Modelling Hand Gestures to Test Leap Motion Controlled Applications

Thomas D. White
Department of Computer Science
The University of Sheffield
Sheffield, South Yorkshire
tdwhite1@sheffield.ac.uk

Gordon Fraser
Chair of Software Engineering II
University of Passau
Passau, Germany
gordon.fraser@uni-passau.de

Guy J. Brown
Department of Computer Science
The University of Sheffield
Sheffield, South Yorkshire
g.j.brown@sheffield.ac.uk

***Abstract*—Programs that use a Natural User Interface (NUI) are not controlled with a mouse and keyboard, but through input devices that monitor the user's body movements. Manually testing applications through such interfaces is time-consuming. Generating realistic test data automatically is also challenging, because the input is a complex data structure that represents real body structures and movements. Previously, it has been shown that models learned from user interactions can be used to generate tests for NUI applications controlled by the Microsoft Kinect. In this paper, we study the case of the Leap Motion input device, which allows applications to be controlled with hand movements and finger positions, resulting in substantially more complex input data structures. We present a framework to model human hand data interacting with applications, and generate test data automatically from these models. We also evaluate the influence of the training data, as well as the influence of using a single model of the complete user data vs. multiple models for the different aspects of hand movement (e.g., finger positions, hand positions, hand rotations). Experiments on five applications controlled by the Leap Motion demonstrate that our approach generates effective test data. The quality and quantity of the training data used to derive the models is the main factor that determines their effectiveness. On the other hand, the effects of using multiple (as opposed to single) models are minor and application specific.**

I. Introduction

Natural User Interfaces (NUIs) allow users to interact with software through methods such as body tracking, gestures, or touch interfaces [19]. They are becoming increasingly popular for virtual reality applications, and are important ways of interacting with computers in environments where using a keyboard and mouse is not an option (e.g., surgeons in the operating theatre). Testing applications controlled by NUIs, however, is a challenge: there are currently no frameworks for test automation, meaning that developers need to manually exercise all functionality by physically interacting with the NUI on every update of the software.

When testing manually is challenging, automation is often desired. For example, there are tools that can automatically generate tests for many different types of applications [4], [12], [14], relieving the tester of the manual effort of designing test cases. While tests for standard software typically consist of API calls or simple interactions with GUI elements (e.g., button clicks), the inputs expected by NUI applications are much more challenging to produce automatically. For example, the input for an application controlled by a Microsoft Kinect input device consists of a collection of points in 3D space which collectively represent the body of the program's user; the input for an application controlled by a Leap Motion input device consists of data representing the user's hand and finger joint positions in 3D space. The challenge for automated test generators lies in producing data that represents valid body or hand positions.

One way to produce such input data automatically is to learn models of realistic user input, and then to sample these models for new sequences of input. The feasibility of this approach has previously been demonstrated using the Microsoft Kinect [8]. However, previous work focused only on one aspect of the Microsoft Kinect input, the body joint positions. In this paper we study the Leap Motion controller, which measures hand positions, finger positions, finger gestures, and various other aspects, which collectively create a substantially more complex test generation problem. We present a framework to apply NUI test generation to applications based on this NUI controller, and evaluate it on five different Leap Motion applications.

Using only a single data model to capture all the different aspects of the NUI input data may not be the most effective approach. For example, in the Leap Motion Controller the hand movement and finger joints shape are encoded together, but training one model on the combined result eliminates the possibility of identifying similar finger joint shapes at different positions in 3-D space. In order to determine whether representing the complex NUI data with multiple models is beneficial, we present a methodology in which we split the NUI data into subsets, and learn separate models for each subset. In our experiments we contrast test data generated from these *multiple models* with data generated from a *single model* of the input data.

In detail, the contributions of this paper are as follows:

- A framework to model hand interactions, and automatically generate and replay test cases for the Leap Motion NUI.
- An empirical evaluation of NUI testing on five applications controlled by the Leap Motion controller.
- An empirical evaluation of the influence of the training data on the resulting code coverage.

- An empirical comparison of generating NUI data from multiple models vs. a single model.

Our experiments show that our approach to automated NUI testing can handle the complexity of the Leap Motion controller well, and produces sequences of data that achieve significantly higher code coverage than random test generation approaches. We show that the training data has a large influence on these results, while the benefits of splitting NUI data into multiple models are small and very application dependent.

II. Background

A. Natural User Interfaces

Natural User Interfaces (NUIs) provide a means of controlling software by recording a continuous stream of data that represents the position or motion of the user's body. For example, the Microsoft Kinect employs a depth camera that allows a user to interact with an application through body tracking. Similarly, the Leap Motion Controller is a small desktop device which tracks the hand and finger positions of the user, thus providing a natural interface in which the user can point, draw or gesture with their hands.

NUIs have been used to solve a range of problems. For example, The Microsoft Kinect has been used in medicine for effective stroke rehabilitation, giving doctors access to the body profile of patients from a patient's own home, and making exercises more fun and motivating for patients [20].

NUIs rely on a user's existing knowledge for interactions with applications: if a menu is on the screen, it is intuitive to reach out and touch a desired button for progression through an application. Although intuitive for real users, NUIs are difficult to test with an automated approach.

B. Automated Test Generation

Software tests can be generated automatically to support developers and testers. A common approach is to generate test cases randomly, for example by randomly sampling numerical data [9], random character sequences [13] or by generating random sequences of API calls [14]. Although this approach can also be directly applied to NUI applications, it is very unlikely that randomly generated data will resemble realistic NUI data. For example, a random set of points in 3-D space is unlikely to match the physical constraints of a real hand.

In order to produce data that is closer to real operational data, it is possible to use previous knowledge or sample from specific distributions representing the real data, rather than sampling uniformly. For example, Whittaker and Poore show how exploiting actual user sequences of actions taken from an application specification can be used when creating structurally complete test sequences [17], representing a path from an uninvoked application state to a terminal application state. For this, a Markov chain was used where each state of the chain represents a value from the application's input domain. Further, Whittaker and Thomason [18] generated Markov chain usage models, with values from the expected function, usage patterns, or previous program versions. The models then generated tests that are statistically similar to the operational profile of the program under test. Walton et al. [16] demonstrated that usage models are a cost effective method of testing. Usage data from NUI applications can also be converted into a usage model, and used to generate tests for the application.

