Software testing automation of VR-based systems with haptic interfaces

Cléber G. Corrêa[1], Márcio E. Delamaro[2], Marcos L. Chaim[3] and Fátima L. S. Nunes[3]

[1]*Computer Department, Universidade Tecnológica Federal do Paraná, 1640 Alberto Carazzai Avenue, Downtown, Cornélio Procópio, 86.300-000, Brazil*
[2]*Department of Computer Systems, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 400 Trabalhador São-carlense Avenue, Downtown, São Carlos, 13.566-590, Brazil*
[3]*School of Arts, Sciences and Humanities, Universidade de São Paulo, 1000 Arlindo Béttio Street, Vila Guaraciaba, São Paulo, 03.828-000, Brazil*
Email: clebergimenez@utfpr.edu.br

As software systems have increased in complexity, manual testing has become harder or even infeasible. In addition, each test phase and application domain may have its idiosyncrasies in relation to testing automation. Techniques and tools to automate test oracles in domains such as Graphical User Interfaces are available; nevertheless, they are scarce in the Virtual Reality (VR) realm. We present an approach to automate software testing in VR-based systems with haptic interfaces – interfaces that allow bidirectional communication during human-computer interaction, capturing movements and providing touch feedback. It deals with the complexity and characteristics of haptic interfaces to apply the Record & Playback (R&P) technique. Our approach also provides inference rules to identify possible faulty modules of the system under testing. A case study was performed with three systems: a system with primitive virtual objects, a dental anesthesia simulator and a game. Faulty versions of the systems were created by seeding faults manually and by using mutation operators. The results showed that 100% of the manually seeded faults and 93% of mutants were detected. Moreover, the inference rules helped identify the faulty modules of the systems, suggesting that the approach improves the test activity in VR-based systems with haptic interfaces.

Keywords: Haptic interfaces; software testing; Virtual Reality

Received 27 June 2019; revised 29 December 2019; accepted 14 April 2020

1. INTRODUCTION

Advances on interfaces generated by improvements in input and output devices and new multimedia data have produced more complex systems. Although software testing is well defined for many application areas, some complex domains are still underexplored. Additionally, each testing phase as well as each application domain can present idiosyncrasies relative to testing automation. Complex systems based on Virtual Reality (VR) still represent a challenge to software testing automation.

A mandatory requirement in VR-based systems is the three-dimensional (3D) human-computer interaction in real time [1]. Users can interact with 3D Virtual Environment (VE) by using several types of interfaces, from common mouse and keyboard to complex devices such as haptic interfaces. With these devices, users perform tasks (such as moving an object to a specific place using the mouse device) and receive feedback from the system according to their actions (for instance, visualizing a displacement of an object or feeling the touch feedback) [2]. Figure 1 shows an example of a VR-based system with haptic device, in which the user moves a virtual needle using a haptic device in the dental anesthesia procedure simulation.

Interaction possibilities in the 3D VE are only limited by the characteristics of the interaction devices used and the behavior programmed to the objects present in the VE. This means that users can execute numerous actions, according to their own manner to interact with the VE. For example, to simulate a medical procedure,

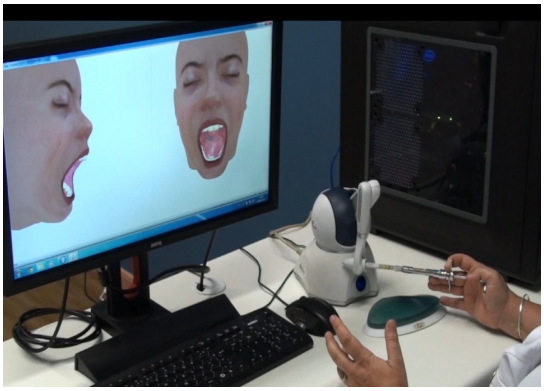

FIGURE 1. User interaction with a haptic device in a VR-based system for dental training using two viewpoints of the same virtual scene [3].

there are several different ways of moving a virtual medical instrument to reach a target, since each user can move the object in a certain trajectory. Thus, input and output data can be very different when comparing two users using the same interface to perform a similar task.

Haptic interfaces provide bidirectional communication (input and output), capturing data regarding the position and rotation in the real space and offering touch feedback to the user. In most VR-based systems, there is also a visual feedback together with haptic feedback. Additionally, these VR-based systems rely on complex algorithms to provide adequate behavior of the virtual objects in response to real time interaction. For example, these algorithms must determine accurate collision detection between virtual objects, virtual objects deformation and haptic feedback calculation according to the features of virtual objects (e.g., stiffness and viscosity) and behaviors (e.g., ability to move) [4, 5].

Every time an algorithm (e.g., collision detection or haptic feedback calculation) is changed, the VR-based system with haptic interface, here called System Under Test (SUT), should be retested with the same input data to check whether the correct outputs are still being generated. This means that automated test oracles or automated procedures to check outputs of a software, are in hand to improve testing, making it faster, cheaper, and more reliable.

Due to their complexity and characteristics, VR-based systems with haptic interfaces may be simultaneously built by teams with several developers in which haptic devices are a scarce resource. Thus, it is often necessary to test algorithms without the physical presence of the device. Some approaches have addressed the challenge of testing complex systems, especially for Graphical User Interfaces (GUIs) [6, 7]. However, VR-based systems involve 3D virtual space, haptic and visual feedback, and multiple virtual objects with different features whose behavior depend on user actions and Physics laws. Additionally, experts in the application's area, such as in the medical training, are needed to validate the applications [8]. From the scenario presented, the literature lacks approaches to automate test oracles in the haptic interfaces domain, including the need to allow tests when a haptic device is not available.

We present an approach for testing automation of VR-based systems with haptic interface. It tackles the idiosyncrasies of haptic devices to apply the Record and Playback (R&P) technique to these systems. Our main contribution regards automating testing of VR-based systems with haptic interfaces through test oracles, even when the haptic device is unavailable and the correct haptic sensations depend on experts' availability. We also provide inference rules that help the tester to locate the typical modules of these systems that are responsible for an erroneous behavior, i.e., for a failure of the system. These rules allow to determine which of the two important modules of the haptic human-computer interaction is faulty: collision detection between virtual objects and haptic feedback calculation algorithms, in this case, force feedback calculation (Section 2). Our approach allows checking whether a software that was changed during development or maintenance processes is correct based on previous correct executions. Thus, it can reduce development time and costs of VR-systems with haptic interfaces, increasing their quality.

We validate our approach using three SUTs: *primitive objects SUT*, *dentistry SUT*, and *game SUT*. For each SUT we derived two sets of versions, including faulty versions: one with manually seeded faults and another seeded with faults generated by applying mutation operators. We used Aspect-Oriented Programming (AOP) to implement R&P in the SUTs. The percentage of versions with manually seeded faults detected and the mutation score were analyzed, as well as the results of the inference rules, indicating that our approach is promising to automate software testing in VR-based systems with haptic interfaces.

This paper is organized as follows: Section 2 presents the background on test oracles and VR-based systems with haptic interfaces; Section 3 discusses the studies in this field and works on automation of non-testable programs. Section 4 presents an overview of our approach. The case study is described in Section 5; Section 6 shows the results and Section 7 presents a discussion about the main advantages and limitations of the proposed approach. We draw our conclusions and present the future works in Section 8.

2. TEST ORACLE AND HAPTIC INTERFACES

Test oracles are procedures created so the tester can check if the outputs of a software execution are correct [9]. These procedures may be used to check faults by observing the program state, according to Reachability, Infection, Propagation, and Revealability

(RIPR) model [10]. In this model, the software state is the current value of all software variables and the current statement during the execution. A test needs to **reach** the location or module of the fault (Reachability). The execution of the statements in the faulty location must cause an incorrect internal software state, i.e., the state must be **infected** (Infection). The incorrect internal software state must **propagate** to an Incorrect Final State or to a failure (Propagation). The complete output state of the software includes the Incorrect Final State. Test oracles strategies are used to observe the final software state, defining an Observed Final Software State. Failures are only **revealed** if the Observed Final Software State includes part of the Incorrect Final State (Revealability) [10, 11].

In the test oracle context, a problem regards predicting the correct outputs for certain inputs and then comparing these correct outputs with the observed outputs of a SUT. Another issue is related to the use of automated test oracles, allowing to reduce costs and time of the software testing activity [9].

Haptic interfaces are related to touch [2]. They can be used in many type of systems, but the most common use is in tools that simulate procedures where the sense of touch is important. Usually such systems are based on VR concepts to build 3D interactive environments. Many application areas can be benefited with such interfaces, such as simulation of medical procedures [12] and entertainment [13, 14].

