ARC-Pad: A Mobile Device for Efficient Cursor Control on Large Displays

by

David McCallum

A Thesis submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfilment of the requirements of the degree of

MASTER OF SCIENCE

Department of Computer Science

University of Manitoba

Winnipeg

Abstract

I present ARC-Pad, a cursor positioning technique for large displays. It allows us to use a touchscreen-enabled phone as a mobile touchpad, and has a hybrid pointing technique to quickly reach distant targets. Users can tap to instantly transport the cursor to a location onscreen, then slide for fine positioning, as with a regular touchpad. This combination makes ARC-Pad especially suited to large displays, even when the touchpad is very small.

I develop several modifications to ARC-Pad and evaluate them, including sending screenshots of the desktop to the phone's touchscreen and using accelerometer data to dynamically modify control-display gain. I perform a series of experiments to compare ARC-Pad to existing cursor positioning systems, and show that ARC-Pad achieves a higher throughput in a target selection task.

Finally, I extend ARC-Pad for 3D object positioning, and show that ARC-Pad comes very close to the performance of a 6DOF input device.

0.1 Acknowledgements

I would like to thank my advisor, Dr. Pourang Irani, and Myron Semegen from the Industrial Technology Center for their support and financial assistance.

I would also like to thank my fellow lab members for their input and assistance, and the participants who volunteered in the experiments.

Finally, I would like to thank Karen Kembel, for her faith and support.

Contents

List of Figures

List of Tables

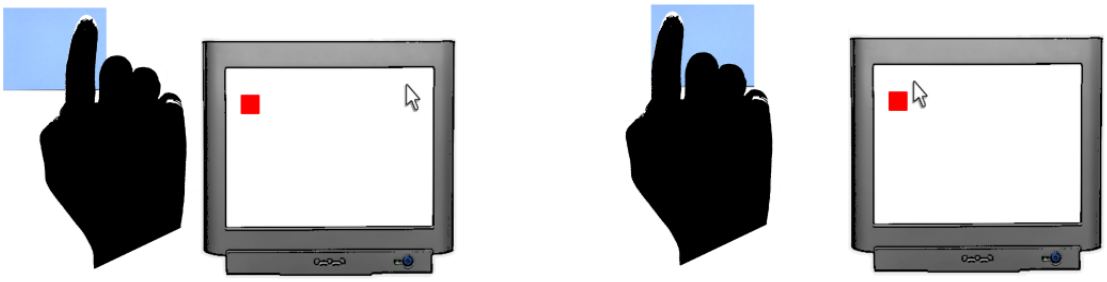
Chapter 1

Background

1.1 Introduction

ARC-Pad (Absolute-Relative Cursor Pad) is a cursor jumping technique that allows users to efficiently control a cursor, even on very large displays [18]. ARC-Pad is a variant of a touchpad, and uses the touchscreen on a mobile device. Sliding a finger over the touchscreen's surface moves the cursor as with a standard touchpad, but quickly tapping and releasing causes the cursor to jump to the corresponding location on-screen, allowing the cursor to quickly travel to distant areas. Users can roughly position the cursor in the proper area with a tap, and then slide to position the cursor more precisely.

Figure 1.1: Absolute Input With ARC-Pad



(Left) The user taps in the upper right corner of the touchpad, making the cursor jump to the equivalent location on-screen. (Right) The user taps in the upper left corner, and the cursor jumps to the upper-left corner on-screen.

ARC-Pad is designed for efficient cursor control where the input area is much smaller than the display area, and is especially geared towards mobile devices. It allows users to approach large displays and interact with them using personal devices that they already own. The cursor can be placed anywhere on the screen with minimal gesturing, and often only two gestures are required: a tap for coarse positioning followed by a brief sliding motion.

In this thesis I will present ARC-Pad's iterative design and evaluation. I first review metrics for evaluating cursor performance in a target selection task (Chapter 3). I then describe ARC-Pad's original implementation on a Windows mobile device, and show experimental results indicating that the absolute jumping improves user performance in a target selection task (Chapter 4). I then introduce several new features, and experimentally evaluate them to see which show benefit in a target selection task (Chapter 6). I combine the beneficial features to create a new version of ARC-Pad, which I experimentally compare to existing commercial solutions for manipulating a cursor on a large display (Chapter 7).

Finally, I introduce modifications that incorporate accelerometer data, and allow

the user to position a 3D cursor. The 3D version is evaluated against a wand that can sense its position and orientation in 3D space.

The experimental results suggest that ARC-Pad compares favourably against existing input devices, and is a viable solution for cursor control that does not require custom hardware. I show that ARC-Pad's selection time does not increase for distant targets, and that it maintains good accuracy for precise pointing.

1.2 Motivation

Mobile devices are becoming increasingly ubiquitous and sophisticated. There are many real-world situations where we can take advantage of the fact that people are carrying around powerful computing systems with built-in touchscreens and wireless communication. For example, when giving a slide presentation, the speaker must normally stand behind a laptop or computer terminal in order to switch between slides, or otherwise interact with the presentation software. The terminal screen may also serve as a kind of cue-card, and lets the speaker see the presentation without having to turn away from the audience and face the presentation screen.

With ARC-Pad, it becomes possible to control the presentation from a mobile device, such as a tablet or touchscreen phone. ARC-Pad is limited to cursor control, and can therefore be used seamlessly with any application that accepts mouse input. It becomes possible to stream images from the terminal to the touchscreen, and the presenter can check the slide show state without needing to stand behind a terminal screen. This allows the presenter to move around, and more fully engage the audience.

ARC-Pad can connect to nearby displays wirelessly, such as interactive screens or information booths. It becomes possible for users to manipulate displays that would normally be out of reach, such as an airport schedule, or download data directly to their personal device, such as the information from a kiosk or the map of a building.

Many interactive displays support touch input, however there are several drawbacks. The user's hand occludes the item that is being interacted with. Rear-projected displays introduce parallax error [8] due to the thickness of the glass, where the cursor does not appear to be directly beneath the user's finger. Items on large displays may be difficult to physically reach, and arm fatigue can occur after extended periods of use.

Fingertips are soft, and deform to cover a round area on the touchscreen, a phenomenon known as the 'Fat Finger Problem'. This makes precise selection on a touchscreen difficult, as there may be multiple targets beneath the area of contact. Vogel et al. [23] found the smallest target that can be reliably selected is a square with an area of 1.05 cm^2. A cursor allows much more precise targeting, as the activation area is typically a single pixel in size.

1.3 Terms

- *Absolute Position Control:* Every position of the input device is mapped to a unique cursor position. For example, a touchscreen.
- *ARC-Pad:* Absolute+Relative Cursor Pad, a mobile touchpad that uses a combination of relative and absolute motion.
- *CD Gain:* Control-Display Gain. A multiplier for mapping device movements to cursor movements. A high gain means that a small device movement results in a large cursor movement.
- *Clutching:* A movement of the input device that does not affect the cursor. For example, a user clutches each time she lifts the mouse and sets it down at a different position on the mouse pad.
- *Degrees of Freedom (DOF):* The number of rotational or translational dimen-

sions reported by an input device. A standard mouse has 2 DOF, as it can move the cursor along the x and y axes. A 3D input wand can have up to 6 DOF, from sensing both movement and rotation along the x, y and z axes.

- *Direct Input:* The input is located directly on the display. For example, a touchscreen.
- *Indirect Input:* The input is separate from the display. For example, a mouse or touchpad.
- *Position Control:* Users directly control the cursor's position with the input device. For example, a mouse.
- *Rate Control:* Users control the cursor's velocity. For example, joystick input.
- *Relative Position Control:* The cursor's movement is based on the distance from the input device's original position. Mice and conventional touchpads use relative control.

Chapter 2

Related Work

2.1 Indirect Input Techniques

These indirect input techniques allow us to control a cursor using a secondary device.

PRISM PRISM stands for Precise and Rapid Interaction through Scaled Manipulation, and is a technique for stabilizing 3D free-hand input by modifying the CD gain of a manipulated object. Frees et al. [9] administered a number of experiments where participants performed a docking task in a 3D virtual environment. They rotated or translated a 3D object into a goal area with the same shape as the object. When the user's hand moved slowly, the object would move more slowly still, gradually falling behind the hand's position. A fast motion would cause offset recovery, and the object would quickly return beneath the user's hand. PRISM employs a similar technique for rotation, except that the CD gain is based on the hand's rotational velocity.

Frees et al. [9] also used PRISM to control a virtual laser pointer which extended from the user's hand. Without PRISM, the laser would always point in the same direction as the hand. With PRISM enabled, slow rotational movements of the hand would decrease the CD gain, making the laser fall behind the hand's orientation. They ran an experiment where participants selected small objects by targeting them

with the laser. PRISM was able to compensate for small, involuntary movements in the user's hands, and had a lower error rate and completion time compared to the laser alone.

Rubber Edge Rubber Edge was developed by Casiez et al. [6] to control a cursor with a touchpad. A solid ring was placed in the center of the touchpad, attached to the touchpad's edges with elastic bands. Sliding a finger within the ring invokes relative positioning, as with a standard touchpad. Pushing a finger against the interior of the ring displaces it, and invokes rate control. The ring provides tactile feedback about the input mode being used, and the direction of the rate control. The cursor will initially move in the direction of motion given by the position control before the circle's edge was reached. It gradually smooths into the direction indicated by the finger's position on the ring.

For example, a user might slide her finger horizontally, and begin pushing against the ring at the 3:00 o'clock position. The rate control will initially move the cursor horizontally towards the right, and gradually adjust the direction of motion to the upper right. This adjustment reduces abrupt changes in the cursor's direction when switching between modes, while allowing the user to intuitively change direction in the rate control mode. Casiez et al. [6] did not report Rubber Edge's performance, but implemented a similar system using a Phantom haptic device, and found that target selection was 20% faster than with position control alone.

Ninja Cursors Kobayashi et al. [13] created Ninja Cursors, where the input device simultaneously moves multiple offset cursors scattered across the display. All cursors can select targets, allowing users to choose the cursor nearest to the intended target, then move it over the target to select it. Ninja Cursors suffers from a drawback in that multiple cursors could be simultaneously pointing at different targets. Kobayashi et al. resolved this problem by introducing a queueing algorithm, where all cursors

pointing at targets are frozen, and only the cursor closest to its target center remains active. If the active cursor leaves its target, Ninja Cursors activates the next-closest frozen cursor, and the process repeats until the desired cursor becomes active.

Kobayashi et al. found that users disliked the queueing algorithm, and that Ninja Cursors suffers when there are many targets on the screen, and when targets are large. However, they consistently found a significant speed improvement when using two cursors, even in situations with high target density.

Butter on a PDA Display Weberg et al. [24] provided cursor control that relied on accelerometer data. The user would push a button, then tilt the PDA in the desired direction.

Drag and Pick is a method of selecting distant targets by bringing them closer to the cursor. Baudisch et al. [4] created a system that incorporates a dragging gesture whose path specifies the angle of the intended target relative to the cursor. Targets with a specified angular distance from the dragging direction are duplicated and placed near the cursor for easy selection.

Drag and Throw was developed by Hascoët et al [11], and allows users to drag and release icons in a 'throwing' gesture, causing the icon to be launched across the screen towards the intended destination. The gesture most closely approximates an archery metaphor, where the item is dragged away from the destination, then released to launch it. A line is drawn on the screen, showing the path the item would take if released. In a sense, the user is controlling the endpoint of the line as if it were a cursor with inverted movement, making it functionally very similar to using cursor acceleration alone. Similarly, Hascoët et al.'s push and throw technique creates a second cursor with a higher CD gain than the original, allowing the user to quickly reach distant locations.

2.2 Direct Input Techniques

Direct input techniques use the surface of the display to manipulate the cursor position.

HybridPointing Forlines et al. [8] developed HybridPointing, which allows users to switch between absolute and relative modes when controlling a cursor on a large touchscreen. They used a Vicon motion tracking system to track a wand's position near the screen. In relative mode, the display acted as a large touchpad. Sliding the wand across the display's surface moved the cursor, and moving the wand away from the surface invoked clutching. In absolute mode, the cursor moved directly beneath the wand whenever it touched the display.

Forlines et al. [8] ran target selection experiments, and found that using absolute mode alone was fastest when the target was no more than 2 meters from the wand. After 2 meters, relative mode became faster, since the users could select the targets without having to physically move to them. They attempted to combine both modes to achieve the best of both worlds, creating HybridPointing.

Forlines et al. added a 'trailing widget' that followed the cursor, represented by a circle. Quickly tapping the circle with the wand switches to relative mode. Lifting the wand more than 8 cm from the screen reverts to absolute mode. They also added substantial visual feedback. They provided an animation of a comet tail between the cursor and wand when switching to relative mode. While in relative mode, they drew a partial line between the wand and cursor, and caused the trailing widget to shrink and move beneath the wand.

Surprisingly, the hybrid mode's performance was virtually identical to using relative mode alone, except that it was consistently 250 milliseconds slower per target. Forlines et al. [8] speculated that this may be due to the cognitive overhead of managing the different modes. Participants would find the target, decide which mode was

most suitable, switch modes if necessary, and then select it. ARC-Pad was strongly influenced by HybridPointing, and the importance of eliminating this overhead. Rather than have users switch between two modes, in ARC-Pad both relative and absolute motion are simultaneously available with two separate gestures.

Offset Cursor Potter et al. [21] introduced the 'take off' cursor for touchscreens, which is now commonly known as the 'Offset Cursor' by HCI researchers. A cursor is displayed half an inch above the user's finger, and follows the finger as it slides across the touchscreen. When the user lifts her finger, the item beneath the cursor is selected. The Offset Cursor significantly increases accuracy, but Potter et al. found that it increased selection times by an average of 4 seconds. More recent studies have found that the extra selection time varies from 250 to 1000 milliseconds [23]. Offset Cursor prevents the selection of targets at the bottom of the display.

Shift Vogel et al. [23] designed the Shift technique, where the circular area below the user's finger is redrawn to a nearby location. The circle contains small cross-hairs that indicate the selection location, allowing the user to finely adjust the position before making a selection. Shift achieved significantly better speeds than Offset Cursor for selecting large targets, while maintaining a low error rate for small targets.

2.3 3D Object Translation

In 3D object translation, users move a virtual object through the x, y and z coordinates. For example, a modelling task might require users to position items within a virtual building. These 3D input techniques are able to extract motion along the three axes from a 2D input device.

First-Person View Plus Map View (FPV+M) Standard 3D editors divide the screen into quarters, each representing a virtual camera's viewpoint on the graphics scene. One quarter shows a perspective view, and the other three are isometric viewports; top-down, forward-back, and left-right. This allows 3 DOF with a 2D input device, as we can use a mouse to manipulate an object by clicking it through the desired viewport [17].

However, the extra viewports add a cognitive load to the user, who must interpret information from different camera angles. Chittaro et al. [7] report that a simplified 2-view layout is the preferred approach for novice users. They created a system that shows a top-down overview map together with a first-person perspective. Furthermore, the zoom level in the map is automatically adjusted so that selected objects are always completely visible.

They found that FPV+M allowed users to perform rotation, scaling and translation tasks with fewer actions compared to a first-person view alone, and provided faster navigation and object manipulation. I built upon their ideas when designing 3D ARC-Pad, and provided a separate overhead view displayed on the iPod touchscreen.

Z-Technique Martinet et al. [17] introduce the Z-technique to provide 3 DOF with a multi-touch surface. The surface displays a first-person view of the 3D scene. When manipulating an object, the first finger on the user's dominant hand performs 2D translation on the camera plane, and a second finger controls depth. Clutching is achieved by lifting and re-positioning the second finger on the screen. Martinet et al. recommend using Z-technique with mobile phones, as we can display a second viewport on the phone without occluding the screen.

Triad Cursor Triad Cursor is a simple method of obtaining 3 DOF from a 2 DOF input device [20]. The graphics scene's x, y and z axes are displayed on the screen. The user moves a 2D cursor along a path parallel to one of the axes, and the object is

moved in the corresponding direction. Triad Cursor only moves the object along one axis at a time, even if there is a little motion from the cursor in the other directions.

Chapter 3

Cursor Control and Metrics

This section describes the metrics I used to evaluate ARC-Pad's performance. These metrics allow us to compare input devices based on their speed and precision, and measure their utility for target selection with a cursor. I used them to validate ARC-Pad's design at each step of its development.

3.1 ISO Guidelines

I chose to use ISO 9241-9 to measure ARC-Pad's performance, which is an international standard for evaluating input devices [12]. The standard lists over 60 design rules that devices should conform to. Rather than list them all, I will provide the few rules that ARC-Pad does not comply with:

Anchoring The ISO standard states that the user should be able to anchor a part of her hand or arm on the device or work area to stabilize the hand. As ARC-Pad is implemented on a mobile device, meeting this guideline is impossible.