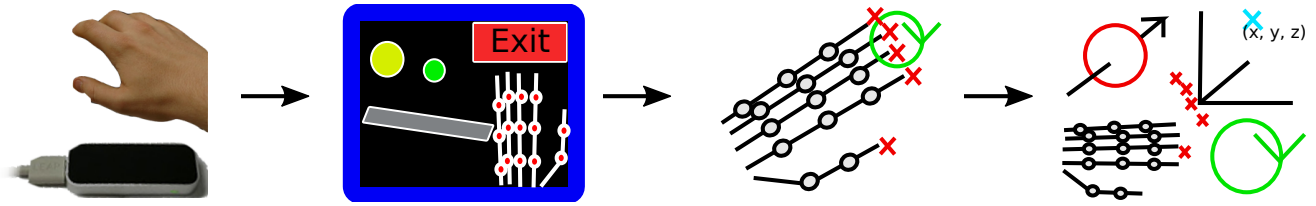
Programs with traditional graphical user interfaces can be tested by sending random events [3] (e.g., random mouse clicks and keyboard events), but effective testing may require more than blindly clicking on apps. Therefore, many GUI testing techniques assume either the availability of information about the concrete set of widgets available for interaction, or a manually constructed model of the GUI. Testing then becomes a matter of choosing from the available widgets to interact with or exploring the model to create test sequences. It is also possible to apply user profiles in terms of probabilistic usage models of an application [2] to generate tests that are statistically similar to real user interactions. While this is an effective approach, it is not applicable to NUI testing: for NUI applications there are no discrete sets of widgets to choose from. Rather, applications just take vectors of numbers representing NUI data structures as input.

C. Natural User Interface Testing

Mobile applications use combinations of regular program inputs (e.g., via touch displays), and NUI inputs (e.g., via external sensory data). To test mobile applications, Griebe et al. describe a framework in which location information [6] and accelerometer data [7] can be replaced with mocked data by developers.

Hunt et al. automatically generated test sequences for the Microsoft Kinect [8]. To generate data, Hunt et al. trained models on data recorded by users. Similarities in the data are identified through clustering, and sequences of clustered data are used to generate a Markov chain. The Markov chain is a probabilistic model that can be used to decide which cluster to seed next during test generation. Clustering was performed on all features of the data structure but this assumes that all features have a static (time-unconstrained) relationship.

Hunt et al. used branch coverage to assess the effectiveness of different NUI data generation methods. The application under test (AUT) was a web browser adapted for Kinect support. Hunt et al. found that using a purely random approach for generation, i.e. using randomly sampled values for each variable in the data structure, performed the worst. Second was an approach involving seeding randomly sampled processed user data. To increase performance further, Hunt et al. generated an N-gram model from the sequence information collected when recording data and used this model during data generation. A single model of NUI data may link different independent aspects of the user movement, resulting in biased test generation. For example, if a NUI was to capture a person running, the body gestures observed may be a repeated sequence of body positions, but the actual data will never be repeated as running displaces the body in 3-D space. Potentially, representing the body location and the joint movement as separate models could be a more effective means of generating new, realistic test input not observed in the training data. New, realistic data will still resemble the



(a) User interacting with the Leap Motion Controller

(b) Application receiving data from the Leap Motion Controller

(c) Data from the Leap Motion Controller saved for model generation.

(d) Feature selection applied to data to create manually identified data subsets.

Fig. 1. Recording user interactions with the Leap Motion Controller and splitting data into subsets.

body part being tracked and also moves smoothly through time. This approach may be particularly important in the case of NUIs where many potentially independent features are present in the common input data structure.

D. Leap Motion Controller

The Leap Motion Controller is a NUI which tracks a user's hand movements and gestures. The device is placed on a desk, and users place hands above the device to interact with software. The Controller tracks properties of a hand such as position in 3-D space, the location of all joints in each finger, the position of the tips of each finger and many other things. Each data frame received from the Leap Motion Controller contains a snapshot of the user's hands at the current time, providing data up to 200 times per second. Because applications expect data at this rate, it is important that testing techniques can match this speed, whilst generating realistic data. The Leap API gives applications a complex relational data structure. The top level of the structure is a Frame, which contains all relevant information observed by the Leap at the current time. However, some aspects of the structure are not only reliant on the current time of capture. For example, to interact with 2-D applications, a developer replaces the cursor with the *Tip Positions* of each finger. However, there is also a *Stabilized Tip Position* for each finger which returns a smoothed version of the fingers tip position, directed at 2-D application interaction, and updates according to the speed which the finger tip was moving. Stabilized positions allow for more consistent 2-D GUI interactions, specifically with micro movements, but how the values are calculated does not appear in the API documentation.

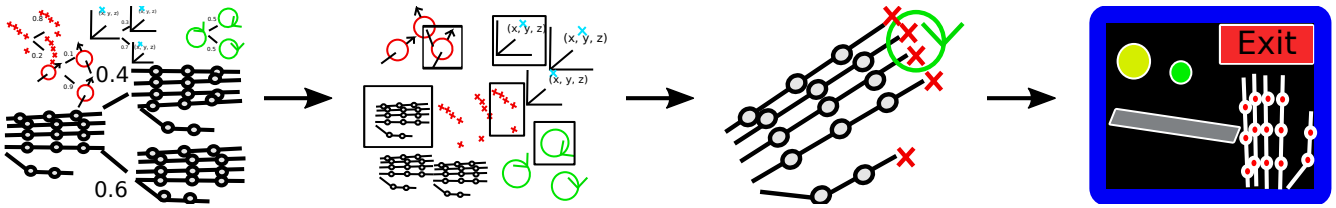
III. Modelling Leap Motion Data

We split the Leap Motion data into 5 parts, where each part is modeled using an N-gram model. An N-gram model represents the probabilities of one element following the N previous elements in a sequence of data. Using such models has distinct advantages and disadvantages in testing compared to manual testing by users. Once created, a model is more cost-effective than manual testing, and can generate long sequences of test data without tiring. However, a model is only as good as the data used to train it, and may not generalize to novel kinds of interaction that were not encountered during training. This may limit the extent to which a NUI application can be explored by model-based test generation.