Haptic interfaces are both input and output devices. Simultaneously, they allow the systems to receive and send data to users. There are several technologies involved in haptic interfaces, namely optical fibers, movement sensors, magnetic sensors, and mechanisms that provide haptic feedback [15]. The haptic sensations can be classified according to their nature: tactile feedback, which indicates the characteristics of the surface of an object (temperature and roughness) when there is contact with the skin; and force feedback, the most common, which indicates the weight or resistance of objects [16].

In the context of VR-based systems, a virtual object is associated to the haptic device. Figure 2 describes the haptic process. It starts with the communication between software and device, then a haptic loop is started, running various frames per second during human-computer interaction. At each frame, the position, rotation, and velocity of the device are received (input), as well as communication data between application and device. Data, such as force, are also sent to device (output), and the behavior of the virtual object associated to the haptic device is changed according to the input. When the haptic loop is over (user finishes the execution or an error occurs), the communication between software and device is interrupted.

The force is calculated using algorithms that consider features of the VE. For example, in medical training,

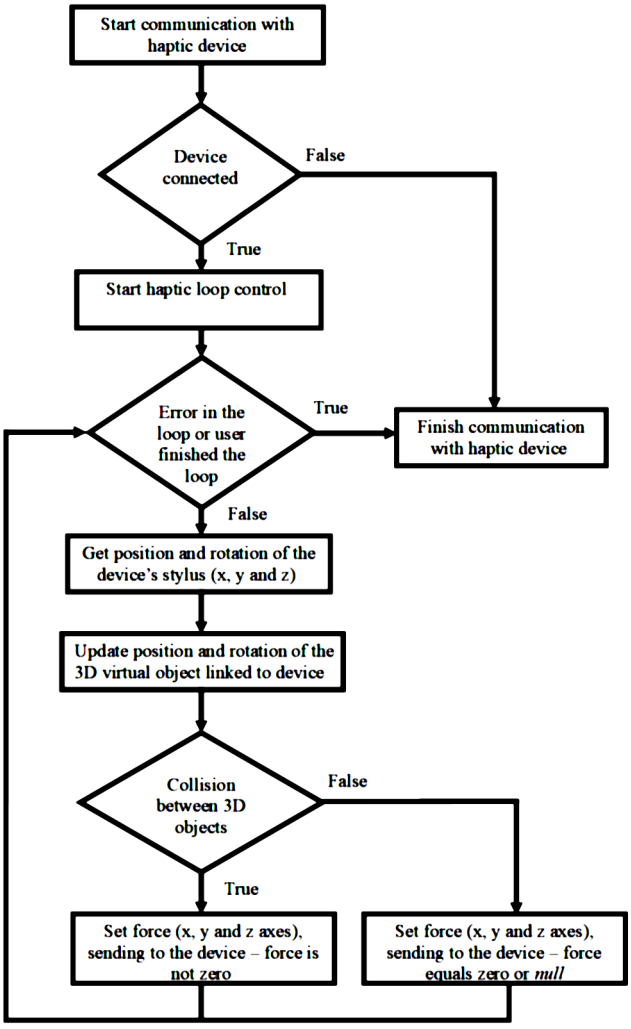

FIGURE 2. Flow chart for the haptic process, describing the steps during human-computer interaction.

algorithms that calculate force use physical features (stiffness, viscosity, elasticity) of the virtual objects that represent medical instruments and tissues, as well as their behavior (predefined motion and shape deformation). Deformation methods and mathematical models are used to calculate haptic feedback [8].

Researchers have studied ways to increase realism and to execute VR-based systems in real time [17, 18]. Virtual objects deformation methods (e.g., finite elements [19, 20]) provide physical accuracy in simulation, but they are time-consuming. Other approaches (e.g., the mass-spring [21]) can be executed in real time, but they do not provide accuracy when compared to finite elements methods. Therefore, new methods are being always created to improve VR-based systems. As consequence, these systems are modified and need to be retested. However, testing this class of applications can be difficult, time-consuming, and limited to the presence of the haptic device and experts of the application area. Our approach overcomes some

of these limitations.

3. STATE OF THE ART

There are several complex domains in the software testing automation: GUIs, with visual information presented in different screen resolutions and sizes; Web applications, whose correct execution depends on the browsers and communication systems used; mobile applications, which depend on resources that are present in the devices; image processing, whose evaluation of the resulting images depends on experts of the application area; and VR and Augmented Reality systems, including haptic interfaces, that allow 3D human-computer interaction in a fully synthetic environment or a mixed environment (real and virtual).

Researchers have employed a variety of techniques to automate software testing in GUIs, such as: R&P and manually written scripts [6]. There are specific tools for each technique. There are approaches that use script-based languages, such as JFCUnit [22], in which testers manually create unit test cases. Such test cases are sets of method calls that invoke GUI events. Selenium WebDriver, Robotium, Abbot or SOAtest are some examples of tools that allow creating this type of test cases [6]. However, this manual work requires high workload and demands considerable time of the tester.

Test Automation FX Selenium Interface Development Environment (IDE) and Quick Test Pro are tools that can be employed in the test activity using R&P technique. They provide functionalities to register and retrieve sequences of events from GUIs [6, 7].

For image processing applications, the test oracle problem can be addressed as metamorphic relations of morphological image operations, such as dilation and erosion [23]. Another approach uses Content-Based Image Retrieval concepts to define graphical test oracles, which are used to compare two images and decide the correctness of the output of an image processing program [24].

Web applications compose another class of applications that deserves special attention in the testing activity. This group of programs encompasses the validation of inputs in the client that can be passed arbitrarily to the server [25], as well as the information quality in online search services [26]. There are also R&P techniques that take into account HyperText Markup Language (HTML) *elements/Attributes* and *Locators*. Selenium IDE is a tool applied in this context [27].

A suite of automated oracle comparator operators for testing Web applications was proposed to reveal failures [28]. Each comparator focuses on certain characteristics of the possibly non-deterministic Web applications in the form of HTML responses. The authors mention that concurrency, non-determinism, dependence on persistent state and previous user sessions, a complex infrastructure, and a large number of output formats require developing different playback and oracle comparison operators for Web applications.

In spite of the growing number of applications using VR concepts, we found few studies concerned with automating the software testing activity for this class of applications. In VR-based systems implemented using Scene Graphs there are efforts that aimed at describing test criteria and automating the software testing activity [29], as well as creating an automated functional testing approach for VR applications [30]. Specifically considering interaction in VR-based systems, Souza [31] applies the R&P technique for body-tracking-based applications using a Kinect sensor; the R&P technique was also applied for the VR Juggler platform, using input from several devices and system's states during interaction [32]. Software that controls devices employed in VR-based systems are also tested. A technique that automatically tests Kinect sensor-based applications by synthesising realistic sequences of skeletal movement was proposed. It generates test cases by means of a statistical model, which is trained on a corpus of common gestures. The results suggest that the generated tests achieve significantly higher code coverage than random test inputs [33]. A framework was proposed to model human hand data that interact with five applications using the Leap Motion device. Test data were generated automatically from this model [34].

To the best of our knowledge, there are no studies aimed at providing techniques to automate tests for VR-based systems with haptic interfaces, strengthening our contribution in this area, as shown in the next section.

4. APPROACH FOR TESTING AUTOMATION OF VR-BASED SYSTEMS WITH HAPTIC INTERFACES

The current section presents the details of our approach for test automation of VR-based systems with haptic interfaces.

4.1. Assumptions

Our approach is based on some important assumptions that must be addressed when VR-based systems are developed with haptic interfaces:

1. These interfaces capture input data during each moment of the human-computer interaction such as position and rotation of the device stylus handled by the user;
2. These interfaces provide output, such as force feedback, which is a result of the inputs, of collision detection and force calculation algorithms. Algorithms that compute force feedback take into account physical features (e.g., stiffness and elasticity), and behaviors (e.g., displacement and deformation) of virtual objects, as well as the data provided by the users (e.g., trajectory and velocity of the virtual object handled by user via haptic

device). These algorithms are usually triggered when a condition occurs (e.g., a collision between two objects) and they must be able to analyze all the VE features in real time to calculate the correct feedback with no noticeable delay;

3. The feedback is highly dependent on the user's actions. It is very difficult for the tester to provide exactly the same input when handling the device, i.e., to maneuver the haptic device exactly the same way, to repeat a given test case. Since the inputs are provided by moving the haptic interface, users cannot accurately control the positions. The correct perception of the feedback can be difficult since the correct sensation is often perceived only by experts, and usually testers are not experts in the application area;
4. VR-based systems can use several input and output devices, such as haptic interface and high-resolution graphical equipment. Additionally, response in real time is required to ensure the realism. All these requirements have guided several research efforts to develop fast and robust algorithms for graphical and haptic feedback. The way to reach these characteristics is to customize the algorithms that calculate the feedback according to the application needs. Thus, such algorithms can be changed several times for improvement, and the new versions of the programs must be retested;
5. A haptic device is a relatively expensive resource and is still considered as non-conventional equipment. Thus, it is not usual that all developers have a specific equipment available throughout the entire development process. This unavailability can delay the development and decrease the effectiveness of the testing activity if an automated alternative approach – preferentially that does not require the physical presence of the equipment – is not available.