Button Force Buttons should require between 0.5 N and 1.5 N of force to activate. The new version of ARC-Pad does not use physical buttons, but the touchscreen serves

the same purpose. Since the touchscreen is capacitive, even the lightest contact is sufficient to register a touch.

Inadvertent Pointer Movement Accidentally pushing a button should not move the cursor. If we consider the touchscreen to act as a button, then accidentally touching the screen will move the cursor to a new location.

3.2 Standard Metrics

While Human Computer Interaction (HCI) researchers commonly use the terms 'error rate' and 'completion time', their definitions often change subtly from paper to paper. The ISO 9241-9 provides a precise definition for each [12]:

Error Rate is given as the percentage of clicks that occurred outside of the target.

Movement Time (t_m) is the interval between the beginning of the user's movement to the time when the target is selected.

3.3 Fitts' Law

Fitts' Law was developed by Paul Fitts in 1954, and predicts the time needed to select a target. It states that the selection time is a function of the target size and its distance to the cursor. Fitts' law has performed remarkably well, regardless of the type of input device or muscle groups used [14]. It is based on Shannon's Theorem for calculating a channel's information capacity in the presence of noise:

$$C = Blog_2((S + N)/N) \tag{3.1}$$

where

S is the signal strength

N is the channel noise

B is the channel bandwidth in Hz

C is the channel capacity in bits per second

Fitts stated that the distance to the target is equivalent to the signal power, and that the target size is equivalent to the amount of noise on the channel. The original formulation of Fitts' Law is as follows:

$$MT = a + blog_2(2A/W) \tag{3.2}$$

where

MT is the movement time

A is the amplitude, or distance to the target

W is the target width in the direction of motion

a and b are constants

Fitts law does very well at predicting movement times for rapid pointing tasks, where the user is attempting to select a target as quickly as possible. It requires the user to be an expert at the task, as learning effects may cause the movement time to decrease as the participant becomes more skilled. While Fitts' Law performs well, it is not intuitive why human movements should match a model originally designed to measure a channel's information capacity.

Perhaps the best explanation we have is the Deterministic Iterative Corrections Model (DICM), which proposes that human pointing is performed through a series of feedback and correction steps [14]. At each step, the pointing device is moved closer to the target. The participant then visually examines the pointing device's new location relative to the target, and performs another movement towards it. The

DICM states that the time to obtain visual feedback and perform the movement is a constant t. Experiments have found that t varies between 135 and 290 ms on different selection tasks [14].

The movement to the target is therefore broken up into n submovements, each with time t. During each movement, participants move $(1 - p) \times$ remaining distance, where p is a constant. p has been experimentally found to lie between 0.4 and 0.7 [14]. Researchers have found that modelling selection time with the DICM gives very similar results to Fitts' law, suggesting that Fitts' law models the same iterative-correction process.

3.4 Throughput

The movement time predicted by Fitts' Law tends to be significantly lower than the actual movement time when the task is easy, and target is large or near the pointing device [14]. Furthermore, Fitts' law assumes that the contributions of amplitude and target width are equal but inverse. However, studies have shown that reducing target width by one half has a greater effect than doubling the amplitude [14]. The throughput formula in the ISO [12] addresses these problems:

$$Throughput = \frac{ID_e}{T_m} \tag{3.3}$$

where

ID_e is the effective index of difficulty, measured in bits

T_m is the movement time

The Effective Index of Difficulty is the precision achieved by the user when selecting a target with the device [12]. It is given in bits:

$$ID_e = log_2 \frac{d + w_e}{w_e} \tag{3.4}$$

where

d is the distance from the cursor to the target

w_e is the effective target width

Finally, *the Effective Target Width* is based on the standard deviation of cursor coordinates in the direction of motion towards the target.

$$w_e = 4.133 s_x \tag{3.5}$$

where s_x is the standard deviation of selection coordinates in the direction of motion.

Formula 3.3 gives us the throughput of the input device, in bits/second. Throughput is a standard metric that allows us to compare the utility of input devices for pointing and selection tasks [22]. For example, a mouse has a higher throughput than a touchpad [16]. The mouse is a 'better' device in the sense that it provides better performance for target selection.

There are two components to device performance: speed and accuracy. These are inversely related; if a user should select targets very quickly, her accuracy will suffer. If she takes care to select each one precisely, her speed will be reduced. This is known as the speed-accuracy tradeoff [15]. Speed and accuracy are functions of both the user and the input device. Some devices may provide fast target selection at the expense of a high error rate, or vice-versa. Throughput combines both speed and accuracy, and allows us to compare the utility of input devices with different characteristics.

The effective target width is a measure of the users' accuracy with the device. Inaccurate devices have a higher deviation in selection coordinates, as users click farther from the target's center, or even outside of the target. The constant 4.133

from formula 3.5 comes from information theory. When participants select targets, they produce a normal distribution of selection coordinates clustered around the target center. The amount of entropy in a normal distribution is given as:

$$log_2(\sqrt{2\pi e}\sigma) \tag{3.6}$$

where σ is the standard deviation of coordinates in the direction of movement

Substituting values for π and Euler's number (e), we get:

$$log_2(\sqrt{2 \times 3.14159 \times 2.71828}\sigma) = log_2(\sqrt{17.0794}\sigma) = log_2(4.133\sigma) \tag{3.7}$$

The constant of 4.133 standard deviations covers 96% of the normal distribution's area. Therefore, in order for the the target width to be equivalent to noise, the selection coordinates must form a normal distribution with 96% of the selection coordinates occurring inside of the target. 4% of selections must be errors. (They occur outside of the target.)

The effective target width normalizes target width, and increases the accuracy of throughput calculations. By basing the width on user performance, and using 4.133 standard deviations, we ensure that the measured error rate is always 4%.

The effective width is used to calculate the effective index of difficulty from formula 3.4. It measures the performance achieved by the user, in bits. It increases by one bit when the distance to the target is doubled, and also when the target size is halved in the direction of motion. The index of difficulty is a measure of the information conveyed to the cursor in order to select a target. As the task increases in difficulty, the amount of information needed to select the target increases as well.

As the effective index of difficulty increases, the movement time increases to keep ID/T roughly constant [15]. In other words, the throughput from formula 3.3 should remain constant regardless of the task difficulty. It allows us to reason about the

performance of different devices used for pointing tasks, even if the method of pointing is very different.

Zhai has noted that the throughput calculation does not preserve the constants a and b from formula 3.2, and that that the value obtained for throughput is therefore dependent on the target sizes and distances used in the experiment [25]. In my experiments, I will use the same target settings for all techniques, allowing me to compare their throughput in a meaningful way.

3.5 Accuracy Measures

MacKenzie et al. [16] have proposed several additional metrics to assist in understanding an input device's performance. The intention is to examine why the device achieves its scores on the standard metrics, allowing us to adjust or re-design the technique to fix problem areas.

Target Re-Entry (TRE) TRE occurs whenever the cursor leaves the target and re-enters it. It is given as the number of re-entries divided by the number of trials. TRE measures fine positioning ability.

Movement Direction Change (MDC) The cursor temporarily moves away from the target before reaching it. A significant number of direction changes can indicate a problem with the input device.

Movement Variability (MV) MV is the extent to which the path followed by the cursor forms a straight line.

3.6 User Preferences

After completing the experiment, users will rate each technique on a 5-point Likert scale, where 1 is "Strongly Disagree", 3 is "Neutral", and 5 is "Strongly Agree". The statements to be evaluated are:

- This technique was easy to use.
- This technique was fast.
- This technique had good control. (few errors)
- This technique was comfortable, with no hand or arm strain.
- Overall, I like this technique.

Chapter 4

Experiment One

The purpose of experiment one was to examine the use of hybrid absolute and relative input for selecting targets on large displays. I wanted to know if absolute jumping showed benefit for selecting distant targets, and was interested in avoiding the performance degradation seen by Forlines et al. [8] in their HybridPointing technique. I also wanted to investigate whether any visual feedback should accompany absolute jumping. To this end, I conducted an experiment comparing ARC-Pad to a standard touchpad.

4.1 ARC-Pad

The original ARC-Pad was a touchpad implemented on an HTC TouchDiamond mobile phone. The phone's touchscreen obtained the finger's position, and sent it to a server running on a desktop. ARC-Pad features relative input, and moves the cursor as the user's finger slides across the screen. It also provides absolute input, and a quick tap and release causes the cursor to instantly jump to the corresponding location on the desktop. For example, tapping in the lower-left corner on the touchscreen would 'jump' the cursor to the lower-left corner on the desktop. ARC-Pad attempts to reduce or eliminate clutching, which has been shown to degrade performance [5].

I found that users were faster at selecting targets with absolute jumping enabled, without any significant loss of accuracy.

4.1.1 Relative Input

Relative input translates sliding gestures on the touchpad into cursor movements using Control-Display (CD) gain and cursor acceleration.

CD gain is given as $\text{velocity}_{pointer}/\text{velocity}_{device}$. A CD gain of 10 means that the cursor moves 10 cm for every 1 cm moved by the input device. To improve selection speed, we might simply increase the CD gain so that only small movements of the input device are required. However, a high CD gain can cause quantization problems [5]. As CD gain increases, it becomes impossible to select some pixels, as the smallest motion of the device corresponds to a movement of more than one pixel on screen. Increasing CD gain also lowers accuracy because of the speed-accuracy tradeoff, making it difficult to precisely position the cursor.

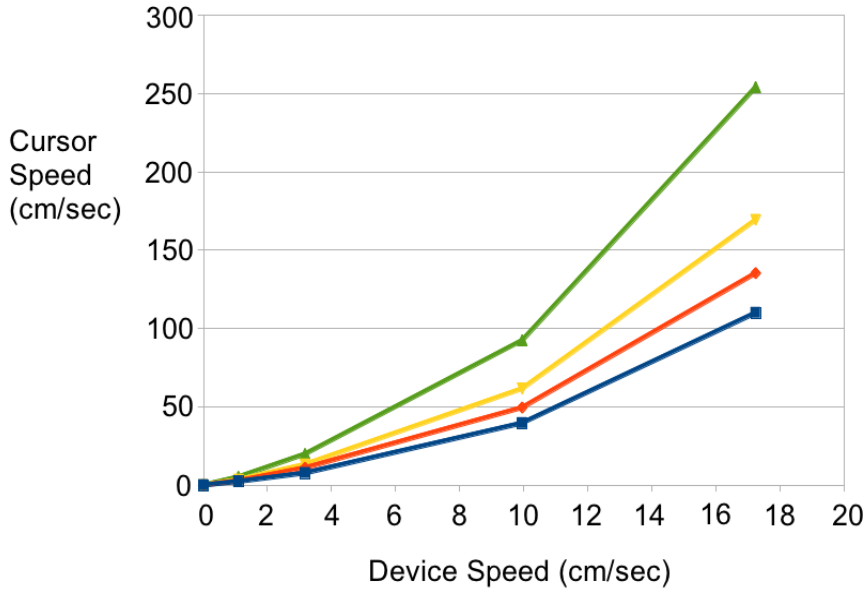
Prior work has shown that adjusting the CD gain has no effect on performance as long as it is high enough to avoid significant clutching, and low enough so the limits of human motor control are not reached [5]. If the display is sufficiently large no value of CD gain can meet these criteria, as a CD gain high enough to let users reach distant targets without clutching is hopelessly inaccurate.

Cursor acceleration is commonly used on modern desktop computers to improve selection times [19], and mitigate the effects of the speed-accuracy tradeoff. It dynamically adjusts the mouse's CD gain based on the mouse velocity. The CD gain is reduced when the mouse velocity is low, since the target is probably nearby and precise movement is desired. It is increased when the mouse velocity is high, as the target is probably far away. The adjustment is based on Fitts' law, which states that the movement time will be higher when the user desires precision [9].

ARC-Pad used the default level of cursor acceleration from Windows XP [19] for

relative motion, which is the red line from figure 4.1.

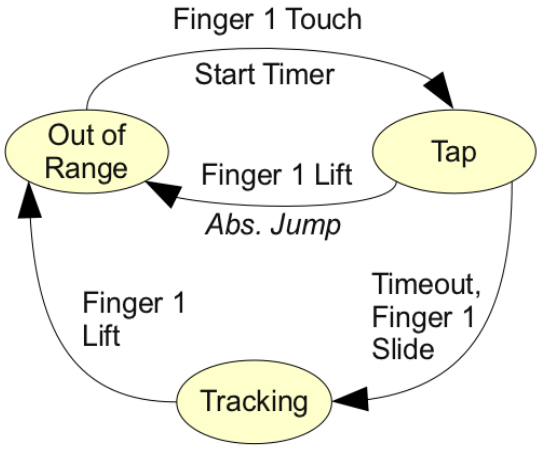
Figure 4.1: The Cursor Acceleration Curves From Windows XP



4.1.2 Absolute Input

Figure 4.2 shows the mechanism for absolute input. A touch quickly followed by a release produces an absolute jump, and relative motion is provided as in a standard touchpad by the Tracking state. The transition from Tap to Tracking can occur in two ways; after a delay of 300 milliseconds, or if the finger slides more than 0.35 cm. These values were obtained from a simple pilot study, and allows the algorithm to reliably distinguish between tapping and sliding gestures.

Figure 4.2: The Logic For Absolute And Relative Motion



If the aspect ratios of the touchscreen and the primary display don't match, the jumping coordinates are scaled appropriately.

4.1.3 Implementation

The mobile device was an HTC TouchDiamond with a 640×480 resolution. It connected to a desktop computer through TCP over bluetooth. The desktop computer powered a 52 inch LCD television with a resolution of 1360×768, and 1.2 pixels per mm. The software was written in C#, using the .NET API.

Since the TouchDiamond is not a multi-touch device, performing a mouse click was accomplished by pushing a physical button at the bottom of the screen.

4.2 Techniques

4.2.1 ARC-Pad

ARC-Pad was implemented as described in section 4.1 above.

4.2.2 Cursor Acceleration

The Cursor Acceleration technique (CA) was identical to ARC-Pad, but did not provide absolute jumps.

4.3 Task

I used a reciprocal tapping task similar to the one developed by Forlines et al [8]. Participants would select a square target, causing another target to appear elsewhere on the screen.

4.4 Experiment Setup

Nine undergraduate computer science students participated in the experiment, in exchange for course credit. We used a within-participants design, and counterbalanced the order of the two techniques.

Target Sizes: 9, 15 and 21 pixels

Target Distances: 625, 937 and 1250 pixels

For each technique, participants were given twenty practice trials followed by 30 trials for each combination of size and distance. 3 sizes $\times$ 3 distances $\times$ 30 trials gives us 180 trials per technique. With two techniques and 9 participants, we get 3240 trials total.

I instructed participants to hold the device at a 90 degree angle to more closely match the television's aspect ratio, and to proceed as quickly and accurately as possible.

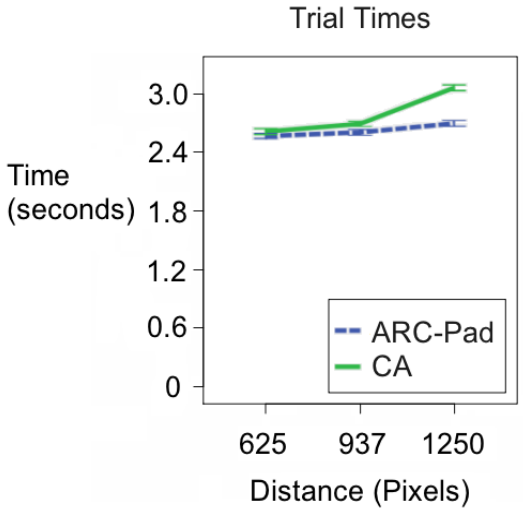
4.5 Experiment One Results

My advisor, Pourang Irani, helped with the analysis of data from Experiment One with SPSS. For the remainder of the experiments, I alone performed the data analysis using R Commander.

We used a univariate ANOVA on trial times, and found a significant effect of target *distance* ($F_{2,16} = 55.1$, $p < 0.001$) and *size* ($F_{2,16} = 97.74$, $p < 0.001$). There was also a significant effect of *technique* ($F_{1,8} = 7.67$, $p < 0.05$), and an interaction effect between *technique* and *distance* ($F_{2,16} = 3.67$, $p < 0.05$).

Pairwise Tamhane comparisons showed significant differences for all combinations of *distance* and *size* ($p < 0.001$).

Figure 4.3: Trial Times For The Original ARC-Pad



Bars are one standard error.

The average completion time across all sizes and distances was 2.623 seconds for ARC-Pad, compared to 2.789 seconds for CA. In figure 4.3, ARC-Pad showed increased benefit at distances of 937 pixels and higher, suggesting that the absolute jumping mechanism successfully improves target selection times over long distances.

4.6 Discussion

ARC-Pad was able to outperform CA for larger distances, and matched its performance for shorter distances. This would indicate that any extra cognitive overhead from managing both absolute and relative gesturing has a negligible effect. I was also concerned about users' abilities to keep track of the cursor after an absolute jump, but did not notice any difficulties. The cursor was easy to track even after a jump, as it would typically be in motion due to continued relative input. Therefore, I did not see the need to add any visual feedback.

