PadCorrect - Correcting User Input on a Virtual Gamepad

Leonardo Torok* Fluminense Federal University
Elmar Eisemann† Delft University of Technology
Daniela Trevisan‡ Fluminense Federal University
Anselmo Montenegro§ Fluminense Federal University
Esteban Clua¶ Fluminense Federal University

Abstract

The processing power of modern smartphones allows publishers to port old and current console titles to these platforms. However, these games were designed to be controlled with a traditional gamepad. Normally, the solution used in mobile ports is a virtual gamepad. This interface adds buttons that imitate the layout of a gamepad as a semi-transparent overlay above the game. While this allows users to play the game, it lacks the necessary haptic feedback to provide an enjoyable experience. Frequently, users will miss buttons or press the wrong ones, which affects the in-game performance and leads to frustration. We present a solution to correct user input. First, we retain a few frames of user input instead of passing the data directly to the game. Using time series analysis, we seek known patterns and detect potential mistakes from the user, correcting actions before the commands are received by the game. We called this new gamepad the PadCorrect. In order to measure the impact on the experience, we performed a user study comparing the PadCorrect with a traditional virtual gamepad. The test results showed a good reception and provided evidence that the new interface is capable of improving the experience with mobile games.

Keywords: Virtual gamepad, touchscreen, mobile games

Index Terms: Human-centered computing—Touch screens; Human-centered computing—Usability testing; Applied computing—Computer games

1 Introduction

Mobile gaming represents the largest market segment of the video gaming industry [21]. Most of this content consists of games developed specially for touchscreen interaction, like Angry Birds (Rovio) or Cut the Rope (Zeptolab). However, since modern smart devices include substantial processing power, developers and publishers are starting to show interest in porting old and even modern titles from traditional platforms, like gaming consoles or PC, to smartphones. The mobile port of "Grand Theft Auto: San Andreas" (Rockstar Games) resulted in millions of paid downloads [20]. Additionally, Sega recently republished several Genesis/Mega Drive titles for Android and iOS [25], using a Unity interface integrated with a console emulator to run the game.

Unfortunately, the default input methods of smartphones differ significantly from traditional console gamepads (also known as a game controller), which are often used by classic games. Redesigning these titles to be fully adapted for touch devices would demand a severe reenginering of the gameplay. Cases like the Sega forever titles can be even worse. The game is not actually ported to Android or iOS, but simply runs on a console emulator, which makes it not viable to change the original game. Commonly, a semi-transparent overlay with virtual buttons (called a virtual gamepad) is overlayed above the game. A virtual gamepad creates a software representation of a physical controller. However, there is a big difference between using real buttons to play games and using a software representation. Virtual buttons lack any significant haptic feedback, impacting the in-game performance and enjoyment of the player [32]. Despite this limitation, virtual gamepads are a common solution to control mobile games.

On different occasions, adapting console games to work with touch controls can also negatively impact the reception of these ports. When the iOS version of Bioshock was announced, it drew significant media attention since it was a relatively recent game. However, game critics considered the experience inferior to the original version due to the touchscreen controls[1]. In the critics score aggregation website Metacritic, the port remains with a score much lower than the classic console versions. This is consistent with research that indicates that issues with controls have a greater impact on the experience than most other factors [5, 6]. Unfortunately, even touch-friendly control schemes based on gestures, like taps and pinch-to-zoom, as implemented by Netherealm studios in their mobile version of the fighting game Mortal Kombat X, can prove insufficient. Users and critics (Metacritic) complained that the smartphone game ended up being an extremely simplified version of the original game, devoid of all depth and gameplay that made the console version enjoyable. Also, redesigning the gameplay is not a viable solution for cases like console games emulated via software.

Physical gamepads, like the Moga Hero Pocket or the Steelseries Stratus, are another solution to control mobile games. However, the marketing penetration of these accessories is extremely low. According to the consulting company IDC, in the third quarter of 2016, only 5.1% of the US mobile gamers connected their devices to a physical gamepad [22]. This highlights that the touch-based interface will still be the main experience for most users and that improving the performance and usability of virtual gamepads has the potential to improve the gameplay of mobile ports.

The objective of our work is to keep ported games enjoyable and playable on mobile devices with similar depth and complexity as the original experience. To achieve this goal, our method includes intelligent algorithms that interpret and correct user input on a virtual gamepad in real time. Developers will train these algorithms with the input patterns that the game uses, teaching the controller to recognize what the user intends to do. They will also be able to introduce new gestures to perform actions in the game. Since the input paradigm is still a virtual gamepad, our method works in cases where the original game cannot be modified. This new virtual gamepad with an input correction system is called the PadCorrect.

The PadCorrect uses an input buffer, that stores all touches on the screen and the buttons that were pressed in the last few frames. Unlike traditional software gamepads, these button presses are not passed immediately to the game. Different features of the touches are stored in a multidimensional time series [26], that is then compared with the trained samples using Dynamic Time Warping [19] (Section 2.2). When a match is found (Section 3), the PadCorrect replaces the button presses on the buffer with the correct command, that is

*e-mail: ltorok@id.uff.br
†e-mail: e.eisemann@tudelft.nl
‡e-mail: daniela@ic.uff.br
§e-mail: anselmo@ic.uff.br
¶e-mail: esteban@ic.uff.br

[1]Metacritic - `http://www.metacritic.com/`

then sent to the game. This method is the main contribution of this work and intends to provide a more enjoyable gaming experience and improve the performance of virtual gamepads. We performed a comprehensive user study to evaluate if our new virtual gamepad achieved those goals, comparing its results with a traditional virtual gamepad. We focused initially on fighting games as a case study, since they use complex and well-defined input patterns to perform combos and special movements and rely on precise timing.

We organized this work as follows. Section 2 contains a discussion about related work on touchscreen interaction, with a special focus on gaming. It also describes our matching algorithm. In Section 3, we will talk about the issues with virtual gamepads and how our method improves their performance. In Section 4, we will present the prototype, the test methodology and our test sessions. The following Sections (5 and 6) will discuss the results and our findings. Sections 7 and 8 will then talk about conclusions and discuss future works.