4.2. Design and implementation

Based on the aforementioned considerations, our approach was designed to:

1. Record test oracle information (inputs and outputs) of correct VR-based systems with haptic interfaces to test new versions (regression testing);
2. Repeat execution (playback) using exactly the same input data;
3. Compare outputs to provide a verdict on the correctness of the new version (test oracle);
4. Test new versions of VR-based systems with haptic interfaces, even if the physical device is not available;
5. Locate faulty modules of VR-based systems with haptic interfaces that have faults.

To reach the aforementioned goals, we used an adapted R&P technique. R&P allows recording the user's actions or commands (inputs) when the user is using an interface, as well as feedback (outputs), creating scripts. Such scripts are used to test the software in subsequent executions, checking whether the previous and current output data are the same when the same input data are provided [35].

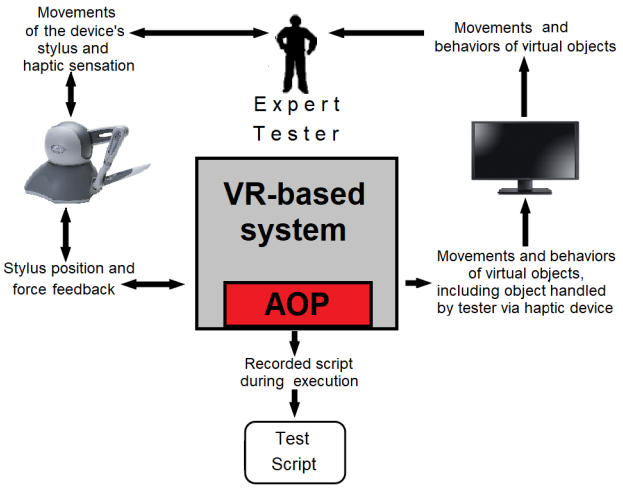

FIGURE 3. *Record* step of the R&P technique for VR-based systems with haptic devices, generating the test script (inputs and expected outputs).

In the *Record* step, the SUT is executed, necessarily using the physical haptic interface. The main actions (displacement of virtual object using haptic device) of the tester, or expert, are recorded, composing a test script. Likewise, the results produced by the program are also recorded to compose the test script. The inputs and outputs (complete test script) of the *Record* step constitute the test oracle information.

Figures 3 and 4 show the steps of our approach. Figure 3 describes the *Record* step, in which the haptic device makes the interface between the tester or expert and the VR-based system whereas AOP records the information. Figure 4 (*Playback* step) shows the script recorded in the *Record* step being executed to reveal failures and the inference rules to find the modules where the faults are located. In the *Playback* step, a physical haptic device is not needed since AOP enables the execution of the test script, as well as the comparison of the outputs (recorded and current) and the application of the inference rules. A graphical interface is used in both steps.

4.3. Generation of scripts and re-execution

In the *Record* step, a test script is generated with input and output descriptions to compose the test oracle information for posterior executions. Each line in the script records information collected in each frame during the human-computer interaction using the haptic device. Figure 5 shows part of a test script

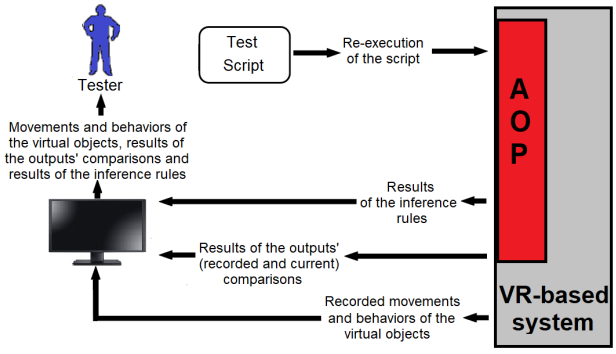

FIGURE 4. *Playback* step of the R&P technique for VR-based systems with haptic devices, using the test script to provide the verdict, i.e., the results of the comparison between outputs current and recorded outputs and inference rules.

recorded using one of the SUTs from Section 5.

In this paper, the record consists of the command name to receive information from a haptic device (e.g., **getPosition**) and the positions of the haptic interface in the three axes (x, y, and z). In addition, the command name to send force information (e.g., **setForce**) and the force calculated in the three axes (output) when a collision is detected are also recorded. The fields are separated in the file using semicolons.

The number of lines in the script depends on the frequency or rate of frames of the haptic loop. Usually a frequency of 1,000 Hz (1,000 frames per second) is required to give the user the sensation of response in real time, but there is a discussion on this subject in which different contexts require different frequencies [12]. The frequency may be influenced by the VE characteristics, namely, number and polygonal complexity of the virtual objects, the computer's memory and processor, the algorithms to calculate collision detection, objects deformation and haptic feedback, and the trade-off between speed and accuracy in the simulation.

In the *Playback* step, each line of the script is read and the four initial fields (command name **getPosition** and position of the haptic device in the axes x, y, and z) are used as inputs for the SUT, which allows moving the virtual object without a haptic device. The last four fields in the script contain the command name **setForce** and the force in the three axes (force information in these three axes are outputs for comparison). Thus, the output generated by the SUT (force in the three axes) is compared with the last three fields in the script generated in the first step; that is, they are compared with the three values presented after the **setForce** command in the test script. When force feedback values coincide with force feedback values of the SUT, a correct verdict is presented. Otherwise, an incorrect verdict is informed to the tester, who can stop the test to correct the SUT. Thus, correct and incorrect outputs can be observed during and after the *Playback* step.

4.4. Verdict composition

The verdict reveals failures and modules (collision detection or/and haptic feedback calculation modules) where the faults causing these failures are located. We know that VR-based systems with haptic devices work with collision detection and haptic feedback calculation (or, in our case and commonly in various systems, force feedback calculation).

The force feedback values (recorded and current) must be exactly the same for a correct verdict since a difference in only one axis and in one frame during execution is considered incorrect.

To compose the final verdict, we defined inference rules to analyze the comparisons between outputs recorded in the *Record* step and the current outputs obtained during the *Playback* step at each frame recorded. The rules are based on two important algorithms mandatory in systems with haptic feedback: collision detection and force feedback calculation. The force feedback algorithm is executed after the collision detection algorithm identifies a collision.

The Decision Table presented in Table 1 shows the rules, as well as their conditions and actions. The first column shows the conditions and the actions to be carried out in the case of the conditions to be satisfied. Other columns show the values (false or true) for the conditions of each rule, regarding the current force and the force recorded; and the actions (messages to testers), indicating the module of the SUT where the fault is possibly located (in this case, collision detection or/and force calculation algorithm).

In **Rule 1**, SUT calculated the force, the recorded force was also calculated and the values (current and recorded) are equal, i.e., values calculated for the SUT during the test and values in the test script generated previously are equal, indicating that the SUT offers correct output. In this case, force is not zero since a zero force means that there is no feedback. Force can be positive or negative at each axis; and if only one of the values of the three axes is different from zero, indicates there is force. Furthermore, if there is a difference between current and recorded values, SUT calculated different force comparing to test script.

In **Rule 2**, there is no current force (current force equals zero), there is no recorded force (recorded force equals zero), and the values (current and recorded) are equal, indicating that the SUT did not calculate the force when it is not necessary, offering correct output.

Regarding **Rule 3**, SUT calculated the current force, the recorded force was also calculated (recorded force is not zero), but the values (current and recorded) are different, indicating incorrect output. The collision was detected, though the force calculation algorithm may have a fault.

In the case of **Rule 4**, there is no calculated current force (current force equals zero), the recorded force was calculated (recorded force is not zero), and the

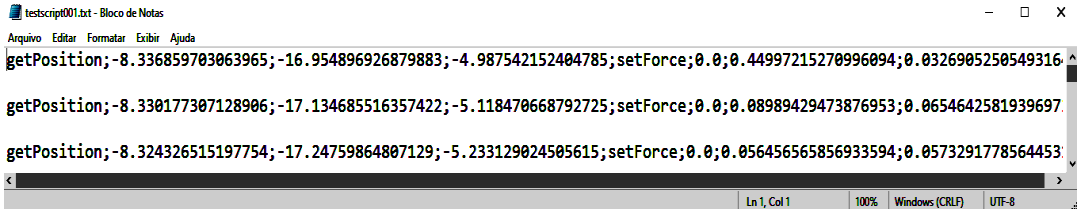

FIGURE 5. Part of a test script — position in millimeters and force in Newtons (three axes) in three frames.