This section shows how user data from the Leap Motion is split into five separate models, each model representing a unique aspect of the Leap Motion data structure. The five models are as follows:

- Position: The 3-D position of the palm, relative to vector (0, 0, 0) in Leap Motion Controller space. This is the physical position of the hand in 3-D space. Position data is denoted in Figure 1 by three axes and a point labelled with (x, y, z).
- Rotation: The rotation of the palm, stored as Euler angles by the Leap Motion, we convert to quaternions for modelling. A quaternion is a 4-D unit vector that represents an object's rotation in 3-D space. Rotational data is denoted in Figure 1 by a circle with an arrow through, representing the quaternion angle of rotation.
- Joints: The 3-D position of each bone joint in the fingers of each hand, respective to the palm position. All fingers were stored in the same feature to preserve anatomical constraints between fingers. Joints are denoted in Figure 1 by the circles on the fingers of hands.
- Gestures: The sequence of pre-defined Leap Motion gesture types performed by the user (Circle, Swipe, Key tap and Screen tap). This is also split into four child models, one per gesture type. For the applications tested here, only the circle gesture is used, which triggers when a *Finger* performs a circular motion. A circle gesture consists of circle center, normal, radius and the duration that the gesture has been performed for. Circle gestures are denoted in Figure 1 by a green circle with an arrow.
- Stabilized Positions: Each hand also has stabilized data, which are vectors targeted towards 2-D menu interactions and rely on time. One example are stabilized tip positions for the tips of each finger, being a variable amount of time behind the actual hand data. Stabilized positions are stored in a separate model to preserve 2-D interactions. Stabilized data are denoted in Figure 1 by red "X"s representing the stabilized tips of each finger.

For the five models defined, we use user data to train each model. Figure 1 shows how user data is stored as separate data subsets, one subset per model. User Recording is the process of capturing user interaction with the Leap Motion and hence the application under test. We intercept this data and use it to train models. First, the data is split into data subsets. This



(a) Model generated from data subsets (Figure 1).

(b) Each model outputs a cluster centroid, all output is combined into a single data object

(c) Generated Leap Motion data object to seed to application under test

(d) Application receiving generated data

Fig. 2. Our approach: generating features before combining them into a realistic data object.

involves applying feature selection [15] to the data and training each model on the selected features.

A. Model Generation

For each data subset, the same technique is applied to generate models. Firstly, the volume of user data is reduced using K-means clustering. K-means clustering groups together related records by Euclidean distance, using all features in the calculation. The result labels each record with a cluster $0..k$ where the label is the cluster with the nearest centroid (mean of all elements in the respective cluster).

Each record is now labelled but the quantity of data has not changed. To reduce the data, we substitute each record with the centroid of the assigned cluster. This reduces the total amount of user data to K centroids.

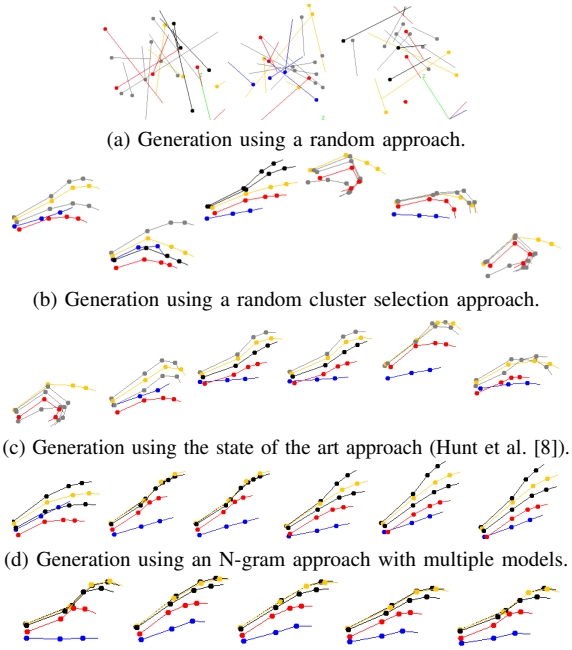
When recording data, the chronological sequence in which each record was received is stored. This sequence can be replaced by the assigned cluster labels and used to train an N-gram model, a model containing the probabilities of all transitions of length N in a sequence.

To generate an N-gram model, a probability tree is constructed from sequences of data. The tree is of depth N and contains all transitions of length N from one element of the sequence to other neighbouring elements. For example, assuming that the cluster label sequence is as follows: $1, 2, 1, 2, 1, 3$. Using this model with $N = 2$, the probability of observing a record in cluster 2 following a record in cluster 1 is $2/3$, and the probability of observing 1 after observing 2 is 1.0. Values of K and N were chosen through parameter tuning.

Each Leap Motion data frame can have an undefined amount of gestures, linked to different fingers i.e. it is possible for a single Leap Motion frame to have three circle gestures and a swipe gesture. We use an additional N-gram model to decide which gestures go in which frames. Specifically, the gesture N-gram model gives the following information: when to start and stop a gesture; which finger each gesture should be linked to; and the types of each gesture.

IV. Generating Leap Motion Data

Once models are trained, Figure 2 shows how data produced from each model can be recombined into valid Leap Motion data. Each model produces a cluster centroid for the area of the Leap Motion data structure the model is representing. The centroids from all models are combined into one data object,



(a) Generation using a random approach.

(b) Generation using a random cluster selection approach.

(c) Generation using the state of the art approach (Hunt et al. [8]).

(d) Generation using an N-gram approach with multiple models.

(e) A sequence from the original user data.

Fig. 3. Sequences of hands generated as input data using different techniques

and seeded back to the application during test generation. This is where our approach differs from the technique by Hunt et al. [8], which only uses a single model to reconstruct data. Our approach to testing NUI applications has the advantage that generated data still resembles the original user data, but is also diverse enough to test parts of a program not necessarily tested by users. We identify common patterns in the user data per model, retaining some relationships that would otherwise be lost when using a single model.

Using the models generated, we propose three methods of generating mock Leap Motion data:

1) Random: Sample numbers in the Leap Motion's view range domain for all properties of all features. See Figure 3a for an example sequence of hands generated using this approach.
2) Random Clusters: Randomly select a cluster for all models and seed the centroid of the clusters. This produces realistic Leap Motion data at a single point in time, but not over time. All time-related data is discarded producing 'mechanical' hands with no animation. See

Figure 3b for an example sequence of hands generated using this approach.

3) N-gram Model Generation: Use the generated N-gram models to select the next cluster centroid to seed. This preserves time-related data, but using separate models eliminates the static relationships between models preserved in the single model method by Hunt et al. However, the benefit is that a wider range of data can be produced, e.g., a single hand shape at various positions in the Controller's 3-D space can be generated, including positions that the user did not provide for the respective hand shape. See Figure 3c for example data generated using a single model (Hunt et al.) or Figure 3d for an example using multiple models.