2 Related Works

2.1 Touchscreen Interfaces for Games

Different studies have tried to evaluate the impact of the limited accuracy of touchscreens on usability and proposed methods to address this issue [3,7]. However, few works have tried to measure the difference in performance and usability between virtual gamepads and physical controllers. Zaman et al. [32] compared the in-game performance of users playing the game Assassin's Creed: Altair's Chronicles (Ubisoft) on a Nintendo DS (a handheld game console with physical controls) and on an iPhone. The only difference between both versions is the use of a virtual gamepad on the mobile game. Users died 150% more on the iPhone, while level completion time was 72% higher. In a post-test questionnaire, all 12 participants affirmed that they preferred to play on the DS.

Zaman and MacKenzie [31] evaluated the differences between a virtual gamepad, a Wii Remote, an analog stick attached to the screen with a suction cup, and buttons glued to the screen. Users then completed two tasks playing commercial console games on an emulator running on the mobile device, for all controllers. The initial task was to complete the first mission of the action game Metal Slug (SNK). For this task, the virtual gamepad resulted in more deaths and a higher completion time, with all differences being statistically significant. The second task consisted in performing different special attacks (button combinations that must be executed in the correct timing) in the Street Fighter 2 fighting game (Capcom). Each user had 50 attempts to perform the attacks. While the software gamepad ranked lower, the difference was not significant this time. However, players did report the virtual gamepad as the worst option for both tasks. This also motivated us to include subjective evaluations, since simply counting how many successful movements the user performed still fails to detect the nuances of how the user really felt about the interface.

Baldauf et al. [1] compared different mobile controls when playing commercial games like Pac-Man (Bandai Namco) and Super Mario Bros (Nintendo). The mobile-friendly tilting controls were considered the least accurate by users and also resulted in longer completion times than the virtual gamepad. The work concluded that tilt controls are more adequate for games designed from scratch for mobile platforms. This is an evidence that console games ported to mobile may still have to rely on a virtual gamepad.

Little work has tried to specifically improve the performance of a virtual gamepad. Torok et al. [28] increased the size of the most used buttons and improved the position of the buttons according to the location of touches on the screen. The last 100 touches were clustered with K-means [27] and the resulting centroid for each cluster was paired to the closest button. Gradually, the controller would move the buttons until they reached the centroid position. These adaptations were performed in real time and would change according to new interaction patterns. Usability tests compared two variants of the same virtual gamepad: one with the adaptive interface and the other with a fixed layout when playing Super Mario Bros (Nintendo) and Streets of Rage (Sega). The success rate of the adaptive version was significantly higher in all cases. An improved version of this approach led to a significantly higher success rate of the users when playing Super Mario Bros. and Sonic Wings (Tecmo) [29]. In both works, the users answered subjective questionnaires, rating the adaptive version with better scores than the non-adaptive interface. The approach, however, only corrects the gamepad to decrease the possibility of future mistakes. It also considers any kind of touch that hits a button as a correct event and cannot assist in situations where the user pressed a different button than intended. Our approach focuses on avoiding errors before they are sent to the game. Our solution also estimates if a user really intended to press a button, using knowledge about usual commands to predict the correct action.

2.2 Dynamic Time Warping

We represent touches as a multidimensional time series. A time series can be defined as a set of points or observations organized by the time they occurred [26]. A traditional method used to compare time series is Dynamic Time Warping (DTW), which measures the similarity of two time series with different speeds [2]. DTW works as follows; given two time series: Q (length n) and C (length m), defined as:

$$Q = q_1, q_2, \ldots, q_i, \ldots, q_n \tag{1}$$

$$C = c_1, c_2, \ldots, c_i, \ldots, c_m \tag{2}$$

An n-by-m matrix D is constructed. The (i^{th}, j^{th}) element of D is the squared distance $d(q_i, c_j) = (q_i - c_j)^2$, that is the alignment between points q_i and c_j. The best match between the sequences is a path through D that minimizes the total cumulative distances between them. The optimal path to reduce the warping cost can be defined as [19]:

$$DTW(Q,C) = min \left\{ \sqrt{\sum_{k=1}^{L} w_k} \right. \tag{3}$$

Where w_k is the element $(i,j)_k$ of the matrix, that is also the k^{th} element of the warping path W. W is determined as a dynamic programming problem seen in Equation 4:

$$\gamma(i,j) = d(q_i, c_j) + min\{\gamma(i-1,j-1), \gamma(i-1,j), \gamma(i,j-1)\} \tag{4}$$

In Equation 4, $d(q_i, q_j)$ is the current cell distance and $\gamma(i,j)$ is the cumulative distance of the current cell and the 3 adjacent ones. $\gamma(n,m)$ will contain the full warping distance between the series. The two time series will be more similar if $\gamma(i,j)$ value is lower. We used the warping distance to compare the current buffer to known gestures, selecting a gesture that best matches the current buffer (Section 3). Time series can also be used to represent data that is actually not connected to time. Keogh et al. [15] showed how to compare two-dimensional shapes mapped as a false time series (sampled in a 360 degrees arc) using DTW. This approach is similar to the one we will use.

DTW was used to solve several other problems such as gesture recognition. Ibañez et al. [13] showed that DTW was a good fit to track the joints of a player detected using a Kinect. The movement of each one resulted in a time series. Different methods were used to compare the gesture to a database containing samples of the desired movement: DTW, Procrustes analysis, Naïve Bayes and Hidden Markov Models. The database started with four different gestures, increasing to 12 gestures for the final test. In all scenarios, DTW had a superior performance, with a larger advantage when the database had more gestures.

Modern smartphone keyboards allow users to input text by sliding their fingers between the letters that compose a word. Kristensson and Zhai [16] proposed the SHARK^2 system for pen-based input. They used proportional shape matching to compare the performed gesture with a database containing 10,000 words and corresponding gestures. In a subsequent work [33], the authors used DTW and template matching to recognize strokes for swipe keyboards. They expanded the system by also including shortcuts to common actions, such as copy/paste and save. De Zoeten [9] implemented a swipe-based keyboard, using a DTW variant known as Greedy Asymmetric DTW, that reduces the computational cost of DTW by adding constraints to the search path.