TABLE 1. Decision Table for the inference rules.

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**
current force value == 0	False	True	False	True	False
recorded force value == 0	False	True	False	False	True
current force value == recorded force value	True	True	False	False	False
Actions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**
Show message about faults in the collision detection algorithm	No message - success	No message - success	No message - success	Message - Rule 4	Message - Rule 5
Show message about faults in the force calculation algorithm	No message - success	No message - success	Message - Rule 3	Message - Rule 4	Message - Rule 5

values (current and recorded) are different, indicating incorrect output of the SUT. It is possible that the collision is not detected; **or**, if the collision is detected, the force algorithm has a fault.

In **Rule 5**, SUT calculated current force (current force equals zero), the recorded force was not calculated (recorded force is equal zero), and the values (current and recorded) are also different, indicating incorrect output of the SUT. Similar to **Rule 4**, it is possible that the collision is not detected; **or**, if the collision is detected, the force algorithm has a fault.

Others three rules were not created because whether the current force is zero and the recorded force is not zero or vice versa, i.e., one of the two first conditions is true, the third condition can not be true (current force equals recorded force). Whether the two first conditions are true the third condition can not be false.

Based on the five rules previously presented, a final result is composed, providing guidance to the tester to identify where the faults are (in which modules). The verdict is presented to the tester in the video monitor during the execution, with partial results at each frame.

5. CASE STUDY

In order to validate our approach, we conducted a case study with three SUTs: *primitive objects SUT*, a system with primitive virtual objects (boxes and spheres); *dentistry SUT*, a dental anesthesia VR-based training system [36, 4, 5]; and *game SUT*, a game for breast biopsy training using VR [37]. For each SUT we created two sets of versions, including faulty versions. In next subsections, we detail the study.

5.1. Objective

The objective of this case study is to investigate the application of our approach in the haptic interface domain that is, VR-based systems which capture position information and provide force feedback. Thus, we intend to analyze the benefits and limitations of our approach towards establishing ways for automating the software testing activity in the domain here considered.

5.2. Research questions

We established two research questions:

RQ1. *Is the R&P technique effective to test VR-based systems with haptic interfaces?* The objective is to check whether our approach can be used to test VR-based systems with haptic interfaces and whether the test oracle information is effective at detecting the presence of faults.

RQ2. *Do the inference rules help find faulty modules in VR-based systems with haptic interfaces?* The objective is to check whether simple rules help identify faulty modules. Beyond collision between virtual objects and force feedback calculation modules, a VR-based system has others, for instance, for loading virtual objects. Thus, if effective, the inference rules will narrow down the search for faults to specific modules.

5.3. Assumptions

We assume that the setup of the systems in both steps of the *Record* and *Playback* process must be equal, i.e., features of the virtual objects, such as scale, rotation and position in the three axes (x, y, and z), as well as the shape, cannot be changed between these two steps. Especially those objects not directly handled by users, such as the objects representing boxes and

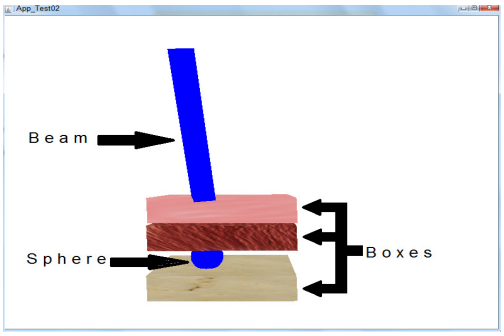

FIGURE 6. Graphical interface of the *primitive objects SUT* – simple objects (boxes and sphere).

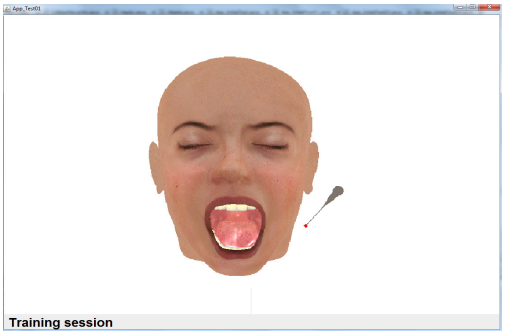

FIGURE 7. SUT graphical interface for dental anesthesia training.

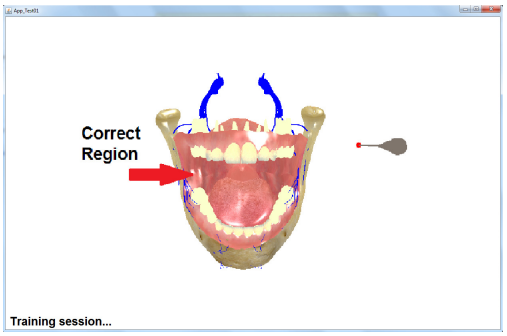

FIGURE 8. Graphical interface of the *dentistry SUT* - objects representing internal anatomical structures and needle.

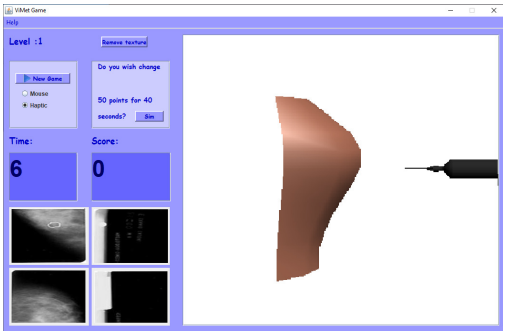

FIGURE 9. Graphical interface of the *game SUT* - objects representing breast and syringe.

sphere, as shown in Figure 6 (*primitive objects SUT*); the objects representing anatomical structures (bones, skin, tongue, muscles and teeth), as shown in Figure 7 and in Figure 8 (*dentistry SUT*); and the object representing the breast, as shown in Figure 9 (*game SUT*). Other features previously assigned to the virtual objects (e.g., stiffness and viscosity values), must also be unaltered between recording and playing back. Thus, the rationale is to keep the features of the VEs and add faults in the source code.

The failures can be caused by faults in the collision detection, force feedback calculation, communication initialization between system and device, haptic loop control and motions capture (position). The beam (*primitive objects SUT*), virtual needle (*dentistry SUT*), and virtual syringe (*game SUT*) are the virtual objects that will be handled by user during human-computer interaction.

In the *Record* step, the VR-based system with haptic interface is considered as the correct version, so the input/output data that it produces is the oracle information used in executions *a posteriori*.

5.4. Study execution

Our approach was analyzed using three SUTs and two sets of versions of the SUTs, including faulty versions (the first set with manually seeded faults and the second set seeded by applying mutation operators).

For the first set of versions, the first step (*Record*) was to generate the correct outputs according to the inputs, using AOP resources. We ran the *Record* step 15 times, creating 15 tests scripts: 5 scripts were recorded using the *primitive objects SUT*, 5 using *the dentistry SUT*, and 5 using *the game SUT*.

The *Playback* step, also using AOP resources, was executed according to Table 2.

The repetition was performed to analyze whether different frame update rates would influence the results of the tests. The rates can vary during human-computer interaction and between *Record* and *Playback* steps due to computer resources (memory and processor) and due to parallel processes (e.g., software modules for collision detection and force feedback calculation executing in parallel).

During the *Playback* step, the failures were revealed and the verdict was shown to the tester for each execution. The faulty versions in which our approach did not reveal the failures, the correct versions that our approach suggested existing failures, and the use of the inference rules were analyzed and discussed.

For the second set of versions of the SUTs (set seeded by applying mutation operators), we used a tool to generate mutants. In total, 160 mutants for the *primitive objects SUT* and the *dentistry SUT*, and 80 mutants for the *game SUT* were generated. We used three scripts (generated in the *Record* step), one for

TABLE 2. Seeded faults manually for each SUT, version, showing the number of scripts generated in the *Record* step and the executions of these scripts in the *Playback* step.

SUT	**Version**	**Scripts number**	**Executions for each** script	**Total**
Primitive objects	Original	5	5	25
	Fault in the force calculation algorithm	5	5	25
	Fault in the collision detection algorithm	5	5	25
	Different collision detection algorithm	5	5	25
Dentistry	Original	5	5	25
	Fault in the force calculation algorithm	5	5	25
	Fault in the collision detection algorithm	5	5	25
	Different collision detection algorithm	5	5	25
Game	Original	5	5	25
	Fault in the force calculation algorithm	5	5	25
	Fault in the collision detection algorithm	5	5	25
Total				275

each SUT.

During the *Playback* step, the presence of failures was checked and the verdict was shown to the tester. The mutants, especially those that were not killed, were analyzed and discussed. The use of the inference rules was also analyzed.