Finally, over half the users preferred ARC-Pad to CA. Those users who preferred CA commented that ARC-Pad was unfamiliar. Because the TouchDiamond is meant to be used with a stylus rather than a finger tip, the sensitivity was low and I witnessed multiple unwanted absolute jumps. The fact that we exceeded the CA technique even with this limitation suggests that we should get very favourable performance with more sensitive hardware.

Chapter 5

Modifications to ARC-Pad

In this section I apply lessons learned from experiment one, and attempt to improve ARC-Pad's perormance by changing settings, and implementing it on a more suitable platform. I change the cursor acceleration settings, use a touchscreen that is more sensitive to touches, and add multi-touch gestures for clicking.

5.1 Cursor Acceleration

I conducted a pilot study to determine the ideal amount of cursor acceleration to use. Microsoft has published a model of the cursor acceleration used in Windows XP, and I duplicated it in the ARC-Pad software [19]. I found that the highest level of acceleration had the lowest completion time, and did not significantly increase error rate. The pilot was conducted using an iPod touch as a mobile touchpad.

5.2 Hardware

I implemented the new ARC-Pad on an 8 GB iPod Touch, model MB528C [2]. The iPod touch has a capacitive-based screen, where the TouchDiamond from the previous version of ARC-Pad was designed to be used with a stylus. The TouchDiamond did

not reliably detect fingertips, and I instructed participants to touch the screen with their nails. This resulted in several problems. Some participants had very short nails, and the TouchDiamond would periodically lose track of their finger position. This made the cursor appear to suffer from latency, as the number of cursor updates per second was reduced. Other participants had longer nails, and found that their fingers tended to slide on the screen, reducing absolute jumping accuracy.

To assist users in determining the edges of the iPod's screen, I placed a layer of scotch tape around its border, creating a raised surface. Without the tape, users tended to slide their fingers beyond the screen edge, and the cursor would stop moving for no apparent reason. I found that the iPod touch will ignore touches where the finger first touches to an off-screen location, and then slides onto the screen. Again, the raised border helped users avoid this situation.

The iPod screen is 5 x 7.1 cm, with a resolution of 480 x 640 pixels. The primary display (containing the cursor being manipulated) is a back-projected screen 7.26 x 3.03 m in size, with a resolution of 2560 x 1024 pixels. The ARC-Pad server ran on an HP wx9400 Workstation, with 4 dual-core processors and 8 GB of RAM.

5.3 Software

The new ARC-Pad server consists of over 30,000 lines of C++ code, and makes extensive use of the Qt library. I used Qt to move the cursor in response to gestures on the iPod, and the xdotool utility to simulate mouse clicks whenever a selection was made. I implemented the 3D experiment graphics with OpenGL. The server ran on the Red Hat Enterprise Linux 5 operating system.

Finger touches, moves and lifts were received from the iPod with TCP over WiFi 802.11n. The latency to the device was sampled once every 500 ms, and added to a running average with five items:

$$pA = pA\frac{hs-1}{hs} + \frac{pT}{hs} \tag{5.1}$$

where

pA is the running average

hs is the size of the running average

pT is the currently measured latency

The value of pA was stored at the end of each trial.

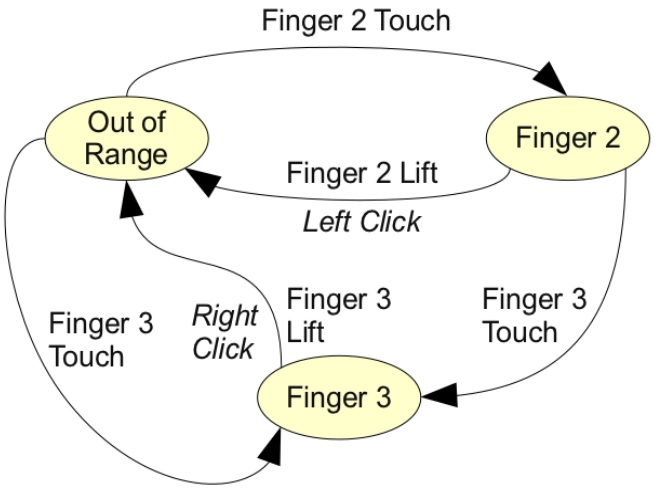
5.4 Multitouch

I developed a variant of ARC-Pad suitable for use on an iPod touch. As I am using the tap gesture to indicate an absolute jump, additional gestures are required for mouse clicks. The original ARC-Pad relied on a separate, physical button located on the device. In the new version, I allow the user to tap with a second or third finger to initiate a click.

Touches are disambiguated into primary, secondary, and tertiary touches before being sent to the finite state machines in figures 4.2 and 5.1, allowing higher-level reasoning about touch states. The primary finger is the first finger to touch the screen, the secondary touch is the second finger, and so forth.

The Out of Range state indicates that no fingers are touching the device. In figure 5.1, notice that it is possible to receive a touch from the secondary finger while in the Out of Range state. I employ an algorithm that matches fingers to touch coordinates based on distance. If it receives a lift event that is rapidly followed by a touch at the same position, it assumes that the touches belong to the same finger. Thus, it is possible to have touches from the secondary and tertiary fingers, even when in the Out of Range state.

Figure 5.1: The Logic For Right And Left Clicking



5.5 Matching Fingers to Touches

When touch information is received by the server, it checks if it is a new touch, or a continuation of a previous event sequence. For example, the user might be sliding his finger across the device, producing a sequence of new touch data. The following algorithm determines if the touch belongs to an existing event sequence, or if it is the beginning of a sequence from a new finger.

The server stores touches in three variables, $T0$ through $T2$. Each touch contains the following information:

T_x Touch T's x-coordinate.

T_y Touch T's y-coordinate.

T_{type} Touch T's type, one of LIFT, PRESS, SLIDE or NONE. It indicates the most recent event for this touch, or NONE if we are not currently tracking this finger.

T_{match} Is set to TRUE if this touch is a candidate to match to a previous event, FALSE otherwise.

T_{time} The time when we received a LIFT event for this touch, in milliseconds.

The variables are ordered such that $T0$ always contains the primary touch, $T1$ contains the secondary touch, and so forth. When a group of one or more touch events is received, we first check which of $T0$, $T1$ or $T2$ are potential candidates to match to the event(s).

for each touch i

if Ti_{type} = LIFT

if current time - Ti_{time} >MAX_MATCH_TIME

$T_{type} \leftarrow$ NONE

if Ti_{type} = NONE

$Ti_{match} \leftarrow$ FALSE

else

$Ti_{match} \leftarrow$ TRUE

Here, we're disregarding any touches where the finger has been lifted longer than MAX_MATCH_TIME, which is 300 milliseconds. If that finger should touch the screen again, it will be counted as a new event sequence. All touches whose types are not NONE are valid candidates to match to the current sequences.

The events we receive are stored in three variables $E0$ through $E2$, with the same data fields as the touches. The following algorithm sorts through the touch data, and determines which events belong to the primary, secondary and tertiary fingers.

for each event k

$f \leftarrow$ FALSE

$s \leftarrow \infty$

for each touch i

if Ti_{match} = TRUE

if match_valid(Ek, Ti)

$d \leftarrow$ distance between Ek and Ti

if $d < s$

$s \leftarrow d$

$I = i$

$f \leftarrow$ TRUE

if f = TRUE

$TI \leftarrow Ek$

$TI_{match} \leftarrow$ FALSE

We iterate through each event Ek, and search for the closest existing touch Ti that is a valid match. If we find a match, f is set to true, and the event is stored in the appropriate touch variable. We determine if a match is valid based on the type of event versus the type of the touch variable:

match $\leftarrow$ TRUE

if Ti_{type} = LIFT and $Ek_{type} \neq$ PRESS

match $\leftarrow$ FALSE

if Ti_{type} = PRESS and Ek_{type} = PRESS

match $\leftarrow$ FALSE

if Ti_{type} = SLIDE and Ek_{type} = PRESS

match $\leftarrow$ FALSE

return match

For example, if the most recent event in a sequence was a lift, we know that the only possible matching event is a press. Once the algorithm has completed, the variables $T0$ - $T2$ will be set to the closest valid event.

Chapter 6

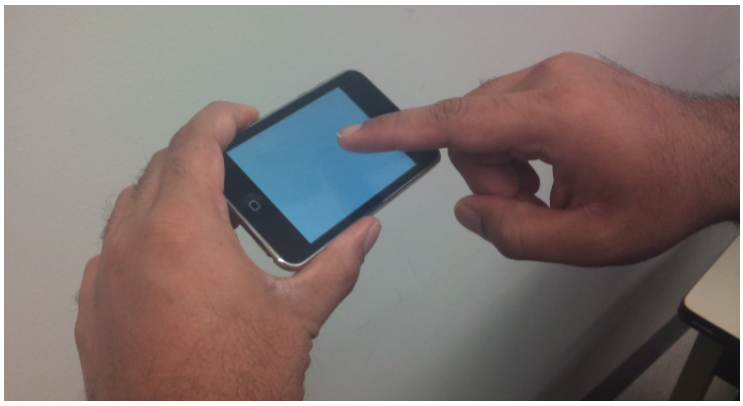
Experiment Two

In experiment two, I continue ARC-Pad's iterative design, and introduce several new features to the modified ARC-Pad version from section 5. Each feature is associated with one of the techniques seen below. In this experiment, I evaluate each individual feature to see which have benefit in terms of accuracy and error rate. ARC-Pad's final design will be based on the techniques that show benefit here.

6.1 Techniques

6.1.1 Touchpad

Figure 6.1: A User Holding The iPod Touch



Touchpad was the base condition, as described in chapter 4.1. However, absolute jumping was not activated, and tapping with the primary finger had no effect. Users were instructed to hold the the device at a 90 degree angle as in figure 6.1, so that the aspect ratio on the device was closer to the aspect ratio of the experiment screen.

6.1.2 Tiltpad

Tiltpad is similar to Touchpad, but allows users to change the amount of CD gain by tilting the iPod. The CD gain is first calculated using cursor acceleration as in the the Basic ARC-Pad technique. I then apply a multiplier to the gain, which varies from 0.5 to 2 depending on how the device is tilted. When the device is horizontal, the multiplier is 1. The multiplier is 2 when the device is tilted 45 degrees away from the user, and 0.5 when it is titled 45 degrees towards the user. Tilting the device more than 45 degrees has no effect.

The idea is that users can manually control the cursor speed, adjusting the device to obtain either increased speed or accuracy.

6.1.3 ARC-Pad

ARC-Pad was implemented as described in sections 4.1 and 5.

6.1.4 Screenshots

The Screenshots technique is similar to ARC-Pad, but continually streams screenshots of the experiment screen to the device. The goal is to improve the accuracy of absolute jumping by displaying the targets on the iPod screen. The user can tap directly on the target to move the cursor there, and slide if additional fine positioning is required. The potential downside is that it divides the user's focus between the primary display and the iPod, which may be distracting.

Screenshots are sent over the same TCP connection as the touches, and the bandwidth is limited to 25 kilobytes/second. I achieved 2.4 frames per second, and the primary limiting factor was the time to obtain a screenshot and scale it. Graphics processing is performed in a separate thread to keep the experiment responsive. I used the 'xwd' tool to save the screenshot to the ramdrive located in /dev/shm, which takes roughly 210 milliseconds. The exact command is "xwd -root -screen -silent -out /dev/shm/screenshot".

The screenshot is then loaded, scaled and rotated to fit the device screen, another expensive operation as the screenshot's original resolution is 2560 by 1024. Note that the image is slightly stretched to fill the iPod screen, as the aspect ratios of the iPod and experiment screens do not match. I then compress the screenshot to the JPEG format, varying the image quality to restrict the bandwidth to 25 KB/second. Finally, the image is sent to the iPod, which decompresses and displays it.

Under this scheme, the average compressed image size was around 20 KB. Sending such a large amount of data in a single burst interfered with the reception of touch coordinates, and introduced noticeable latency in cursor movements. I addressed this problem by splitting the image into 'chunks' of 8 KB each, and periodically sending

them at 300 millisecond intervals.

The image quality is varied using an algorithm inspired by the TCP congestion control mechanism [1]:

1. $increase \leftarrow reduce \leftarrow$ FALSE
2. $send \leftarrow$ TRUE
3. if $f_{size} > 65535$

   halve_quality()
4. else if $f_{size} > 49152$

   $reduce \leftarrow$ TRUE
5. if $send =$ TRUE and queued bytes >8192

   halve_quality()
6. $ideal\_size \leftarrow$ bandwidth / images per second
7. if $send =$ TRUE and $send\_limiter > ideal\_size \times 2$

   $reduce \leftarrow$ TRUE

   $send \leftarrow$ FALSE

The above algorithm uses the following variables:

increase Set to TRUE if we should incrementally increase image quality.

reduce Set to TRUE if we should incrementally reduce image quality. *increase* has no effect when reduce is TRUE.

send Set to TRUE if we should send this frame, FALSE if we should skip it, lowering our frames per second.

f The current frame to send, compressed to the JPEG format.

f_{size} The number of bytes in the compressed frame.

ideal_size The frame size that would result in 25 KB per second of bandwidth being used.

send_limiter Is periodically decremented by 25 KB per second, and incremented by each f_{size} from the frames we send. We attempt to incrementally adjust the image quality such that *send_limiter* is between *ideal_size* and *ideal_size* $\times 2$.

Lines 1-4 The above algorithm begins with the current frame already compressed, using the image quality computed the previous time the algorithm was run. It begins by reducing the image quality by one half if the frame size is larger than 64 KB. 64 KB is imposed as an extreme upper limit on the frame size, as at 25 KB per second we would only be sending one frame every 2.56 seconds. Next, it incrementally reduces the image quality if the frame size is larger than 48 KB. When the server's desktop is a complex image, the compressed and scaled frame can be larger than 40 KB, even with very low-quality JPEG compression. However, I prefer to reduce image quality as much as possible at this point to maintain a reasonable number of frames per second.

Line 5 Next, the algorithm checks for network congestion. If there are more than 8 KB of data to send from previous screenshots, the current screenshot is not sent and image quality is halved. (8 KB is the size of one chunk, as described above.) This allows us to reduce our send rate if the network is unable to provide transfer speeds of 25 KB per second.

Lines 6-7 The algorithm keeps track of the number of items it has sent in the last five seconds, and uses this information to calculate the 'images per second' seen on line

6. We then calculate the image size that would produce the desired bandwidth, given the number of images per second. This value is stored in *ideal_size*. The *send_limiter* variable ensures that we consistently use 25 KB per second. It is decremented by 2560 ten times per second, but is not allowed to go below zero. It is incremented by the size of each compressed frame that is sent. On line 7, we will only send this frame if the *send_limiter* is no more than twice the *ideal_size*. Otherwise, we reduce the image quality and discard this frame. The *send_limiter* can be thought of as the available send capacity. As it is continually decremented at a rate of 25 KB per second, it produces a fixed send rate, even if f_{size} or the images per second vary.

Finally, the frame is sent and the quality is adjusted:

```
if send = TRUE
    1. send_limiter ← send_limiter + f_size
    2. if send_limiter > ideal_size
           increase ← FALSE
           if send_limiter > prev_limiter
               reduce ← TRUE
    3. prev_limiter ← send_limiter
```

On line 1, we increase the *send_limiter* by the size of the current frame. On line 2, we set *increase* to FALSE if the *send_limiter* is greater than the *ideal_size*. The reasoning is that the current *quality* value appears to be producing the desired bandwidth; we're lower than the threshold of $ideal\_size \times 2$, but higher than the *ideal_size* size. The *quality* value is maintained until the *send_limiter* either falls below the *ideal_image* size, or exceeds $ideal\_size \times 2$.

We then compare the *send_limiter* against its value the previous time this algorithm was called. If it is greater, then we know that *send_limiter* is increasing, even

though it is already larger than *ideal_size*, and *reduce* is set to TRUE to compensate. In line 3, we save *send_limiter* in *prev_limiter*, to be used the next time a frame is sent.

The JPEG *quality* ranges from 1 (most lossy compression) to 100 (least lossy compression). When *reduce* is TRUE, we reduce the *quality* by 4 after the frame is sent. If *increase* is TRUE and *reduce* is FALSE, we increase it by 1. The above algorithm mimics TCP congestion control in that it reduces the image *quality* by half when congestion is detected. However, we also know the predetermined amount we should be sending over this channel, which in this case is 25 KB per second. Therefore, we can adjust image *quality* incrementally as we approach this limit, resulting in less *quality* variability. If *quality* is 1 and either congestion is detected or the 25 KB limit is exceeded, the algorithm starts skipping frames.

