ABSTRACT

DULBERG, MARTIN S. A Task-based Framework for the Quantitative Evaluation of Input Devices. (Under the direction of Robert St. Amant and David F. McAllister.)

This research describes the development of a conceptual framework and methodology that will permit the evaluation of input devices in graphical user interfaces in a more meaningful context than previous studies. We provide a procedure for the reuse of performance characteristics in expanded domains. Individual performance differences are analyzed and their implications for current evaluation methods are discussed. We have built an interactive simulation for domain-independent testing of the suitability of different input devices for specific tasks, based on the demand characteristics of the task and the performance characteristics of the device. The simulation will allow researchers and practitioners to evaluate the suitability of particular input devices in a given user interface with a severely restricted role for usability testing. Using the system, it will be possible to select a device based upon optimal task completion time or estimated error rate. The role of inter-task transition times is explored. A methodology for prediction of performance with the use of execution graphs is described.

A TASK-BASED FRAMEWORK FOR THE QUANTITATIVE EVALUATION OF INPUT DEVICES

by
MARTIN S. DULBERG

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

COMPUTER SCIENCE

Raleigh

2003

APPROVED BY:

_____	_____
Co-chair of Advisory Committee	Co-chair of Advisory Committee

PERSONAL BIOGRAPHY

Martin Dulberg was born in 1960 in New York City. He received a Bachelor of Arts in Computer Science from the City University of New York, Queens College in 1994 and a Master of Science in Computer Science from North Carolina State University in 1996.

ACKNOWLEDGEMENTS

This research was partially supported by the National Science Foundation, award number IIS-0083281. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright notation hereon.

Table of Contents

List of Tables

List of Figures

Chapter 5

1. Introduction

A graphical user interface (GUI) is a visual environment through which a user can interact with a software application. GUIs derive their effectiveness from their support for direct manipulation [Shneiderman, 1983]. Direct manipulation interfaces are associated with

- The visibility of objects of interest.
- The replacement of language with action.
- Incremental action and rapid feedback.
- Reversibility of actions.
- Syntactic correctness of all actions.

These properties encourage users to explore the functionality of the interface, promote learning and retention, and enable users to gain mastery of the interface more quickly than through most other forms of interaction.

Direct manipulation interfaces have become the dominant paradigm in human-computer interaction, and GUIs are the most common form that they take. In a GUI, virtual objects represent their counterparts in the domain of the application. In GUIs based on the common desktop metaphor, for example, we find visual representations of manila folders, paper documents, calculators, sticky notes, address books, and other office tools and artifacts. GUIs for word processing applications generally contain visual representations of rulers, typefaces, bulleted lists, and other properties of text and documents. These types of

presentation are sometimes generalized as the "WIMP" model, being based on windows, icons, menus, and a pointer (e.g., a mouse or touchpad.)

The visual representation is only part of the GUI; interaction with the GUI must support problem-solving activities in the application domain. A desktop GUI thus supports the dragging of file icons from one spatial location to another, representing file movement in the directory system; a word processing GUI supports the activation of buttons that change the visual properties of text. A key issue for developers is designing visual representations that appropriately reflect the support of the GUI for the tasks that the user needs to carry out. Ideally, the GUI will provide a set of conventions for interacting with the application domain that can be easily comprehended and remembered, quickly learned, and efficiently executed [Dix 1998].

Producing such interfaces is non-trivial. For a given GUI, significant performance differences can arise due to natural physiological variation in users, differences in domain knowledge and reasoning abilities between users, and differences in the ways that users interpret the capabilities of the interface based on its visual representation. In comparing different GUIs developed for the same task, further performance differences are due to the alternative mappings of tasks to GUI interactions, the spatial structuring of screen elements, and even variations in the size and appearance of those elements. The performance properties of a given GUI are so complex and tightly interrelated that it is practically impossible to predict them in advance, from first principles. Evaluation has thus come to occupy a critical role in user interface development.

The specific evaluation issue addressed in this dissertation deals with the match between the tasks that a user interface is designed to support and the low-level performance properties of the physical input device through which the interaction occurs. For example, consider the task of steering a car in a simulation. The task in this case would be indicating one of two directions (steer left or steer right), and possibly a magnitude (e.g. bearing left versus a hairpin left turn), depending on the simulation software. Any number of input devices might be used for this task: a mouse, a touchpad, a joystick, arrow keys, even a steering wheel. Intuitively, some of these devices will be more effective than the others, especially if the task is considered in the context of other tasks that must be carried out simultaneously, such as adjusting the speed of the car, attending to road signs, and so forth. The problem of selecting the best match between the steering task and the input device can involve a significant amount of analysis and empirical experience. (Early steering devices in the history of the automobile include steering bars, steering sticks, and even leather reins comparable to those used for horse-driven carriages; modern steering devices in prototype cars include joysticks and immobile force sensitive steering wheels). Every year novel computer input devices, either general-purpose or specialized for particular domains, are described in the human-computer interaction literature, each with distinctly different capabilities and performance differences. An important system design issue is determining how effective an input device will be for performing a particular task with a particular GUI.

The suitability of input devices as they relate to task performance is generally evaluated by empirical means through controlled experiments. For example, MacKenzie, Sellen, and Buxton [1991] studied the suitability of mice, trackballs and tablets for pointing and

dragging tasks. In such experiments the user is asked to perform a task or series of tasks with different devices to determine which one performs better for the task being evaluated. Better is usually defined as faster task completion time or fewer errors, while errors are usually defined by the experimenter within the context of the task being performed. For their study MacKenzie et al. developed an experiment and wrote software that provided the targets for the users to point at, as well as collected the data from each trial. Twelve subjects were recruited and presented with trials of both tasks (dragging and pointing), using four different target amplitudes with four different widths, for a total of 32 different conditions. Once all of this data was collected, it was analyzed and adjusted to look at movement time, error rate, movement time adjusted for errors, and fit of data to existing performance models.

This study is an excellent model for empirical research in this area, and it has been adopted widely in HCI. Generally, to conduct research that will yield meaningful results, it is necessary to design and implement an experiment, recruit participants, have the participants perform the experiment while data is being collected, and finally analyze the data to determine the performance characteristics of the devices. Implementing the experiment usually involves writing a customized version of the application with data logging procedures built in. Somewhere between four and twenty subjects must be recruited and monitored during the experiment. Once the experiment is done, the data are filtered, aggregated, and analyzed to get a result. The MacKenzie et al. study points out a number of broad, general findings about the performance characteristics of the above devices in relation to the two tasks analyzed. Dragging takes longer than pointing with the devices

tested and it involves a higher error rate than pointing, for example. We can even conclude that a tablet is superior to a mouse for pointing tasks but not for dragging.

For all the information that we can glean from such experiments, however, there are significant missing elements. These provide the motivation for our research.

- *Ecological validity.* Interaction tasks are usually tested in isolation, for the sake of experimental control. Real user interfaces generally involve much more complex tasks than those used in testing the input device, however, which means that individual tasks carried out in an experimental testbed may or may not provide a good match for task combinations that are required in deployed interactive systems.

- *Task coverage.* Device evaluations almost always rely on a set of tasks that include pointing and dragging, with many being limited to just these two tasks. The scope of actions possible in GUIs is much wider than this; some action taxonomies include rotation and path following as elements, for example, and it is not always clear that these can be treated as composite tasks whose performance is precisely predictable from an analysis of their individual component tasks.

- *Standards for cross-device comparison.* Although comparative studies of devices usually lay out their procedures in clear detail, they are often implicitly tied to those devices under consideration. When tasks and procedures are described that go beyond pointing and dragging, we find variation across the devices being evaluated.

This can make it difficult for those considering the adoption of a novel device to determine its comparative advantages in practice.

- *Individual differences*. By focusing on broad patterns in device comparisons, evaluations usually neglect individual performance differences. For example, do trends exist across users? Will all individuals that do well using a mouse will also do well using a tablet?

This dissertation addresses these issues and a number of related questions that they raise.

1.1 Contributions

This dissertation describes the development of a conceptual framework and methodology that will permit the evaluation of input devices in GUIs in a more meaningful context than previous studies. We provide a procedure for the reuse of performance characteristics in expanded domains. Individual performance differences are analyzed and their implications for current evaluation methods are discussed. We have built an interactive simulation for domain-independent testing of the suitability of different input devices for specific tasks, based on the demand characteristics of the task and the performance characteristics of the device. The simulation will allow researchers and practitioners to evaluate the suitability of particular input devices in a given user interface with a severely restricted role for usability testing. Using the system, it will be possible to select a device based upon optimal task completion time or estimated error rate. In order to perform this evaluation with current methods it would be necessary to exhaustively explore the space of device-task pairs. This

type of analysis would require an inordinate number of comparisons and time, and it would yield an evaluation that would be of value only to the domain-specific task that was tested. Our research provides a framework that enables a researcher or designer of interactive systems to quickly evaluate device suitability for a wide range of tasks.

The approach taken in this dissertation involves the following steps:

- *Development of a taxonomy of elemental tasks.* We have identified a set of core, fundamental tasks that users will perform in a GUI. The tasks in the taxonomy are low-level operations that can be chained together to allow the modeling of more complex interaction tasks. Examples of elemental tasks are key presses, clicks, and moving the cursor to a particular location.

- *Selection of representative input devices.* We have chosen a representative sample of input devices in our evaluation, each device associated with a characteristic set of appropriate tasks. Although the input devices we have chosen are conventional, they have not undergone the type of analysis presented in this dissertation. The set is furthermore easily extensible to other less conventional input devices.

- *Modeling: task execution paths.* Once a task list and a device list have been created, a set of execution steps can be derived for each task on a device by device basis, assuming that the device is capable of performing the task. A series of execution steps represents an execution path for a particular device and task pairing. We have analyzed device task pairings and produced appropriate task execution paths.

- *Modeling: task execution graphs.* We have developed task execution graphs to combine execution paths into a concise, convenient representation. Nodes in the graph represent states of the user interface. Outgoing arcs from each node represent the elemental actions or steps that the user performs. Two weighting factors are associated with the transitions in each task graph. One value represents an execution time for the particular action represented by the transition, the other an error probability. Task execution graphs, when parameterized in this way, can be applied as predictive models of user performance.

- *Simulation:* A GUI test bed was created with a trial presentation interface, extensive timing, and event logging mechanisms and reporting capabilities, which permit the observation and analysis of user actions within the test bed. The user is presented with a series of curve matching tasks with randomized orderings and position requirements. By altering the task orderings to complete a trial, we are able to capture performance data for all possible combinations of task orderings and transitions between tasks.

- *Implications of user differences to conventional evaluation methods.* Our results demonstrate that individual user performance varies widely from median performance and even with respect to the relationships between different devices. This leads us to conclude that conventional evaluation methods are of very little benefit in comparing device usage characteristics for substantial segments of the user population.

The contributions of this thesis are the development of a simulation environment, the design of a new model for representing and analyzing interaction, and the empirical results of applying the interaction model to data produced by users using the simulation. Our research has led to interesting findings in the area of modeling task and transition performance, the role of individuality of user performance to prediction findings and a new methodology for modeling user interaction. We have made contributions in the following areas:

- *Analysis: inter-task transition times.* Inter-task transition times are predictable. Models in common use in user interface performance modeling vary widely in the level of detail at which they represent user behavior. The most popular approaches represent tasks individually, rather than in combination. Our work shows that individual task performance depends on the sequencing of the tasks involved, and that this performance can be characterized naturally and concisely. The result is an improvement in modeling accuracy with only a small increase in modeling complexity.

- *Analysis: device-dependent transition times.* Transition times depend on device. Common modeling approaches generally neglect the interactions between modeling predictions and devices under consideration. In the keystroke level model [Card 1980], for example, pointing operations are generic, generating predictions that are independent of whether pointing is carried out with a mouse, a touch pad, a trackball, or some other device. Performance models for these different devices

exist in the literature; we show how they can be incorporated into general modeling procedures, again taking account of transitions between tasks of different types.

- *User performance differences.* Current evaluation methodologies do not take individual performance characteristics into account, preferring instead to generalize over a large population. This generalization has the effect of diminishing the applicability of the results to any particular user in the population. We find that while there appear to be some general trends with regards to performance, a substantial minority of users cut against that trend.

1.2 Outline

The remainder of this thesis is organized in the following manner:

- Related Work: A review of relevant research is done. We look at studies related to interaction and device usage in GUI's in section two.

- Interaction modeling: In section three we present a new model for interaction in a GUI and explain the steps necessary to construct a model and populate it with user performance data.