Subsection 5.4.1 presents the Systems Under Tests (SUTs), Subsection 5.4.2 describes AOP resources used in the SUTs. Subsection 5.4.3 shows how the test scripts were generated for each SUT, and Subsection 5.4.4 describes how the faults were manually seeded and the mutants were created.

FIGURE 10. Haptic device with Carpule syringe - instrument used by dentists in the anesthesia procedure [36].

5.4.1. Systems Under Tests (SUTs)

The SUTs used in the case study allow communication with the haptic device Geomagic Touch [38], which provides a resolution of 0.055 mm (millimeters), six Degrees of Freedom (DoF) of motion, three Degrees of Freedom of Force Feedback (DoFF), a maximum force of 3.3 Newtons and workspace of 170 mm (width), 120 mm (height) and 70 mm (depth).

The SUTs are VR-based systems with different features (e.g., virtual objects with different shapes and scales). All the SUTs were developed using Java programming language and Java3D API (Application Programming Interface) for VE rendering, as well as collision detection computation. Java Native Interface (JNI) was used to integrate software modules written in Java and C programming languages, since C was employed to access the functions from the device software library. The SUTs require human-computer interaction in real time, usually, at a frequency of 1,000 Hz. The haptic rendering is usually combined with graphical rendering, allowing the user to view and feel virtual objects.

These SUTs detect collision between virtual objects and calculate force feedback, allowing the haptic sensation. In Figure 7 we can see the interface of the *dentistry SUT*.

The haptic device has a stylus that is handled by the user (Figure 10) and it offers data such as: position in three axes (x, y, and z), rotation in three axes, motion velocity in three axes and force feedback in three axes. In Figure 10, the stylus of the haptic device was replaced with Carpule syringe, an instrument used by dentists in the anesthesia procedure.

Figures 6, 8 and 9 show the virtual objects of the three SUTs, especially the objects handled by user at each SUT (beam in the *primitive objects SUT*, needle in the *dentistry SUT* and syringe in the *game SUT*).

The class called *Haptic* controls the information of the haptic device, receiving the position of the stylus and sending data regarding the resistance of virtual objects when a collision occurs between the object handled via device and other virtual objects, causing the force feedback. Collision detection and force calculation were implemented in other different classes (*Collision* and *ForceFeedback*). When the needle, syringe or beam reach the virtual anatomical structure or the simple virtual object (collision detection), the force calculation is activated.

The main difference in the human-computer interaction between SUTs is the precision of movements and haptic feedback required in the dental anesthesia and breast biopsy procedures comparing to system with primitive objects. In these cases, anatomical structures (e.g., muscles, nodules, blood vessels and nerves) cannot be viewed by users, only felt using haptic percep-

tion; and the user must know where the needle or needle/syringe is during the insertion.

5.4.2. Aspect-Oriented Programming (AOP)

To carry out the *Record* and *Playback* steps without changing or instrumenting the source code of the SUT, we used AOP. This type of programming allows to modularize transverse interests (called aspects) affecting *Classes* and *Objects* independently of the original code. Since the major part of the system was implemented using Java, we chose the AspectJ library, a Java extension that provides aspect resources [39].

We used these resources (such as *advices* and *pointcuts*) to intercept methods of the *Haptic* class. In the *Record* step, the interception allows to record data returned by some methods (e.g., to get the position in the 3D space) and data to be used by methods (to set the force).

In the *Playback* step, the goal is to test the software without the presence of a physical haptic device. In a normal execution the method would indicate to the user that there is no device connected and would finish the program. To cope with the lack of the device, resources provided by the AspectJ library were chosen to identify the call for particular methods of the *Haptic* class (e.g., the method that starts the communication of the software with the device); and to indicate that this method must be executed.

Another method that depends on the presence of the haptic device is the method that checks whether the device is working correctly; a similar implementation to the method that starts the communication was employed for this verification method. In the *Playback* step, the method that sends the force to the haptic device was intercepted, avoiding its call and obtaining the force calculated by system; and the method that captures the position was intercepted to read this data in the script generated in the *Record* step.

To use AOP, the tester must be familiar with a specific library of the programming language, such as AspectJ for Java, as well as the signature of the methods (names, parameters, data that they return) of the software. However, the tester does not need to modify the source code. Java and AspectJ are the best known languages to support AOP but many other languages (e.g., Python, C/C++, C#, Lua, PHP and JavaScript) have support to AOP.

5.4.3. Test scripts

To generate the test oracles for the *primitive objects SUT*; that is, to create scripts from its correct version, we defined a unique start contact point (the top of the upper third box) and different end points (five end points), that the user should reach with the virtual beam (handled according to haptic device). The tester performed five different trajectories for the virtual beam, that is distant from other virtual objects before

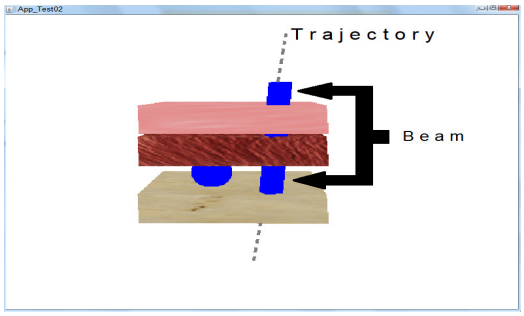

FIGURE 11. Beam passing by three virtual objects - the beam in contact with the boxes.

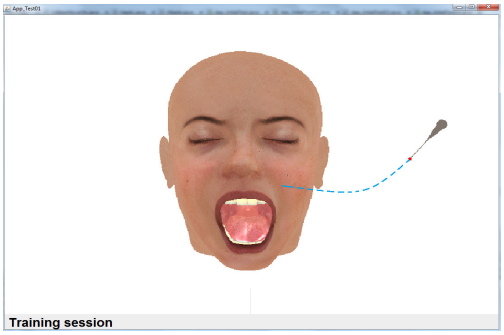

FIGURE 12. Example of trajectory with needle reaching the skin.

the interaction. End points were the upper second box; upper third box; sphere; bottom box passing by two upper boxes and sphere; bottom box passing by two upper boxes. In the last two cases, three and four virtual objects were reached by the virtual beam, generating more than one force feedback value different from zero in the same simulation. In Figure 11 we can observe the beam reaching the three boxes. 6

For test oracles of the *dentistry SUT*, we defined five end points that the user should reach with the virtual needle (handled using haptic device). The tester should carry out five different trajectories of the virtual needle for creating five test scripts. The virtual needle is distant from other virtual objects before the interaction (Figures 7 and 8). Figure 12 shows a recorded trajectory (between needle and skin). End points were the central right superior tooth; buccal mucosa; gum; buccal mucosa and jaw (bone) together; and skin and jaw together. In the last two cases, two virtual objects were reached by the virtual needle, generating more than one force feedback value different from zero at different moments in the same simulation. In Figure 13 we can see a trajectory where the needle crosses the mucosa and reaches the jaw.

In Figure 14 we can see a trajectory where the virtual syringe reaches the virtual breast.

For test oracles for the *game SUT*, the graduate students in Computer Science used the system, handling the syringe using the haptic device to reach the virtual object that represents the breast, specifying five

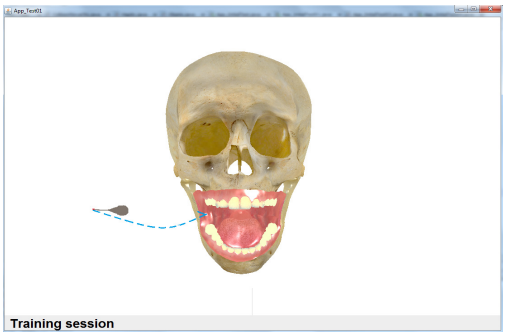

FIGURE 13. Trajectory: needle passing by two virtual anatomical objects. The skin was removed to show internal structures.

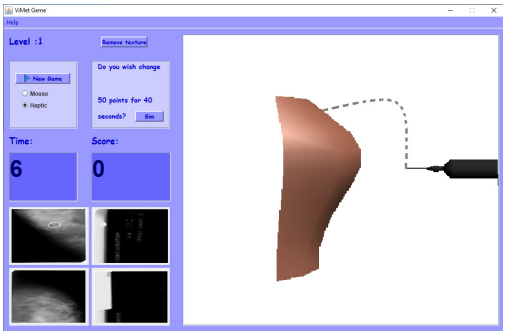

FIGURE 14. Trajectory: syringe is moved to reach the virtual breast.

trajectories according to their preferences. The virtual syringe is distant from other virtual objects before the interaction.

Each object of the SUTs has a stiffness value according to its composition. These values were assigned by experts in the case of the anesthesia simulator and breast biopsy training game (*dentistry SUT* and *game SUT*), and were recorded in a database. *Primitive objects SUT* had random values, also recorded in a database.