The main differences between our problem and swipe keyboards can be observed on the insertion or replacement of the match. For keyboards, the recognized word is inserted at the end of the phrase. The consistency of the match is validated using language models. In our case, the match can replace any part of the buffer and its necessary to perform changes to the start and end of an inserted segment to keep it consistent with the rest of the commands. The methods to validate the match are based on the timing of the inputs instead of a linguistic model. This replacement logic is discussed in Section 3.3.

De Luca et al. [8] used DTW to add an extra layer of security to the gesture-based unlock system of Android smartphones. A common attack vector is that, if another person oversees the owner performing the gesture, it can be easily reproduced to obtain access. De Luca et al. used DTW to compare several features of the unlocking gesture, like speed and pressure, with a stored pattern performed by the owner. A correct shape was no longer enough to unlock the phone but the manner of the gesture counted as well. Our problem also involves features of touchscreen gestures to infer if a given action has a match in a database of samples.

3 Correcting Input in a Virtual Gamepad

The most important aspect of our input correction technique is the input buffer. When a user touches the screen, a position is registered in pixels as (X, Y), starting from the lower left screen corner. If a user's touch is inside the area of a button, a button press occurs. We refer to the state of all buttons in a single frame as button events. In a regular virtual gamepad, this events would be directly passed to the game. The positions of the touches and the button events are captured once per frame. The gamepad tracks up to 2 fingers at once, typically both thumbs. We normalize the values to be resolution-independent (discussed later). Each movement performed by sliding your finger or simply touching the screen, either with one or two fingers, will be called a gesture.

Instead of directly passing the input on to the game, we store it in a buffer. The touches are organized in a multidimensional time series, with the (X, Y) coordinates of each finger consisting in four dimensions. The buffer contains the entry of the time series and the associated button events for each frame.

We are using a false time series [15], since we are not using the time each event occurred to index our points, but the total delta Δ_n for the n^{th} entry. This value is the sum of the Euclidean distances between the n points in the series. We define Δ_n as follows:

$$\Delta_n = \Delta_{n-1} + \sqrt{(X_n - X_{n-1})^2 + (Y_n - Y_{n-1})^2} \qquad (5)$$

Where X and Y are the finger coordinates for the points n and $n-1$, while Δ_{n-1} is the same equation applied to the previous point in the series. For the starting point, we have $\Delta_0 = 0$. If more than one finger was active, we used the closest finger in relation to the directional pad as the reference, since it was the button used in the trained commands. If, during a frame, no finger is touching the screen, the time series will receive an entry with all dimensions set to -1 and the delta will be equal to the last non-empty entry.

Each set of button press events contains the state of all buttons on the screen. For each frame, we will have one entry for the multidimensional time series and the button states for that moment. When DTW matches a subsequence of the time series, we know we have to replace the button press events for the same frames.

To detect known gestures in the real time buffer, our method uses a database with entries for each potential gesture [8], discussed in the next subsection.

3.1 Gesture Database

The gesture database contains multiple gestures, using the same representation as the real-time buffer. To generate each entry, we performed the same gesture ten times and used the approach of De Luca et al. [8] and Biswajit et al. [14] to select the best sample. We do this by comparing the n collected entries to all the other $n-1$ samples using DTW. The warp values are used to calculate a mean warp value. The sample with the lowest mean warp value is considered the reference for the gesture. The mean warp value is also defined as the matching threshold for that gesture. The gamepad constantly compares the buffer with all gestures in the database to find the result with the lowest warping distance. If this distance is also lower than the gesture threshold, it is considered a match and the corresponding segment of the buffer is replaced. During the development, these thresholds seemed to be excessively strict. So we empirically selected a multiplier (9 times) to increase the gesture threshold to a more reasonable value. We intend to use the results to improve these limits for a second iteration of our method.

3.2 Normalization

The action performed by the user may differ significantly in size and position from the sample stored in the database, even if it should be a perfect match when using differing devices with different dimensions. To address this issue, we normalize and translate both the buffer and the database entries. We normalize each gesture by dividing the X and Y coordinates, as well as the total delta by the screen height (measured in pixels). Implicitly, we assume the size of the virtual gamepad is relatively defined to the screen height, which handles screens with differing aspect ratios. With this normalization, our approach is also independent of a phone's screen resolution.

The coordinates are translated to a neutral starting point. For the X-axis, we use the minimum X value. Similarly, the minimum Y-value is used for the vertical axis. We subtract these values from the respective dimensions for all points of the gesture. This means that DTW will match a gesture even if it is performed in a different part of the screen than the trained sample, as soon as it has a similar shape. The normalization is applied to the buffer every frame before comparing to the already normalized database entries. This is important because it addresses one limitation of current virtual gamepads: since users cannot feel the position of the buttons, they will end up performing the correct movements on the wrong part of the screen, instead of where the buttons actually are [31].

3.3 Replacement Logic

Figure 1 shows the replacement logic. The squares represent the buffer, in a queue. The orange boxes are the commands that already expired and were passed to the game. The yellow boxes are the actual buffer, the commands that were performed but not sent to the game. The top queue are the button press events while the bottom one are the touch coordinates. To simplify the scheme and avoid a cluttered image, we are showing a shortened buffer and only the X and Y dimensions for a single finger. The buffer also shows the deltas that serve as references for the false time series. The final prototype uses a buffer with 10 frames of input data. In the first frame, we can see the coordinates being normalized and translated. A segment of the buffer, surrounded by a red line, resulted in a positive DTW match. The database gesture is represented as green boxes. The button presses are replaced, as shown on Frame 2. The time series also changed, with all values set to -1, which represents an