Each technique reconstructs hands using the following method: 1) generate model data in isolation by selecting cluster centroids; 2) combine generated features into single data objects. Using an N-gram model produces a sequence of data that are statistically similar to the order of hands seen during user recording. In contrast, selecting random clusters produces more uniform data.

A. Executing Leap Motion Tests

Our technique generates tests for applications which use the Leap Motion Controller [10]. The Controller allows interaction with applications through hand tracking. The Leap API supports many target source code languages, and works through a background service installed on a machine, which provides a continuous stream of data to applications registered as listeners.

Our framework functions as a layer that sits between the application under test and the Leap Motion background service, replacing the Leap Motion's stream of data with automatically generated data. We use a full mock of the Leap Motion's Java API. During test generation, when applications register as a listener for the Leap Motion, our framework now provides a stream of data in place of the Leap Motion background service.

To save tests, we store the ordered cluster labels of each model and the execution time which the generated data frame was seeded to the AUT. Replaying a test involves using these stored cluster labels to select the cluster centroids for all models at the appropriate point in time, before combining all centroids into a data frame. Currently, playback of tests takes the same time as generation, but future work is to minimize the generated tests by removing sub-sequences which have no impact on final code coverage. Tests produced by our tool currently produce sequences of hands that can be played back into an application. This is useful for regression testing: ensuring that the current program state after seeding data on the modified application is equal to the state seen during generation.

V. Evaluation

To study NUI testing on the Leap Motion in more detail, we investigated the following research questions:

- RQ1: How well does NUI testing with N-gram models work on Leap Motion apps?
- RQ2: How does the quantity of training data influence the effectiveness of NUI testing?
- RQ3: How does separation of NUI data into multiple models influence the effectiveness of NUI testing?

A. Experimental Setup

To answer RQ1, we compare the test generation techniques outlined in Section IV, i.e., random test data, random clusters, and N-gram based test generation. For the Microsoft Kinect, Hunt et al. [8] observed that the use of an N-gram model resulted in substantial code coverage increases over the random baselines, and the main question is whether this effect can also be observed on Leap Motion applications, where input data is more complex than on the Microsoft Kinect.

To answer RQ2, we compare models created using only a single user's data, against models created using data from many users. Intuitively, assuming an equal value of K when clustering, using data from many users should lead to N-gram models which are less sparse, and have a higher diversity in the set of centroids. However, anatomical differences (e.g., different hand sizes) could have unexpected effects in the clustering process. To evaluate the effect of user data in test generation, we use the N-gram model technique with multiple models.

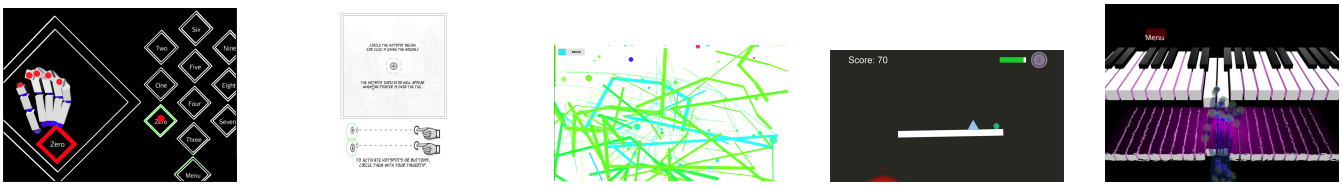
To answer RQ3, we evaluate the effects of splitting the Leap Motion data structure into multiple models. The baseline is the approach outlined by Hunt et al. [8] for the Microsoft Kinect, i.e., creating a single model with the complete Leap Motion data structure interpreted as a flattened vector of features. To evaluate the effectiveness of splitting Leap Motion data into multiple models, we use the N-gram model generation technique with models trained on data from many users for each application.

Our metric for comparison is line coverage; the amount of lines executed in an application divided by the total lines of the application. We measured line coverage using instrumentation provided by an open source tool[1]. To test for significance, we used a Wilcoxon rank-sum test, with a significant result occurring when $p < 0.05$. To find the better approach, we use a Vargha-Delaney $\hat{A}_{12}$ effect size, with a value trending towards 1 indicating an improvement over the baseline, 0.5 being no improvement and trending towards zero being a negative impact. To account for the random nature of our generation techniques, we ran each configuration for 30 iterations [1], and the code coverage achieved at the end of a one hour period was used in comparisons.

A one-hour generation time was selected due to coverage increases still being attained at this point for certain applications. To select the value of K for clustering, tests were generated using predefined sets of clusters between $200 - 2000$ and the value of K achieving the highest coverage was used in experiments. The value of N was tuned in the same manner but for values between 2 and 4. As data is recorded separate for each application, values are also tuned separate.

For evaluation, we chose five applications: four from the Leap Motion Airspace Apps Marketplace, and one open source application.

[1] https://github.com/thomasdeanwhite/Scythe



(a) ASLDigits (b) GorogoaPuzzle (c) PaintLeapMotion (d) PolyDrop (e) Virtual Piano

Fig. 4. The five applications under test, used to empirically evaluate our framework.

- ASLDigits (Figure 4a) is an educational game teaching American Sign Language for the numbers 0-9. There is also a game in which users score points for using the correct signs for numbers displayed in a given time limit. ASLDigits contains around 4213 Lines of Code (LOC)
- GorogoaPuzzle (Figure 4b) is a puzzle game where unique interactions are performed with the Leap Motion in order to advance the story, and thereby move to different program states. GorogoaPuzzle contains around 19633 LOC.
- PaintLeapMotion (Figure 4c) is an open source app published on GitHub. This application allows users to paint onto a canvas with a selection of tools using the Leap Motion. PaintLeapMotion contains around 1579 LOC.
- PolyDrop (Figure 4d) is a physics game in which blocks fall on to the screen and the player needs to catch them on a bridge controlled by Leap Motion interaction. PolyDrop contains around 8212 LOC.
- Virtual Piano for Beginners (*VPfB*, Figure 4e) is an application which allows users to play an "air piano". There is a free play mode and also an educational mode which teaches users to play certain songs. VPfB contains around 2276 LOC.