The five trajectories for each SUT were used in the executions of the first set of versions, including faulty versions (faults manually seeded). We choose one out of these five trajectories for each SUT to use in the executions of the second set of versions (mutants and equivalent mutants). We selected the trajectory in which the top box was the end point for the *primitive objects SUT*, the trajectory in which the gum was the end point for the *dentistry SUT*, and a trajectory specified by graduate students for the *game SUT*.

5.4.4. Seeding of faults

We used two strategies to seed faults in the SUTs: manual faults insertion and mutant generation, creating faulty versions for the two sets of versions of the SUTs. Thus, some versions of these sets did not have faults, allowing to evaluate the approach with correct versions.

In the first set, we used two seeded faulty versions, a version with a different collision module and the original version of each SUT, i.e., four versions in the first set. In the first faulty version, we changed the collision detection, setting a *boolean* variable with a constant *false* value when the value should be *true*. Thus, when there is collision, the algorithm indicates that there is no collision.

In the second faulty version, we changed the force feedback calculation algorithm, adding a multiplication factor (0.25) to the three values of an array, which stores the force in the three axes, calculated in the force feedback module, that must be sent to the device. This value was defined to modify the force feedback compared to the force data previously recorded.

The third version was not actually faulty. We used a collision software module from the Java3D library (using the *Bounding Box* method), different from the software module that was tested initially (using the *Octree* method) [40]. *Bounding Box* method envelops the 3D virtual objects in imaginary boxes, aligned to the coordinate axis. It uses the coordinate points that are farthest from the objects, to check the intersections between boxes, considering their centers and their sizes [41]. The *Bounding Box* accuracy is lower than that provided by the original module. There are no faults in the source code. However, the source code is different from the original, used in the *Record* step. A version with a different collision module was not created for the *game SUT* because it originally uses the *Bounding Box* method. The fourth version was the original SUT.

In the second set, we seeded faults in the SUTs of the case study by generating mutants with MuJava [42, 43]. We applied MuJava on the methods of the *Haptic* class (the same class in all three SUTs), using the following operators:

(A) Arithmetic operators: Arithmetic Operator Replacement (AOR_B) to replace basic binary arithmetic operators with other binary arithmetic operators ($+$, $-$, $*$, $/$, and $\%$.); and Arithmetic Operator Replacement (AOR_S) to replace short-cut arithmetic operators with other unary arithmetic operators ($op{+}{+}$, ${+}{+}op$, $op{-}{-}$, and ${-}{-}op$).

(B) Relational operator: Relational Operator Replacement (ROR) to replace relational operators ($>$, $>=$, $<$, $<=$, $==$ and $!=$) with other relational operators, as well as to replace the entire predicate with true and false.

(C) Logical operator: Logical Operator Replacement (LOR) to replace binary logical operators with other binary logical operators (&, | and !).

(D) Deletion operators: Statement DeLetion (SDL), which deletes each executable statement, but does not delete declarations - when this operator is applied to control structures that include a block of statements (if, while and for), the entire block is deleted, as well as each statement; Variable DeLetion (VDL), where all occurrences of variable references are deleted from every expression; Constant DeLetion (CDL),

where all occurrences of constant references are deleted from every expression. Likewise VDL, in CDL, when needed, operators are removed to preserve the compilation; and Operator DeLetion (ODL), where each arithmetic, relational, logical, bitwise and shift operator is deleted from expressions and assignment operators.

We chose these mutant operators to emulate possible faults in VR-based systems with haptic devices. These systems use arithmetic operations to calculate the force feedback and conditional commands to check force, position of the haptic device, the haptic loop, and the communication between the system and the device. Thus, we selected mutation operators to seed faults on these operations and structures. These are also well known operators, used in several previous experiments.

6. RESULTS

To assess our approach, we calculate the percentage of versions with manually seeded faults (first set cited in the previous section) that were detected by the test oracle. Regarding the mutants (second set), the ability of a test oracle for detecting the presence of faults is measured by the percentage of killed mutants in relation to the total of mutants, the *mutation score* or $ms(P,T)$, according to Equation 1 [44].

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)} * 100 \qquad (1)$$

where $DM(P,T)$ is the number of killed mutants considering the tests set T for the program P; $M(P)$ is the total number of generated mutants; and $EM(P)$ is the number of equivalent mutants generated for program P.

Thus, after applying our approach on both SUTs, following the steps to record and playback, the percentage of versions detected with manually seeded faults was 100%; and the mutation score was 93%.

Figure 15 shows an example of outputs presented to the tester, informing that a failure was revealed, and the number of the inference rule. The figure also shows the output or the force in the current frame during the execution in the three axes. In this example, the force in axis x is correct; the force in axis y is incorrect (the recorded and current values are shown; and the force in axis z is incorrect (the recorded and current values do not appear). A failure is revealed if a difference between the recorded and the current force value is found in one frame and in one axis.

During the *Playback* step, the tester could observe the behavior of the beam, of the needle and of the syringe in the VEs using a video monitor. Since information about force values is also shown on the video monitor, it is possible to visually check whether the force is correct according to the virtual object and whether there was collision between virtual objects.

For the first set of versions, Table 3 shows the

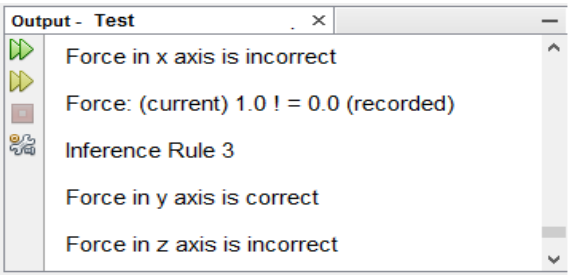

FIGURE 15. Visualization of failures revealed using a faulty version of a SUT and a test script.

results. The failures were expected in 150 executions (in the versions with modifications in the collision detection and force calculation algorithms), except in 125 executions (in the original versions and versions with another collision detection algorithm).

We detected the 150 executions with faults (failures revealed). However, other 25 cases with no faults were detected (failures revealed). This happened because of the different source codes for collision detection at each step of the R&P (different algorithms for collision detection), since the haptic feedback calculation is performed only when there is a collision between the object handled by user and other objects in the VE. We used a method implemented in the Java3D software library (*Bounding Box* method) for the collision detection in the *Playback* step, whose precision is inferior for irregular objects when compared to the *Octree* method, used in the *Record* step. The method from the Java3D library involves objects with transparent boxes to simplify the calculations. The *dentistry SUT* controls a VE with objects with irregular edges and the 25 executions presented failures. Due to these objects, collision was detected in different points from those points detected in the original version. Consequently, failures were revealed in relation to the *dentistry SUT* when they do not really exist.

Regarding the second set of versions of the SUTs, 400 mutants were generated, being nine equivalent mutants (manually observed), of which 362 were killed using three test scripts (one script for each SUT). Many mutants were timed out, because of infinite loops. The 29 cases that were not killed were manually analyzed, and it was observed that faults in the variable that controls the haptic loop were not detected using our approach. This variable allows to end the system when a problem in the haptic device during human-computer interaction is found. It should be pointed out that the haptic process encompasses, as mentioned: initializing communication between the software and the device, loop control (3D motion capture, collision checking, force calculation, and errors checking), and ending communication with the device. We believe that whether the values of the loop control variable were recorded in the test script during the human-computer

TABLE 3. Results - seeded faults manually.

SUT	Version	Executions number	Failures expected	Failures revealed
Primitive objects	Original	25	0	0
	Fault in the force calculation algorithm	25	25	25
	Fault in the collision detection algorithm	25	25	25
	Different collision detection algorithm	25	0	0
Dentistry	Original	25	0	0
	Fault in the force calculation algorithm	25	25	25
	Fault in the collision detection algorithm	25	25	25
	Different collision detection algorithm	**25**	**0**	**25**
Game	Original	25	0	0
	Fault in the force calculation algorithm	25	25	25
	Fault in the collision detection algorithm	25	25	25

interaction in the *Record* step, these mutants would be killed.

After the *Playback* step, the messages showed, and the inference rules helped identify, the faulty modules, especially in the collision detection module, except when the faults were in the variable that controls the haptic loop. Certain values attributed to this variable indicates problems and causes the interruption of the execution of haptic software.

Finally, the research questions were answered. Regarding RQ1, the application of the adapted R&P technique showed to be effective for automating the software testing activity in the haptic interface domain with several benefits. However, there are limitations that will be addressed in Section 7.

Regarding RQ2, there is evidence that the set of inference rules help identify the software modules where the faults causing failures are located: in the collision detection or in the force feedback calculation module, according to the source code analysis after revealing failures.