- Simulation design and experiment procedures: We look at the constructional issues related to the simulation in section four and provide details of how data was collected for the 24 participants that were in our study.

- Data Exploration and Analysis: In section five we review the results of the analysis of the participant data looking at task and transition based performance at the aggregate and individual level.

- Conclusions: In section six we discuss the implications of our findings.

2. Related Work

This proposal draws upon a large body of work in the HCI and psychology literature. We have broken this up into six general areas of interest.

- *Evaluation and calibration of input devices:* Numerous studies have evaluated how long it takes a user to complete a task and how well they are capable of doing it. The findings of these studies inform our efforts to create a model of human performance that supports the evaluation of device suitability in terms of task completion time and some metric of the quality of performance. The data in these studies provide us with a body of empirical data that we can use to both create and validate our model.

- *Comparison of input devices:* Many studies have analyzed the performance metrics of a sampling of devices. We aim to extend this body of research with a new framework for the evaluation of these devices. It is therefore important to understand the strengths and limitations of work in this area. Each study in this area has been done with a customized piece of experimental software, using a different methodology and different analysis methodologies. This has the effect of limiting the applicability of the results of these studies and diminishing the ability to generalize about the findings. Our aim is to provide a more consistent approach to device evaluation.

- *Device taxonomies:* The range or type of input devices as well as the characteristics that make devices similar or different have also been studied in detail. These classification dimensions have helped us structure our model appropriately by giving us a schema for covering the design space of relevant devices for a particular task.

- *Models of device usage:* Various modeling techniques attempt to explain different aspects of device usage through biomechanical, cognitive and probabilistic means. These models are successful at describing device usage in different domains and different contexts. We intend to use these models both as guidance for our model and for verification that our model is reliable.

- *Task taxonomies:* Taxonomies have been developed that describe what type of tasks are typically performed and how they are defined. We intend to use these task taxonomies as a starting point for our model, to provide for a wide range of evaluation conditions that might reasonably be encountered in the course of an evaluation.

- *Other Related Work:* Finally, there is additional research that does not fit into any of the categories above which we believe is important to understand the context of our work and the current state of the art in device evaluation.

2.1 Evaluation and Calibration of Input Devices

Input devices have been evaluated for many different performance characteristics. These evaluations have been used to determine the performance characteristics of devices, usually with a single task, in order to uncover the underlying nature of user performance. Task completion time has been studied extensively [Albert 1982, Card 1978, MacKenzie 1991, MacKenzie 1992, Meyer 1988, Murata 1991, Walker 1993, Worden 1997], time to complete a task along paths of different angles [Dulberg 1999, Jagacinski 1985], and motion along a curved path [Accot 1997]. Error rates or mistakes in performance have also been calculated in the above work as well as other studies. User preferences (including the phenomenon that users may not prefer the most efficient devices) have also been examined [Andre 1995].

A large body of literature starting with Paul Fitts's seminal work [Fitts 1954] investigates the time required to complete a pointing task with a variety of different of apparatus, under a multitude of different conditions assuming expert performance with a low error rate. Most of these works have validated some form of Fitts' law equation, which takes the form of a log relationship such as the following:

$$T=a+b \log_2 (2A/W)$$

where T is time, a and b are experimentally derived constants, A is the distance to the center of the target, and W is the width of the target. The index of performance (IP) of a device is the reciprocal of b. Different devices have a different (IP) [Douglas 1994, MacKenzie

1992]. Fitts' Law has been experimentally validated for a variety of tasks such as pointing, tapping a stylus, hand movement, wrist rotation, joystick use and mouse positioning [Albert 1982, Card 1978, MacKenzie 1991, MacKenzie 1992, Meyer 1988, Murata 1991] and can thus be used as a tool for the predictive measures of performance times for many cursor prediction tasks. For example, MacKenzie, Sellen and Buxton [MacKenzie 1991] showed that pointing and dragging tasks fall under the category of Fitts' Law tasks. They also analyzed the rate of dropping errors for the dragging task.

The log form of Fitts' law has been the target of some investigation. Meyer et al [Meyer 1988?] is one representative sample. They proposed a stochastic optimized-submovement model which characterizes target acquisition as a gross movement followed by a corrective submovement. Their work yields a formula for predicted time of:

$$T = A + B\sqrt{D/W}$$

where A and B are experimentally derived nonnegative constants, D is distance and W is target width. This model, like some of the other modifications to Fitts' law yields slightly more accurate predictions at shorter distances with very similar predictions to Fitts' law at longer distances. For theoretical reasons, however, the log form of Fitts' law remains the standard. As Newell explains it [Card 1983], the time to acquire a target is based on a series of movements by the user until the goal is achieved. Each movement requires that the user make use of their perceptual processor to observe where their hand or cursor is (perceive the current state of the environment), then their cognitive processor to decide on any correction needed (goal formation) and finally use their motor processor to carry out

the action. This process is repeated *n* times until the goal is achieved. He goes on to show that the relationship between the time to complete the action and the environment is based on a logarithmic term of the distance to, and width of, the target.

Researchers do not always agree on the value of IP for a particular device. For example, the IP of the mouse varies from 1.1 to 10.4 [MacKenzie 1991]. What accounts for this discrepancy? Two factors are at work here: Different researchers use a different term for the IP, and there are also quality issues with regard to the manufacture of the device. Some mice are superior to others, employing finer quality mechanisms for the tracking of movement, higher sampling rates to provide better feedback and perhaps a better ergonomic shape to allow the user to manipulate the mouse more easily. We do not concern ourselves with this aspect of device evaluation other than to demonstrate that it exists and to provide some recommendations for compensation methods. We believe that it represents itself as a constant factor in task performance time and so can be accounted for. What is less clear is the effect that such differences in quality will have on errors.

2.2 Comparison of Input Devices

The comparison literature specifically looks at inter-device differences to determine which devices are best suited to a particular task in either the time, variability of time, or error rate dimensions. All of the following studies contain numerical results regarding device comparisons. We have used these studies as the starting point for our model; they have also influenced our empirical approach.

One of the earlier papers that investigated the different performance characteristics of various input devices was Card, English and Burr's work [Card 1978], which studied the performance characteristics of the mouse, isometric joystick, step keys (which resemble a diamond layout of arrow keys with a home key in the center), and text keys (which have dedicated functionality like shortcut keys). The paper primarily looks at how rapidly and accurately text can be selected with the four devices. The authors took five novice users (one of whom was discarded for being significantly slower than the others) and discarded results from the first 200 trials to control for practice effect (the subjects ran approx. 800 trials for each condition). The mouse was found to be faster and less error prone with less of an increase in positioning time for greater distances, which is consistent with Fitts' Law. The positioning time for the isometric joystick grew at the same rate as the mouse, but the values were consistently higher regardless of the distance. The step keys and text keys saw performance degrade at a much faster rate as the distance increased. The mouse had a significantly lower error rate than the other three devices regardless of the target size.

The authors also discussed the effects of approach angle to the target. They found no significant difference in positioning time when using a mouse regardless of the angle of approach, whereas the other three devices had a noticeable increase in performance time as the angle of approach to the target became more oblique, because the target was a piece of text and hence was only one character high. The reason that they might not have found a difference in the mouse performance was that they were only looking at the interval 0-90 degrees broken up into 3 bins (0-22.5 , 22.5-67.5 , 67.5-90) and might have missed more subtle differences.

Albert [Albert 1982] compared seven different input devices for accuracy, speed, and a subjective evaluation of each device that involve a graphical input task that required the participant to select a target. There were differences in the groups of devices that were considered more or less direct. In this context we refer to directness as having a one to one correspondence to the world, without an intermediary [Norman 1986]. For example, given a drawing task, most direct to least direct approaches might include: drawing with pen and paper; using stylus and tablet, a mouse; a joystick; a set of typed, drawing commands.

Albert found that the direct manipulation devices, a touch screen and a light pen, afforded faster positioning speed and higher positioning accuracy than less direct devices such as track balls and joysticks. Even within the same device, a touch screen mounted *on* the display provided superior performance over one that was mounted next to the display. Direct hand-eye coordination is presumed to be superior to indirect. A separate "enter" button was also found to degrade the performance of participants due to the homing time required both to context switch to the button, then return to the input device.

MacKenzie, Sellen and Buxton [MacKenzie 1991] looked at pointing and dragging tasks with mouse, trackball and stylus. Fitts' law modeled both tasks well with a higher IP for pointing than dragging. The mouse was superior for dragging while the stylus was faster for pointing. The track ball was third in both cases.

Errors were also investigated, particularly accidentally "dropping" an object while dragging it. The mouse produced the fewest percentage of errors in the dragging task while the

trackball was the most error prone device. There was little difference in errors across devices for the pointing task.

Murata [Murata 1991] looked at pointing tasks and error rates between six devices. The task was for the participants to sort five three-digit numbers. This task was paired with a mouse, trackball, joystick, joycard, light pen and touch screen. A joycard is a type of isometric joystick. The author used 10 naive subjects. He found that the joystick was fastest while the light pen was most accurate. This study however only looked at these devices in the context of a one-dimensional task. The participants were simply picking the row in which the number was located. The author also gives no insight into whether he controlled for learning with regard to task times. The findings in this paper run counter to previous device comparisons in the literature.

Kurtenbach and Buxton [Kurtenbach 1993] did an evaluation of marking menus using pen and mouse input, looking at error and response times based on the number of segments or different selection areas in the circular pie menu and depth of submenus. A marking menu provides a pie menu for novice users but allows experts to simply make the gesture that corresponds to a selection in the novice menus. This gesture would correspond to the path that would need to be traversed through the menus to make a particular selection. They found the pen more accurate and faster than mouse input for this task. The error rate increased as segments, depth or the combination increased while the response time increased at a slower rate.

Guimbretière and Winograd [Guimbretière 2000] discuss an extension of marking menus called FlowMenus. In a suggested usage, the menu appears with eight submenus appearing radially arranged around the center menu at the time the menu is invoked. Using pen based input, users can use a path to select like a marking menu but also use a path to quantify a degree of selection based on path properties. No empirical evaluation was provided.

Douglas and Mithal [Douglas 1994] showed that isometric devices (devices which sense pressure but don't move or change shape as a result of pressure being applied) are Fitts' Law devices. The authors believed that a finger controlled isometric device would operate faster than a joystick (the finger has a higher IP than the hand: Langolf 1976). This supposition did not hold. Even though the keyboard joystick that they tested reduced homing time, it had a longer pointing time component than the mouse. The authors question the assumption in the Keystroke Level Model that homing time to a mouse (or other device) is a constant by suggesting that Fitts' law would dictate otherwise (relationship to distance of the device from the hand).

Zhang et. al. [Zhang 1988] evaluated single and multimodal interaction in a CAD system. Multimodal findings aside (which are beyond the scope of our model), they found that mouse input was superior to pen based input in task completion time accuracy and subjective satisfaction.

While all of the results described thus far in this section have been empirical, there are some qualitative results available as well. van de Pol, Ribarsky, Hodges and Post [van de Pol 1998] looked at interaction techniques for navigation, selection and manipulation

particularly on the virtual workbench, but the findings can be generalized to other large display immersive environments.

They mention five different types of selection interaction techniques:

- Direct Picking: The user reaches out till they actually "touch" the object in question. This was implemented with a glove with positioning info. Direct picking was the most intuitive and easiest selection technique but suffers from the limitation that objects can't be out of reach.

- Ray Casting: By having a ray shoot out from the tip of a finger, the user can use their finger like a laser pointer. This technique is useful for picking distant objects. Feedback must be provided to the user to indicate the current position of the beam in the virtual world; otherwise there can be discrepancies between the perceived and actual position of the ray.

- Gaze directed selection: The user selects an object by looking at it. The ray starts from a point mid-distance between the eyes. There is an uncited mention in the article that it was found to be more intuitive to have the gaze point down rather than straight ahead. A small cursor in the form of a cube is placed on the workbench surface to provide feedback to the user. In the case of occluding objects, the closest one is selected.

- Pointing: The user selects an object by covering it with their fingertip.

- Virtual Hands: A representation of one's hands is provided in the scene much like direct picking; however, a nonlinear mapping technique is employed to allow the user to select objects outside of their reach.

An unknown method was used to evaluate the effectiveness of these methods. Ray casting was found to be most effective for distant object selection.

All of the studies that we have seen that relate to device comparisons have demonstrated that performance is based on Fitts' law with few minor differences. None of these studies look at individual performance differences, concentrating instead on the overall patterns of device usage across populations.