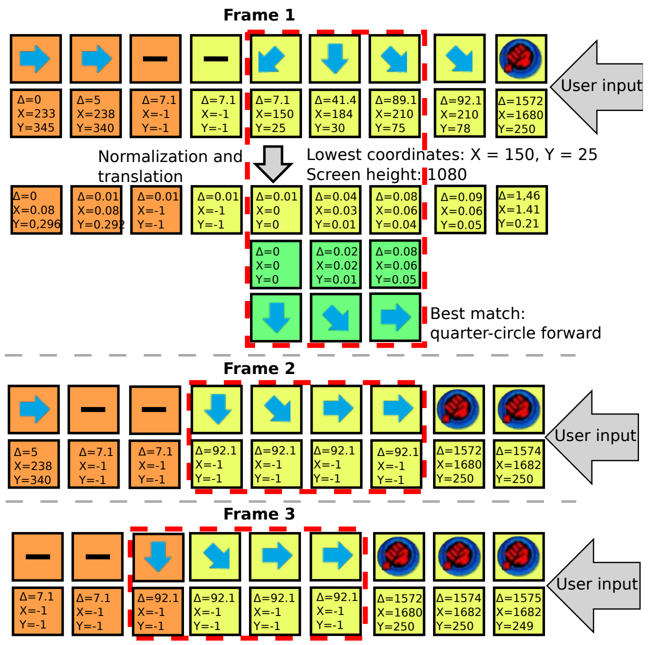


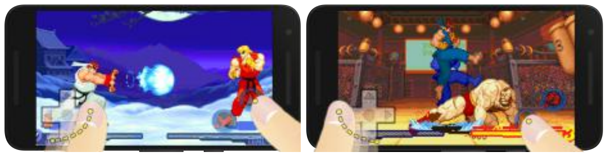
Figure 1: Diagram showing the replacement process in 3 frames after a positive match.

empty event. This prevents DTW from matching a sequence that was already corrected in the previous iteration. All matched points have their deltas replaced by the last delta of the segment, so new events will continue from that value. In the last frame, the game started receiving the replaced match.

A longer input buffer would allow our method to use more information to find matches. This is useful to detect long gestures. However, it is not viable to introduce a significantly large input lag. We decided to use a secondary input buffer, storing the last 20 events that left the main buffer and were already sent to the game. This allows the DTW matches to include portions of both buffers. If a match includes part of the secondary buffer, our method checks if the button presses on that segment were already correct (equivalent to the database match). In this case, only the events on the primary buffer will be replaced. If the expired events were incorrect, then we insert the entire database match in the primary buffer so it is sent to the game in the next frames.

A button press may be registered in several sequential frames. This results in a risk for our match: the boundaries (start and end position) defined by DTW may end up in the middle of one of these sequences. Since holding a button is a single action, it would be inconsistent to replace only part of the frames corresponding to that command. Our method addresses this issue by moving the start or end of the DTW match while it finds equivalent events (same state for all buttons), increasing the size of the match. The adjustment of the boundaries is shown in the second frame of Figure 1, where the end of the match increases to include a repeated button press.

The matched gesture can also be much smaller (measured in pixels, normalized in relation to the screen height) than the database entry or have a significantly different ratio (width versus height) on the screen. DTW ignores the size and ratio differences, since it can compare time series of different characteristics and still finds an optimal warping [19]. To include limits to these matches, each one must pass size and ratio validations before being used as a replacement. A gesture must have at least 70% of the size of the database gesture (measured by the normalized total delta) to pass size validation. For the ratio validation, we calculate the individual values for the gesture and the matched portion of the buffer. We do this by dividing the width (difference between the maximum and minimum coordinates



(a) Quarter-circle forward (⬇ ⬊ ➡) (b) Quarter-circle backward (⬇ ⬋ ⬅)

Figure 2: The gestures trained for the test demonstrated with Street Fighter Alpha 2 (Capcom). The user must press the buttons in a quick sequence and follow with an attack button.

on the X-axis) by the height (same approach, for the Y-axis). We then divide the two, selecting the larger value as the denominator. If the value is greater than 0.5, it is a valid match.

3.4 Limitations

The main limitation of the current approach is dealing with discrete input, like a single button press. This kind of action results in a single coordinate on the screen for a touch, while our method works by comparing relatively large sequences of touches. In a fighting game, like our case study, this means that the PadCorrect will not improve simple actions like moving the character or pressing a single attack button. Games that do not rely on complex button sequences, like platforming games, would not benefit from the current approach.

For this work, we decided to focus on simplifying the execution of complex commands, that are harder to be performed on touchscreens in comparison to single button presses.

4 User Study

4.1 Prototype

We used Unity to develop a prototype. Unity is a cross-platform game engine, used to develop 2D and 3D games and simulations. Unity is the engine used to develop around 34% of all new games[2].

The game runs on a console emulator (Libretro[3]), wrapped in a Unity interface. All code was developed in C#. We used a DTW implementation based on Rakthamanon et al [18], that proposed the UCR Suite for DTW, with massive performance gains over the former state-of-the-art solutions. Erez [23] ported the C++ UCR Suite to C#. This is the version used in our prototype.

The size of the buffer we used was 10 frames. Since our prototype runs at 60 frames per second, this means roughly 167 milliseconds of input lag. Our approach proposes increased input lag to gain higher accuracy when performing movements. The buffer size can be easily adjusted and future work could evaluate different parameters.

Figure 2a and 2b shows the two patterns trained for our tests. These commands are frequently used in fighting games [31]. We represented these gestures using the fighting game convention (where right is forward). If a player changes position with the adversary, the movements will be inverted (quarter-circle forward becomes backward and vice-versa). To perform them on a traditional directional pad or on a touchscreen, the user slides his fingers between all correct buttons, in an arc that roughly corresponds to a quarter of a circle, starting with the first button and finishing on the last one (Figure 2a).

4.2 Methodology

The user study intends to evaluate if the PadCorrect can improve the performance of the user when performing a set of tasks. It also evaluates his satisfaction and the amount of physical and mental effort necessary to complete the tasks. We will consider our correction approach successful if it improves the performance or the usability of a virtual gamepad.