We chose these applications due to their variety of use in the Leap API. These applications include use of the gestures API, 2-D menu interactions, advanced processing of the Leap data structures and other areas. The applications are also dissimilar to one another. The only information that our technique has of each application is the data from the Leap background service when user interaction occurred.

Data was recorded from five users. The users first practiced interacting with the Leap Motion on the "Leap Motion Playground", a training app provided by Leap Motion. Then, users explored each application in sequence for five minutes. We did not instruct them to perform specific tasks with the applications but allowed them to freely explore applications.

B. Threats to Validity

We chose a subset of available applications which use the Leap Motion Controller. To decide if our framework was applicable to an application, we use the following criteria:1) the applications must be in Java, and use the Leap Motion Java API; 2) the application must be available publicly, either on the Leap Motion Airspace Apps Store or open source. The applications chosen use different areas of the Leap Motion API. Some applications, like PolyDrop, make use of the stabilized vectors for menu interactions, whereas others like ASLDigits use the raw finger joint positions. Only GorogoaPuzzle uses gestures, and only a circle gesture. The variance in usage of the API means that our technique can be used on a wide range of applications which use the Leap Motion Java API.

A threat to external validity is whether the data used in training models is representative of data that actual users would provide. To mitigate this, we use data from five users, each interacting with the application under no guidance. Users were given a short training session on how to use the Leap Motion Controller, but not on how to use each application. Users recorded data for five minutes per application, with breaks in between each app. It is possible that the order of data recording gave users a chance to learn more about the Leap Motion Controller and improve usage on later applications. The order in which application data was recorded changed per person to mitigate against this.

As we are recreating and mocking an API, there is a question as to whether our mimic API represents the real Leap API. The version of the Leap API used for these experiments does not support replay of data, so playback of data through the physical device cannot occur, therefore the mock API must be used. The Leap API is sparsely documented, and it is infeasible to recreate the API exactly without knowledge and calculations that are missing from the documentation. To mitigate against this threat, we have techniques of reconstructing the raw data using our API before clustering occurs and we ensure that the reconstructed data seeded through our framework performs similar to the original user data.

With any developed software there is a potential for faults to occur. To mitigate against this threat, we have a unit test suite and also make all the artifacts available publicly on GitHub[2].

TABLE I
CODE COVERAGE FOR DIFFERENT DATA GENERATION TECHNIQUES FOR EACH APPLICATION. **BOLD** IS SIGNIFICANT ($P < 0.05$).

				N-gram Model Comparison			
	Random	Random Clusters	N-gram Model	Random		Random Clusters	
Application	Cov.	Cov.	Cov.	A12	P-value	A12	P-value
ASLDigits	0.425	0.441	0.468	**0.963**	< 0.001	**0.884**	< 0.001
Gorogoa	0.364	0.371	0.371	**1.000**	< 0.001	0.366	0.109
PaintLM	0.625	0.706	0.689	**1.000**	< 0.001	**0.080**	< 0.001
PolyDrop	0.459	0.505	0.534	**1.000**	< 0.001	0.513	0.838
VPfB	0.589	0.663	0.778	**1.000**	< 0.001	**0.849**	0.002

C. RQ1: How well does NUI testing with N-gram models work on Leap Motion apps?

Table I shows the line coverage achieved by different techniques of data generation. The two right-most columns show the

[2]https://github.com/thomasdeanwhite/NuiMimic/tree/nuimimic

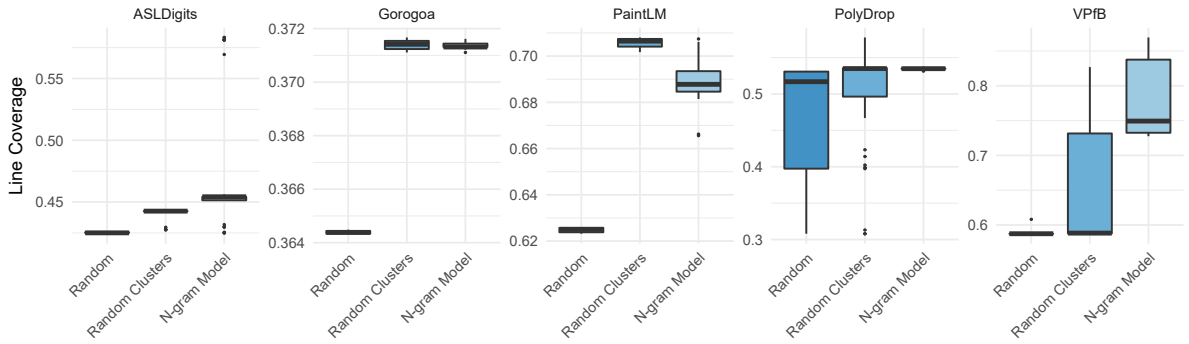


Fig. 5. Line Coverage for different data generation techniques for each application.

A12 effect size when comparing the N-gram model technique to random and random clusters respectively. Random generation achieves the overall lowest code coverage, compared to both random clusters and N-gram based generation. Statistical comparison between N-gram based generation and random generation shows that the difference is significant in all five cases, as can be seen in Figure 5. Using random clusters for test generation leads to substantial coverage increase on all 5 apps. The difference between the random and random clusters approach is that the random clusters approach exploits domain knowledge, selecting random cluster centroids from the model generation stage. Combining these centroids generates data similar to that which the original user provided i.e. real data that the Leap Motion could provide to an application. However, the random approach generates unrealistic hands and is very unlikely to generate something resembling actual human data from the Leap Motion under normal use. This demonstrates how important it is to generate realistic data. Compared to random clusters, the N-gram based generation adds animation. N-gram based generation allows not only the current hand to appear realistic, but a sequence of hands to be human-like. This leads to a significant coverage increase in two of the applications. For Paint Leap Motion the use of the N-gram interestingly leads to a significant decrease in coverage; for GorogoaPuzzle and PolyDrop there is no significant change. While overall there is a small average coverage increase, this result justifies a closer look at the individual applications under test. ASLDigits and Virtual Piano for Beginners (VPfB), the applications where the N-gram based approach performs best, use a Java game engine with Leap Motion integration. They both require the hand data to represent specific positions and gestures. For example, ASLDigits uses a machine learning approach to determine if signs are correctly shown, and Virtual Piano for Beginners requires specific hand shapes with minute changes over time. Furthermore, both applications use complex menus which require precise interactions with menu elements. All these aspects are more likely to occur with N-gram based test generation, leading to around *114* and *262* more lines of code being covered for ASLDigits and VPfB respectively. PaintLeapMotion, the application where the random cluster technique achieved higher coverage than the N-gram based approach, is a painting application, where users paint on a canvas using hand gestures. While the N-gram based approach generates more realistic hand sequences, these do not matter for this application: PaintLeapMotion only uses the finger tips area of the Leap Motion API. Users can change tools by moving their hand towards the back of the Leap Motion Controller's view and selecting a new tool from the pop-up menu. Here is a code snippet from PaintLeapMotion:

```
if (minDepth < ...DRAWING_DEPTH) {
  menuPanel.hide();
  draw(i);
} else if (minDepth > ...MENU_DEPTH) {
  menuPanel.show();
} else {
  tool.stopDrawing();
  menuPanel.hide();
  setLastPosition(i, null, null);
}
```