2.3 Taxonomies of Input Devices

A taxonomy of input devices is useful as a means for classifying and understanding the underlying nature of input. By classifying devices according to characteristics like dimensionality, degrees of freedom, or activation methods, the taxonomy provides insight into the nature of how devices are used, as well as hints at where one might start looking for similarities and differences in the performance characteristics of the devices.

Buxton [Buxton 1983] proposed a taxonomy that classifies devices along two dimensions, the property that is sensed and the number of dimensions that the device can track (figure 2.1). Devices are then further classified according to whether they sense input via

mechanical or touch sensitive means. The distinction that Buxton was making is very similar to the difference between direct and indirect manipulation, as discussed above.

The number of dimensions relates to the number of degrees of freedom along which the device affords movement. A slider or rotary control allows or controls a single dimension of movement, while a mouse, joystick, tablet or light pen allows motion along two dimensions.

The properties that can be sensed are position, motion and pressure. A tablet, slider or light pen sense position; a mouse, trackball or joystick sense motion; while an isometric joystick, some styli on tablets and a space ball sense pressure or torque.

The device taxonomy allows us to group devices that are similar in performance characteristics with the hope of simplifying our model. It allows us to establish relevant relationships between devices that are close to each other in the possible design space.

One of the major drawbacks of the Buxton taxonomy is the lack of descriptive ability for discrete devices. Without that capability, it is limited to a subset of all input devices. Another drawback relates to the way that devices are categorized according to how they sense input. While it might seem appropriate, there is little evidence provided to indicate that it is a meaningful distinction. The author also mentions the notion of being able to make analogies between devices according to their relative positions in the taxonomy. There is no evidence to suggest that these might be meaningful relationships.

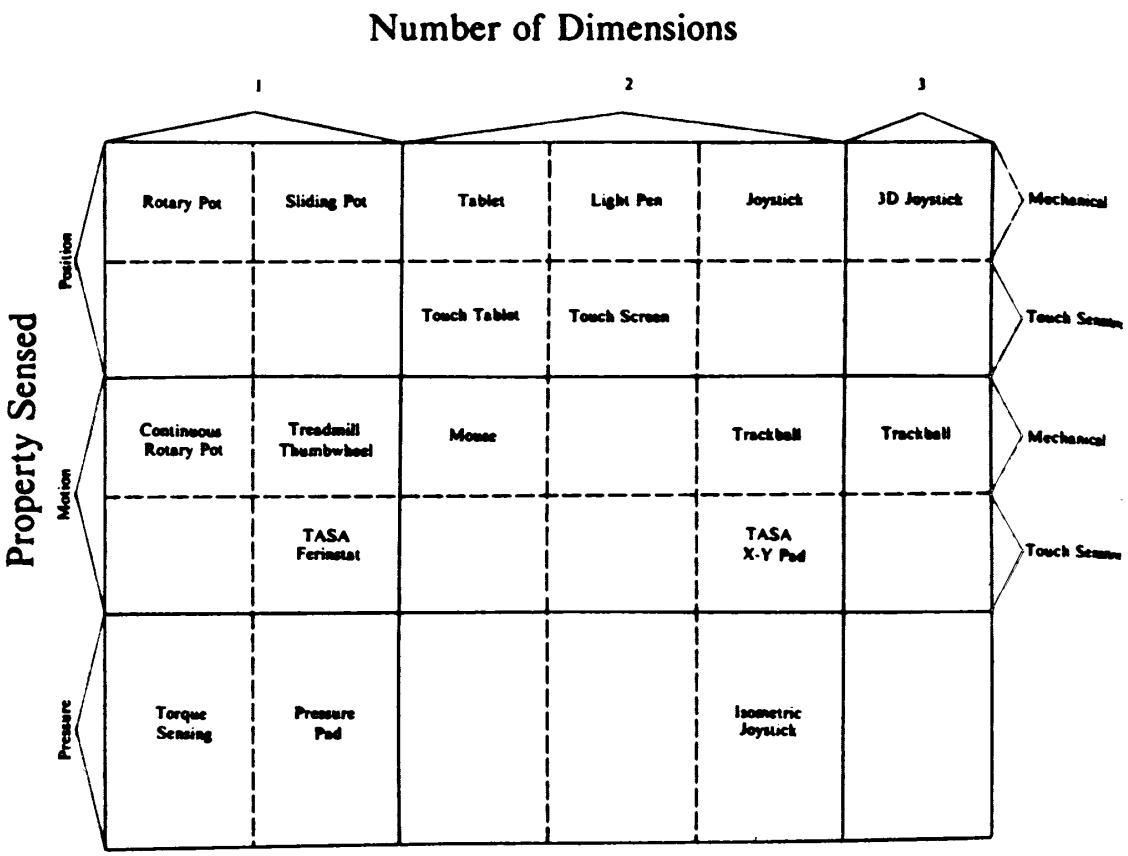


Figure 2.1: A selection of devices plotted on Buxton's taxonomy [Buxton 83]

Simpson and St. Amant [Simpson 2003] developed a taxonomy based on the physical properties of devices with the idea in mind of being able to automatically find more efficient mappings of device controls to tasks.

Foley et al. [Foley 1984] provide more of a design space rather than taxonomy for input devices within the context of computer graphics interaction techniques. A variety of devices commonly used in two-dimensional GUI's are described in terms of the tasks that are commonly performed in graphics software, so devices are analyzed in terms of their ability or suitability of performance for a task, rather than their physical performance

characteristics. For example, a mouse might be classified as excellent for selection but poor for text entry.

Due to the informal classification scheme in the Foley paper, the possible design space of input devices is quite constrained and somewhat arbitrary. While they have performed a reasonable coverage of devices that were deemed appropriate for use within a GUI, the scope of the design space is limited to devices that were in existence at the time of publication. The Foley taxonomy is not designed to be extensible. There is also no consideration of three-dimensional and hybrid devices that have been developed subsequent to the publication of the paper.

Card, MacKinlay and Robertson[Card 1991, 1992] provide an interesting scheme for the classification of devices based on the physical characteristics of the device as well as the method of operation and interaction with said device. The Card et al. scheme classifies devices along the following dimensions (Figure 2.2):

- Axes that can be manipulated: A rotary dial or slider can manipulate along a single axis, a mouse or joystick can manipulate along two axes, usually referred to as x and y, while a space ball or data glove can manipulate along all three axes simultaneously. When manipulation along an axis is mentioned, it refers to the device's native abilities, not what is possible with overloading functionality. There exist seven different possible combinations for either one, two, or three axes. Of course, many of these are functionally equivalent when we simply consider transformation of coordinate spaces.

- Linear vs. Rotary: A device either allows a user to make movements that can be tracked in a positional manner or makes use of the angle of rotational information of the device. A mouse is a linear device since it keeps track of the change in x and y coordinates of movement, while a dial is a rotary device since it is controlled by how much or how far it has been rotated. It is also possible to have hybrid devices such as the Magellan mouse or a data glove where all six degrees of freedom are being utilized.

- Position or Angle Utilization: If a device tracks movement or rotation it will do so in either an absolute or relative fashion. A mouse tracks relative movement. Every time that the mouse is polled, it reports the distance that it has moved since the last time it was polled. On the other hand, a graphics tablet is frequently set up in absolute mode. Each time the tablet is polled, it reports the actual location, if any, of the stylus. There exists a correspondence between the screen and the tablet. A mouse has no such correspondence. If a user picks up the mouse and places it down on a different part of the desktop then no movement has taken place.

- Positional or Relative Force and Torque: For input devices that utilize force such as a pressure sensitive stylus or space ball, the device can either measure the absolute or relative force and torque that is applied.

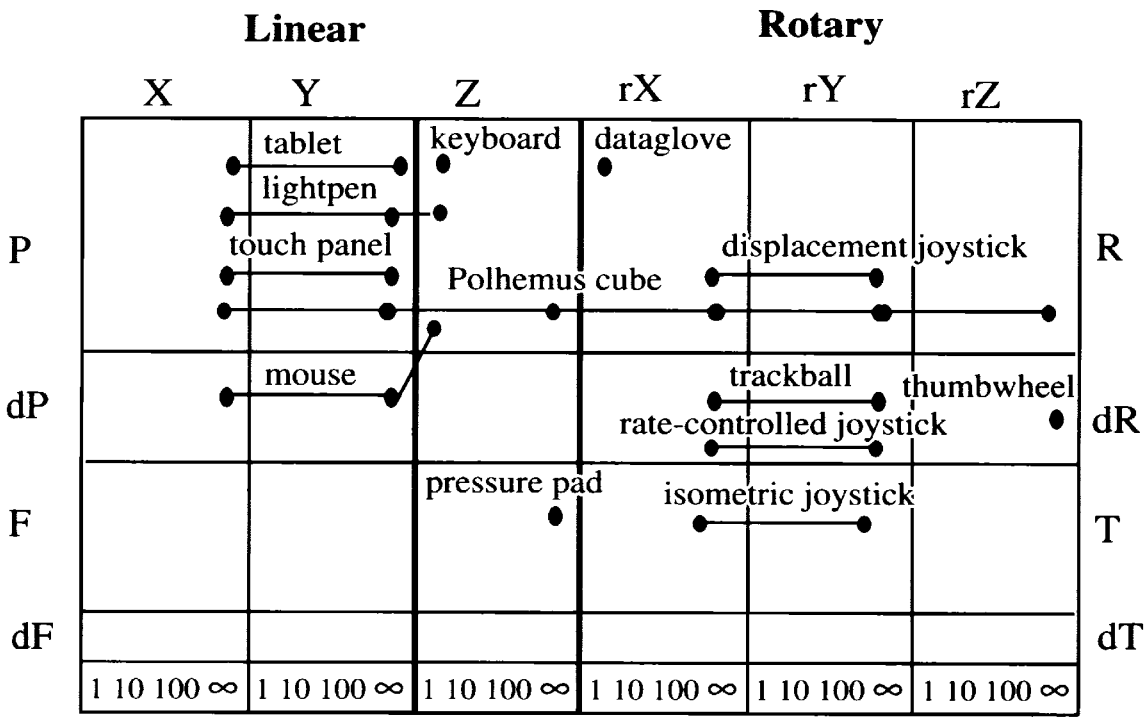


Figure 2.2: A selection of devices plotted on Card et al's. taxonomy [Card 91]

The authors clearly intended that the above taxonomy be used to describe the design space for all possible input devices regardless of practicality. As such it is capable of describing all input devices regardless of whether they are discrete or continuous unlike the taxonomy proposed by Buxton [Buxton 1983] and Foley [Foley 1984]. Both Buxton's and Foley's taxonomies or design spaces exist as a subset of the Card taxonomy.

2.4 Models of Device Usage

The modeling of device usage is concerned with arriving upon a "formal description of activity which can be used for predicting some future activity." [Baber 1997] Several techniques have been used. The primary focus is usually transaction time (time to complete a specific task or subtask). Models can be used to test a particular interaction task without the need for building the actual system and testing it with human subjects. For example, Fitts' Law can be used to produce quantitative predictions that buttons three pixels wide are not a good idea in an interface because the user would have a great deal of difficulty clicking on them with a conventional mouse and hence the task completion time would be unreasonably long. All this can be accurately predicted without writing a single line of code or running a single subject in an experiment.

Fitts' Law [Fitts 1954] is an early model for rapid aimed pointing tasks with a low error rate. Paul Fitts did his original work with a stylus tapping task where participants were required to alternately tap two targets of fixed width, separated by a fixed distance, as quickly as possible. His work has subsequently been extended to a variety of tasks including mouse movement and dragging.

The three state device approach [Buxton 1990] takes an automata or state transition diagram approach for the generic actions allowed by user interface widgets and interaction devices. Examples are given to show the state diagrams for simple tasks such as selecting (figure 2.3) and dragging. The three state device approach does not allow for the comparison of different devices since their state diagrams may be the same, yet in practice

they will have different characteristics with regards to usability. It is also limited in terms of tasks and devices that it can describe, a pressure stylus being one example of a device that cannot be modeled. It is possible that one could focus on the transitions and look at how they were actually performed by the different devices. Note that the arcs in Buxton's model might be annotated by performance characteristics. Although not explored in his work, the state diagrams can be expanded on by adding performance measures.