[2]Unity Fast Facts: `https://unity3d.com/public-relations`
[3]Libretro: `https://www.libretro.com/`

We followed an approach similar to Zaman and MacKenzie [31], with simulated game tasks for the test sessions. One observation the authors made was that using an evaluation based on the ISO 9241-9 standard (like on Natapov et al. [17]) would not be ideal since it focuses on targeting tasks. Since our case study is a fighting game, this kind of task is not adequate.

We set up different tasks and registered information like the number of attempts to accomplish a movement or the amount of victories against a CPU-controlled opponent, which were part of the objective evaluation. To track these statistics, we recorded the device screen during the test session and revised the video afterwards to manually count each occurrence. During the session, the smartphone was configured to show the user touches as a semi-transparent circle, allowing us to see when the user attempted the pattern of the movements. Event detection was annotated after the test. During the tests with a CPU opponent, the user could be interrupted during a combo attempt by an enemy attack. In those situations, we did not considered it as a failed attempt and simply discarded the occurrence

We also used different subjective evaluations. The first was the NASA Task Load Index. NASA-TLX is a workload assessment tool that is based on a weighted average of self-reported scores for mental demand, physical demand, temporal demand, performance, effort and frustration [12]. Our choice was motivated by the large scale use of NASA-TLX in user evaluation in the last decades [11] and the coverage of aspects that are relevant to our research. After completing each set of tasks, the users would fill out the NASA-TLX questionnaire. At the end of each trial, the users would fill out the System Usability Scale [4]. SUS consists of 10 questions that can be rated according to an 1 (strongly disagree) to 5 (strongly agree) Likert scale. These questions are then consolidated into a 100 points score that aims to provide a global view of usability. A SUS score below 68 generally indicates usability issues on the interface being evaluated [24]. Additionally, the post-test questionnaire included extra questions about other aspects, using an 1 to 5 Likert scale. The additional questions were:

- How comfortable was it to perform the tasks using the controller? - 1 (very uncomfortable) to 5 (very comfortable)
- How would you rate the finger fatigue to perform the tasks? - 1 (very high) to 5 (very low)
- How would you rate the response time of the controller? - 1 (very bad) to 5 (very good)
- How would you rate the accuracy of the controller to perform the commands you wished? - 1 (very low) to 5 (very high)
- How easy was it to perform the commands you wished when fighting against the CPU? - 1 (very hard) to 5 (very easy)

The additional questions are meant to test the trade-off between accuracy and response time discussed in Section 4.1, while also verifying if our method can avoid finger fatigue or increase comfort. The first part of the test consisted of tasks based on performing specific combos. The first movement was the "fireball". The attack can be performed with a quarter-circle forward followed by a punch button (↓ ↘ → ●). The second movement is a more complex pattern, the "super fireball". The user needed to input two quarter-circle forwards and a punch (↓ ↘ → ↓ ↘ → ●) in quick succession. For the super fireball, he also must have at least one full special meter. For the tasks, users already started with 3 full bars. If the user managed to use all bars, we simply reloaded the stage to continue the test.

We also tested our correction method using the shortcut detailed in Figure 3. We find this an interesting case because it highlights how the PadCorrect can add new gestures to the game without having access to the source code of the game. Our input correction method keeps the same input paradigm and the feel of the original control

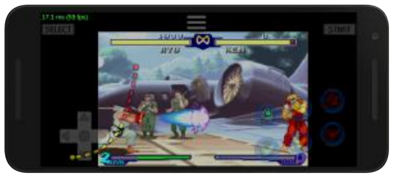


Figure 3: Example of the shortcut to perform the super special. The yellow line is matched as the first quarter-circle forward while the red line is matched as the second one. The green dot is the player pressing the punch button.

scheme. Hereby, developers can expand the controls of an old game even when running on an emulator. We evaluated the usage of this shortcut separately in our analysis.

All tests were performed twice for each user. Once with the PadCorrect and once with an identical interface without any kind of correction. The latter served as a baseline, representing the current state of virtual gamepads. Figure 2 shows the gamepad and the game Street Fighter Alpha 2 (Capcom), used for testing. The character on the left was used by all participants. The CPU difficulty and opponent were kept constant for all users. We set the opponent's difficulty to slightly less than half the maximum. No task had any kind of time limit and users were free to perform them at their own pace.

The test started with a training session, where we would teach the required movements and the game rules to the user. Each volunteer would then try to perform each attack and play the game until he felt confident to start the test. In all cases, the user was not informed about the input correction system, they were just told that they would test two different controller prototypes. In some tasks, a CPU-controlled opponent was present to add extra pressure. Each user was instructed to not try to fight the CPU and concentrate on the task. The user would fail the task if the enemy drained his entire health bar before performing all combos. The initial part of the evaluation included the following tasks:

- 10 attempts to perform fireballs. We counted how many successful movements he managed to perform.
- 10 attempts to perform super fireballs.
  - 10 attempts to perform the super fireball with the shortcut.
- 5 fireballs against the CPU. For this test we counted how many attempts were needed to correctly perform the action. The volunteer had to do it while a CPU opponent attacked him.
- 2 super fireballs against the CPU. Users were allowed to use the shortcut if they wanted to, when using the PadCorrect (since the regular gamepad does not have this functionality).

Each volunteer performed these 4 tasks for one controller and then answered the NASA-TLX survey about the workload. Next, the users did the same for the other controller. We were concerned that the learning effect could impact the results. The second controller could have an advantage in the tests since the user already practiced playing the game with the first gamepad. To compensate this effect, half of the users started all tests with the PadCorrect while the other half started with the regular gamepad.

The second part of the test was a three-round fight against the CPU opponent. The players could fight freely and use any movement they learned. They were allowed a new training fight before starting the test. We tracked the rounds that were won or lost. Once again, users did this twice. After each fight, they answered another NASA-TLX survey.

After the tests, users answered the SUS questionnaire and the 5 Likert scale questions, for both controllers. In the end of the test, they would also inform if they preferred one of the interfaces or if both were the same for them. We debriefed the participants at the end of the study as to the purpose of our experiment.