In this code, `minDepth` is the minimum position of a finger tip. The Leap Motion API uses a negative Z-axis so this is the front-most part of the hand. The selection of random clusters more uniformly samples combinations of cluster centroids, therefore more rapidly changing between the branches in this function. In the application, this is reflected by alternating between showing the menu, selecting new tools, and painting on the canvas very quickly. This leads to an increase of around 43 lines of code over the N-gram model approach. Using the N-gram model technique can also change tools, but does so at much less speed, following realistic movement. Random generation of test data is unlikely to move all points in the hand behind the threshold to activate the pop-up menu so can only paint on the screen using the default tool, and thus performs poorly on this application. For GorogoaPuzzle and PolyDrop the likely reason that coverage does not increase with the use of N-gram models is that both apps require very specific and complex interactions (e.g., balancing elements on a horizontal bar in PolyDrop). While N-gram based generation may produce more realistic data sequences, these sequences would need to be tailored to the specific state of the gameplay. Consequently, both random clusters and N-gram based generation are likely stuck at the same point in the application. Overall, the benefits of using an N-gram model in data generation are application specific. On applications such as PaintLeapMotion, which do not rely on a steady stream of data with small change over time, random clusters performs well. Other applications such as Virtual Piano For Beginners require precise gestures and slow interactions with menu items, which the N-gram based

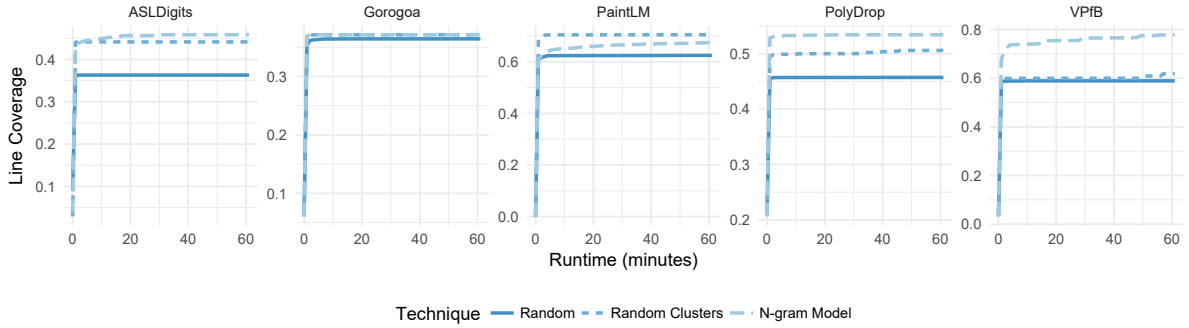


Fig. 6. Line Coverage for different data generation techniques for each application across time.

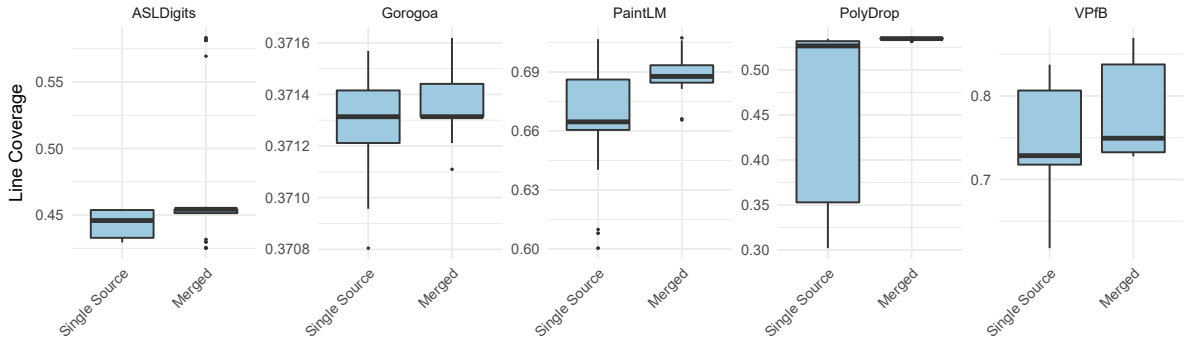


Fig. 7. Line Coverage for models using either multiple or single person data when training for each application.

approach handles well, while it is unlikely that the random cluster approach will generate a hand that remains still long enough to activate a button. Figure 6 shows the change in line coverage during test generation. In three of five applications, line coverage is still increasing after 30 minutes for an N-gram model based approach to generation. In two of five, line coverage is still increasing after 50 minutes. Given enough time, it is plausible that the N-gram model based approach will achieve an equal level of coverage on PaintLeapMotion than random clusters.

Although code can be executed by seeding NUI data, it is impossible to achieve 100% coverage in certain circumstances. For example, GorogoaPuzzle has defensive programming when loading images, ensuring that the image exists. The cases where an image does not exist cannot be executed by seeding Leap Motion data alone. Another example of unreachable code is in PaintLeapMotion, which contains both NUI and mouse interactions. For our experiments, no mouse interaction could take place hence there is no possible way to test this code.

RQ1: NUI test generation approaches increase coverage on Leap Motion applications, but applications may only use subsets of the complex NUI input data structures, limiting benefits achievable with N-gram modeling.

D. RQ2: How does the training data influence the effectiveness of NUI testing?

Table II shows the mean coverage for different generation techniques using models trained on both a) a single user's data or b) all users' data for the respective application. For all five applications, the mean coverage was greater for a 'merged' model that was trained on all users' data, as confirmed in Figure 7. Of the five applications tested, three applications achieved a significantly higher code coverage when tested with the merged model. From this, we can make two conclusions. Firstly, models that have been trained with more data yield higher code coverage. Secondly, a greater volume of training data is beneficial even when it originates from a number of different users.