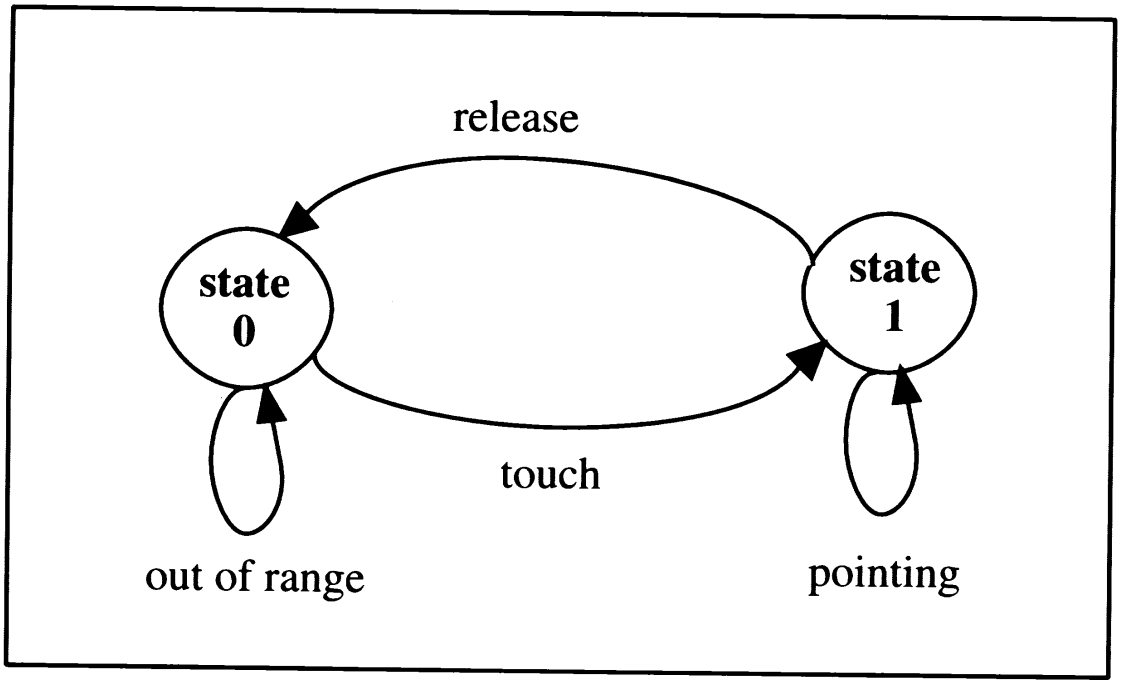


Figure 2.3: Selection using a direct pointing device [Buxton 1990]

The keystroke level model (KLM) [Card 1980, 83] takes a summation of a task decomposition approach to the prediction of task completion times. Given a particular task *A*, *A* would be broken down into a series of elemental subtasks. Times can be computed for each subtask and then the summation of all subtask completion times will give you the time to complete task *A*. Certain constant times are used to determine the subtask times, such as 1.35 seconds per mental operation, 0.2 seconds per keystroke (based on an average typing speed of 55 words per minute), or 1.1 seconds to move the cursor to a particular area. The KLM has advantages as well as disadvantages in predicting user performance. It is fairly easy to apply if a task decomposition can be created. It gives a fairly reasonable approximation of actual performance [Baber 1997, Card 1983] but does not account for the wide differences in individual performance. There is a lot of disagreement as to what the appropriate constant values should be for different devices. Different constants have been used [Card 1978, Epps 1986, MacKenzie 1991] for different devices to provide a greater or lesser degree of fidelity to the actual user performance. This suggests that perhaps the keystroke model does not always capture the essence of the task that is being measured. For example, one assumption is that tasks are practiced and efficient; there is no accommodation for users stopping to think about what to do next. In addition, there is also the question of equivalence of devices. Two mice, for example, might not have the same performance characteristics and would therefore require different weights [Buxton 1986].

Markov chain models are based on the idea that it is possible to describe the interaction between a user and a device with a finite state machine. If each one of these actions takes a user to a particular state then times can be assigned to the transitions between states.

Probabilities can be assigned to these transitions. An expected duration can then be calculated by looking for the most likely path through the state diagram given that the user is trying to accomplish a particular task. This method is most frequently used in voice recognition systems [Rudnicky 1991].

Task network models are similar to Markov models in that they have states with projected task completion time except that they are constrained by time; they expect action to occur within a specified period of time [Baber 1997]. The time constraint is used to allow for the inclusion of an error factor for unsuccessful completion within the projected time.

In some ways both of these approaches are similar to the KLM approach, with the inclusion of probabilities between transitions. In addition, differences in performance can be modeled with the possibility that different paths are taken.

GOMS (Goals, Operators, Methods, Selection) and the Model Human Processor (MHP) [Card 1983] incorporate an approach to modeling that consists of three basic components: The MHP which attempts to describe the cognitive aspects of the process that humans use to store and retrieve information as well as process and act upon the information, a set of tasks with a description of the knowledge required by the human to perform them, and a set of performance times to allow performance prediction much like the KLM. The MHP is a systems level way of describing the human mind. It consists of three subsystems: the perceptual system which brings sensory information into the MHP, the motor system which controls the output of the body, and the cognitive system which is responsible for mediating

between the perceptual system and the motor system as well as solving more complex problems requiring stored information and goal formation.

GOMS suffers from a number of different drawbacks. One problem is that it is a serial based model and so does not allow for any parallelism in the MHP. It is not clear to what extent a human may be forming the intention for a goal while still processing the results of a previous action. What is clear is that some form of parallelism does occur [Olsen 1990]. GOMS also does not factor in or account for error. Even expert performance is not error free. Finally GOMS is also subject to a high degree of complexity particularly when alternative means of task completion are taken into account.

The Cognitive Complexity Theory (CCT) [Kieras 1986] extends GOMS by adding production rules to describe the user knowledge and a state transition diagram of the system. CCT suffers from many of the same problems as GOMS and has not been empirically evaluated [Knowles 1988] although it is still in use for its qualitative properties.

Cognitive models such as EPIC [Hornof 1999] are frequently employed for the purpose of simulating human interaction in a system. EPIC requires as inputs to its system a cognitive strategy to complete a task, the perceptual features of the interface and the details of the task environment. While models like EPIC are useful tools to understand the nature of an interaction with a system, they require too much a priori knowledge of what the user wishes to do. The direction of our research is not concerned with predicting a user's intent but rather determining the user's ability to accomplish specific goals.

Different models may be more appropriate for different research goals. Fitts' Law may be very useful for predicting performance time for elemental pointing tasks and gives designers valuable guidance as to positioning and size criteria of controls in interfaces. The KLM or GOMS are more useful for representing more complex interactions with interfaces. Markov models are particularly useful for speech recognition or for probabilistically calculating likely user intent, while task network models are frequently used to measure error as opposed to task completion time. Cognitive models are useful for simulating human interaction with a system. Some of these models are composites of other simpler modeling techniques. For example, KLM or GMS incorporate aspects of Fitts' Law in their models.

2.5 Task Taxonomies

Very little work has been done in the area of developing a task taxonomy that can be used to specify the range of user actions in a GUI. By range of user actions we mean a complete description or taxonomy of possible interaction methods and tasks that can be accomplished in a GUI without consideration of context. To our knowledge, there is no prior work that adequately describes the full range of tasks or actions possible in a GUI and provides a framework for evaluating their similarities and differences although Simpson and St. Amant [Simpson 2003] are using a set of low level tasks called positioning, orientation and confirmation as a set of basic primitives to build larger tasks. There are however quite a number of articles that mention specific tasks within the context of performance evaluation. Unfortunately the definition of a task is not clearly agreed upon, so some work looks at tasks from a primitive viewpoint while others have fairly high level monolithic tasks that

seem specific to a particular interface. For example, Szalavari [Szalavari 1998] mentions Navigation, System control and Object manipulation as possible user tasks. Poupyrev [Poupyrev 1996] investigated methods for Manipulation and Selection of objects. Some work takes a more elemental view of tasks and considers cursor movement [Shneiderman 1983], pointing, and typing [Graham 1996, Douglas 1994] to be primary tasks. We are looking for a middle level task taxonomy that will permit us to express a reasonably complete set of user interactions in a GUI without having to resort to a complete task decomposition scheme that would require an unwieldy amount of task analysis before someone could use our system.

Foley [1984] has what appears to be a comprehensive set of tasks as they relate to interaction in a graphics package:

- Select: Select from a finite set of alternatives (see Figure 2.4)
- Position: Indicate a position on an interactive display
- Orient: Manipulate the angle of orientation of an object in two space or three space
- Path: Generate a series of points or orientations over time (Considered distinct because of time dimension)
- Quantify: Specify a quantity either through text input or manipulation of a control
- Text: Input a string from the keyboard or other text entry device

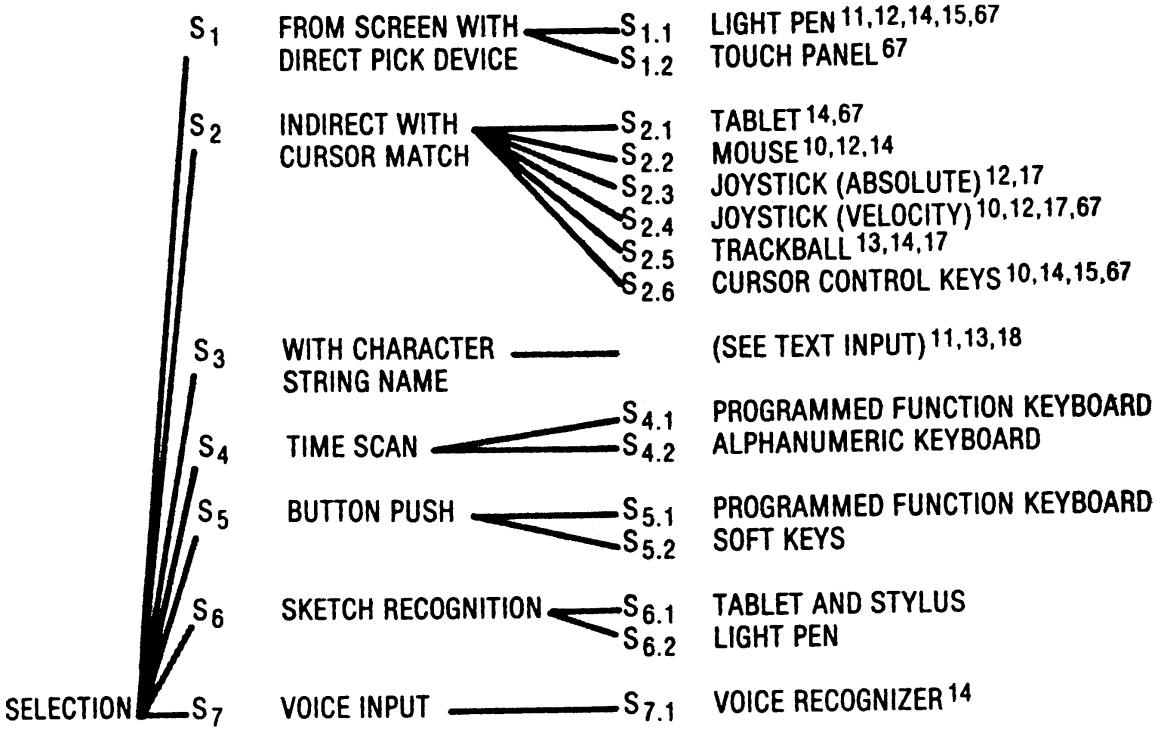


Figure 2.4: The selection task from Foley et al. [Foley 1984]

The related work described has done a good job of advancing the field to the point where it is possible to develop the system that we are proposing. A lot of work has been done in the area of evaluation and calibration that can be used both for the purpose of model construction and model validation by making sure that it is consistent with earlier research. The comparison studies that have been done are also very useful as verification tools for a

model. They all suffer however from a lack of generality that can be overcome by making the results more applicable to complex systems rather than simply focusing on a specific, narrowly defined task.

We use prior device taxonomy work to clarify likely differences in performance based on the differences in the underlying device, to aid us in developing our model. All of our work is experimentally validated. Since it is not provable that these taxonomies are indicative of performance, they merely serve as descriptors of the possible design space of input devices. The task taxonomy work that we have looked at serves as a starting point for the task descriptions that we develop for our model. Since we are not suggesting that we can describe this area in its entirety, we are not concerned with the lack of empirical evaluation in this area.

2.6 Other Related Work

Not all GUI interaction situations can be modeled accurately without greatly increasing the complexity of the model. For example, Farris et. al. [Farris 2001] describe a situation where targets can be placed at the edges of the screen to decrease target acquisition time. Placement of targets at the edge of the screen (with no pixel border between the edge and the target) is superior because it eliminates overshoot and theoretically allows users to maintain velocity rather than slowing as they approach the target [Tognazzini 1999], although in practice users still slowed down as if they were approaching a non-edge target. Techniques such as this can be incorporated into our model with a corresponding increase in accuracy. This dissertation will not address interaction techniques such as these because

we feel that it is not desirable to add special cases such as edge targets since it will yield marginal improvements in model accuracy but a combinatorial increase in the complexity of the model. After adding relatively few special cases, the effort required to build a model will eclipse the cost of testing the actual system.