6.2 Tasks

The tasks were selected from the International Organization for Standardization (ISO) document 9241-9: "Ergonomic requirements for office work with visual display terminals (VDTs) Part 9: Requirements for non-keyboard input devices" [12]. The ISO 9241-9 provides a standard methodology of testing input devices that control a cursor, and was suggested to me by the reviewers of the original ARC-Pad paper [18].

The two tasks that are appropriate for ARC-Pad are the one direction tapping task and the multi-directional tapping task.

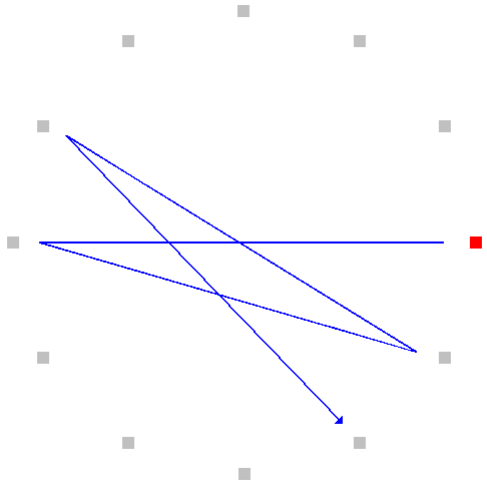
6.2.1 1D Tapping

Figure 6.2: 1D Tapping Task Screenshot

The 1D Tapping task measures horizontal positioning. In figure 6.2 there are two targets, each of which has width w, and a height that covers the entire screen. The goal is to select the red target, which will cause it to turn Gray, and the opposite target to turn red. The task continues until a pre-determined number of targets have been selected. Selection occurs when the mouse button is lifted, as in common desktop operating systems. Selections made outside of the targets are errors. The task begins when the first target is selected, and the first target is not counted in the results.

6.2.2 Multi-Directional Tapping

Figure 6.3: Multi-Directional Tapping Task Screenshot



The blue line shows the direction of motion, but was not present in the experiment.

The Multi-Directional Tapping task is similar to 1D Tapping, but measures the device's capabilities in all directions. The targets, shown in figure 6.3, are arranged in a circle. The target to select is highlighted red. Once selected, it turns gray, and the opposite target is highlighted. The sequence of highlighted targets is shown by the blue line. This task forces the user to move her cursor in many different directions to select targets, in case the input device's performance varies based on the direction of cursor movement.

As with 1D Tapping, the task ends after a pre-determined number of targets have been selected.

6.3 Experiment Setup

Fourteen undergraduate computer science students volunteered for the experiment. They were given 1% added to their final grades in exchange for participating. There were four techniques for each task: ARC-Pad, Tiltpad, Screenshots, and Touchpad, as described above.

Task	Target Sizes	Target Distances
1D Tapping Task	4 and 8 cm	3.5 and 7 m
2D Tapping Task	4 and 8 cm	1.5 and 2.5 m

Each task was separated into blocks of 20 trials each, with a rest period of 10 seconds between blocks. There were an additional 20 practice trials before the start of each combination of task and technique. 4 techniques $\times$ 20 trials $\times$ 2 sizes $\times$ 2 distances = 320 trials per task. 4 techniques $\times$ 20 practice = 80 practice trials, for a total of 400 trials per task. I used a within-participants design, and the order of the techniques and tasks were counterbalanced to compensate for learning effects. 14 participants $\times$ 2 tasks $\times$ 400 trials = 11200 trials total.

I instructed participants to hold the iPod in their non-dominant hand, and to use the pointing finger on their dominant hand to interact with the touchscreen. I told them to proceed as quickly and accurately as possible. I also told them to jump at least once per trial in ARC-Pad and Screenshot, to expose the potential benefit of absolute jumping. I told participants to look down at the screen when jumping in the Screenshot technique. Participants performed the experiment while standing 5.5 meters away from the screen's center. The experiment lasted approximately 45 minutes.

6.4 1D Tapping Results

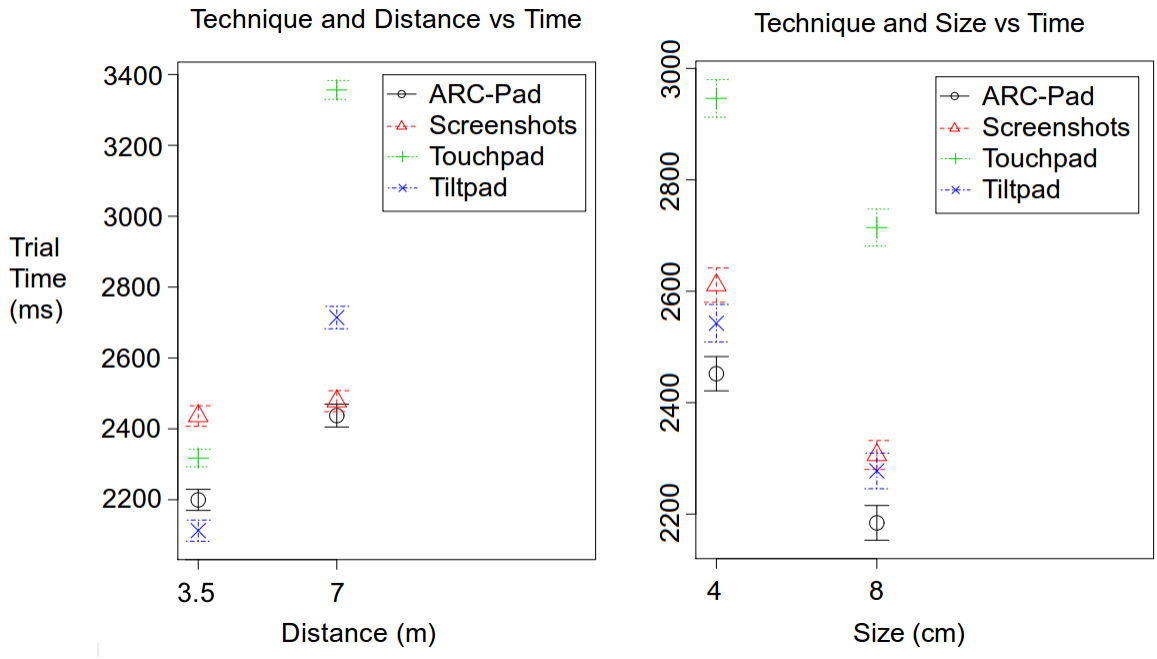
I removed outliers with a trial time greater than 3 standard deviations from the mean, which was 1.6% of trials.

6.4.1 Trial Times

A normal Q-Q plot showed that the trial times were not normally distributed, so I performed an ANOVA using $\log_{10}$(Trial Time). After applying the Bonferroni correction, I found a significant effect of *distance* ($F_{1,13} = 645$, $p < 0.001$), *size* ($F_{1,13} = 220$, $p < 0.001$)and *technique* ($F_{3,39} = 142$, $p < 0.001$) on trial times. Furthermore, there was an interaction between *distance* and *technique* ($F_{3,39} = 114$, $p < 0.001$). Interestingly, there was no significant interaction between *size* and *technique*.

I used Levene's Test for Homogeneity, and found equal variances across techniques ($F_{3,39} = 19.2$, $p < 0.001$). Post-hoc pairwise comparisons using Tukey's HSD revealed significant differences between all techniques, at a confidence interval of 95%.

Figure 6.4: Experiment Two - 1D Trial Times As A Function Of Target Distance And Size



Bars are 1 standard error.

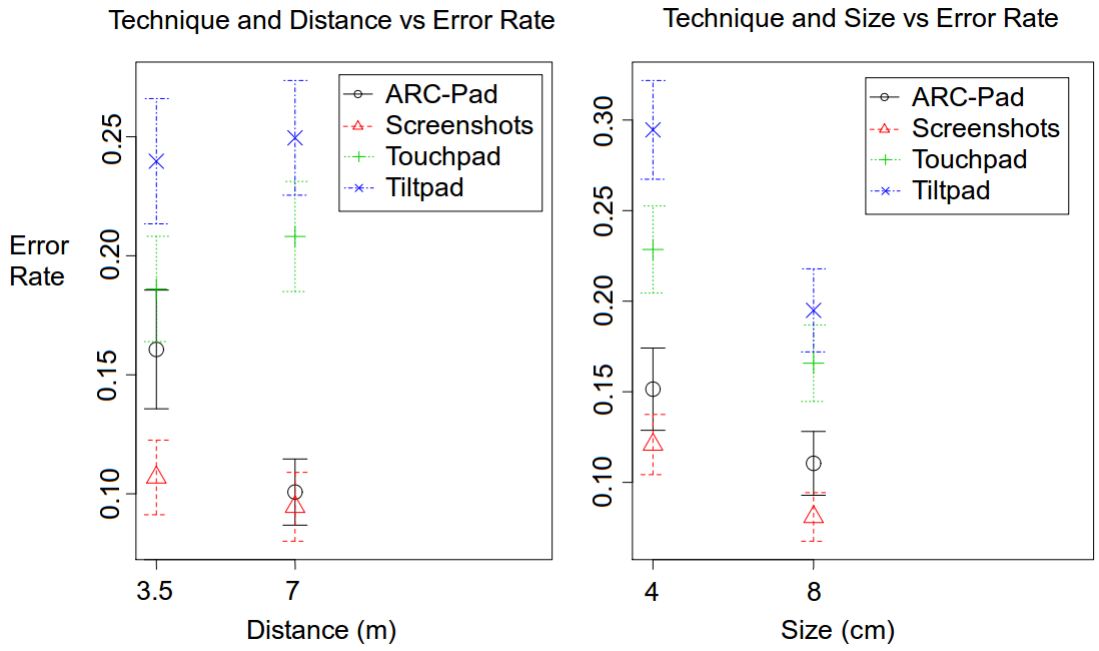
In figure 6.4, we can see that all four techniques caused similar increases in selection times for the smaller target size, confirming a lack of interaction between technique and size. The distance-related performance shows a different story. The longer distance impacted touchpad the most, as even the highest cursor acceleration curve required additional clutching to reach the target. The screenshots technique was not affected by the distance, confirming that the screenshots assist users in performing precise absolute jumps. ARC-Pad performed similarly to Tiltpad on the shorter distance, and somewhat better on the longer distance.

6.4.2 Error Rate

I performed a multi-way ANOVA to examine the effects of *technique*, *size* and *distance* on the error rate. I applied the Bonferroni correction and found a significant effect of *technique* ($F_{3,39} = 18.8$, $p < 0.001$) and *size* ($F_{1,13} = 16.6$, $p < 0.001$), but no significant effect of *distance* ($F_{1,13} = 0.426$, $p = 0.541$). I did not find any interaction effects.

Levene's Test for Homogeneity found equal variances across techniques ($F_{3,39} = 18.8$, $p < 0.001$). Post-hoc pairwise comparisons with Tukey's HSD yielded significant differences between all *technique* combinations at a 95% confidence interval but two: Screenshots - ARC-Pad and Touchpad - Tiltpad. I speculate that the similarity of the techniques within these pairs caused them to have roughly equivalent error rates.

Figure 6.5: Experiment Two - 1D Error Rate As A Function Of Target Distance And Size



Bars are 1 standard error.

The right side of figure 6.5 shows how the error rate varies with the target size.

ARC-Pad and Screenshots were significantly less error-prone than Tiltpad and Touchpad. Tiltpad is affected the most by the target size, moving from 0.195 at 8 cm to 0.295 at 4 cm. While observing participants, I found that they overwhelmingly tended to use the maximum amount of cursor acceleration. As we can see, Tiltpad's increased speed reduces the participants' ability to accurately position the cursor.

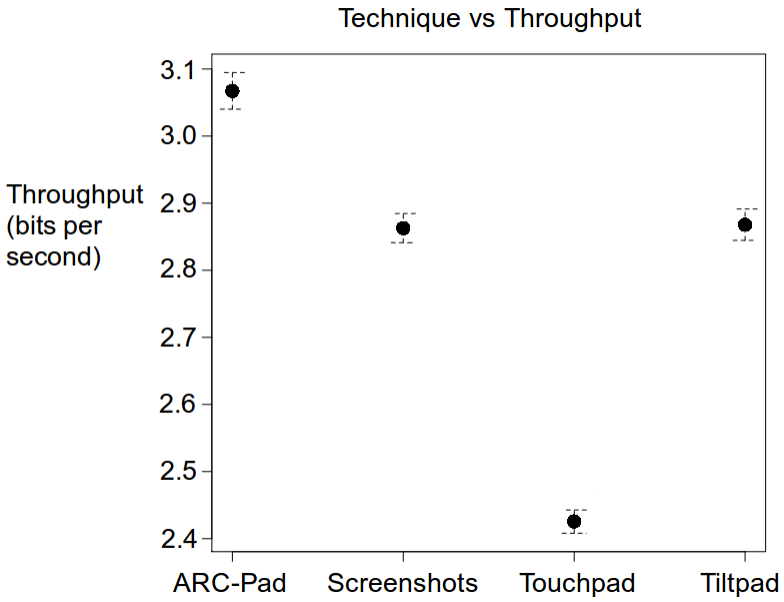
In figure 6.5 (left side), ARC-Pad's error rate actually goes up when the target is nearby, moving from 0.1 at 7 m to 0.161 at 3.5 m. When observing participants, I noticed that they tended to overshoot the target by a significant amount in the 3.5 m condition. I speculate that this may be related to the higher error rate, especially since Screenshots eliminates the issue of overshooting, and the Screenshots error rate is more or less constant.

6.4.3 Throughput

I performed an ANOVA to examine the effect of *technique* on throughput, and found a significant effect ($F_{3,39} = 142$, $p < 0.001$). Levene's Test for homogeneity found equal variances across techniques ($F_{3,39} = 71.4$, $p < 0.001$). Pairwise comparisons with Tukey's HSD found significant differences between all pairs of techniques except one; Tiltpad and Screenshots.

The mean throughput of each technique is shown in figure 6.6.

Figure 6.6: Experiment Two - The Throughput Of Each Technique, In The 1D Task



Bars are one standard error.

6.5 2D Tapping Results

For the following results, I removed outliers with trial times greater than 3 standard deviations from the mean, which amounted to 1.4% of trials.

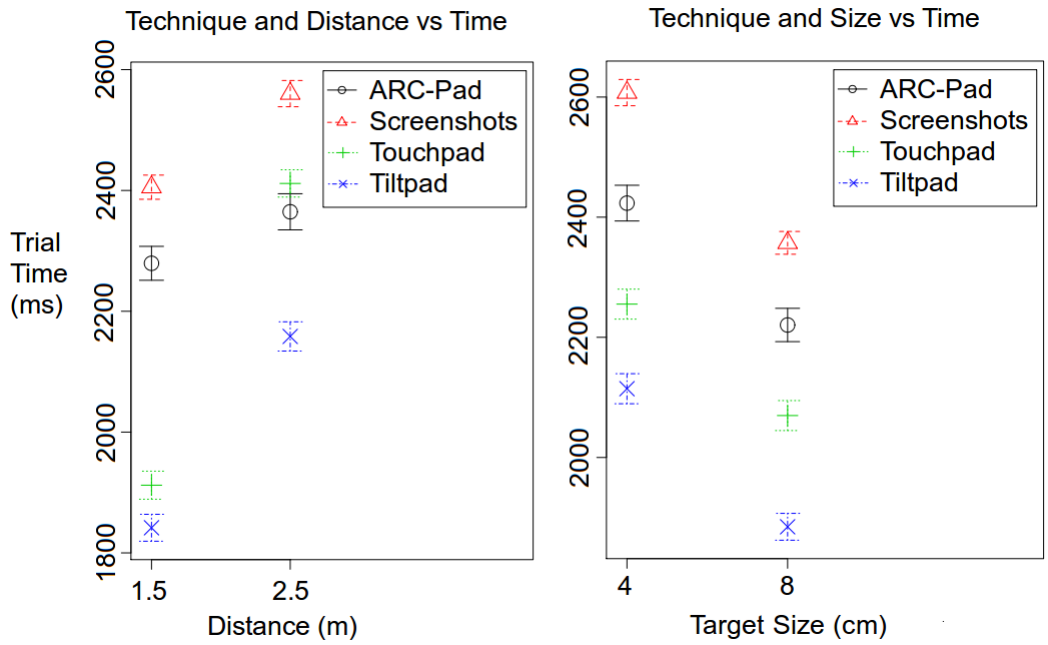
6.5.1 Trial Times

A normal Q-Q plot revealed that the trial times were not normally distributed, so I performed an ANOVA using $\log_{10}$(Trial Time). I used the Bonferroni correction, and found a significant effect of *distance* ($F_{1,13} = 306$, $p < 0.001$), *size* ($F_{1,13} = 185$, $p < 0.001$) and *technique* ($F_{3,39} = 193$, $p < 0.001$), and an interaction effect between *distance* and *technique* ($F_{3,39} = 44.5$, $p < 0.001$).

