activity," in *Lecture Notes in Computer Science*, 2007, pp. 192–201.
40. W. Ching, *Markov chains: model, algorithms and applications*. Springer Science, 2006.
41. D. Kleppner and R. Kolenjow, *An Introduction to Mechanics*. Cambridge University Press, 2013.
42. C. Nello and Shawe-Taylor John, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, vol. 22, no. 2. 2001.
43. N. S. Altman, "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression," *Am. Stat.*, vol. 46, no. 3, pp. 175–185, 2007.
44. C. i. Wang and S. Dubnov, "The Variable Markov Oracle: Algorithms for Human Gesture Applications," *IEEE Multimed.*, vol. 22, no. 4, pp. 52–67, 2015.
45. Z. Moghaddam and M. Piccardi, "Training Initialization of Hidden Markov Models in Human Action Recognition," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 394–408, 2014.
46. J. B. MacQueen, "Some Methods for classification and Analysis of Multivariate Observations," *5th Berkeley Symp. Math. Stat. Probab. 1967*, vol. 1, no. 233, pp. 281–297, 1967.
47. P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, pp. 53–65, 1987.
48. J. D. Williams and S. Young, "Partially observable Markov decision processes for spoken dialog systems," *Comput. Speech Lang.*, vol. 21, no. 2, pp. 393–422, Apr. 2007.
49. L. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains," *Annu. Math. Stat.*, vol. 41, no. 1, pp. 164–171,

1970.
50. T. Tunys, “Gesture detection and NFC for Android OS,” Czech Technical University in Prague, 2012.
51. J. Ravikiran, K. Mahesh, S. Mahishi, R. Dheeraj, S. Sudheender, and N. V Pujari, “Finger detection for sign language recognition,” in *International MultiConference of Engineers and Computer Scientists*, 2009, vol. I, pp. 0–4.
52. R. Cross, “Enhancing the Bounce of a Ball,” *Am. Assoc. Phys. Teach.*, vol. 48, no. 7, p. 450, 2010.
53. L. J. Lin, “Reinforcement Learning for Robots Using Neural Networks,” Carnegie Mellon University, 1993.
54. M. Torchiano, “R Package: ‘effsize,’” 2015.
55. A. Jagannatam, “Mersenne Twister – A Pseudo Random Number Generator and and its Variants,” 2008.
56. A. Shahbazi, A. F. Tappenden, and J. Miller, “Centroidal voronoi tessellations-a new approach to random testing,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 163–183, 2013.
57. A. Arcuri and L. Briand, “Adaptive Random Testing : An Illusion of Effectiveness ?,” in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, 2011, pp. 265–275.
58. S. Yoo, M. Harman, and S. Ur, “GPGPU Test Suite Minimisation : Search Based Software Engineering Performance Improvement Using Graphics Cards,” *Empir. Softw. Eng.*, vol. 18, no. 3, pp. 550–593, 2013.
59. J. Bach, “Exploratory Testing Explained,” *Den Bosch UTN Publ.*, pp. 253–265, 2003.
60. M. Dirk, L. Begona, van der P. K. Rob, and W. Alan, *Software Quality and Software Testing in Internet Times (High-tech software quality management)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.
61. C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing*. New York, NY, USA: John Wiley & Sons, Inc., 2001.
62. A. Beer and R. Ramler, “The Role of Experience in Software Testing Practice,” in *Proceedings of Euromicro Conference on Software Engineering and Advanced Applications*, 2008, pp. 258–265.
63. J. Itkonen, M. V Mäntylä, and C. Lassenius, “Defect Detection Efficiency : Test Case Based vs . Exploratory Testing,” in *Proceedings of International Symposium on Empirical Software Engineering and Measurement*, 2007, pp. 61–70.
64. L. Williams, G. Kudrjavets, and N. Nagappan, “On the Effectiveness of Unit Test Automation at Microsoft 1,” in *20th International Symposium on Software Reliability Engineering*, 2009, pp. 81–89.
65. A. Arcuri and L. Briand, “A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering,” in *33rd International Conference onSoftware Engineering (ICSE)*, 2011, pp. 1–10.
66. T. Y. Chen, H. Leung, and I. K. Mak, “Adaptive Random Testing,” in *Annual Asian Computing Science Conference*, 2004, pp. 320–329.
67. R. Singh, J. Miller, and M. Smith, “Random Border Centroidal Voronoi Tessellations (RBCVT) Improved by Parallel Selection using Regular Sampling,” *IEEE Trans. Reliab. Under Review*.
68. J. A. Whittaker and M. G. Thomason, “A Markov chain model for statistical software testing,” *IEEE Trans. Softw. Eng.*, vol. 20, no. 10, pp. 812–824, 1994.
69. S. S. Emam and J. Miller, “Test Case Prioritization Using Extended Digraphs,” *ACM Trans. Softw. Eng. Methodol.*, 2015.

AUTHORS PROFILE