Accot and Zhai [Accot 2001] investigated control gain and scale between monitor and input device in steering law tasks. They determined that the error rate went up sharply for both linear and steering law tasks as scale was increased. There was a less pronounced but significant decrease in Index of Performance indicating that increasing scale above a factor of two was detrimental to both performance time and accuracy in either type of task. Our model handles interaction issues such as these through device specific calibration. The same mouse set to two different C/D ratios is considered to be two different devices.

Various work has looked at optimization of target layout to minimize interaction time [Schmitt 1999, Tognazzini 1999]. With the exception of techniques that "overcome" Fitts' Law by altering characteristics such as gain [Worden 1997] or edge placement, these techniques are simply optimizations of Fitts' Law and are subsumed in our model. While our model is designed to give an overall measure of suitability to purpose for the various interactions possible in a GUI given a particular device, the optimization literature tends to take a more frequency of task approach, optimizing the time and error rate for more frequently performed tasks while allowing sparsely used functions to be more difficult. These types of studies are more germane to particular interfaces while we strive for a greater level of generality and application independence in our model.

Douglas et. al. [Douglas 1999] investigated the predictive power of the ISO 9241, Part 9 Draft International Standard for testing computer pointing devices for performance and comfort [ISO Draft 1999, Final 2000]. While the ISO standard addresses a variety of ergonomic issues related to muscle load and fatigue, which are beyond the scope of our model, it does little to address the suitability to task of a particular device. Its greatest contribution perhaps is to specify a particular framework for the experimental evaluation of devices to conform to a range of biomechanical parameters and to develop a single instrument for the evaluation of subjective user comfort. This will obviously have the benefit of permitting more cross-study comparisons which is also a goal of our work. Douglas et. al. have determined that there are a number of areas in which the ISO standard still falls short of its goal, particularly in experimental evaluation.

MacKenzie et. Al. [MacKenzie 2001] take a novel approach to evaluating accuracy of pointing devices. Rather than looking at a single error measure they propose seven new accuracy measures that are designed to elicit more subtle differences between devices. All of these quantitative measures look at some aspect of the deviation between a “perfect” target selection task and actual performance. The measures are designed to allow the researcher to augment an error rate, which simply gives you the magnitude of the problem with the ability to measure why the problem is occurring. For example, is a high error rate the result of a lack of smoothness in motion or a propensity for a device to allow cursor acceleration thus overshooting the target? While this work is fascinating and will likely lead to a better understanding of design tradeoffs in input devices, it is beyond the scope of our model's purpose.

3. Interaction Modeling

3.1 Introduction

In this section we describe an interaction model that provides our methodology for analyzing user actions. The model has several components. It relies on a taxonomy of elementary interaction tasks, and supports quantitative predictions of execution performance. The model is based on the concept of execution graphs, in which user actions are interpreted as transitions along weighted edges in a graph. A complete traversal of an execution graph produces a prediction of performance in terms of duration or error rate. The parameterization (i.e., the specific edge weights) of an execution graph depends on the properties of a specific input device.

3.2 Task Taxonomy

We have identified a set of core, fundamental tasks that users will perform in a GUI. This set is an extension of existing task taxonomies taken from the literature. The taxonomy is not restrictive; it allows extension to new tasks as needed by the evaluator, by following a methodology outlined below. For generality, the tasks in the taxonomy are of a fairly primitive nature; they can be chained together to allow the modeling of more complex tasks in an interface.

We have looked for a balance between the range of expressiveness permitted and the degree of complexity that our taxonomy imposes upon the specification of a task. At one extreme, we might have defined our tasks such that they were the basic components of a model such

as the KLM. This would force an unnecessary degree of complexity upon the user of our modeling system; and would therefore add only marginal value to what already exists with the KLM. At the other extreme, we might have described and provided measurements for high-level application specific tasks. Doing so would eliminate what we see as one of the key advantages of our system, the promise of generality across applications. By tying our tasks to specific applications, it would be necessary to define an arbitrarily large number of tasks and then perform an exhaustive series of tests to validate each task. This sort of brute force approach will not work in the changing world of interface development, nor would it provide any real benefit over prototyping and usability testing.

The direction we have therefore chosen is to provide the system evaluator with a reasonable set of tasks that will successfully model higher-level tasks. We provide a set of three tasks that can be chained together to model more complex tasks. We also provide a means for the extension of our model by way of adding additional tasks to our task set to meet the changing nature of interface design and input devices.

For the purposes of this study we defined the actions selection, position and orient as follows:

- Selection: A cursor movement starting with a minimum of a 3 pixel movement [Worden 1997], ending with an activation while the cursor was on a target.

- Position: Movement of a target from position A to position B. Position is commonly associated with dragging in a GUI.

- Orient: Dragging the needle of a dial control to change the orientation of an object and deactivating the pointing device when the desired position is reached. The performance characteristics of orient are heavily dependent on the type of control used and would need to be empirically evaluated for each different interaction style.

Since the verification and activation time are constants [Meyer 1988], a selection could be defined in an execution graph as ending with the cursor stopped over a target. Our operational definitions of tasks above were based on the ease of separating the tasks in data analysis as well as commonly accepted interactions in current GUI's. We could just as well have used the alternate definition of selection by parsing the data collected in a different fashion.

3.3 Execution Paths

Consider a common task in a GUI: menu selection. The user moves the pointer to a header in the menu bar, clicks, waits for a menu to appear, then moves to a menu item and clicks on it. This sequence can be modeled in the abstract as follows (figure 3.1):

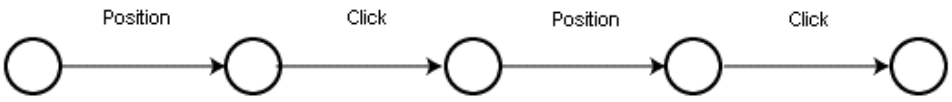


Figure 3.1: Menu selection abstract model

This can be treated as a task execution path. If all GUI states were represented (possible in theory, not in practice) we would see user interactions as paths through a very large graph.

Because clicks of a mouse (or other pointing device) are similar enough in performance, these actions are usually abstracted away, with a constant factor substituting for it. We can further reduce the model by collapsing similar nodes, as with Buxton's model.

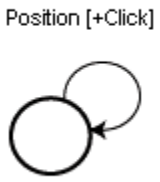


Figure 3.2: Simplified menu selection, abstract model

Adding the other types of actions in the taxonomy, we have the generalized model shown in figure 3.3. This model can obviously be tailored to other menu activation/selection schemes.

We can generalize this example as follows. Once a task list and a device list have been created, a set of execution steps are derived for each task on a device by device basis, assuming that the device is capable of performing the task. A series of execution steps represents an execution path for a particular device and task pairing. The execution paths represent the most efficient possible series of interaction steps by the user, given a particular interaction style, since we are trying to model optimum performance rather than all possible interactions.

An execution path consists of a series of elemental interactions or execution steps that must be performed in order to perform the desired action, using a particular device. These elemental steps share much in common with the KLM and are based upon the smallest measurable subtasks that are reasonable to describe in an interaction with a GUI. An example of some elemental steps might be key presses, clicks, and moving the cursor to a particular location. We decompose our tasks from our task taxonomy into a series of execution steps for each device that we include in our device list. Each of these task/device pairs will yield at least one execution path if it is possible to perform the desired task with that particular device. It is entirely possible and quite probable that a particular task/device pairing will yield more than one execution path if the device can be used in a different fashion to complete the same task. Because there are an infinite number of these possible execution paths available (if we consider repeated actions), we do not consider any execution path that contains additional or non-optimal elemental tasks within the execution path to remove any pathological examples of inefficient or non-optimal interactions. We also can't predict what new methods of task completion might be devised in the future and hence can't provide an execution path for that particular interaction style prior to its inception. One example of this is the work in the area of the activation of objects [Dulberg 1999] with the flick gesture by using a mouse to throw or flick the cursor at a target. In a conventional GUI interface it would not have been reasonable to consider an execution path where target activation could be accomplished by a mouse down, mouse move and mouse up performed in sequence with temporal and movement constraints since most operating systems are not designed to support this sequence of steps for this task, preferring instead to

use them to indicate that either a drag, or a selection cancellation has taken place. We therefore provide a mechanism for the creation and inclusion of new execution steps and paths for the users of our system as the need arises.

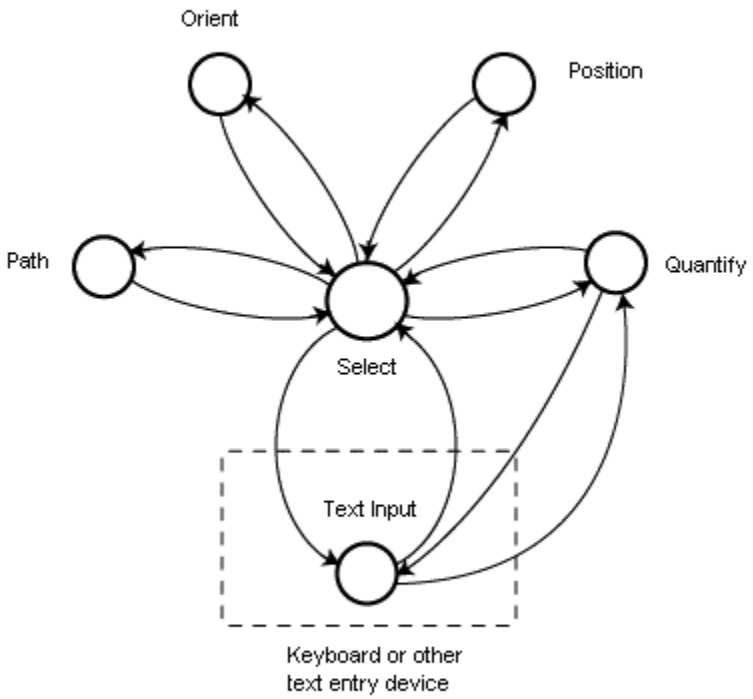


Figure 3.3: Generalized execution graph

3.4 Execution Graphs

An execution graph consists of a set of circular nodes which represent tasks, and directed arcs, representing transitions. An example of a generalized task execution graph appears in

figure 3.3. Loosely based on Card's taxonomy of tasks, this graph provides a model for the execution of most tasks in a GUI. Each node would have an associated performance time based on distance as well as an error rate in percent. The transitions would have an empirically derived constant time and error rate associated with them. An execution graph can be made as large as necessary to incorporate appropriate task taxonomy. Based on the actual definition of said taxonomy and permissible orderings, the appropriate transitions would need to be included as well.

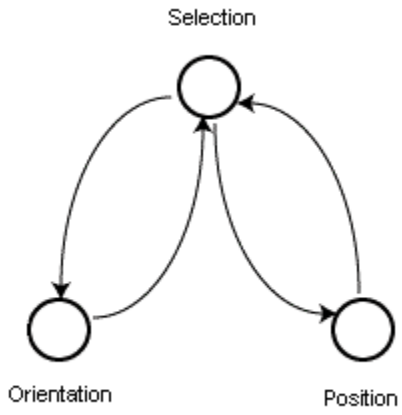


Figure 3.4: Execution graph for simulation

An example of a three-task execution graph, which is modeled upon our test bed, appears in figure 3.4. Notice that there are no transitions between the position and orient states since in our experimental test bed it was necessary to select something before you positioned or oriented it.

Execution paths for each task are represented as a directed graph. The actual elemental actions or steps required to be performed by the user are represented as nodes in the graph. The transitions between nodes are annotated by time and error weights. Our experience suggests that while individual nodes may be used by more than one device in the course of completing the task, each path through the directed graph is unique to the particular device that generated it, unless the two devices are equivalent. In this case the two devices can be treated as one. There may of course be multiple execution paths for a single device if the device can be used in different manners to complete the task, as discussed in the previous section.

Execution graphs are constructed so as to insure that all paths from the start state to the accepting state are valid execution paths; however they presume that a user will take a path that is not duplicitous. For example, if a user were to repeatedly select and release an object by clicking on it, before positioning the object, this would result in an execution path of several selections followed by a position. Our model would not accept this because the additional selections do not have anything to do with a goal directed behavior that we might be studying. A user could however select an item and position it, release it then select and position it again to reach a final goal. Our modeling technique would simply see this as two distinct selections and positions since we are not concerned with the context in which they are performed.