Levene's Test for homogeneity showed equal variances across techniques ($F_{3,39} = 56.4$, $p < 0.001$). Pairwise comparisons with Tukey's HSD showed significant differ-

ences between all techniques at a 95% confidence interval.

Figure 6.7: Experiment Two - 2D Trial Times As A Function Of Target Distance And Size



Bars are 1 standard error.

Figure 6.7 (right side) shows that the trial times increased by a roughly constant amount for each technique when the target was 4 cm. Tiltpad had the best speed, with an average of 2114 ms in the 4 cm condition, and 1886 ms in the 8 cm condition. ARC-Pad and Screenshots were both significantly slower than even the base Touchpad condition. The story is similar when examining movement times based on distance. Tiltpad again outperforms the other techniques, with a mean completion time of 2158 ms for the 2.5 m distance, and 1842 in the 1.5 m distance.

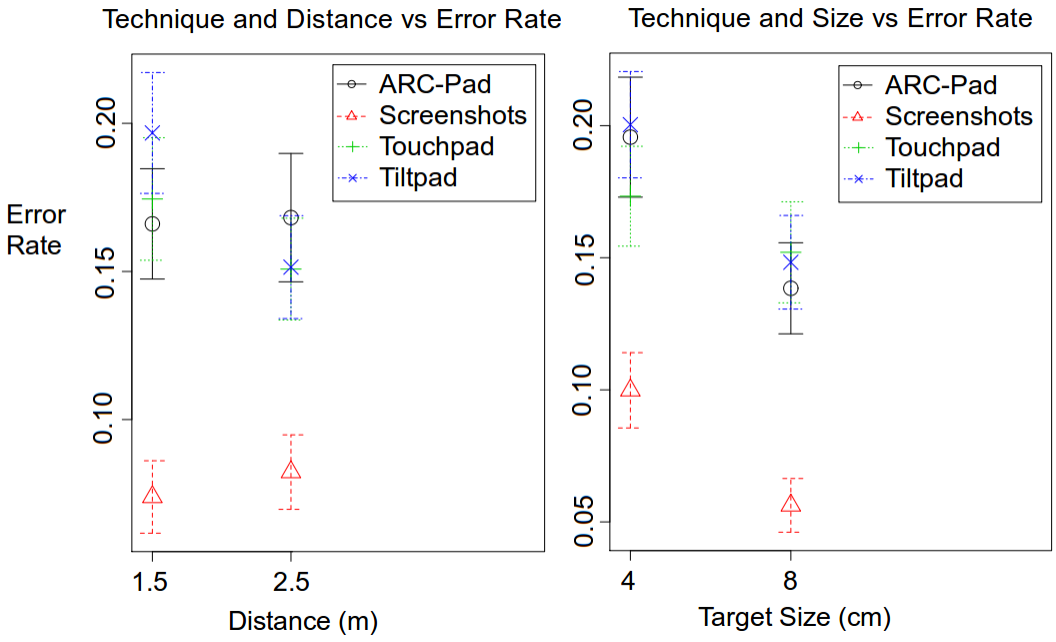
6.5.2 Error Rate

As before, I applied a multi-way ANOVA with the Bonferroni correction. The results were similar to the 1D task, with a significant effect of *size* ($F_{1,13} = 11.8$, $p < 0.001$)

and *technique* ($F_{3,39} = 12.8$, $p < 0.001$) on error rate. There was no significant effect of *distance* ($F_{1,13} = 1.34$, $p = 0.248$) and no interaction effects.

Levene's Test for homogeneity found equal variances across techniques ($F_{3,39} = 12.8$, $p < 0.001$). Post hoc pairwise comparisons with Tukey's HSD showed a significant difference between Screenshots and the other techniques at a 95% confidence interval.

Figure 6.8: Experiment Two - 2D Error Rate As A Function Of Target Distance And Size



Bars are 1 standard error.

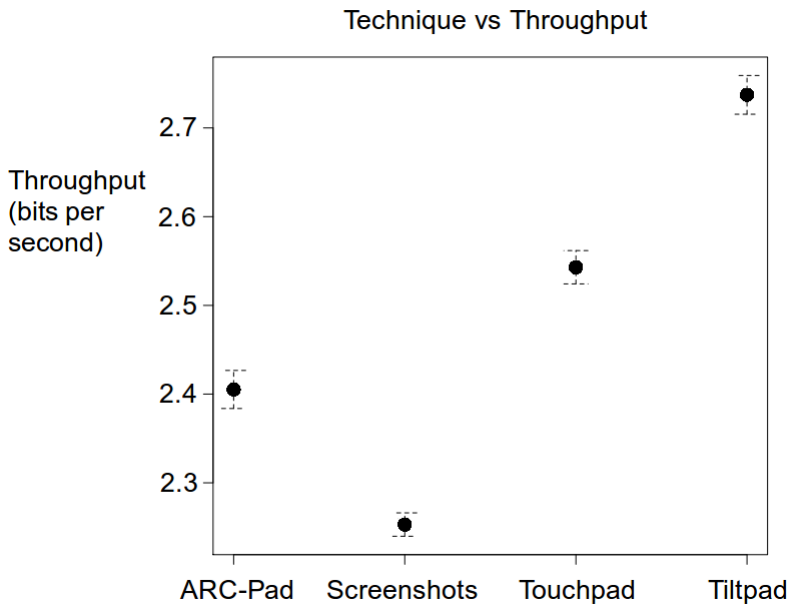
In figure 6.8, we can see that Screenshots consistently outperforms the other techniques. The Screenshots error rate was only 0.1 for small targets, and 0.056 for large targets.

6.5.3 Throughput

A one-way ANOVA found a significant effect of technique on throughput ($F_{3,39}$ = 115, p <0.001). Levene's Test of homogeneity found equal variances across techniques ($F_{3,39}$ = 76.5, p <0.001). Tukey's HSD showed significant differences between all pairs of techniques at a 95% confidence interval.

The 2D throughput is shown in figure 6.9.

Figure 6.9: Experiment Two - The Throughput Of Each Technique, From The 2D Task



Bars are one standard error.

6.6 Throughput Summary

ARC-Pad had the highest throughput in the 1D tapping task, at 3.07 bits/second, but suffered in the 2D condition, with only 2.41 bits/second. We can see that the throughput is lower than in 1D tapping, which makes sense. The throughput is based on the target width in the direction of motion, but in the 2D case users must also

position the cursor based on the target's height.

Table 6.1: 1D and 2D Throughput.

Technique	1D Tapping	2D Tapping
ARC-Pad	3.07	2.41
Screenshots	2.86	2.25
Touchpad	2.43	2.54
Tiltpad	2.87	2.74

6.7 Preferences

User preferences were collected with a 5-point Likert scale:

Table 6.2: Mean User Preferences.

	Ease of Use	Speed	Control	Comfort
Touchpad	4.23	3.00	4.08	3.23
Tiltpad	3.77	4.31	4.15	3.00
ARC-Pad	3.62	4.08	2.77	3.77
Screenshots	4.46	4.54	4.15	3.92

Screenshots was the overall highest rated technique. It had the highest rating in ease of use, speed and comfort, and tied Tiltpad for control. Users found touchpad to have the lowest speed, which is confirmed in the data. ARC-Pad had the lowest score for control, although its error rate was actually lower than Touchpad and Tiltpad in the 1D task, and not significantly different in the 2D task. Tiltpad had the lowest comfort score.

6.8 Discussion

In the 1D Tapping task, ARC-Pad clearly surpasses the other techniques. It had the lowest trial completion times, and did not significantly differ from Screenshots in terms of error rates. This is backed up by ARC-Pad's throughput of 3.07 bits/second, the highest in that task.

In the 2D Tapping task, the picture is less clear. Tiltpad had the best times, but Screenshots had the best error rate. The throughput lets us resolve this situation. Tiltpad's throughput was the highest, at 2.74 bits/second. Screenshots actually had the lowest throughput of all techniques, with only 2.25 bits/second.

Despite Tiltpad's performance in the 2D case, I decided not to include it in the final ARC-Pad design. The error rates were far too high when selecting small targets, at 20% in the 2D task and 30% in the 1D task. Tiltpad also had the lowest comfort rating, but I believe this problem can be resolved. Participants tended to continually tilt the device to obtain the maximum CD gain, and so I could simply use that CD gain by default, and avoid the need to tilt the phone at all. In essence, Tiltpad turned out to be equivalent to doubling the CD gain. However, we can see that the extra cursor speed made it far more difficult to select targets, as described by the speed-accuracy tradeoff. At this time there does not appear to be any way to reduce Tiltpad's error rate, other than simply removing the Tiltpad technique altogether.

However, it did appear that ARC-Pad's performance could be improved. The targets in the 2D case were relatively close together, as the physical screen's dimensions were 7.26 m x 3.03 m. The ISO's 2D tapping task requires targets to be placed in a perfect circle, so the maximum distance between targets could only be around 2.5 meters. (We want to make it possible to overshoot the targets, so they can't be placed right at the screen edge.) With the more distant targets in the 1D task, we can see that Tiltpad's throughput falls slightly below ARC-Pad's. It seems likely that ARC-Pad's performance in the 2D case was a result of forcing the users to jump at

least once per trial, even when the target is very close. The 2D task targets could typically be reached with a single sliding gesture, without clutching. If I should stop requiring at least one jump, ARC-Pad's performance should rise for nearby objects, while maintaining high performance for distant targets. In essence, this is the original combination that made ARC-Pad successful [18].

The only technique with a reasonable error rate in the 2D task was Screenshots, which was also the highest-rated technique. And yet, I was reluctant to include it in my final design without modifications because of its low throughput. In the end, I added Screenshots, but removed the requirement that users look down at the device before making an absolute jump. Instead, I told them they could look at the touchscreen whenever they felt the need for more accurate positioning. I felt that this combination had the best chance of maintaining high performance and user ratings.

Chapter 7

Experiment Three

In experiment three, I evaluate ARC-Pad's final design by comparing it to existing solutions for target selection on large displays. I attempt to show that ARC-Pad is a viable solution for cursor control, one that can compete against techniques already in use.

7.1 Techniques

7.1.1 ARC-Pad 2 Technique

ARC-Pad is described in sections 4.1 and 5. ARC-Pad 2 (AP2) consists of ARC-Pad with the Screenshots technique detailed in section 6.1.

7.1.2 Air Mouse Technique

The Gyration Air Mouse is a commercial solution for portable cursor control. It is a standard optical mouse with accelerometers to detect the mouse's changes in yaw orientation, as well as horizontal acceleration [10]. Mouse model GC15M was used. The Air Mouse appears as a basic pointing device to the operating system. I examined the Air Mouse's bluetooth output and found that the conversion of accelerometer data

to mouse coordinates must occur on the device, as only the x and y cursor movements are sent to the desktop.

I conducted a small pilot with the Air Mouse, and found that the default cursor settings in Red Hat Enterprise Linux 5 were far too sensitive for the Air Mouse, though they seemed appropriate for regular mice. I modified the cursor settings through the gnome-mouse-settings utility, greatly reducing the cursor speed and slightly reducing the cursor acceleration settings. As the other techniques use my custom server to set the mouse position directly, these settings only impacted the Air Mouse.

7.1.3 PRISM

PRISM (Precise and Rapid Interaction through Scaled Manipulation) is a method of smoothing 3D input to remove hand jitter [9]. It dynamically adjusts the CD gain in a 3D system based on the user's hand speed. The effect is similar to cursor acceleration for direct 3D manipulation. The max CD gain is 1, and is reduced for slow hand movements. PRISM allows the user to move an object or cursor through a combination of direct and indirect manipulation. The object will lag behind the hand's position when movement is slow, to improve accuracy. The object snaps back beneath the hand when movement is fast.

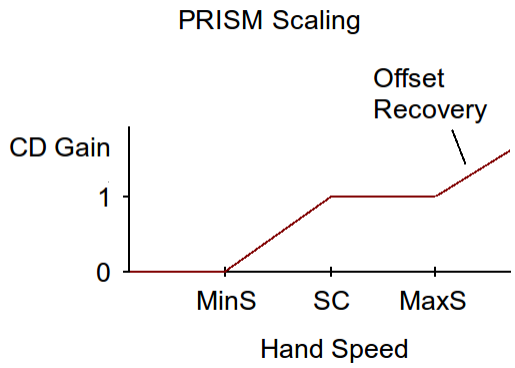
Frees et al. [9] introduced three constants to achieve this effect, as seen in figure 7.1:

MinS (Minimum Speed) Any hand movements below this speed are considered to be involuntary, and the CD gain is set to zero.

SC (Scaling Constant) The CD gain is proportional to the hand speed, up to a maximum CD gain of one.

MaxS (Maximum Speed) The CD gain is increased above one, allowing the object to 'catch up' to the hand.

Figure 7.1: PRISM's CD Gain-Scaling Mechanism



While Frees et al.'s focus was on direct object manipulation in an immersive 3D environment, they also applied the PRISM algorithm to improve target selection with a virtual laser pointer. The laser extended from the user's virtual hand, and moved as the hand was rotated. The CD gain was applied to the laser's rotational velocity, and was modified based on the rotational velocity of the hand, as in figure 7.1. Frees et al. [9] showed that using PRISM to stabilize a virtual laser pointer resulted in significantly improved performance compared to an unstabilized laser for small, distant targets.

I implemented the same system using an Ascension Flock of Birds tracking system, which can accurately detect the position and orientation of a wand in 3D space [3]. I used a MinS of 0.6 degrees per second, rather than 0.2 as in the original PRISM, since there was a small amount of signal noise in the wand's position. I also smoothed out noise in the sensor data by taking the average of the three most recent wand orientations. I prefer this approach rather than increasing MinS further because I found that a higher MinS causes intentional movement to be discarded, and makes it difficult to select small targets. The wand's orientation was sampled 100 times per second, so taking the average of the three most recent orientations increased latency by at most 30 ms, which is below the ISO's guideline of 40 ms [12].

I ignored roll when calculating the wand's rotational speed, as I found that small hand gestures were causing large increases in speed. As the roll is not used to control the laser beam, I did not notice any negative effects from the removal.

SC and MaxS were set to 25 degrees per second and any motion higher than SC resulted in immediate offset recovery, which is the approach taken by Frees et al. for rotational velocity. I also added the arc used in the original PRISM, which appeared in the top-right corner of the screen and indicated the amount of CD gain currently in use. The arc length increases along with the wand's rotational velocity, and varies from a tiny sliver at MinS, to a semi-circle at SC.

The laser's position was calculated with Spherical Linear intERPolation (SLERP), using the algorithm provided by Frees et al [9, p. 10]. However, I found that line 4 of their formula did not always return the shortest distance between two quaternions. I added the following code between lines 4 and 5 to correct this problem:

1. A $\leftarrow$ absolute value(A)
2. if A $>$ 180

   A $\leftarrow$ 360 - A

7.2 Tasks

7.2.1 1D Tapping

I used the 1D reciprocal tapping task from experiment two. See section 6.2 for details.

7.2.2 Distant Tapping

Distant Tapping was adapted from the ISO 9241-9's multi-directional pointing task [12]. In experiment two, I found that arranging the targets in a perfect circle did not take

full advantage of the large display, as the screen was very wide but not very tall, and the circle only covered a small area in the center of the screen. To examine the benefits of the techniques over long distances, I developed a task comprised mainly of horizontal motion.

In Distant Tapping a target is initially placed near the right edge of the screen. The program generates a circle whose center is the target center, and whose radius is the distance between targets. The next target is randomly placed along the part of the circle that intercepts the screen. As I kept the minimum target distance greater than the screen height, the targets alternated between the left and right sides of the screen. I simultaneously displayed two targets; the target to select was highlighted in red, and the next target was shown in Gray The Gray target reduced visual search time, and more closely approximated the ISO's 2D tapping task.

7.3 Experiment Setup

Eleven undergraduate computer science students volunteered for the experiment. They were given 1% added to their final grades in exchange for participating. There were three techniques for each task: AP2, Air Mouse, and PRISM.

Table 7.1: Experiment Factors

Task	Target Sizes	Target Distances
1D Tapping Task	4 and 8 cm	3.5, 5.25 and 7 m
2D Tapping Task	4, 6 and 8 cm	3.5, 4.37, 5.25, 6.12 and 7 m

I added additional distances and sizes to examine the trends as the size and distance increase. The 1D task was separated into blocks of 10 trials each, and the distant tapping task had 15 trials per block. Because of the large number of conditions

in Distant Tapping, I varied the target sizes and distances within each block. I took care that each distance and size appeared an equal number of times. I was unable to add the same number of conditions to 1D Tapping, since the target size and distance must remain constant within each 1D block.

There were 15 practice trials before the start of each new task and technique combination, and a rest period of 10 seconds between blocks. The order of the techniques and tasks was partially counter-balanced. To summarize:

1D Tapping: 3 techniques $\times$ 2 sizes $\times$ 3 distances $\times$ 10 trials = 180 trials

Distant Tapping: 3 techniques $\times$ 15 trials $\times$ 4 blocks = 180 trials

Adding three practice sessions gives us 225 trials per task, for 450 trials in the experiment. With 11 participants, we have 4950 trials total. The experiment duration was under 40 minutes.