7. DISCUSSION

Automating the software testing activity is important to ensure productivity and quality of the software product. However, certain domains of applications have characteristics that hamper performing this task.

In this context, VR-based systems, in particular those with haptic interfaces, are still a challenge for testers. In the literature few studies have shown convincing results in this domain. Haptic devices are still considered non-conventional and expensive equipment. The communication is bidirectional, capturing information (position, rotation and velocity) and providing haptic feedback (force feedback). Along with the characteristics of haptic devices, VR-based systems have to address the visual feedback and different frequencies or frame update rates during human-computer interaction. Moreover, in these interactive systems, frame update rates can vary during the execution, which is an additional aspect that makes the testing activity difficult. Additionally, as mentioned before, experts are who usually test the VR-based systems, but the reproduction of their actions, in order to test these systems with the same data is difficult, even when it is performed with the same user. Experts also determine the correct force feedback or haptic sensation. These considerations evidence how complex is to test VR-based systems with haptic interfaces.

R&P techniques have been applied in the complex interfaces, but their application in VR-based systems with haptic interactions has never been explored in the literature, probably because it requires non-trivial adaptations, which requires extensive knowledge of the domain. We adapted the R&P technique and used it along with AOP and inference rules to, respectively, automate VR-based systems testing with haptic interfaces, reveal failures, and locate faulty modules in these systems.

We answered RQ1 and RQ2 by verifying that R&P can be effective for automating the software testing activity in the haptic interfaces domain, and that inference rules help the tester to locate the faulty module that gives origin to certain failures. There are benefits and limitations in our approach. We used position in three axes of a virtual object and force in three axes to compose the scripts, reducing computational resource consumption in the application of the R&P technique. However, we noted that it is important to include the information about the haptic loop control variable, verifying whether the haptic device is working correctly, since those mutants not killed affected the variable that controlled this loop.

A limitation is related to the users' profile. Experts in the field where the VR-based system will be used should define the main trajectories the device should go through and the correct haptic sensations. Thus, test scripts relevant for testing could be specified. This creates a dependence on expert's participation; though the experts are interested in reliable simulators and can help in the development.

Another limitation is that there is the need to check VE settings (objects features, such as scale, position and rotation), indicating whether they are similar to

the software used in the *Record* step. Such differences in settings may change the outputs of the software being tested; though they are not faults. For example, a trajectory that causes a collision of two virtual objects with a S scale, when it is reproduced in a VE where the same objects have a $0.5S$ scale, may not collide because the objects are smaller and the trajectory previously recorded is not enough to cause the collision. These features of the virtual objects, as well as stiffness and viscosity were not recorded in the test scripts, but they can be added.

The VE settings may also include the shape of virtual objects (vertices and polygons). The tester must know the memory variables that store settings values to check the test conditions. Thus, the virtual objects must have the same features at the beginning of the *Record* and *Playback* steps. For instance, in the case study almost all objects in the VE are static (they have the same features – position, rotation and scale), only the virtual beam, needle and syringe (*primitive objects SUT*, *dentistry SUT* and *game SUT*, respectively) are moved (the position is changed during human-computer interaction) using the haptic device. However, these features can be recorded in the *Record* step and compared in the *Playback* step to inform these differences to the tester.

With regards to time consumption, it can be a problem in the observation, since additional commands in the source code or interceptions could cause delays, a situation that did not happen and which is undesired in real time systems. This is solved including delays during *Playback* step for waiting certain commands.

Regarding the inference rules, they are related to collision detection and force feedback calculation software modules; though, these rules can be applied in most VR-based systems with haptic devices. These rules must be improved to consider the control of the haptic loop and setup conditions of the VE, according to case study (tests using mutation and different settings of the VE).

Although limitations are perceived, there are the following advantages. R&P techniques-based tools are very useful because they prevent the tester from manually repeating the same sequence of actions on the interface. In addition, using the outputs generated by the original execution as a test oracle, they make possible automatically check if the program behaves as expected.

AOP, in turn, prevents the direct instrumenting of the original source code, contributing to its use in practice. If the signatures of the methods are known, it is possible to test the system even if the source code is not available.

An effective benefit is that the tester can perform automated tests even in the absence of the haptic device, which is considered a non-conventional and expensive device. This benefit is a contribution to the practice, since the practitioners usually test the VR-based systems with haptic interfaces by manually repeating the trajectories with the physical device. Our approach also contributes to ensure the exact repetition of the input data during the execution of a trajectory, reaching the same structures in the VE, and using the same speed in the movement. For a human tester, this is an almost impracticable and complex task. Different speed values used in the haptic device manipulation can generate different force feedback, hampering the feedback analysis. Therefore, the tester should apply the same velocity to analyze the correct force feedback as well as to reach the same object. Thus, our approach allows testing new versions of the VR-based system with the same previously used data, whose respective correct outputs are known, even without the physical presence of a haptic device.

Finally, our approach can be used to identify lack of force feedback or inadequate force feedback values. This is possible because the inference rules can be customized to reveal failure in some intervals of force values and of frames, which could be pre-defined by the tester.

8. CONCLUSION

Haptic interfaces are deemed non-conventional devices and allow a bidirectional communication during human-computer interaction. Haptic devices are able to capture motions in the real space and provide tactile or force feedback. These interfaces are employed in VR-based systems, mainly for training in several areas.

We investigated and adapted the use of the *Record* and *Playback* (R&P) technique for haptic interfaces domain, composing an approach that puts together characteristics of the R&P and assumptions that are intrinsic to systems that use haptic devices in order to automate the test oracles in this domain. Sets of versions, including faulty versions of three SUTs or VR-based systems with haptic interfaces and AOP were used to assess the approach. One hundred percent of faulty versions was detected and a mutation score of 93% was achieved using the new approach. Thus, the case study showed that R&P and AOP can be applied to automate VR-based systems with haptic interfaces testing, and simple inference rules can help locate faulty modules.

We worked with important haptic information (position and force in the three axes), but additional information can be added such as rotation. Furthermore, most of the objects in the VE (e.g., virtual anatomical objects) were static during *Record* and *Playback* steps, a scenario that can change depending on the objective of the simulation and that can cause problems in applying R&P because the trajectories previously recorded may cause collision and haptic feedback at undesired moments.

As future work we plan to apply our R&P technique in other systems with haptic interfaces to corroborate the results found in our current case

study. Additionally, we intend to apply machine learning techniques to analyze the context, mainly when the virtual objects features (position, scale and shape) change between *Record* and *Playback* steps. An analysis of the inference rules with several testers and the use of metamorphic testing are also planned. Many VR-based systems with haptic interfaces are implemented in parallel or concurrent mode, generating different outputs for the same inputs. Another line of research is to extend our approach to tackle these non-deterministic cases; as well as to improve the inference rules.

Another future work is to study whether a threshold of the recorded information can be useful to compare force values in the R&P technique in a certain set of frames, since the execution frequency can change, depending on computer resources, causing delays in the force feedback calculation, for example. Small differences of force feedback can be observed when recorded and actual values are compared, and for some applications they do not represent a real error. So, it is interesting to analyze which differences in such values are perceptible or not and should or not be considered as failures in the *Playback* step [5]. Studies are also needed to analyze the influence of faults in other modules besides the two (collision detection and haptic calculation).

Thus, this paper proposes the first approach to automatically test the complex domain of systems with haptic interfaces in a systematic manner. Nowadays, applications in this domain are tested by developers or users in an ad hoc manner, without maintaining similar conditions of input data and actions. Our approach can be a contribution to effectively carry out automated software testing in this domain.

9. FUNDING

This work was supported by the Post-Doctoral National Program of the Brazilian National Council for the Improvement of Higher Education Personnel (PNPD–CAPES), School of Arts, Sciences and Humanities - Graduate Committee [EACH/CPG number 88/2015 to C.G.C.]; the National Institute of Science and Technology - Medicine Assisted by Scientific Computing (INCT–MACC) [Second step - 2016–2021]; and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) [process number 308615/2018-2].

REFERENCES

[1] Bowman, D. A., Kruijff, E., Joseph J. Laviola, J., and Poupyrev, I. (2005) *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

[2] Salisbury, J. K. (1999) Making graphics physically tangible. *Communications of the ACM*, **42**, 74–81.

[3] Corrêa, C. G., Machado, M. A. d. A. M., Ranzini, E., Tori, R., and Nunes, F. d. L. S. (2017) Virtual Reality simulator for dental anesthesia training in the inferior alveolar nerve block. *Journal of Applied Oral Science*, **25**, 357 – 366.

[4] Corrêa, C. G., Nunes, F. L. S., and Tori, R. (2014) Virtual reality-based system for training in dental anesthesia. *Proceedings of the 16th International Conference on Human-Computer Interaction (HCII)*, Creta Maris, Heraklion, Crete, Greece, 21 - 27 June, pp. 267–276. Springer.