3.5 Performance analysis and prediction

Each transition in a task graph has two weighting factors associated with it. One value represents an execution time for the particular action represented by the transition, the other an error probability. These factors must be calculated for each transition of the task graph. We draw heavily upon prior, published work based upon well-accepted HCI principles such as Fitts' law. Fitts' law describes the performance characteristics for the execution of a rapid aimed movement such as a mouse move.

Once the transition weights have been calculated, it is then possible to examine all possible paths through the directed graph in order to evaluate the relative performance characteristics of the different devices and/or interaction styles that originally generated the graph.

This work is based upon the underlying assumption that we are dealing with an optimal device. By optimal, we are referring to a device that is working properly and consistently. A mouse that was dirty or joystick with erratic contacts would most likely exhibit linear or worse performance. While we have conducted a preliminary investigation into the area, it is beyond the scope of this work to determine the specific difference between the performance characteristics of two different devices of the same type such as two mice or keyboards. We look at this area only to the extent that is necessary to determine its effect on our model and to be able to provide an appropriate adaptation mechanism for those following our work. Our model handles device differences within a class by presuming that they are of different classes. This is to say that we would assume that two different mice with different

indices of difficulty would be treated as different devices and no generalizations would be made from one to the other. It is entirely possible that a quick calibration between the two devices might yield a less cumbersome approach but this has not been determined.

4. Simulation Design and Experiment Procedures

4.1 Method

4.1.1 Subjects

A total of 24 students from the College of Engineering at North Carolina State University served as subjects. Participants were selected on a first-come, first-serve basis in response to a call for participation. Subjects were paid US $40.00 for their participation. Each subject was randomly assigned to one of six orderings of the 3 input devices. There were 17 males and 7 females, 6 were left-handed and 18 were right-handed. The subjects' median age was 28 with a range of 20 to 49.

4.1.2 Design

The experimental software was developed in Visual Basic. A trial consisted of manipulating control points, positioning and orientation of a bezier curve to match a randomly generated exemplar that appeared on their screen. This application was motivated by Barham and McAllister's [Barham xx] work in 3D cursor positioning with bezier curves. The subject had a maximum of 30 seconds to complete each trial. The experimental software logged the position of all objects on the screen as well as the cursor at 55 ms. intervals. In addition, any user action or change in state of the environment was logged when it occurred as well.

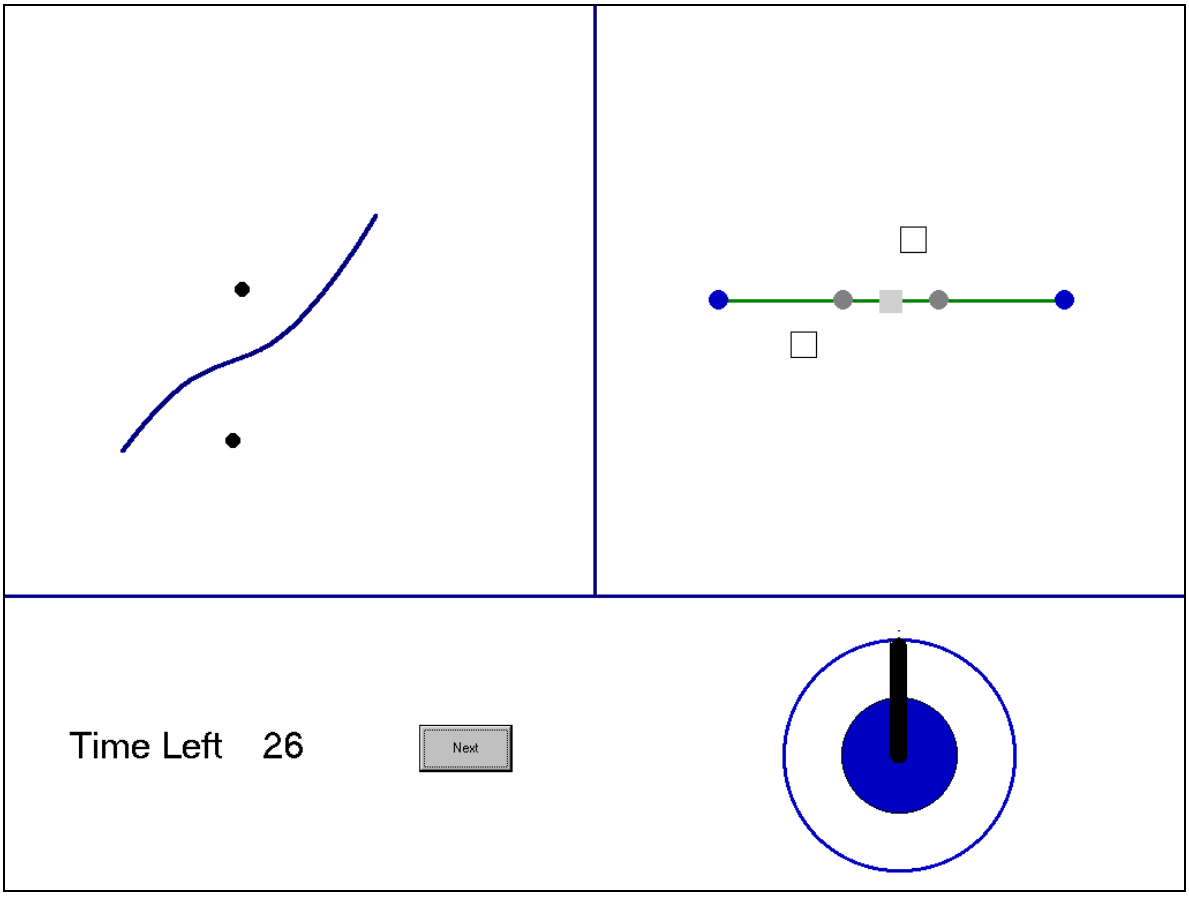


Figure 4.1: Participant view of a random trial

Three basic actions are required to match the curves:

1. Move the two interior control points to their respective boxes
2. Position the entire curve
3. Orient or rotate the curve to the correct position.

We selected these actions for the simulation because they allow us to look at selection, position and orientation tasks in a variety of different combinations within an ecologically valid task. These three primitive tasks can be combined to create a wide variety of interactions in GUI's.

Figure 4.1 shows a random trial displayed by the system. On the left-hand side of the screen is the reference curve. Participants manipulate the controls on the right hand side of the screen to match the user curve in shape, position and orientation as closely as possible to the reference curve. There are four controls that the user must manipulate to accomplish the task: drag the interior control points (represented as medium gray circles in figure 4.1) to their respective box, drag the square to an appropriate place on the screen to reposition the curve (represented as a light gray square) and rotate the dial to orient the curve.

Figure 4.2 shows a trial in progress. The interior control points have been dragged to their respective positions and the curve has been oriented. The curve has not been positioned yet.

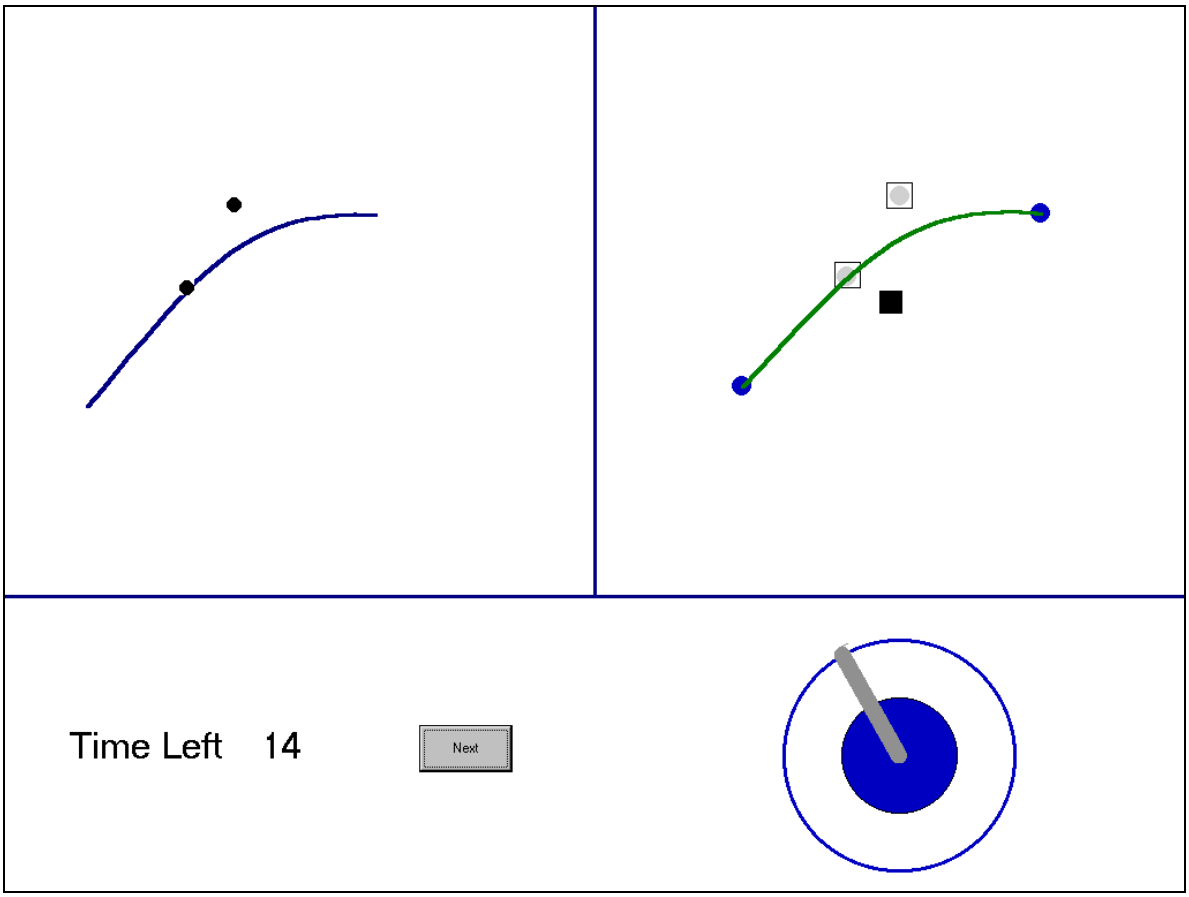


Figure 4.2: User view of a random trial in progress

The order in which the actions were performed was randomly determined by the experimental software. The participant was forced to perform the actions in a particular sequence based on the color-coding provided by the interface starting out with the lightest gray control, then the medium gray control and finally finishing with the black control. Once a subject completed an action and moved to the next one, they were not permitted to perform the previous action again in that trial. If the subject finished the trial before the allotted time, they were able to click a "Next" button to proceed to the next trial. The two interior reference control points were randomly perturbed from a position one third of the

distance along the straight line formed by the outer control points within a range of 150 pixels in any direction, while the outermost control points were fixed as the endpoints. The angle of the reference curve was randomly generated between the range of 0 and 2π radians. The entire curve was rotated by the random angle about the midpoint between the endpoints. The entire curve was then translated by a random distance in the range of 0 to 75 pixels along a randomly chosen vector. The user curve was always placed in a horizontal orientation, centered on the right side of the screen with the two manipulatable control points uniformly spaced in a straight line. It should be noted that while guidance was provided to the participant with respect to where the interior control points could be placed, it is possible to generate curves that are exactly the same, or closely resemble the reference curve by placing the points in other areas. If, for example, the user rotates the curve 180 degrees and places the points 180 degrees from the suggested positions at the same distance from the center, an identical curve will be generated. When looking at the participant's qualitative performance we do take this into account by checking the quality of both orientations and selecting the closest fit.

The experiment was run on a Dell Inspiron 3800 laptop, which has an integrated track point and touch pad. A Logitech two button mouse was attached via a serial port. Subjects were permitted to position the apparatus to their liking.

4.1.3 Procedure

Participants performed 4 blocks of 25 trials with a rest period of 2 minutes between each block during each one-hour session. Participants engaged in three sessions each scheduled

one week apart. Each week a different input device was used. The ordering of the input devices was randomly assigned to each participant so that each of the six possible orderings of the three input devices was uniformly accounted for. Subjects were presented with a set of written directions beforehand. They were also given a brief demonstration and permitted to perform a few trials before they began the experiment.

4.2 Data Collection

For each block of trials that a participant completed two comma-delimited files were created, a *Trials* file and a *Test* file. The *Trials* file (table 4.1) contained a single record for each trial performed with the initial state of the trial and relevant parameter settings. The *Test* file was used to log participant actions and poll the interface to keep a running history of all activity. Records were written to the *Test* file every 55 milliseconds and whenever the participant clicked, pressed or released an actuating switch on the input device.