I instructed participants to perform the experiment while standing 5.5 meters from the center of the screen. I told them to hold the iPod in their non-dominant hand, and to use their dominant hand to touch the screen. I gave a brief explanation of PRISM, and said that the cursor would lag behind their hand when moving slowly to improve stability, but would 'snap back' if they moved the wand quickly.

7.4 1D Tapping Results

I removed trials whose completion time was more than 3 standard deviations from the mean, resulting in less than 1.8% of trials deleted.

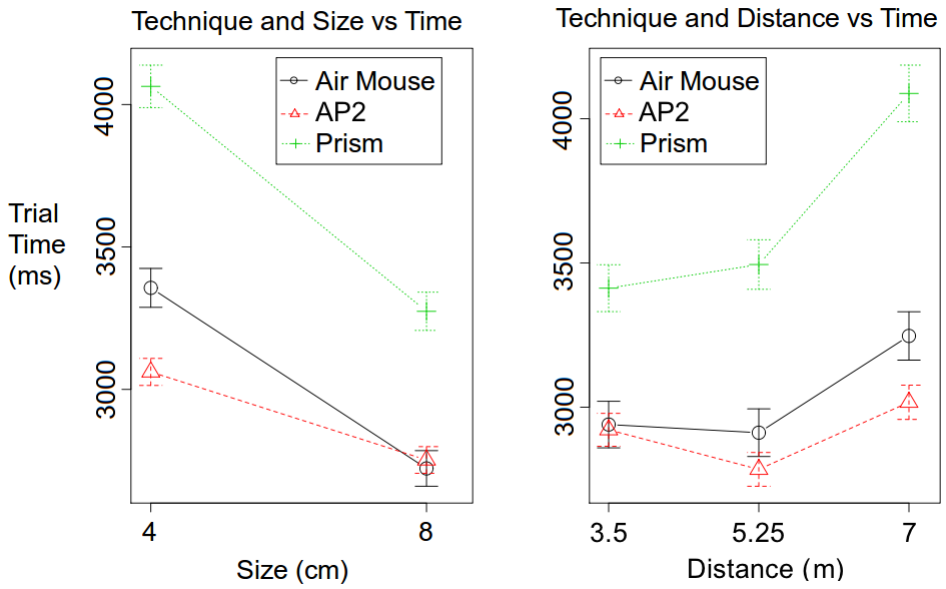
7.4.1 Trial Times

I applied a multi-way ANOVA to examine the effects of target *distance*, *size* and *technique* on trial times. A normal Q-Q plot showed that the distribution was not

normal, so I used $\log_{10}$(trialTime). I applied the Bonferroni correction to the results, and found a significant effect of *size* ($F_{1,10} = 176$, $p < 0.001$), *distance* ($F_{2,20} = 30.1$, $p < 0.001$) and *technique* ($F_{2,20} = 77.0$, $p < 0.001$). There was also an interaction effect between *size* and *technique* ($F_{2,20} = 7.98$, $p < 0.01$).

Levene's Test for homogeneity found equal variances across techniques ($F_{2,20} = 27.6$, $p < 0.001$). Pairwise comparisons with Tukey's HSD showed a significant difference between PRISM and AP2, and between PRISM and Air Mouse at a 95% confidence interval. There was no significant difference between AP2 and Air Mouse.

Figure 7.2: Experiment Three - The Effect Of Technique, Size And Distance On 1D Trial Times



Bars are one standard error.

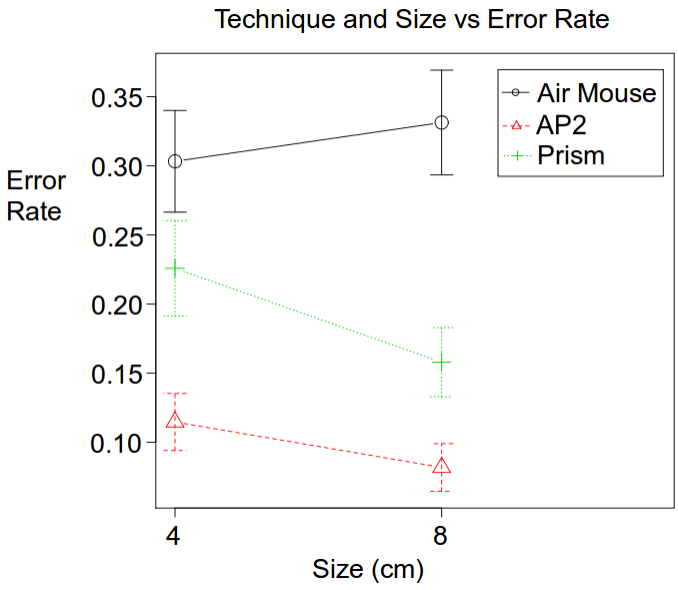
Figure 7.2 shows that AP2 and Air Mouse both significantly outperformed PRISM. All three were affected by the target size, and the general trend was that selection times increased for the largest distance.

7.4.2 Error Rate

As before, I used a multi-way ANOVA to look at the effect of target *size*, target *distance* and the *technique* on error rates, and applied the Bonferroni correction to the results. I found a significant effect of *technique* ($F_{2,20} = 31.6$, $p < 0.001$) and *size*($F_{1,10} = 5.50$, $p < 0.05$). There was an interaction effect between *distance* and *technique* ($F_{4,40} = 3.07$, $p < 0.05$). *Distance* alone had no significant effect on the error rate ($F_{2,20} = 0.616$, $p = 0.540$).

Levene's Test for homogeneity found equal variances across techniques ($F_{2,20} = 31.3$, $p < 0.001$). Tukey's HSD showed significant differences between all combinations of techniques, and figure 7.3 shows that the technique had a much larger effect on error rate than size.

Figure 7.3: Experiment Three - The Effect Of Technique And Size On 1D Error Rate

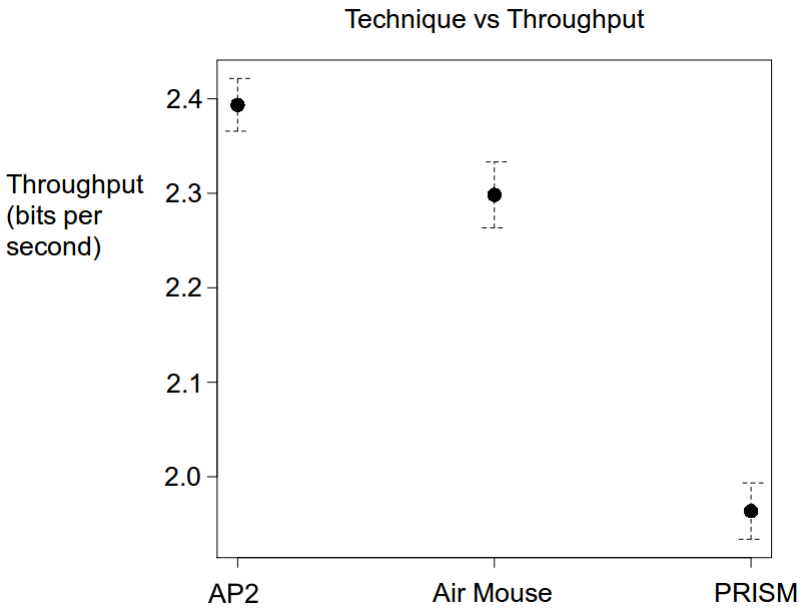


Bars are one standard error.

7.4.3 Throughput

A multi-way ANOVA showed a significant effect of *technique* on throughput ($F_{2,20}$ = 53.0, p <0.001). Levene's Test for homogeneity found equal variances across techniques ($F_{2,20}$ = 17.6, p <0.001). Tukey's HSD found a significant difference between PRISM and the other techniques at a 95% confidence interval, but no difference between AP2 and Air Mouse. Figure 7.4 shows that PRISM did relatively poorly, with only 1.96 bits per second. AP2 and Air Mouse had 2.39 and 2.3 bits per second, respectively, but were not significantly different.

Figure 7.4: Experiment Three - The Effect Of Technique On 1D Throughput



Bars are one standard error.

7.5 Distant Tapping Results

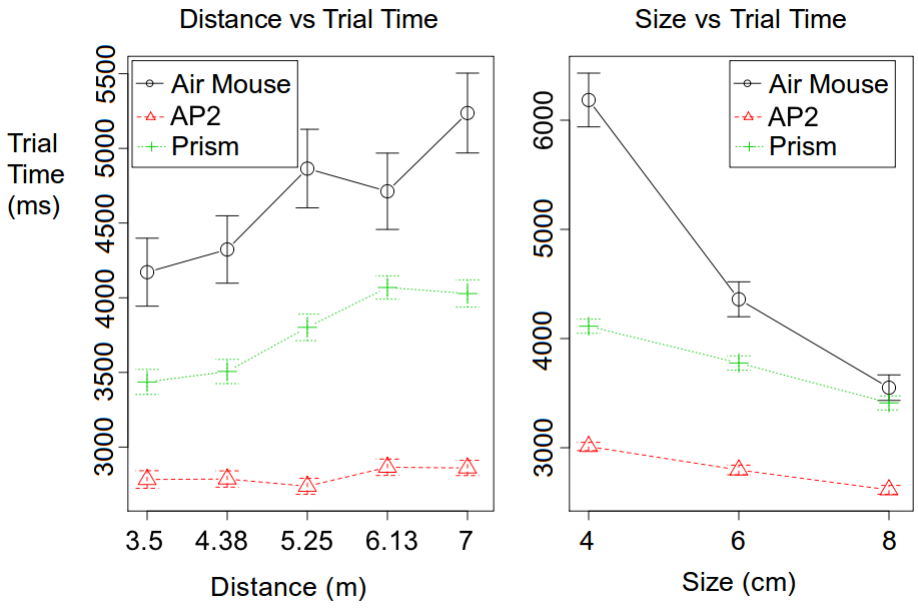
Before running ANOVAs, I removed outliers whose trial time was greater than 3 standard deviations from the mean, which was less than 1.9% of all trials.

7.5.1 Trial Times

I used a multi-way ANOVA to evaluate the effect of *size*, *distance* and *technique* on trial times. After applying the Bonferroni correction, I found a significant effect of *distance* ($F_{4,40} = 14.5$, $p < 0.001$), *size* ($F_{2,20} = 132$, $p < 0.001$) and *technique* ($F_{2,20} = 262$, $p < 0.001$). There was an interaction effect between *size* and *technique* ($F_{4,40} = 20.1$, $p < 0.001$), and an effect between *distance* and *technique* ($F_{8,80} = 2.80$, $p < 0.01$).

Levene's Test for homogeneity found equal variances across techniques ($F_{2,20} = 130$, $p < 0.001$). Post hoc comparisons with Tukey's HSD showed significant differences between all pairs of techniques at a 95% confidence interval.

Figure 7.5: Experiment Three - The Effect Of Technique, Target Distance And Target Size On Distant Tapping Trial Times



Bars are one standard error.

Figure 7.5 shows that AP2 was the highest-performing technique. AP2's trial times remained roughly constant as target distance and size increased, suggesting that participants were able to rapidly cross long distances without any corresponding

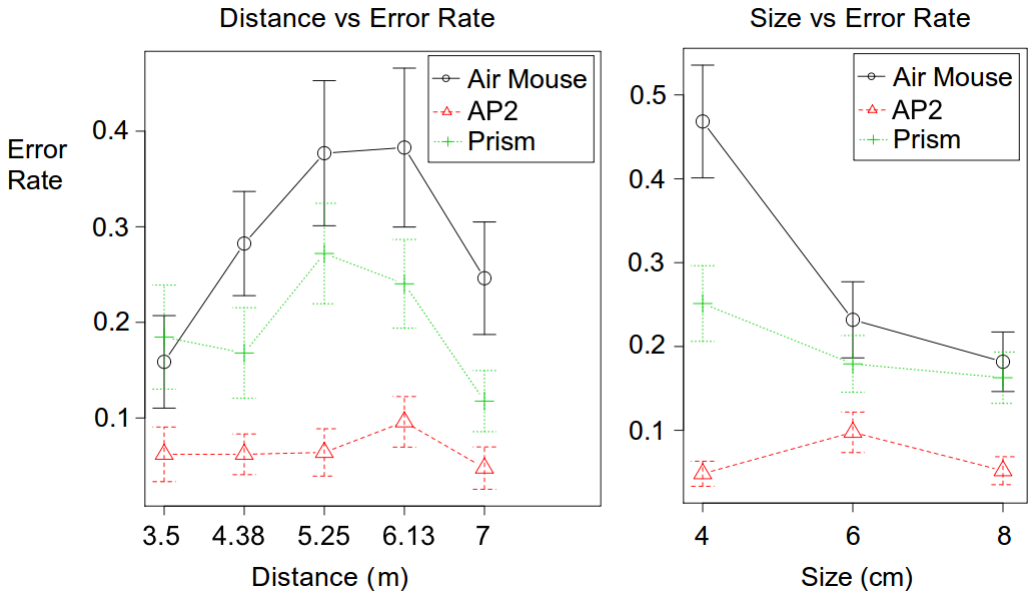
decrease in accuracy. Both PRISM and Air Mouse saw their trial times increase along with target distance, and Air Mouse suffered a large increase in trial times when the target size was small, moving from 3550 ms at 8 cm to 6236 ms at 4 cm. From the data, we can see that Air Mouse had a high speed but low accuracy, and suffered when precise pointing was required.

7.5.2 Error Rate

As before, I used a multi-way ANOVA with the Bonferroni correction to examine the effects of target *size*, *distance* and the *technique* on the error rate. I found a significant effect of target *size* ($F_{2,20} = 8.19$, $p < 0.001$), *distance* ($F_{4,40} = 3.14$, $p < 0.05$) and *technique* ($F_{2,20} = 24.6$, $p < 0.001$). There was an interaction effect between *size* and *technique* ($F_{4,40} = 4.59$, $p < 0.01$), and between *technique*, target *distance*, and *size* ($F_{16,160} = 1.67$, $p < 0.05$).

Levene's Test for homogeneity found equal variances across techniques ($F_{2,20} = 23.4$, $p < 0.001$). Post hoc comparisons with Tukey's HSD showed significant differences between all pairs of techniques at the 95% confidence interval.

Figure 7.6: Experiment Three - The Effect Of Technique, Target Distance And Target Size On Distant Tapping Error Rate



Bars are one standard error.

Figure 7.6 shows how the error rate varies with target distance and size. Both PRISM and Air Mouse have increasing error rates for smaller targets, with Air Mouse at 0.47 for 4 cm targets. AP2 is virtually unaffected by target size and distance, which indicates to me that its hybrid pointing mechanism functions as intended. Participants can jump the cursor across the screen with a single gesture, quickly covering long distances. They can then slide their finger for slower, precise pointing when selecting small targets.

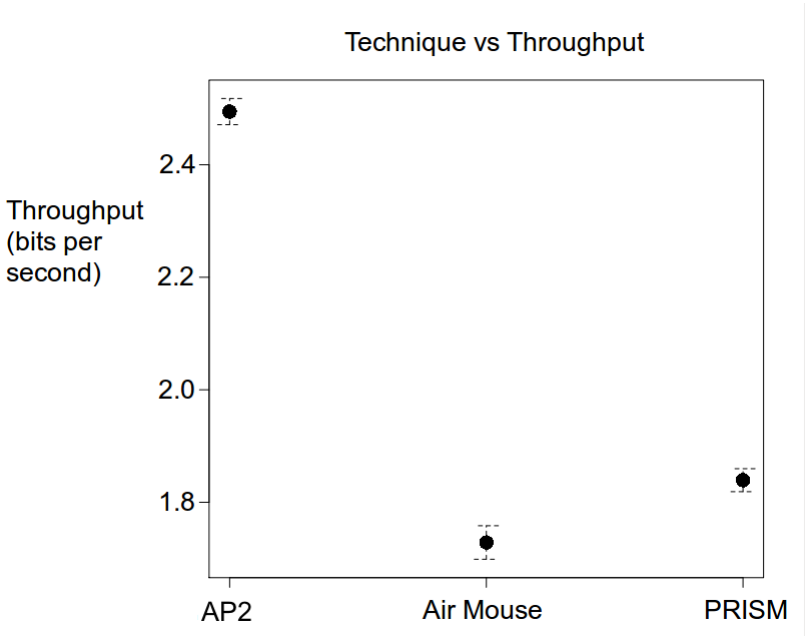
The error rate increases with distance as I expected, but then drops at the 612.5 and 700 cm distances. I suspect that this may be due to the target's proximity to the edge of the screen, which could act as a guide for cursor positioning.