TABLE II
CODE COVERAGE DIFFERENCE BETWEEN SINGLE AND MERGED DATA SOURCES FOR EACH APPLICATION. **BOLD** IS SIGNIFICANT ($P < 0.05$).

Source Application	Single Source Cov	Merged Source Cov	A12	P-value
ASLDigits	**0.444**	**0.468**	**0.725**	0.017
Gorogoa	0.371	0.371	0.536	0.590
PaintLM	**0.672**	**0.689**	**0.759**	< 0.001
PolyDrop	**0.467**	**0.534**	**0.965**	< 0.001
VPfB	0.738	0.778	0.711	0.080

Increasing the amount of data available to produce models increases the individuals assigned to each cluster, producing a more diverse set of cluster centroids to be chosen by models when generating data. As each cluster contains more elements, the N-gram models representing transitions between clusters are less sparse, allowing a greater variance in the sequences generated. The finding that a benefit accrues from a larger amount of training data, even when it originates from a diverse pool of users, is not entirely expected. Users interacting with the Leap Motion have different anatomy (e.g., hand sizes, finger lengths) and may interact with the controller in specific ways. Apparently, the benefits of generalizing over a diverse pool of data outweigh the disadvantages that might be expected

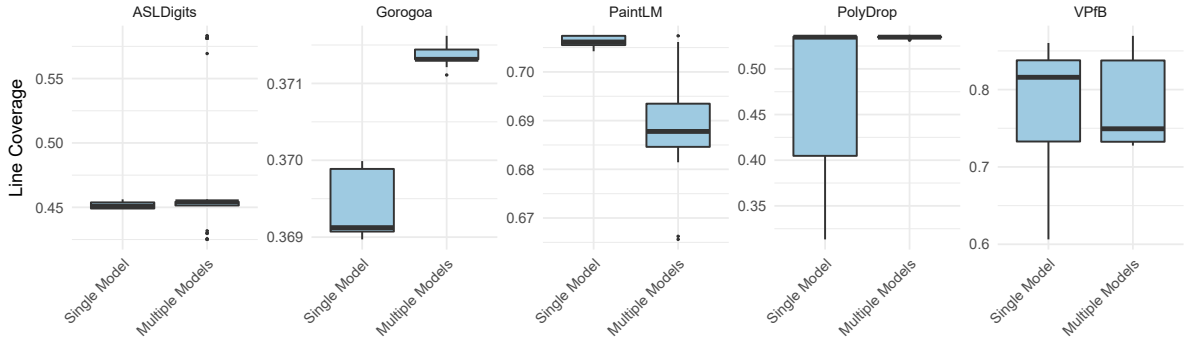


Fig. 8. Line Coverage for models using either single or multiple model generation for each application.

TABLE III
CODE COVERAGE FOR SINGLE OR MULTIPLE MODEL GENERATION FOR EACH APPLICATION. **BOLD** IS SIGNIFICANT ($P < 0.05$).

Data Generation Application	Single Model Cov	Multiple Models Cov	A12	P-value
ASLDigits	**0.452**	**0.468**	**0.751**	< 0.001
Gorogoa	**0.369**	**0.371**	**1.000**	< 0.001
PaintLM	**0.706**	**0.689**	**0.070**	< 0.001
PolyDrop	0.479	0.534	0.510	0.926
VPfB	0.781	0.778	0.483	0.879

from anatomical differences. This suggests that in future work, crowd-sourcing interactions from a large pool of users should be an effective way of building models for NUI testing.

RQ2: Test generation using models trained with more than one source of training data outperformed those using only a single data source. This suggests that pooling data across a number of users is beneficial, even though the users differ in their anatomy (e.g., their hand sizes and finger lengths).

E. RQ3: How does separation of NUI data into multiple models influence the effectiveness of NUI testing?

Table III shows the mean code coverage after testing the two forms of model generation: a single model or multiple models. The single model approach generates entire data frames at once, by selecting a centroid from Leap Motion data clustered as a complete set of features. The multiple model approach generates data from models clustered from subsets of the data set, then combining data from each model into a data frame. On ASLDigits and GorogoaPuzzle the multiple model based approach achieves a significantly higher code coverage; on PaintLeapMotion the coverage is significantly lower. The coverage difference can be seen in Figure 8. On the other two applications the mean coverage is slightly higher with multiple models, but differences are not significant. These results show that the decision to use multiple models for generating data is application dependent. The application which benefits mostly from use of a single model is PaintLeapMotion. From RQ1 we already know that random clusters perform better at interacting with the tool menu items of this application. Similarly, using a single model is more likely to reproduce the interactions with the tool menu in the training data, while creating separate models leads to less reproduction, and exploration of new combinations. For example, on PaintLeapMotion we used 1200 clusters, and the single model simply learns the temporal relationships between these clusters. In contrast, when splitting the data into five models, we end up with substantially more possible ways of interactions (i.e., 1200^5 possible combinations). A single model approach explores the input space much quicker, leading to 27 more lines of code being covered than using multiple models. Consequently, applications with simple interactions may be more suited to a single model approach, whereas applications which require more complex sequences of inputs are better suited for a multiple model approach.

GorogoaPuzzle benefits from the use of multiple models. It uses two main forms of interaction: circle gestures and hand movements. The first screen of GorogoaPuzzle requires a specific circle gesture before progression in the story can occur. However, advancing in the story does not necessarily increase code coverage, as the same code is used to handle all circle interactions. To achieve a higher coverage, tests need to advance far into the storyline, where complex sequences of interactions are introduced and needed to advance further. Using multiple models allows for more degrees of freedom in the generated data, and thus succeeds slightly more often in progressing in the GorogoaPuzzle storyline, achieving around 39 more lines of code covered.