4.3 Test Sessions

For this test, 12 volunteers were randomly selected. We did not require any prior gaming experience and users had different gaming backgrounds. Before the test, each user read and signed a free and clarified consent release form. They did not received any compensation for their participation. We used the same device for all tests, a Samsung Galaxy S8 with a 5.8 inches screen (13.23 cm x 6.44 cm) and a resolution of 2960 x 1440. All buttons occupied square areas. The action buttons (punches and kicks) measured 20% of the screen height, while the directional pad measured 40% of the screen height.

Volunteers started by answering a profile questionnaire. 75% of the users were between 18 to 30 years old, with the remaining volunteers in the range of 31 to 40. Users were asked to report their gaming preferences regarding different platforms. On average, users played 3.54 hours on PC, 1.41 on consoles and 2.04 on smartphones per week. When inquired about which platform they preferred to play games on, 8 users chose the PC, 2 chose consoles and 2 reported preferring to play on smartphones.

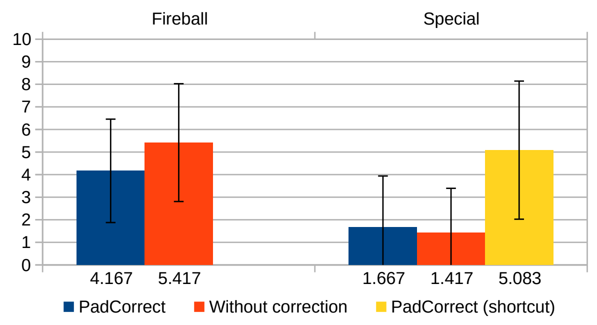
To compare the numeric results when using both controllers, we used the Wilcoxon signed-rank test [30], a non-parametric statistical test. We used a 95% confidence interval. If the p-value for a test is lower than 0.05, the difference is considered significant. The Wilcoxon test was used because it does not depend on the distribution of the data (while a t-test demands a normal distribution, for instance) and has a high precision even if the distribution is normal [10]. We will use a precision of three decimal places for our results.

5 Results

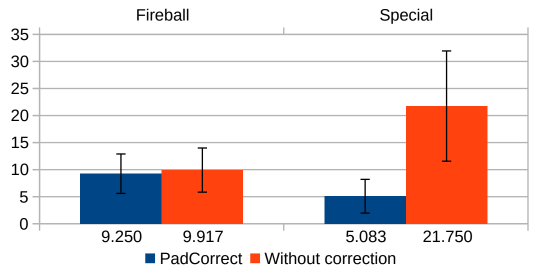
The test started with the users being asked to perform 10 fireballs. Figure 4a, shows the average number of movements performed with each gamepad. This difference is not significant according to the Wilcoxon test (p-value = 0.116) for the regular fireballs or the super special. However, when using the shortcut (large half-circle) to perform the super special, the difference is significant when compared to the version without correction (p-value = 0.006).

While users were able to perform more fireballs without the correction system on the first test, the results changed when they had to repeat the task while facing constant pressure of a CPU opponent. Figure 4b shows that they needed fewer attempts when using PadCorrect. This difference however is not significant (p-value = 0.542). For this task, users were capable of failing if the CPU opponent managed to drain all their health bar before they performed the required amount of correct combos. Two users failed this task for both controllers. When the task was to perform super specials (shortcuts allowed in the version with correction), users needed fewer attempts when using the PadCorrect. This difference is significant (p-value = 0.003). The amount of users that accomplished the task is also different. With the PadCorrect, all 12 users managed to perform 2 super specials, while only 4 did it without the correction system.

The final part of the test was a three-round fight against the CPU, with users fighting freely and being allowed to perform any movement, including the fireballs or the super fireball (users were free to use the shortcut). Of the 12 users, 8 won the fight when using the PadCorrect while 6 did it without the correction. Users reported the difficulty of the task in the NASA-TLX evaluation and also on the SUS scores, as well as the final questions. The NASA-TLX workload scores (Figure 5a) when doing the initial tasks were higher for the gamepad without input correction. This difference is significant (p-value = 0.015). The results for the workload of fighting the CPU are also higher and show a significant difference (p-value = 0.023). For these scores, a lower NASA-TLX value is a better result, representing a lighter workload to complete the same tasks.



(a) Number of correct movements (fireball and super special) performed after 10 attempts. More correct movements represent a better result.



(b) Number of attempts necessary to perform the required movements (5 fireballs or 2 super specials) while facing a CPU-controlled enemy. Fewer attempts represent a better result.

Figure 4: User performance for the predefined tasks.

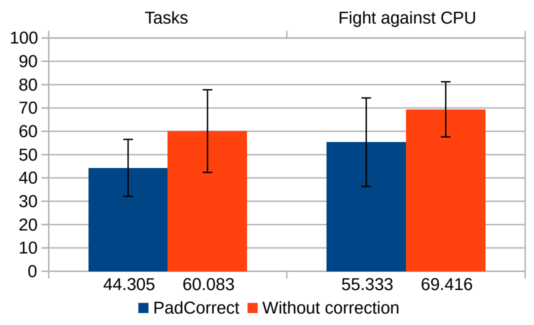
Figure 5b shows the average SUS scores. The difference is significant (p-value = 0.004). The PadCorrect had an average score higher than the baseline. An interface that scores less than 68 is generally said to present a below average or poor usability [24]. Of the 12 volunteers, only three scored the PadCorrect less than the baseline, while ten volunteers scored the regular virtual gamepad less than the baseline.

Users reported comfort, finger fatigue, response time and the accuracy of the controller to perform commands (with and without a CPU enemy). Figure 6 shows the average scores for each one of these aspects. Since the Likert score only allows 5 different answers, it was not possible to calculate a p-value for many factors since ties do not count for the Wilcoxon test. The PadCorrect had higher scores on average for comfort, accuracy with no CPU enemy and accuracy when facing an enemy. The difference is statistically significant for comfort (p-value = 0.012). Users rated the PadCorrect higher when considering finger fatigue (which means lower finger fatigue), but the difference is not significant. Surprisingly, users also rated the PadCorrect slightly higher when evaluating the response time. Even if the difference is not significant, it is noteworthy that the 10 frames of input lag did not impact this metric.