7.5.3 Throughput

I performed an ANOVA, and found a significant effect of *technique* on throughput ($F_{2,20} = 278$, $p < 0.001$). Levene's Test for homogeneity found equal variances across techniques ($F_{2,20} = 50.0$, $p < 0.001$), and Tukey's HSD found significant differences between all techniques at a 95% confidence interval.

Figure 7.7 shows that AP2 and PRISM had similar results in both the 1D and Distant Tapping tasks. However, Air Mouse had its throughput drop to only 1.73 bits per second in Distant Tapping. I believe this is due to the increased difficulty of the 2D task, where the cursor must be accurately positioned in both the x and y axes. It appears that Air Mouse was barely accurate enough to provide a reasonable throughput in the 1D case, but was insufficient in the 2D experiment. This is backed up by the Air Mouse's very high error rate.

Figure 7.7: Experiment Three - The Effect Of Technique On Distant Tapping Throughput



Bars are one standard error.

Table 7.2: 1D and Distant Tapping Throughput.

Technique	1D Tapping	Distant Tapping
AP2	2.39	2.50
Air Mouse	2.3	1.73
PRISM	1.96	1.84

7.6 Preferences

I collected preferences on a 5-point Likert scale:

Table 7.3: User Preferences.

	AP2	Air Mouse	PRISM
Ease	4.27	2.36	2.82
Speed	4.45	2.54	3.09
Precision	4.72	2.09	3.27
Comfort	4.18	2.72	3.00
Overall	4.36	2.27	2.82

AP2 had the highest score in each category, as well as the highest overall rating. Air Mouse was the lowest-rated in each category, with PRISM in between the two.

7.7 Additional Metrics

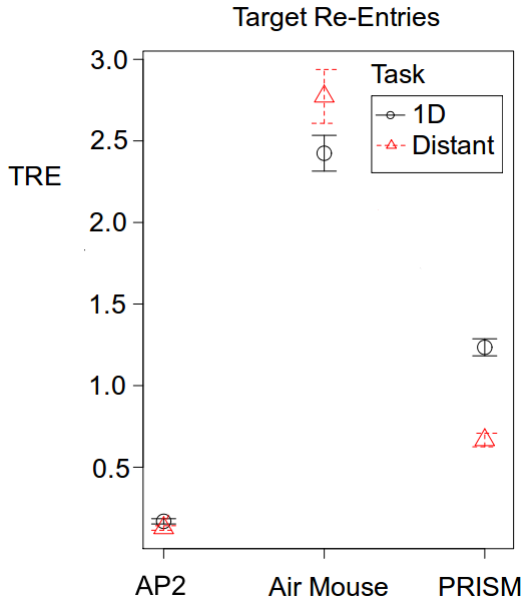
These metrics are described in section 3.5.

7.7.1 Target Re-Entry (TRE)

I used a multi-way ANOVA with the Bonferroni correction to look at the relationship between the *task* type, *technique* and the number of target re-entries. I found a significant effect of *technique* ($F_{2,20} = 424$, $p < 0.001$) on TRE, and an interaction effect between *technique* and *task* ($F_{2,20} = 14.4$, $p < 0.001$). There was no significant effect for the *task* alone ($F_{1,10} = 1.54$, $p = 0.215$).

Levene's Test of homogeneity found equal variances across techniques, with respect to the number of re-entries ($F_{2,20} = 319$, $p < 0.001$). Tukey's HSD found significant differences between all pairs of techniques at a 95% confidence interval, but the only technique that was significantly affected by the task alone was PRISM.

Figure 7.8: Experiment Three - The Target Re-Entries For Each Technique And Task



Bars are one standard error.

Figure 7.8 shows that AP2 had the lowest number of re-entries, at an average of 1.14 across both task types. Air Mouse had the highest TRE, averaging 3.60 across the task types. PRISM had a TRE of 1.66 for the Distant Tapping task, but it

increased to 2.23 for 1D Tapping. This is somewhat interesting, since PRISM's error rate did not significantly change across task types. PRISM's 1D error rate was 0.22, while the Distant Tapping error rate was 0.21.

Imagine a line connecting the cursor's starting location to the target center. I believe that PRISM must be less accurate in the direction perpendicular to this line than AP2 or Air Mouse. In the 1D task, any time the cursor overshoots the target the number of target entries increases by one, since the target extends across the entire height of the screen. However, in the 2D case the target will only be entered if the cursor passes through it when an overshoot occurs. AP2 and Air Mouse tend to cross the target while overshooting, while PRISM goes around it. Therefore, PRISM has a lower re-entry rate in Distant Tapping than in 1D tapping, even though its error rate is unchanged.

7.7.2 Movement Direction Change (MDC)

Again, I used a multi-way ANOVA with the Bonferroni correction to look at the interaction between the *task*, *technique* type and the number of movement direction changes. I found a significant effect of *task* ($F_{1,10} = 50.9$, $p < 0.001$) and *technique* ($F_{2,20} = 697$, $p < 0.001$), and an interaction effect between *task* and *technique* ($F_{2,20} = 43.4$, $p < 0.001$). The results were almost identical to the number of target re-entries, suggesting that the vast majority of direction changes were caused by corrections after overshooting the target.

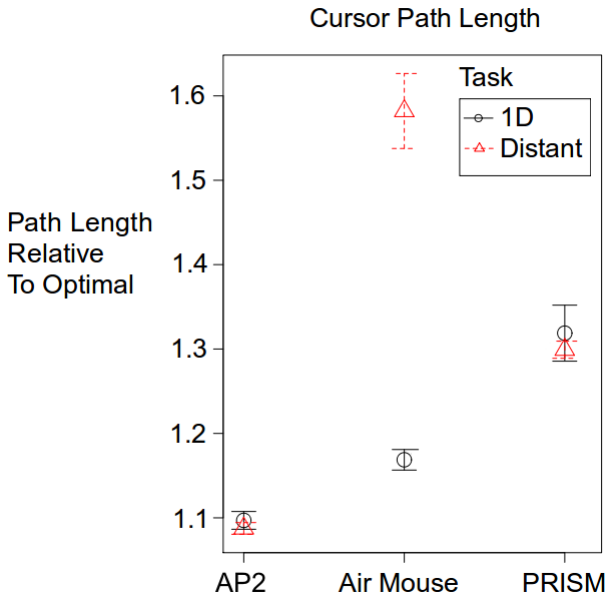
In the future, I would prefer to measure the direction changes that occur before the cursor passes the target, so the MDC would fill its intended purpose of finding problems with the input device.

7.7.3 Movement Variability (MV)

I used a variation of MacKenzie et al.'s metric [16] for calculating MV. MacKenzie et al. [16] take the standard deviation of cursor points away from the shortest path to the target. I prefer to compare the length of the optimal path to the the length of the path actually travelled, to get a better understanding of the device's efficiency. A value of one indicates that the cursor travelled along the optimal path, a value of two means it travelled twice the distance, etc. This is similar to Martinet et al.'s coordination metric for 3D docking tasks [17].

I used a multi-way ANOVA with the Bonferroni correction, and found a significant effect of *task* ($F_{1,10} = 42.3$, $p < 0.001$) and *technique* ($F_{2,20} = 75.5$, $p < 0.001$), as well as an interaction effect between *task* and *technique* ($F_{2,20} = 52.5$, $p < 0.001$). Tukey's HSD showed significant differences between all pairs of techniques at the 95% confidence interval. In figure 7.9, we can see that Air Mouse performs significantly worse in the Distant Tapping task, with a path 1.58 times longer than a straight line to the target. The MV seems to run contrary to Air Mouse's error rate, which stands at 0.29 in Distant Tapping, and 0.37 in 1D Tapping.

Figure 7.9: Experiment Three - The Length Of the Cursor's Path Relative To The Optimal Path



Bars are one standard error.

I believe this conflict can be explained by the increased difficulty of positioning the cursor over the target in the Distant Tapping task. Once the cursor is positioned, participants were relatively good at selecting the target without inadvertently causing the cursor to leave. However, getting the cursor over the target in the first place was more challenging. In 1D tapping, positioning the cursor was simpler as it was constrained to move in the x-axis only. Therefore, the cursor's path was relatively shorter relative to the optimal path, since only movement along the x-axis was counted.

Both AP2 and PRISM had similar MVs in 1D and Distant Tapping, as they did not have as much difficulty selecting targets in the Distant Tapping condition.

7.8 Discussion

All techniques had trial times inversely correlated with target size. The effect was most pronounced in Air Mouse in the Distant Tapping task, where Air Mouse's low accuracy prevented users from accurately selecting small targets. However, there was no significant difference between AP2 and Air Mouse's trial times in the 1D Tapping task, which were both faster than PRISM. AP2 had by far the best trial times in Distant Tapping.

AP2's performance was not significantly affected by either target size or distance in Distant Tapping, and its error rate and trial times remained roughly constant when target size and distance were varied. Increased target distance significantly increased Air Mouse's and PRISM's trial times and error rates in the Distant Tapping task. All techniques were affected by reduced target sizes.

In the end, AP2 had the highest throughput in Distant Tapping, and tied with Air Mouse in 1D tapping. AP2 was by far the highest-rated technique, and had the lowest number of target re-entries. The path followed by AP2's cursor was the shortest of all techniques, only 1.09 times longer than a straight line connecting the cursor's starting position to the final selection coordinates.

To conclude, it seems that AP2 is superior to the other two techniques by every metric I have considered in the Distant Tapping condition. While Air Mouse and AP2 share the same throughput score in 1D Tapping, Air Mouse's low accuracy and user rating make it an unappealing choice. AP2 had consistently good results regardless of the technique and task type, and was unaffected by the target distance, indicating that the hybrid absolute/relative input scaled well, even to targets seven meters apart.

I feel that AP2 performed very favourably compared to the existing techniques, making it a useful and inexpensive method of controlling a cursor on a large display.

Chapter 8

3D Experiment

In this experiment, I extended ARC-Pad's hybrid absolute and relative positioning system for 3D object manipulation. I believed it would be feasible to control a 3D cursor in a reasonably intuitive and efficient manner by incorporating the accelerometer data from a mobile device.

8.1 Techniques

8.1.1 3D-Pad

I used the AP2 technique described in experiment three, but added accelerometer data to position objects in the Z-direction. In standard AP2, the touchpad lets us move a cursor in the x and y axes. However, it is not clear how we could move the cursor in and out of the scene, along the z axis. I modified AP2 to alter the axis the cursor moves along based on the orientation of the phone.

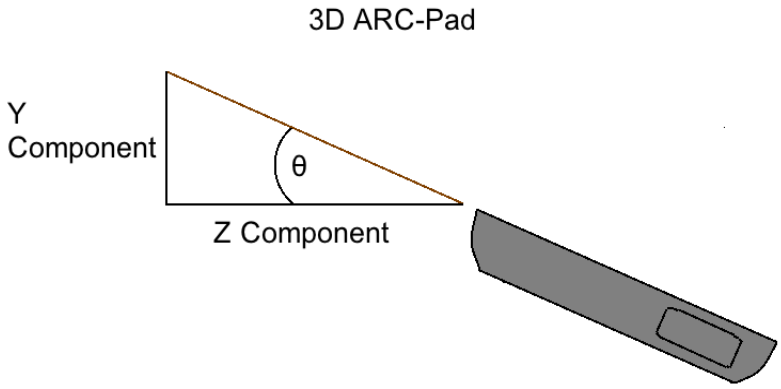
When the phone is horizontal, with the touchscreen facing the ceiling, moving a finger along the phone's y-axis will move the cursor in the z-axis. When the phone is vertical, the y-axis on the phone corresponds to the y-axis on the large display. The cursor always moves in the same direction as the user's finger; when the user's

finger moves vertically, the cursor moves vertically as well. When the user's finger moves towards or away from the display, the cursor moves along the z-axis in the same direction.

There is no distinct mode-switch between moving along the y or z-axis. For example if the phone is held at a 45 degree angle, the cursor will move along the same plane as the touchscreen. Only the phone's pitch is used to set the cursor's direction of motion. Yaw is ignored, and the roll value from the accelerometers is only used to obtain an accurate measure of the phone's pitch.

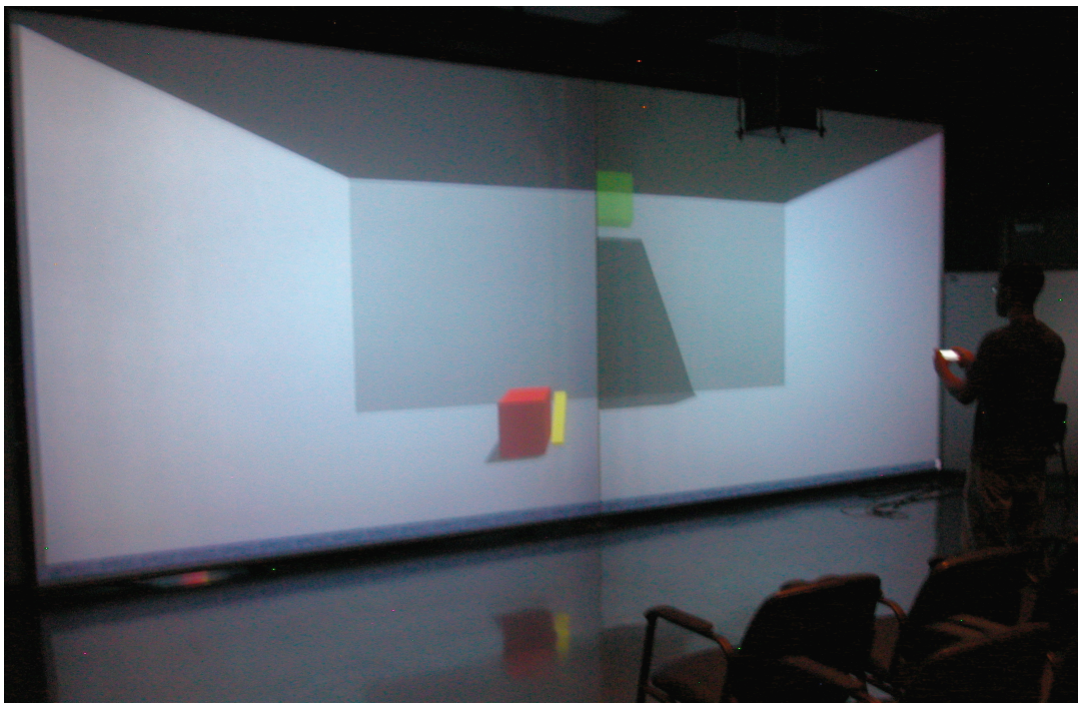
The screenshots are similarly affected by the phone's orientation. When the phone is vertical, the phone shows a view from the front of the display, as in standard AP2. When the phone is horizontal, the phone shows a top-down view of the scene. Again, the viewpoint changes smoothly as the phone is tilted.

Figure 8.1: 3D ARC-Pad Tilting Mechanism



Both relative and absolute motion occur on the plane of the phone's touchscreen. When the phone is tilted between the vertical and absolute positions, motions of the finger along the y-axis will move the cursor along the y and z axes. This is shown in figure 8.1. The screenshot orientation always gives a constant viewpoint of the plane the cursor will move along, allowing the screenshots to assist with absolute positioning as in standard AP2.

Figure 8.2: 3D ARC-Pad Implementation



The red cube (bottom left) is the cursor object, the green cube (upper right) is the goal area.

The phone's orientation is shown by a yellow bar next to the cursor, as in figure 8.2. It is intended to provide visual feedback as to the direction the cursor will move when the finger slides along the y-axis.

8.1.2 Blank Pad

This technique is identical to 3D-Pad, but does not employ screenshots.

8.1.3 PRISM

I used the PRISM technique for object translation described by Frees et al [9]. It involves the same modifications to CD gain as seen in figure 7.1 from experiment three, but modifies the CD gain based on the user's hand speed, rather than the

wand's rotational velocity. See experiment three, section 'Techniques' for a more complete description of PRISM's scaling mechanisms.

I employed the PRISM constants used by Frees et al [9]; 1 cm/second for MinS, 15 cm/sec for SC, and 25 cm/sec for MaxS. Any movement faster than 25 cm/second invokes immediate offset recovery. I did not require any additional smoothing mechanisms other than PRISM itself, as the position coordinates received from the wand had less noise than the orientation.

Frees et al [9] calculated the hand speed by comparing its current position to its position 500 milliseconds in the past, which allowed them to ignore back-and-forth motion caused by hand jitter. I used a position 200 milliseconds in the past to make the changes to CD gain more responsive, and did not notice any significant increases from hand jitter in the computed speed.

I multiplied PRIMS' CD gain by 4, which allowed users to cross the screen with a single movement. If multiple movements were required, users could clutch by pressing a button on the wand.

8.2 Task

I used a standard 3D docking task, where an object must be moved to be completely inside a 3D goal area. The object was a red cube, and the goal area was a transparent green cube. I used OpenGL's lighting and custom 2D shadowing to add depth cues to the scene, as seen in figure 8.2.