[5] Corrêa, C. G., Tokunaga, D. M., Ranzini, E., Nunes, F. L. S., and Tori, R. (2016) Haptic interaction objective evaluation in needle insertion task simulation. *Proceedings of the ACM 31st Symposium on Applied Computing (SAC)*, Pisa, Italy, 3 - 8 April, pp. 149–154. ACM.

[6] Nguyen, B. N., Robbins, B., Banerjee, I., and Memon, A. (2014) Guitar: An innovative tool for automated testing of GUI-driven software. *Automated Software Engg.*, **21**, 65–105.

[7] Adamoli, A., Zaparanuks, D., Jovic, M., and Hauswirth, M. (2011) Automated GUI performance testing. *Software Quality Journal*, **19**, 801–839.

[8] Corrêa, C. G., Nunes, F. L., Ranzini, E., Nakamura, R., and Tori, R. (2019) Haptic interaction for needle insertion training in medical applications: The state-of-the-art. *Medical Engineering & Physics*, **63**, 6 – 25.

[9] Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., and Yoo, S. (2015) The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, **41**, 507–525.

[10] Ammann, P. and Offutt, J. (2016) *Introduction to Software Testing*. Cambridge University Press, Cambridge.

[11] Li, N. and Offutt, J. (2017) Test oracle strategies for model-based testing. *IEEE Transactions on Software Engineering*, **43**, 372–395.

[12] Coles, T. R., Meglan, D., and John, N. W. (2011) The role of haptics in medical training simulators: A survey of the state of the art. *IEEE Transactions on Haptics*, **4**, 51–66.

[13] Sithu, M., Ishibashi, Y., Huang, P., and Fukushima, N. (2015) Qoe assessment of operability and fairness for soft objects in networked real-time game with haptic sense. *Proceedings of the 21st Asia-Pacific Conference on Communications (APCC)*, Kyoto, Japan, Oct, pp. 570–574. IEEE Computer Society.

[14] Wong, C. Y., Chu, K., Khong, C. W., and Lim, T. Y. (2010) Evaluating playability on haptic user interface for mobile gaming. *Proceedings of the International Symposium on Information Technology*, Kuala Lumpur, Malaysia, 15 - 17 June, pp. 1093–1098. IEEE Computer Society.

[15] Burdea, G. C. and Coiffet, P. (2003) *Virtual Reality Technology*, 2 edition. John Wiley & Sons, Inc., New York, NY, USA.

[16] Okamura, A. M. (2009) Haptic feedback in robot-assisted minimally invasive surgery. *Current opinion in urology*, **19**, 102–107.

[17] Fortmeier, D., Wilms, M., Mastmeyer, A., and Handels, H. (2015) Direct visuo-haptic 4d volume rendering using respiratory motion models. *IEEE Transactions on Haptics*, **8**, 371–383.

[18] Goksel, O., Sapchuk, K., and Salcudean, S. (2011) Haptic simulation of needle and probe interaction with tissue for prostate brachytherapy training. *Proceedings of the IEEE World Haptics Conference (WHC)*, Istanbul, Turkey, June, pp. 7–12. IEEE Computer Society.

[19] Sutherland, C., Hashtrudi-Zaad, K., Abolmaesumi, P., and Mousavi, P. (2011) Towards an augmented ultrasound guided spinal needle insertion system. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Boston, USA, Aug, pp. 3459–3462. IEEE Computer Society.

[20] Dimaio, S. P. and Salcudean, S. E. (2005) Interactive simulation of needle insertion models. *IEEE Transactions on Biomedical Engineering*, **52**, 1167–1179.

[21] Choi, K.-S., Chan, S.-H., and Pang, W.-M. (2012) Virtual suturing simulation based on commodity physics engine for medical learning. *Journal of Medical Systems*, **36**, 1781–1793. cited By (since 1996)2.

[22] Caswell, M., Aravamudhan, V., and Wilson, K. (2004). Introduction to jfcUnit. Available at http://jfcunit.sourceforge.net/. Accessed on 21 dec. 2017.

[23] Jameel, T., Lin, M., and Chao, L. (2015) Test oracles based on metamorphic relations for image processing applications. *Proceedings of the 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Takamatsu, Japan, 1 -3 June, pp. 1–6. IEEE Computer Society.

[24] Delamaro, M. E., Nunes, F. L. S., and Oliveira, R. A. P. (2013) Using concepts of content-based image retrieval to implement graphical testing oracles. *Software Testing, Verification and Reliability*, **23**, 171–198.

[25] Offutt, J., Papadimitriou, V., and Praphamontripong, U. (2014) A case study on bypass testing of web applications. *Empirical Engineering*, **19**, 69–104.

[26] Zhou, Z. Q., Zhang, S., Hagenbuchner, M., Tse, T. H., Kuo, F.-C., and Chen, T. Y. (2010) Automated functional testing of online search services. *Software Testing, Verification and Reliability*, **22**, 221–243.

[27] Hammoudi, M., Rothermel, G., and Tonella, P. (2016) Why do record/replay tests of web applications break? *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Chicago, IL, USA, 11 - 15 April, pp. 180–190. IEEE Computer Society.

[28] Sprenkle, S., Gibson, E., Sampath, S., and Pollock, L. L. (2005) Automated replay and failure detection for web applications. *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, Long Beach, CA, USA, 7 - 11 November, pp. 253–262. ACM.

[29] Bezerra, A., Delamaro, M. E., and Nunes, F. L. S. (2011) Definition of test criteria based on the scene graph for VR applications. *Proceedings of the XIII Symposium on Virtual Reality*, Uberlândia, MG, Brazil, May, pp. 56–65. Computer Brazilian Society.

[30] Souza, A. C. C., Nunes, F. L. S., and Delamaro, M. E. (2018) An automated functional testing approach for virtual reality applications. *Software Testing, Verification and Reliability*, **28**, 1–31.

[31] de Souza, B. F. R. (2013) Kina: an enhanced development model and toolkit for kinect applications. Master's thesis. Universidade Federal de Pernambuco (UFPE).

[32] Bierbaum, A., Hartling, P., and Cruz-Neira, C. (2003) Automated testing of virtual reality application interfaces. *Proceedings of the Workshop on Virtual Environments (EGVE)*, Zurich, Switzerland, 22 - 23 May, pp. 107–114. ACM.

[33] Hunt, C. J., Brown, G., and Fraser, G. (2014) Automatic testing of natural user interfaces. *Proceedings of the IEEE Seventh International Conference on Software Testing, Verification and Validation*, Cleveland, OH, USA, March, pp. 123–132. IEEE Computer Society.

[34] White, T. D., Fraser, G., and Brown, G. J. (2018) Modelling hand gestures to test leap motion controlled applications. *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Vasteras, Sweden, April, pp. 204–213. IEEE Computer Society.

[35] Fewster, M. and Graham, D. (1999) *Software Test Automation: Effective Use of Test Execution Tools*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[36] Corrêa, C. G., Nunes, F. L. S., and Tori, R. (2013) Haptic simulation for virtual training in application of dental anesthesia. *Proceedings of the XV Symposium on Virtual and Augmented Reality (SVR)*, Cuiabá, MT, Brazil, 27 - 30 May, pp. 63–72. Computer Brazilian Society.

[37] Torres, R. S., Bíscaro, H. H., Araújo, L. V., and Nunes, F. L. S. (2013) ViMeTGame: A serious game for virtual medical training of breast biopsy. *SBC Journal on 3D Interactive Systems*, **3**, 12–22.

[38] 3DSystems (2017). Touch. Available at https://br.3dsystems.com/haptics-devices/touch. Accessed on 21 dec. 2017.

[39] Kiselev, I. (2002) *Aspect-Oriented Programming with AspectJ*. Sams, Indianapolis, IN, USA.

[40] Hearn, D. and Baker, M. (1997) *Computer Graphics, C Version*, 2nd edition edition. Prentice Hall, Upper Saddle River, NJ, USA.

[41] Klosowski, J. T., Held, M., Mitchell, J. S. B., Sowizral, H., and Zikan, K. (1998) Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, **4**, 21–36.

[42] Offut, J. and Li, N. (2013). mujava home page. Available at https://cs.gmu.edu/ offutt/mujava/. Accessed on 27 dec. 2018.

[43] Ma, Y.-S., Offutt, J., and Kwon, Y. R. (2005) Mujava: An automated class mutation system. *Softw. Test. Verif. Reliab.*, **15**, 97–133.

[44] DeMillo, R. A. (1980) Mutation analysis as a tool for software quality assurance. Technical report. Georgia Institute of Technology, School of Information and Computer Science.