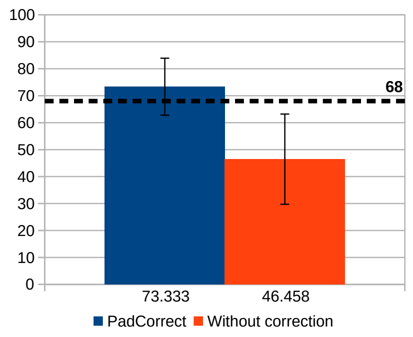
Finally, users were asked to chose which controller they liked more to use. Nine preferred the PadCorrect, 2 were indifferent and only 1 would rather use the version without correction. After this question, we revealed that one of the controllers used an input correction system and we asked users to try to guess which one it was. Ten users got it right, 1 guessed it wrong and 1 could not decide. This last question intended to verify if users could notice the input correction in action.

6 Discussion

The first batch of tasks showed some interesting results. Surprisingly, users did not perform better when doing the fireballs using the PadCorrect. We observed that most users tried to make the combos as slowly as possible, while our matching algorithm was trained



(a) NASA Task Load Index workload scores for each controller, for the in-game tasks and for the three-round fight against the CPU enemy. Lower scores represent a better result.



(b) System Usability Score (SUS) ratings for both controllers. Higher scores mean better usability. The dashed line is the SUS baseline, that defines that any interface that scores lower than 68 has a poor usability.

Figure 5: Subjective scores reported by the users after the tests.

using fast gestures, a difference that impacted the recognition. When performing fireballs against an enemy, users had to do combos faster and the PadCorrect actually demanded slightly less attempts. This information will be useful to adjust and improve the next version of our gamepad. The super special movement is more complicated, so the help provided by the correction was more apparent. The shortcut (large quarter-circle that matches as two sequential regular half-circles) significantly improved the performance of users and was well received. When adding the extra pressure of having a CPU controlled enemy, the PadCorrect fared much better. Users also won more matches when using the PadCorrect.

The subjective evaluation presented results that support the idea that the PadCorrect improved the experience for our volunteers. The SUS score for the regular gamepad averaged 46.458, substantially lower than the minimal desirable score of 68. Users rated the PadCorrect with a higher score (73.333), above the SUS baseline. This shows that, according to the SUS evaluation, the input correction system increased the usability to the point that a poorly performing interface managed to improve significantly and surpass the baseline.

The NASA-TLX questionnaire measured the workload of performing the given tasks. We separated the evaluation with the first batch of predefined tasks and the second part with the free three-rounds fight. In both cases, users reported significantly lower workloads when using the PadCorrect, which indicates lower mental and physical effort, less time pressure, better performance and decreased effort and frustration.

The end questionnaire also resulted in higher scores for the PadCorrect in all cases, with comfort presenting a statistically significant difference, supporting the idea that our system increased the comfort of playing mobile games. The response time category is especially interesting: while the PadCorrect fared better on average, we expected it to have a worse response time due to the increased input lag. However, users felt that the regular version was not responding to their commands and interpreted the lack of response as a poor response time or input lag. We believe that this could indicate that the trade-off (increasing input lag to improve the accuracy) was successful. Further studies could help to clarify exactly how perceptible the input lag is. With the positive results in our subjective evaluations, it was also expected to see a higher preference for the PadCorrect controller, with 75% of the volunteers reporting that they preferred to play the game with the input correction system.

In general, users complained about the difficulty of performing the movements with the regular version. It is interesting that some users even asked if we changed something in the regular controller when facing the CPU enemy, since it was much harder to perform the combos. In those situations, the PadCorrect led to a better performance since it was adjusted to faster inputs.

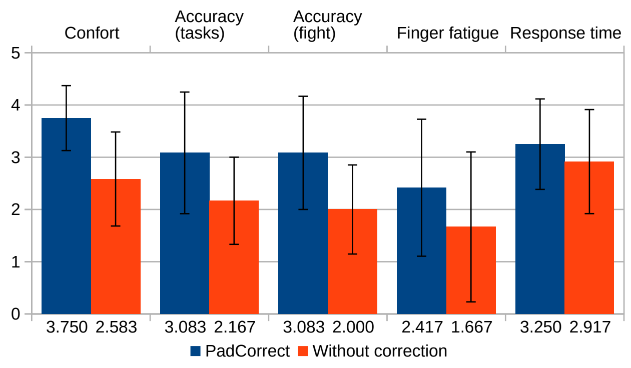


Figure 6: Ratings for controller attributes, reported using an 1 to 5 Likert scale.

7 Conclusion

Companies keep pushing the boundaries of mobile games with ports of console titles and downsized versions of modern videogame franchises. Adapting games created to be played with a gamepad to a touch interface, while keeping the same gameplay and general feel of the original version, is a difficult task. While a virtual gamepad emulates the original controls, it results in a poor gaming performance and frustrates the consumers. We presented an input correction system (PadCorrect) based on a buffer that delays inputs while analyzing the touch patterns and correcting interaction errors. The intention is to increase the performance of virtual gamepads to the point that more games become playable (and enjoyable) on smartphones.

The response from our tests was encouraging, showing a good reception of our system. The increase of the SUS score above the baseline and the lower workload are good evidences to support the claim that we were able to improve usability while decreasing effort, frustration, and general workload, providing a more enjoyable experience. Users also had a clear preference for the PadCorrect, showing that a better gaming experience was achieved. With further adjustments to better detect slow actions and a more comprehensive sample database, we believe these results can be further improved, making several classic console titles effectively playable on smart devices.

8 Future Works

To detect slow gestures, one could increase the size of the secondary buffer, which does not affect the general responsiveness since it only contains events that were already sent to the game. A more complete database, with more gestures, is also a natural direction.

The PadCorrect is currently limited to correcting long command sequences and will not help in games where the input is composed of single button presses (as discussed on Section 3.4). One possible solution to this limitation is to use a more sophisticated machine learning approach, with the PadCorrect actively evaluating what is happening on the game at the moment, predicting the best actions and correcting user input accordingly.