Participants had 45 seconds to move the object to the goal, otherwise the trial would be marked as 'failed', and the next trial would start. I provided feedback with either a green checkmark indicating success, or a red cross for failed trials.

8.3 Experiment Setup

15 participants volunteered for the study, and were given a small amount of course credit in exchange for participating. I used a within-participants design, and counterbalanced the order of the techniques. The object was a cube 75 cm across. I used three goal area sizes: 3 cm larger than the object, 6 cm larger than the object, and 9 cm larger than the object. At the beginning of each trial the goal and object were randomly placed within the scene, and were spaced either 5.5, 4.25 or 2.75 meters apart.

Since 3D-Pad and 3D-Pad Blank are very similar, I doubled the number of trials for PRISM. In this way, participants used PRISM half the time and a 3D-Pad technique half the time. I used a block size of 9 trials, and 5 blocks per technique. There was a 10-second break in between blocks. Participants had a block of 9 practice trials at the start of each new technique.

5 blocks $\times$ 9 trials $\times$ 4 techniques = 180 trials

The target size and distance were varied within each block. With 36 practice trials per participant, we get 216 trials per participant, and 3240 trials total.

8.4 Results

I removed failed trials from the results before examining the data, resulting in less than 0.26% of trials removed.

8.4.1 Trial Times

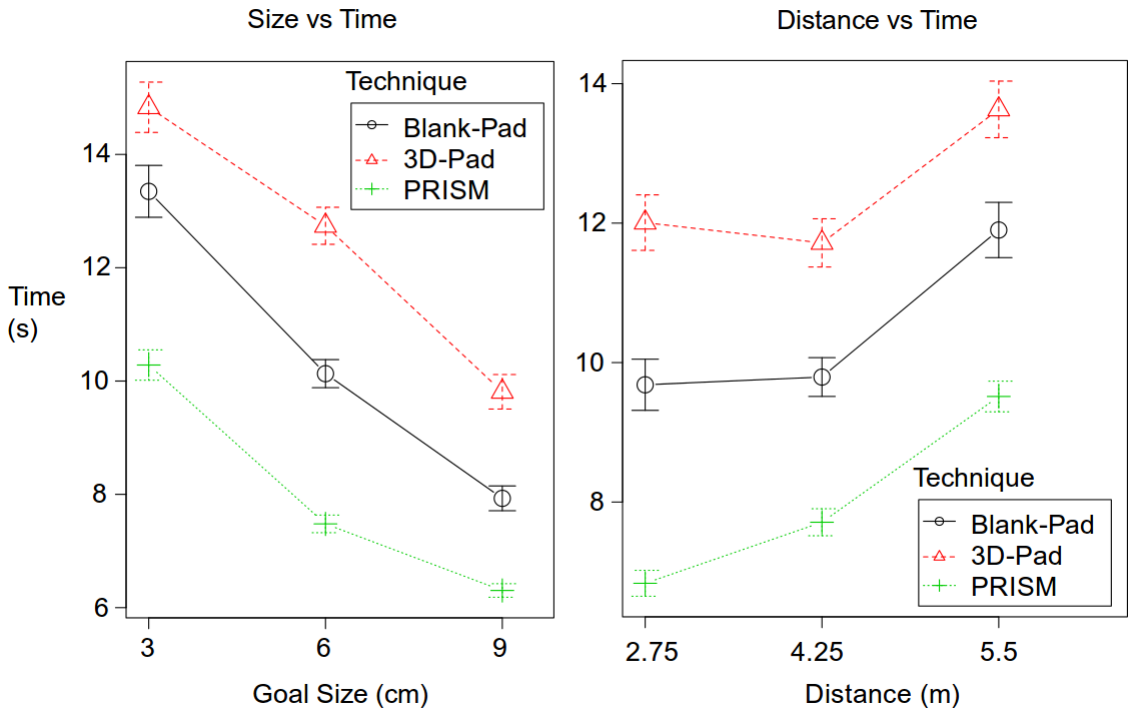
I applied a multi-way ANOVA to examine the effect of *technique*, target *distance* and goal *size* on trial times. I applied the Bonferroni correction, and found a significant effect of *distance* ($F_{2,28} = 88.8$, $p < 0.001$), *size* ($F_{2,28} = 263$, $p < 0.001$) and *technique* ($F_{2,28} = 397$, $p < 0.001$). I found an interaction effect between *distance* and *size* ($F_{4,56}$

= 13.4, p <0.001), between *distance* and *technique* ($F_{4,56}$ = 3.96, p <0.05), between *size* and *technique* ($F_{4,56}$ = 6.27, p <0.001), and between *distance*, *size* and *technique* ($F_{8,112}$ = 5.20, p <0.05).

Levene's Test of homogeneity found equal variances across techniques ($F_{2,28}$ = 5.14, p <0.01). Post hoc comparisons with Tukey's HSD revealed significant differences between all pairs of techniques at the 95% confidence interval.

Figure 8.3 shows how the trial time varies along with goal size and distance. We can see that all three techniques are negatively affected by smaller target sizes, and all three have increased trial times at longer distances. Overall, PRISM was the fastest technique, with a mean completion time of 8.02 seconds across all sizes and distances. Blank Pad was slower, at 10.4 seconds. 3D-Pad was the slowest technique, taking 12.5 seconds on average.

Figure 8.3: 3D Experiment - The Effect Of Target Distance And Size On Trial Times



Bars are one standard error.

8.4.2 Preferences

There were only minor differences between user preferences in most categories, though PRISM had a substantially lower rating than the other techniques in terms of comfort. This may be a result of the continual arm gesturing required to translate the object with PRISM. 3D-ARC-Pad received the highest overall rating at 4.1, despite having the slowest performance.

Table 8.1: User Preferences.

	Blank Pad	3D-Pad	PRISM
Easy	3.9	4.1	4.1
Fast	3.6	3.7	3.8
Precise	3.8	3.7	3.4
Comfortable	4	4.1	2.9
Overall	4	4.1	3.6

8.5 Discussion

PRISM's had the best performance, and Blank Pad required an average of 2.2 additional seconds to complete a trial. I find this result to be encouraging, since I came very close to the performance of a dedicated 3D input device with 6 degrees of freedom, using only an iPod touch.

3D-Pad's screenshots hindered performance, and there are several possibilities as to why this might be. First, the screenshots may have been distracting rather than helpful, and moved users' focus away from the primary display. Second, the screenshots introduced roughly 40 milliseconds of latency to the iPod's network connection, slightly increasing the object's response time to cursor commands. And finally, Screenshots may have suffered from its low frames per second.

By taking screenshots through OpenGL, I was able to obtain an average of 4.2 screenshots per second in the 3D task, as opposed to only 2.4 in the 2D experiments. However, in experiments one and two the scene was static. The target positions only changed at the start of a new block of trials. The low frames per second did not have any impact on 2D performance, since the target position in each frame remained constant. In the 3D task, the target object and scene viewpoint are continually moving, making the latency more critical.

Without further experimentation, I can not say which explanation is valid. I would like to simulate the extra latency caused by screenshots in the Blank Pad technique, to eliminate latency as a confounding variable. I would also like to achieve a higher number of frames per second on the device screen, though this is not possible with my current hardware configuration.

Next to the target object is a yellow bar, indicating the device's orientation. In the future I would also like to modify that bar by stretching it to cover the entire screen, and making it transparent. The user could tilt the phone until the bar intersects the target, then tap on the target to accurately jump to it.

Chapter 9

Conclusions and Future Work

I have presented ARC-Pad, which allows users to quickly reach distant targets without a corresponding loss of accuracy. It uses a hybrid absolute/relative approach, where tapping invokes absolute jumping, and sliding allows for accurate cursor positioning. I presented the work in three experiments; in experiment one, I ran a small study to verify that the basic idea of cursor jumping has merit, and showed that it outperforms a standard touchpad on a medium-sized display.

I saw the potential to greatly improve ARC-Pad's performance with several modifications. I implemented it on a more responsive touchscreen, incorporated multi-touch gestures for left and right clicking, allowed users to manipulate the CD gain by tilting the device, and streamed screenshots from the desktop to the device. In experiment two, I evaluated the new features individually. I found that tilting the device to adjust the CD gain had a very high error rate, and so I discarded it.

ARC-Pad and ARC-Pad with screenshots did very well in 1D target selection, but had poor results in the 2D tapping task. I felt this was because the targets were not as far apart, and because I had instructed participants to jump even if they did not feel the need. I used these results in my final design, which had all of the proposed features except for tilt.

In experiment three I compared my final design, ARC-Pad 2, to existing solutions for cursor control. I performed a thorough evaluation of ARC-Pad 2, Air Mouse and PRISM, measuring trial times, error rate, throughput, target re-entries, and movement variability. I found that ARC-Pad 2 now had by far the best throughput in 2D target selection, with significantly lower error rates and trial times. It also matched the Air Mouse's throughput in 1D target selection, and had a much lower error rate than the Air Mouse.

Participants gave the best ratings to ARC-Pad 2 for ease of use, perceived speed and precision, and comfort. They also rated it as the overall best technique. ARC-Pad 2 had the lowest movement variability, and on average the cursor's path was only about 1.1 times the length of the optimal path to the target (a straight line).

Experiment two evaluated ARC-Pad on a large display, and showed that ARC-Pad can be used to quickly and accurately select targets only 4 cm wide, and up to 7 meters apart. ARC-Pad significantly exceeded the performance of existing solutions for 2D cursor positioning, even though those solutions required specialized hardware.

In the final experiment, I extended ARC-Pad to position a 3D cursor object, using accelerometer data to determine the mobile device's orientation. I compared 3D ARC-Pad to a 6 degrees-of-freedom wand with PRISM's adaptive CD gain. ARC-Pad came close to the wand's performance, taking an average of 10.4 seconds per trial compared to PRISM's 8 seconds. I feel these results are encouraging, especially since 3D ARC-Pad is an inexpensive, portable solution. I outlined several possible methods of increasing 3D ARC-Pad's performance. Future work includes adding visual feedback to assist users in making 3D jumps, and improving the frames-per-second of the streamed screenshots.

Overall, I have demonstrated that ARC-Pad is a compelling technique for cursor control. It has a low cost, high speed, good accuracy, and high user ratings. ARC-Pad is not affected by target distance in 2D object selection, making it ideal for large

displays. I have also shown that ARC-Pad has the potential to be an effective tool for 3D object manipulation, and comes close to the performance of a dedicated 6 DOF input device.

Bibliography

[1] Allman, M., V. Paxson, and E. Blanton (2009, September). TCP Congestion Control. RFC 5681 (Draft Standard).

[2] apple (2011). Apple iPod touch features. `http://www.apple.com/ipodtouch/features/multitouch.html`.

[3] Ascension (2010). Ascension Technology Corporation. `http://www.ascension-tech.com`.

[4] Baudisch, P., E. Cutrell, D. Robbins, M. Czerwinski, P. Tandler, B. Bederson, and A. Zierlinger (2003). Drag-and-pop and drag-and-pick: Techniques for accessing remote screen content on touch- and pen-operated systems. Zurich, Switzerland, pp. 57–64.

[5] Casiez, G., D. Vogel, R. Balakrishnan, and A. Cockburn (2008). The impact of control-display gain on user performance in pointing tasks. In *Human-Computer Interaction*, Volume 23, pp. 215–250.

[6] Casiez, G., D. Vogel, Q. Pan, and C. Chaillou (2007). RubberEdge: Reducing clutching by combining position and rate control with elastic feedback. In *ACM symposium on User Interface Software and Technology (UIST)*, Newport, Rhode Island, USA, pp. 129–138.

[7] Chittaro, L., R. Ranon, and L. Ieronutti (2009). 3D object arrangement for novice

users: the effectiveness of combining a first-person and a map view. In *The ACM Symposium on Virtual Reality Software and Technology (VRST)*, pp. 171–178.

[8] Forlines, C., D. Vogel, and R. Balakrishnan (2006, April). Hybridpointing: Fluid switching between absolute and relative pointing with a direct input device. In *ACM symposium on User Interface Software and Technology (UIST)*, pp. 547–552.

[9] Frees, S., G. D. Kessler, and E. Kay (2007, May). PRISM interaction for enhancing control in immersive virtual environments. In *ACM Transactions on Computer-Human Interaction (TOCHI)*, Volume 14.

[10] Gyration (2010). Gyration. `http://www.gyration.com`.

[11] Hascoët, M. (2003). Throwing models for large displays. In *Human-Computer Interaction*, Volume 2, pp. 73–77.

[12] ISO 9241-9 (2000). ISO 9241-9 Ergonomic requirements for office work with visual display terminals (VDTs) Part 9: Requirements for non-keyboard input devices TC159/SC4. `http://www.iso.org/iso/iso_catalogue`.

[13] Kobayashi, M. and T. Igarashi (2008, March). Ninja Cursors: using multiple cursors to assist target acquisition on large screens. In *Conference on Human Factors in Computing Systems (CHI)*, pp. 949–958.

[14] MacKenzie, I. (1992). Fitts' law as a research and design tool in human computer interaction. In *Human-Computer Interaction*, Volume 7, pp. 91–139.

[15] MacKenzie, I. S. and P. Isokoski (2008). Fitts' throughput and the speed-accuracy tradeoff. In *Conference on Human Factors in Computing Systems (CHI)*, pp. 1633–1636.

[16] MacKenzie, I. S., T. Kauppinen, and M. Silfverberg (2001). Accuracy measures

for evaluating computer pointing devices. In *Conference on Human Factors in Computing Systems (CHI)*, New York, USA, pp. 9–16.

[17] Martinet, A., G. Casiez, and L. Grisoni (2010, September). The design and evaluation of 3d positioning techniques for multi-touch displays. In *IEEE Symposium on 3D User Interfaces (3DUI)*, pp. 115–118.

[18] McCallum, D. and P. Irani (2009, May). ARC-Pad: absolute+relative cursor positioning for large displays with a mobile touchscreen. In *ACM Symposium on User Interface Software and Technology (UIST)*, pp. 153–156.

[19] Microsoft (2002). Pointer ballistics for Windows XP. `www.microsoft.com/whdc/archive/pointer-bal.mspx`.

[20] Nielson, G. M. and J. D. R. Olsen (1987). Direct manipulation techniques for 3D objects using 2D locator devices. In *ACM Workshop on Interactive 3D Graphics*, pp. 175–182.

[21] Potter, R., L. Weldon, and B. Shneiderman (1988). Improving the accuracy of touch screens: an experimental evaluation of three strategies. In *Conference on Human Factors in Computing Systems (CHI)*, pp. 27–32.

[22] Soukoreff, R. W. and I. S. MacKenzie (2004). Towards a standard for pointing device evaluation: Perspectives on 27 years of Fitts' law research in HCI. In *International Journal of Human-Computer Studies (IJHCS)*, Volume 61, pp. 751–789.

[23] Vogel, D. and P. Baudisch (2007). Shift: A technique for operating pen-based interfaces using touch. In *Conference on Human Factors in Computing Systems (CHI)*, pp. 657–666.

[24] Weberg, L., T. Brange, and W. Hansson (2001). A piece of butter on the pda display. In *Conference on Human Factors in Computing Systems (CHI)*, pp. 435–436.

[25] Zhai, S. (2004). Characterizing computer input with Fitts' law parameters - the information and non-information aspects of pointing. In *International Journal of Human-Computer Studies (IJHCS)*, pp. 791–809.

Appendix A: Sample Consent Form

[Target Selection]
Page 1/1

Participant Information

Date / Time: _____

Participant Number: _____ Participant Name: _____

Student No:_____

Informed Consent Agreement
Please read this consent agreement carefully before you decide to participate in the study.

Purpose of the research study: To evaluate the performance of different input devices for selecting targets on a large display.

Time required: The study will require about 30-40 minutes of your time.

Risks: There are no anticipated risks in this study.

Benefits: You will receive one hour of experiment credit in exchange for your participation.

Confidentiality: The information that you give in the study will be handled confidentially. It will be viewable only by researchers working on this project, which may involve faculty members and graduate research assistants.

Voluntary participation: Your participation in the study is completely voluntary.

Right to withdraw from the study: You have the right to withdraw from the study at any time without penalty.

How to withdraw from the study: If you want to withdraw from the study, please inform the experimenter and leave the room. There is no penalty for withdrawing. You will still receive full credit for the study. If you would like to withdraw after your materials have been submitted, please contact one of the researchers listed below.

If you have questions about the study, contact:
David McCallum
Department of Computer Science, E2-560 EITC
University of Manitoba, Winnipeg, Manitoba R3T 2N2, Canada
Email: dmccallum9@gmail.com

Faculty Advisor: Dr. Pourang Irani
Department of Computer Science, E2-580 EITC
University of Manitoba, Winnipeg, Manitoba R3T 2N2, Canada
Phone: +1 (204) 474-8995

Agreement:
I agree to participate in the research study described above.

Signature: _____ Date: _____