Trial Number	Number between 1 and 25 indicating trial number
Reference Points 1-4 (x,y)	X and Y coordinates of points 1-4 on reference curve (randomly generated for each trial)
User Points 1-4 (x,y)	X and Y coordinates of the initial position of the user curve control points (these values are the same every trial)
Dial Points 1 and 2 (x,y)	The initial position of the orientation dial (same for each trial)
Reference Curve Center (x,y)	X and Y coordinates of the central point of the bezier curve (randomly generated each trial)
Task Ordering (a,b,c)	The randomly generated sequence of activities that governed trial where 1=position, 2=orientation, 3= pointmove

Table 4.1 Format of *trials.txt* file

Time	Current timestamp in ms.
Trial	Current trial number
Message	One or two character message indicating a critical event generated by button actuation
State	0-4: indicating degree of completeness of trial
Cursor Position	Current cursor coordinates
User Points Position	X and Y coordinates of the current position of the user curve control points
Dial Position	Coordinates of the current position of the extremity of the dial needle
User Curve Center	Current coordinates of the user curve center

Table 4.2: Format of *test.txt* file

After each block of trials, the files were renamed with the participant's number, block number, and device used. A total of 12 sets of files were generated for each of the 24 participants representing four sets of blocks for each of the three devices tested.

4.3 Data Analysis Procedures

For every block of trials, the *Trials* and corresponding *Test* files were run through an action-parsing program and an actions file was created. The parsing program read in the objectives for each trial (ordering of actions and goal positions) and then attempted to match it to the user actions logged in *Test* for that particular trial. As the data for each user action was processed an event file was generated. Each action was logged in the event file in chronological order to permit the analysis of various sequences. The data collected for each event included the event type, start time, end time, and magnitude (distance in pixels or radians). If an error occurred, an error message was logged instead.

Errors were defined as any action undertaken by the user that did not lead towards a successful execution of the required task. At this stage of the analysis, we did not treat qualitative differences as errors. For example, if the user was moving away from a target rather than towards, this was not considered to be an error. The experimental software logged error messages in the message field of *Test* if the participant's action did not match one of the appropriate actions given the ordering of tasks. For example, if the software was currently in point move mode then permissible actions would include clicking and dragging either user point, clicking on the "Next" button to advance to the next trial, or clicking on the correct control to advance to the next mode if one was available.

Once a block of trials has been action-parsed, the data analysis diverged along two different paths: evaluation and verification of action performance, and computation and analysis of transition times.

The action performance times file was imported into Microsoft Excel. Once in Excel, it was sorted by action and duration time. The quantity of errors were recorded and any anomalous data points, such as a user hovering for three seconds (we presume they were thinking about what they wanted to do next) while they were in the middle of an action, were removed. The removal accounted for less than 2% of the data which is well below the bounds of accepted practice in Fitts' Law research [MacKenzie 1991a]. The cleaned action performance data were then analyzed as described below.

The procedures above produced some 360 megabytes of descriptions of user performance. To provide context for later analysis, it will be useful to give a broad overview of patterns in the data.

Each individual subject engaged in some 100 trials using the simulation. Recall from our discussion above that one key to our research is the ability to predict performance based on the properties of a task.

5. Data Exploration and Analysis

In the introduction to this thesis we identified three questions not otherwise addressed in the HCI literature. First, is it appropriate for a detailed model of performance to neglect the time interval between what are usually considered elementary tasks? That is, models generally do ignore this time, implicitly treating it as an approximately constant noise factor. If we find patterns in inter-task transition times however, specifically patterns that depend on task types, we might build more accurate models at a greater level of detail. Second, if there exist patterns in inter-task transition times, do they depend on the type of device being used? Again, if we find such relationships, we will be able to build better models. These two questions can be treated as testable hypotheses. The third question is whether individual differences in performance should be considered in model building. This is not the type of issue that can be addressed by hypothesis testing, but exploratory data analysis leads to some suggestive findings.

5.1 Modeling inter-task transition times

The first question we address is the most basic: is there a significant difference in the transition times between different types of tasks? To our knowledge this question has not been addressed by an empirical study, though an experienced researcher in HCI would have a strong suspicion that such a difference exists. Our analysis confirms this suspicion.

For our analysis, we extracted all pairs of sequential tasks from the data logs. For each pair we measured the time interval between the conclusion of the first task and the initiation of

the second task. All action-error and error-action transitions are ignored, because they are not valid transitions in our model. This gives us data for the four different types of transitions possible:

- DM/S is a transition between an orientation task and a selection task. For example, in our simulation, the user completes the task of rotating the curve and begins the task of moving the pointer to press a button.

- P/S is a transition between a positioning task and a selection task. For example, the user might complete the task of positioning an interior control point and begin the task of selecting the other interior control point.

- S/DM is a transition between a selection task and an orientation task. This transition can only occur in the interval between the user selecting the dial and actually moving the dial to orient the curve.

- S/P is a transition between a selection task and a position task. . For example, this transition would occur between the time that a user selected an interior control point and actually started to move it.

During the experiment, each user completes several dozen of each type of the transitions above. For each user, we recorded the type of transition in the categorical variable Transition, and collected the median time for completing the transition. To be consistent

with standard practices for human performance research, medians were used rather than means in order to prevent outliers from distorting our results. These median time intervals are stored in the continuous variable Time. In all, there are 96 observations (24 users x 4 transition types). An analysis of variance shows a significant effect of Transition on Time, as given below (table 5.1). The F value of 95.8806 is highly significant. The means between the different groups vary between around 30ms to 175ms, which has some practical significance in modeling terms, justifying our interest in this issue.

Summary of Fit

Rsquare	0.757666
Adj Rsquare	0.749764
Root Mean Square Error	40.84062
Mean of Response	101.9444
Observations (or Sum Wgts)	96

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
Transition	3	479773.94	159925	95.8806	<.0001
Error	92	153451.99	1668		
C. Total	95	633225.93			

Table 5.1: Fit and analysis of variance over all subjects

Figure 5.1 shows the source data for the analysis of variance. A clear pattern is obvious here: we see a differentiation between those task sequences that end with an S task and those that do not. Further, the S and non-S groups of tasks are relatively homogeneous internally.

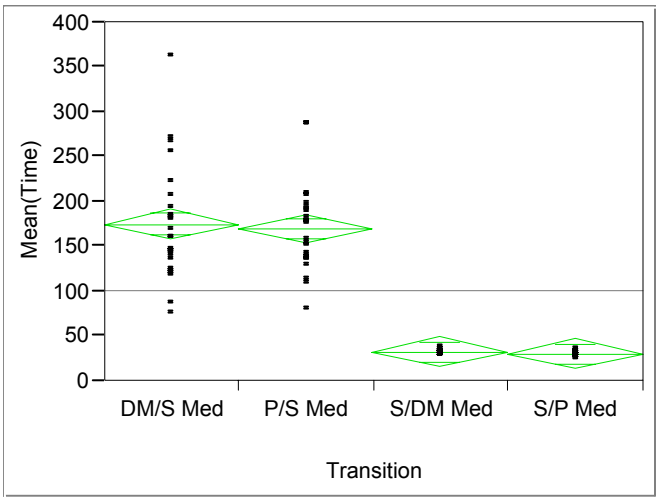


Figure 5.1: One way Analysis of Time (ms) By Transition type

It is straightforward to explain these patterns, in cognitive terms. The S/DM and S/P transactions simply reflect the mechanical time required to activate the pointing device (i.e. click) once on the target. The DM/S and P/S transitions do however involve a more complex set of actions. Once the user has deactivated the pointing device to complete a task, they will typically perceive the current state of the interface and engage in goal formation before beginning their next task. These considerations are taken into account in detailed cognitive models such as ACT-R [Anderson 1997] and Soar [Newell 1990]. However, such models are difficult to construct and evaluate, requiring considerable expertise and knowledge of cognitive psychology. Our results can be applied in simpler models, such as GOMS, as well. In a nutshell, our analysis shows that patterns in inter-task transition times are present, and that they can be characterized in a relatively simple way. To take a straightforward case, for example, a GOMS model might be extended by adding

inter-operator transitions whose durations are constant but depend on the types of operators involved.

5.2 Modeling device dependencies

We can continue the above analysis by breaking down the observations by device. There are three types of devices we considered:

- E is Trackpoint.
- P is Touchpad.
- M is Mouse.

Using the same procedure for Transition and Time, we added another categorical variable, Device, corresponding to the three device types above. An analysis of variance shows significant effects of Device and Transition on Time, as given below (table 5.2); as we might expect, there is also a significant interaction effect.

Summary of Fit

Rsquare	0.678914
RSquare Adj	0.666117
Root Mean Square Error	62.6082
Mean of Response	101.9444
Observations (or Sum Wgts)	288

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
Model	11	2287516.1	207956	53.0529	<.0001
Transition	*3*	*1439321.8*		122.3980	<.0001
Device	*2*	*424114.3*		54.0992	<.0001
*Transition * Device*	*6*	*424080.0*		18.0316	<.0001
Error	276	1081861.0	3920		
C. Total	287	3369377.1			

Table 5.2: Fit and analysis of variance over all subjects by device

5.3 Overall performance patterns

Part of the focus of this thesis is on the role that individual differences in user performance can play in understanding the applicability of an interaction model. To explore this issue, we will first need to see patterns in the aggregated data, over all the users. Summary statistics for all of the participants were collected in tabular form as shown in table 5.3. This summary data was analyzed to look at overall performance characteristics of the group as well as to spot trends and anomalies between subjects.

Column	Description
Subject	Subject number and Device
Sel Intcpt	Y axis intercept of log model for selection task
Sel Log(dist)	Coefficient of log term for log model of selection task
Sel N	Number of selection task observations
D/M Intcpt	Y axis intercept of log model for orientation task
D/M Log(dist)	Coefficient of log term for log model of orientation task
D/M N	Number of orientation task observations
Pos Intcpt	Y axis intercept of log model for position task
Pos Log(dist)	Coefficient of log term for log model of position task
Pos N	Number of position task observations
S/DM Med	Selection-Orientation transition median
S/DM N	Number of Selection-Orientation transition observations
S/DM Stddev	Selection-Orientation transition standard deviation
S/P Med	Selection-Position transition median
S/P N	Number of Selection-Position transition observations
S/P Stddev	Selection-Position transition standard deviation
DM/S Med	Orientation-Selection transition median
DM/S N	Number of Orientation-Selection transition observations
DM/S Stddev	Orientation-Selection transition standard deviation
P/S Med	Position-Selection transition median
P/S N	Number of Position-Selection transition observations
P/S Stddev	Position-Selection transition standard deviation
Device Order	Order in which subjects used devices in experiment

Table 5.3: Summary data collected by participant

The median time for completion of the selection task, over all subjects, is 1043 ms. with the mouse, 1729 ms. with the touchpad and 2019 ms. with the trackpoint. Histograms in figures 5.2-4 show the distributions for selection with the different devices. As we see, the shapes of the distributions are not qualitatively different from one another.