ASLDigits also attained a significantly higher code coverage using a multiple model approach. Multiple models performs better than single model due to the application expecting specific finger-joint shapes corresponding to the ASL sign for 0-9, requested by the application. The single model approach merges hand positions and rotations from all interactions with the application, which decreases the amount of unique finger-joint shapes available; in contrast, the multi-model approach covers this with an explicit model, achieving around 67 more lines of code covered. Virtual Piano for Beginners is an interesting application when comparing single to multiple models. In RQ1, N-gram generation achieved a higher coverage than random clusters because it could generate a still hand to interact with the game menu. However, a single model approach can also generate a steady hand. Single model works well for this application due to the position and rotation being encoded with finger positions. To play the correct key on the piano in a tutorial song, the single model N-gram has to generate a single

sequence corresponding to the user pressing the key. However, similar to with PaintLeapMotion, the multiple model approach has a much higher search space, so is less likely to generate the sequence to activate the key and progress with the song, covering around 7 less lines of code than using a single model.

RQ3: Using multiple models is beneficial when applications use specific features in isolation. If a more precise replication of the training data is required, using a single model approach may be beneficial.

VI. Conclusions

The Leap Motion allows users to interact with applications through hand tracking. We have created a model and test generation framework for the Leap Motion, capable of generating data by learning from real user interactions. This demonstrates that the idea of NUI testing generalizing to other, more complex NUI devices than the previously studied Microsoft Kinect. It is also conceivable that the approach generalizes to other systems which use complex inputs e.g., Autonomous Driver-Assistance Systems, which alert drivers to possible future hazards [5], [11].

Splitting Leap Motion data structures into separate models exponentially increases the amount of data available during generation. Each model generates data in isolation, and interacts with other models when combining data in complex ways, producing data that was never recorded from the original user. However, if applications rely on precise positioning of a user's hands for interaction as captured in the training data, then the increased quantity of possible data can be as much a hindrance than an advantage. In our experiments, two out of five applications showed a clear benefit from splitting data, but we also found an example where coverage decreased. A challenge thus lies in identifying when to split data, and when not to split data. A possible solution might be to use a hybrid approach, where data is sampled from either of the two approaches with a different probability.

When training models from multiple sources of training data, increased data size leads to higher code coverage. This occurs even when the data is from different users. Potentially, this insight opens up the possibility to gather data through crowdsourcing from many individuals, and using that to train user-independent models for data generation. A further angle for future work lies in the generalization of models. We limited training data to individual applications, but will it be possible to create generalized models that can be used on applications *without* previous user data to train models with? Currently, our tool only provides a sequence of Leap Motion data that can be played back into the AUT. Future work involves identifying the current program state from the contents of the screen and providing regression tests with oracles. This can then be used in mutation testing. Finally, our experiments have also shown that programs controlled with complex NUI interfaces may also have complex program behavior, where blindly generating data may not achieve best results. In games like GorogoaPuzzle, thorough testing requires actions that are tailored towards the current state of the application. This suggests future work on identifying such program states, and learning different models for different program states.

References

[1] A. Arcuri and L. Briand. A Hitchhikers Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014.

[2] P. A. Brooks and A. M. Memon. Automated GUI Testing Guided by Usage Profiles. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 333–342, New York, NY, USA, 2007. ACM.

[3] J. E. Forrester and B. P. Miller. An Empirical Study of the Robustness of Windows NT Applications Using Random Testing. In *Proceedings of the 4th USENIX Windows System Symposium*, pages 59–68. Seattle, 2000.

[4] G. Fraser and A. Arcuri. EvoSuite: Automatic Test Suite Generation for Object-Oriented Software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 416–419. ACM, 2011.

[5] D. Greene, J. Liu, J. Reich, Y. Hirokawa, A. Shinagawa, H. Ito, and T. Mikami. An Efficient Computational Architecture for a Collision Early-Warning System for Vehicles, Pedestrians, and Bicyclists. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):942–953, Dec 2011.

[6] T. Griebe and V. Gruhn. A Model-based Approach to Test Automation for Context-aware Mobile Applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 420–427, New York, NY, USA, 2014. ACM.

[7] T. Griebe, M. Hesenius, and V. Gruhn. Towards Automated UI-Tests for Sensor-Based Mobile Applications. In *Intelligent Software Methodologies, Tools and Techniques - 14th International Conference, SoMeT 2015, Naples, Italy, September 15-17, 2015. Proceedings*, pages 3–17, 2015.

[8] C. Hunt, G. Brown, and G. Fraser. Automatic Testing of Natural User Interfaces. In *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*, pages 123–132, March 2014.

[9] D. C. Ince. The Automatic Generation of Test Data. *The Computer Journal*, 30(1):63–69, 1987.

[10] Leap Motion. Leap Motion — Mac & PC Motion Controller for Games, Design, Virtual Reality & More. https://www.leapmotion.com. Accessed: 2016-09-13.

[11] D. F. Llorca, V. Milanes, I. P. Alonso, M. Gavilan, I. G. Daza, J. Perez, and M. . Sotelo. Autonomous pedestrian collision avoidance using a fuzzy steering controller. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):390–401, June 2011.

[12] L. Mariani, M. Pezz, O. Riganelli, and M. Santoro. Automatic Testing of GUI-based Applications. *Software Testing, Verification and Reliability*, 24(5):341–366, 2014.

[13] B. P. Miller, L. Fredriksen, and B. So. An Empirical Study of the Reliability of UNIX Utilities. *Communications of the ACM*, 33(12):32–44, 1990.

[14] C. Pacheco and M. D. Ernst. Randoop: Feedback-directed Random Testing for Java. In *Companion to the 22Nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion*, OOPSLA '07, pages 815–816, New York, NY, USA, 2007. ACM.

[15] M. Shardlow. An Analysis of Feature Selection Techniques.

[16] G. H. Walton, J. H. Poore, and C. J. Trammell. Statistical Testing of Software Based on a Usage Model. *Softw. Pract. Exper.*, 25(1):97–108, Jan. 1995.

[17] J. A. Whittaker and J. H. Poore. Markov Analysis of Software Specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2(1):93–106, 1993.

[18] J. A. Whittaker and M. G. Thomason. A Markov Chain Model for Statistical Software Testing. *IEEE Transactions on Software Engineering*, 20(10):812–824, Oct 1994.

[19] D. Wigdor and D. Wixon. *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. Elsevier, 2011.

[20] S. Wood, K. Reidy, N. Bell, K. Feeney, and H. Meredith. The emerging role of Microsoft Kinect in physiotherapy rehabilitation for stroke patients. https://www.physio-pedia.com/The_emerging_role_of_Microsoft_Kinect_in_physiotherapy_rehabilitation_for_stroke_patients. Accessed: 2017-10-12.