Acknowledgments

The authors wish to thank all volunteers for their participation on the user study. We also would like to thank the incredible support from the Delft University of Technology and the Fluminense Federal University. This work was supported in part by a grant from CAPES (n. 88881.134271/2016-01).

References

[1] M. Baldauf, P. Fröhlich, F. Adegeye, and S. Suette. Investigating On-Screen Gamepad Designs for Smartphone-Controlled Video Games. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 12(1s):1–21, 2015. doi: 10.1145/2808202

[2] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, vol. 10, pp. 359–370. Seattle, WA, 1994.

[3] X. Bi and S. Zhai. Bayesian touch: a statistical criterion of target selection with finger touch. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pp. 51–60. ACM, 2013.

[4] J. Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[5] P. Cairns, J. Li, W. Wang, and A. I. Nordin. The influence of controllers on immersion in mobile games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 371–380. ACM, 2014.

[6] E. H. Calvillo-Gámez, P. Cairns, and A. L. Cox. Assessing the Core Elements of the Gaming Experience. In *Game User Experience Evaluation*, pp. 37–62. Springer, 2015. doi: 10.1007/978-3-319-15985-0_3

[7] J. C. Chang, N. Hahn, and A. Kittur. Supporting mobile sensemaking through intentionally uncertain highlighting. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pp. 61–68. ACM, 2016.

[8] A. De Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann. Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 987–996. ACM, 2012.

[9] R. de Zoeten, B. O. K. Intelligentie, and J. Zuidema. Recognizing input for swipe based keyboards. *Bachelor thesis, Institute for Logic, Language and Computation Faculty of Science, University of Amsterdam*, 2013.

[10] M. P. Fay and M. A. Proschan. Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys*, 4(0):1–39, 2010. doi: 10.1214/09-SS051

[11] S. G. Hart. Nasa-task load index (nasa-tlx); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, vol. 50, pp. 904–908. Sage Publications Sage CA: Los Angeles, CA, 2006.

[12] S. G. Hart and L. E. Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. *Advances in psychology*, 52:139–183, 1988.

[13] R. Ibañez, A. Soria, A. R. Teyseyre, L. Berdun, and M. R. Campo. A Comparative Study of Machine Learning Techniques for Gesture Recognition Using Kinect. In *Handbook of Research on Human-Computer Interfaces, Developments, and Applications*, pp. 1–22. IGI Global, 2016. doi: 10.4018/978-1-5225-0435-1.ch001

[14] B. Kar, P. K. Dutta, T. K. Basu, C. VielHauer, and J. Dittmann. DTW based verification scheme of biometric signatures. In *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 381–386. IEEE, 2006. doi: 10.1109/ICIT.2006.372387

[15] E. Keogh, L. Wei, X. Xi, S.-H. Lee, and M. Vlachos. Lb_keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *Proceedings of the 32nd international conference on Very large data bases*, pp. 882–893. VLDB Endowment, 2006.

[16] P.-O. Kristensson and S. Zhai. Shark 2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pp. 43–52. ACM, 2004.

[17] D. Natapov, S. J. Castellucci, and I. S. MacKenzie. ISO 9241-9 Evaluation of Video Game Controllers. *Proceedings of the Graphics Interface Conference (GI'09)*, pp. 223–230, 2009.

[18] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 262–270. ACM, 2012.

[19] C. Ratanamahatana and E. Keogh. Everything you know about dynamic time warping is wrong. *Third Workshop on Mining Temporal and Sequential Data*, pp. 22–25, 2004. doi: 10.1097/01.CCM.0000279204.24648.44

[20] Rockstar Games. Grand theft auto: San andreas for android. `https://play.google.com/store/apps/details?id=com.rockstargames.gtasa`.

[21] Superdata Research. Year in review (2016). `https://www.superdataresearch.com/market-data/market-brief-year-in-review`, 2017.

[22] Venture Beat. Gamevice inc. raised 12.5 million with its mobile controller. `https://venturebeat.com/2017/05/31/gamevice-inc-raised-12-5-million-with-its-mobile-controller/`, May 2017.

[23] E. Robinson. Dynamic time warping - ucr suite in c sharp. `http://debugitalready.blogspot.com.br/2012/12/dynamic-time-warping-ucr-suite-in-c.html`, 2012. (Accessed on 03/18/2016).

[24] J. Sauro. Measuring usability with the system usability scale (sus). `https://measuringu.com/sus/`, February 2011.

[25] Sega. Sega forever. `http://forever.sega.com/`, 2017.

[26] R. H. Shumway and D. S. Stoffer. *Time series analysis and its applications: with R examples*. Springer Science & Business Media, 2006.

[27] A. Smola and S. Vishwanathan. Introduction to machine learning. *Cambridge University, UK*, 32:34, 2008.

[28] L. Torok, M. Pelegrino, J. Lessa, D. G. Trevisan, C. N. Vasconcelos, E. Clua, and A. Montenegro. Evaluating and customizing user interaction in an adaptive game controller. In *Design, User Experience, and Usability: Interactive Experience Design*, pp. 315–326. Springer International Publishing, 2015.

[29] L. Torok, M. Pelegrino, D. G. Trevisan, E. Clua, and A. Montenegro. A mobile game controller adapted to the gameplay and user's behavior using machine learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9353, pp. 3–16, 2015. doi: 10.1007/978-3-319-24589-8_1

[30] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[31] L. Zaman and I. S. Mackenzie. Evaluation of Nano-stick, Foam Buttons, and Other Input Methods for Gameplay on Touchscreen Phones. In *International Conference on Multimedia and Human-Computer Interaction-MHCI 2013*, vol. 1, pp. 1–8, 2013.

[32] L. Zaman, D. Natapov, and R. J. Teather. Touchscreens vs. traditional controllers in handheld gaming. *Proceedings of the International Academic Conference on the Future of Game Design and Technology*, pp. 183–190, 2010. doi: 10.1145/1920778.1920804

[33] S. Zhai and P. O. Kristensson. The word-gesture keyboard: reimagining keyboard interaction. *Communications of the ACM*, 55(9):91–101, 2012.