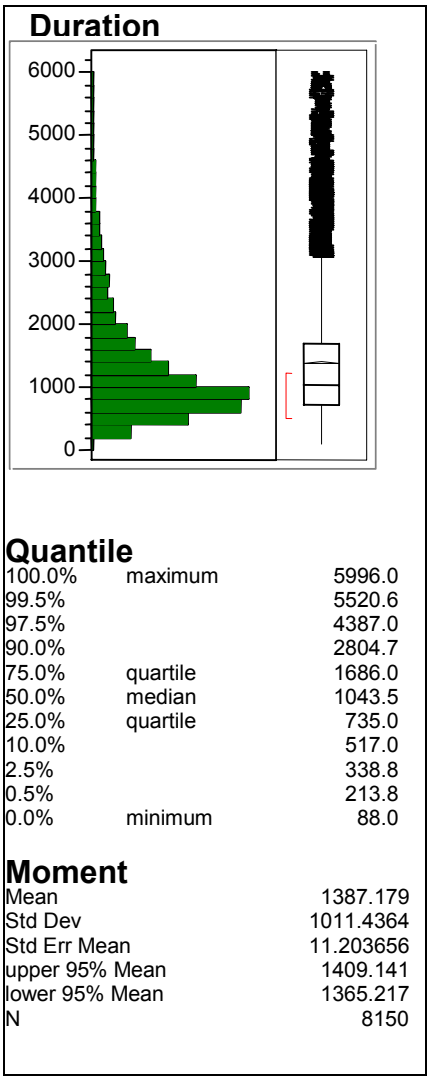


Figure 5.2: Distribution by time (ms.) of selection task with Mouse across subjects

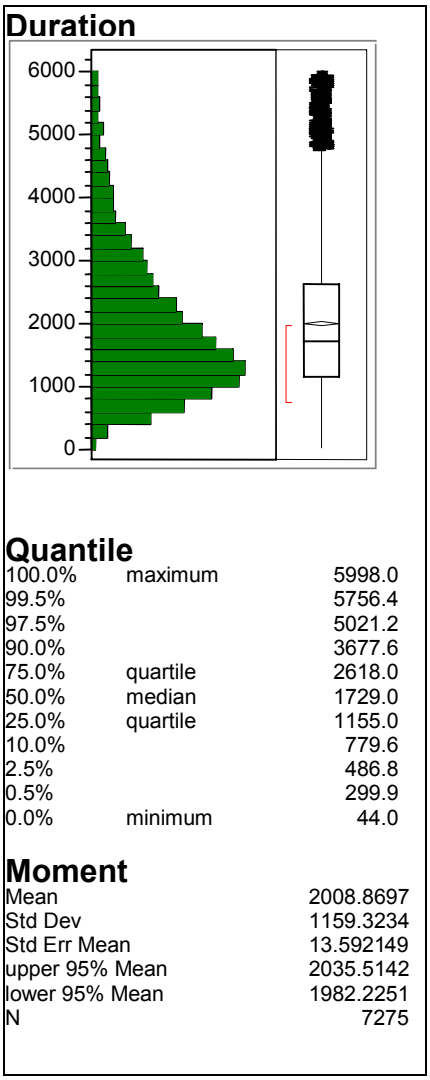


Figure 5.3: Distribution by time (ms.) of selection task with Touchpad across subjects

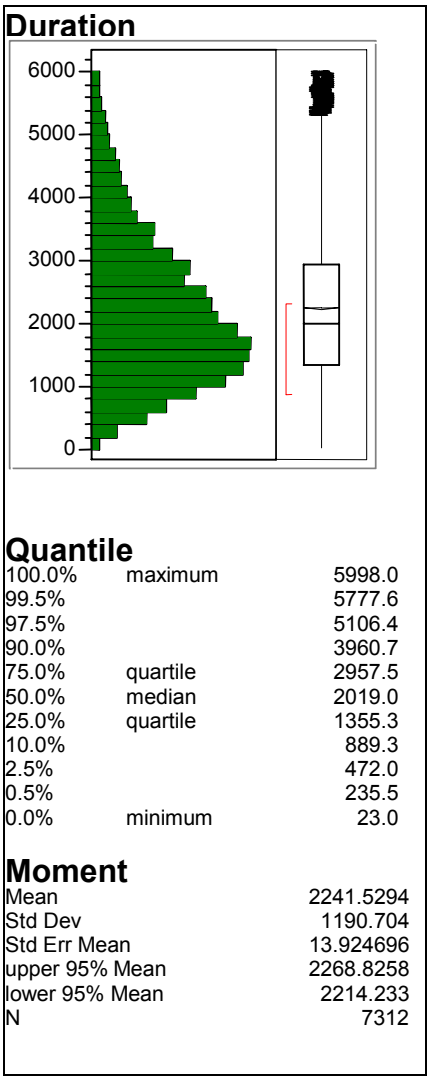
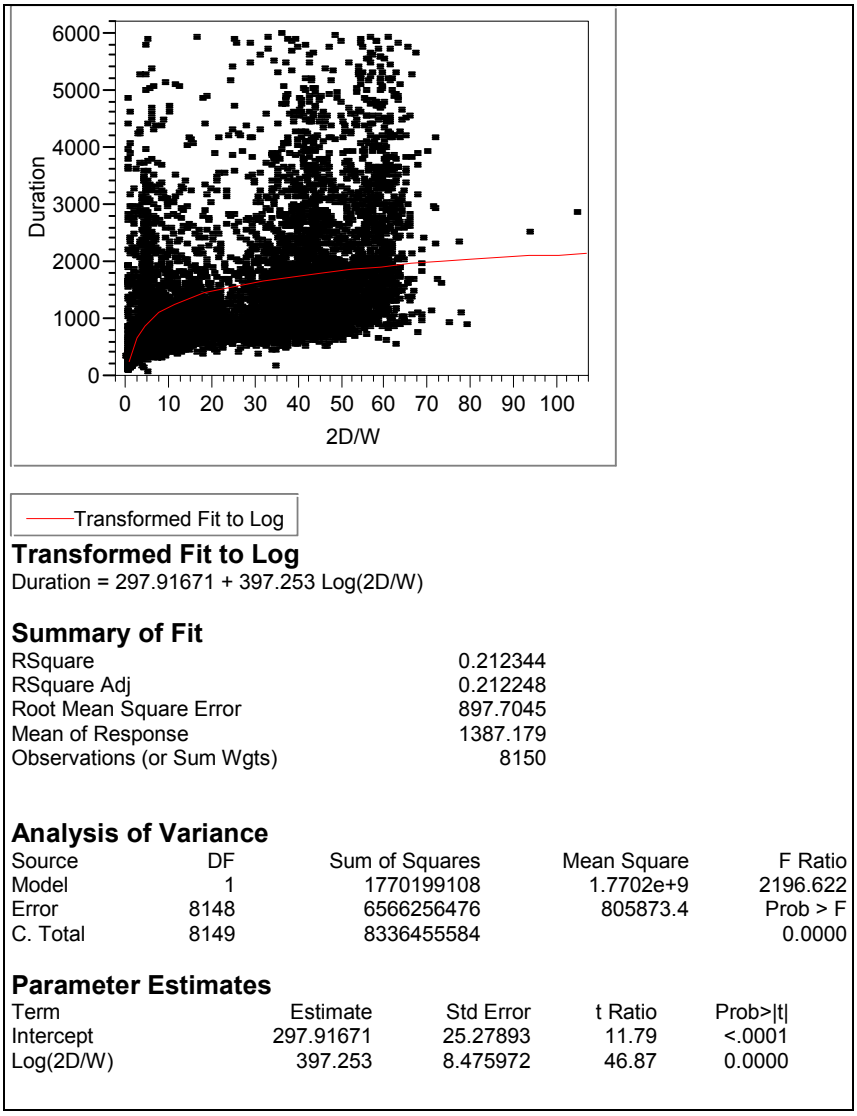


Figure 5.4: Distribution by time (ms.) of selection task with Trackpoint across subjects

Figures 5.5-7 depict scatter plots of Duration versus Distance for all trials for all subjects of the selection task by device. A Fitts' Law model is superimposed over the plot to fit the data. What appears as vertical banding in the graphs can be explained by the fact that the

various controls on the screen that the users were asked to select were not uniformly evenly distributed across the entire range of distances.



Transformed Fit to Log

Duration = 297.91671 + 397.253 Log(2D/W)

Summary of Fit

RSquare	0.212344
RSquare Adj	0.212248
Root Mean Square Error	897.7045
Mean of Response	1387.179
Observations (or Sum Wgts)	8150

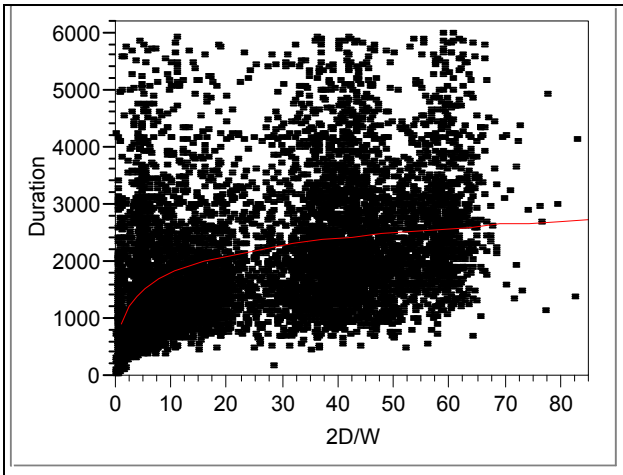
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	1770199108	1.7702e+9	2196.622
Error	8148	6566256476	805873.4	Prob > F
C. Total	8149	8336455584		0.0000

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	297.91671	25.27893	11.79	<.0001
Log(2D/W)	397.253	8.475972	46.87	0.0000

Figure 5.5: Scatter Plot with Fitts' Law Model for Selection with Mouse across subjects



Transformed Fit to Log

Transformed Fit to Log

Duration = 793.6449 + 439.00127 Log(2D/W)

Summary of Fit

RSquare	0.191248
RSquare Adj	0.191137
Root Mean Square Error	1042.659
Mean of Response	2008.87
Observations (or Sum Wgts)	7275

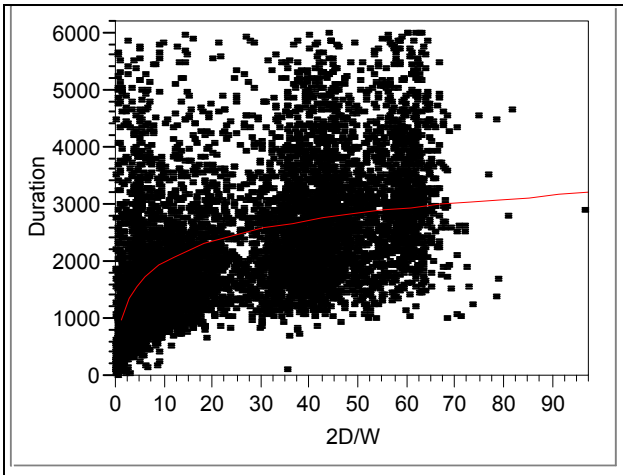
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	1869729456	1.86973e9	1719.865
Error	7273	7906750623	1087137.4	Prob > F
C. Total	7274	9776480078		0.0000

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	793.6449	31.75044	25.00	<.0001
Log(2D/W)	439.00127	10.58568	41.47	0.0000

Figure 5.6: Scatter Plot with Fitts' Law Model for Selection with Touchpad across subjects



Transformed Fit to Log

Transformed Fit to Log

Duration = 760.18858 + 533.60689 Log(2D/W)

Summary of Fit

RSquare	0.281549
RSquare Adj	0.28145
Root Mean Square Error	1009.328
Mean of Response	2241.529
Observations (or Sum Wgts)	7312

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	2918352608	2.91835e9	2864.662
Error	7310	7447007182	1018742.4	Prob > F
C. Total	7311	1.03654e10		0.0000

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	760.18858	30.08884	25.26	<.0001
Log(2D/W)	533.60689	9.969761	53.52	0.0000

Figure 5.7: Scatter Plot with Fitts' Law Model for Selection with Trackpoint across subjects

The median time for completion of the orientation task, over all subjects, is 3051 ms. with the mouse, 3138 ms. with the touchpad and 2880 ms. with the trackpoint. Histograms in figures 5.8-10 show the distributions.

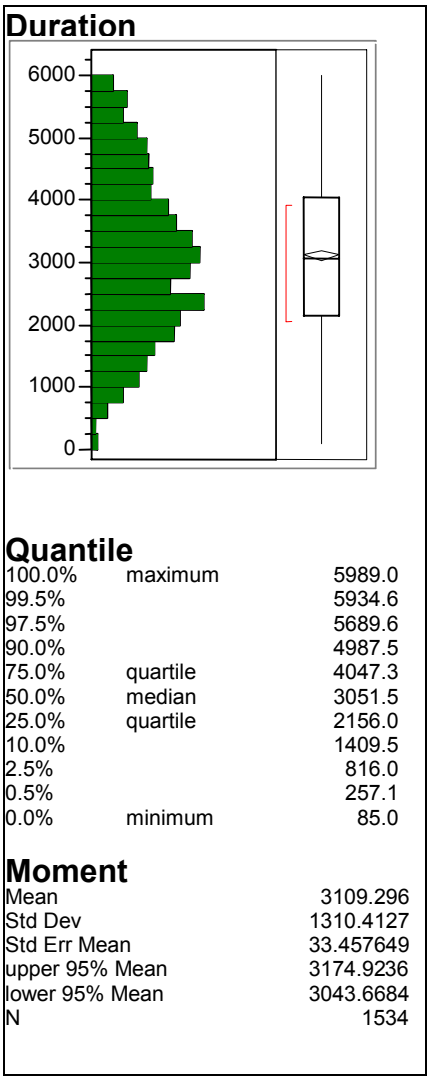


Figure 5.8: Distribution by time (ms.) of Orientation task with Mouse across subjects

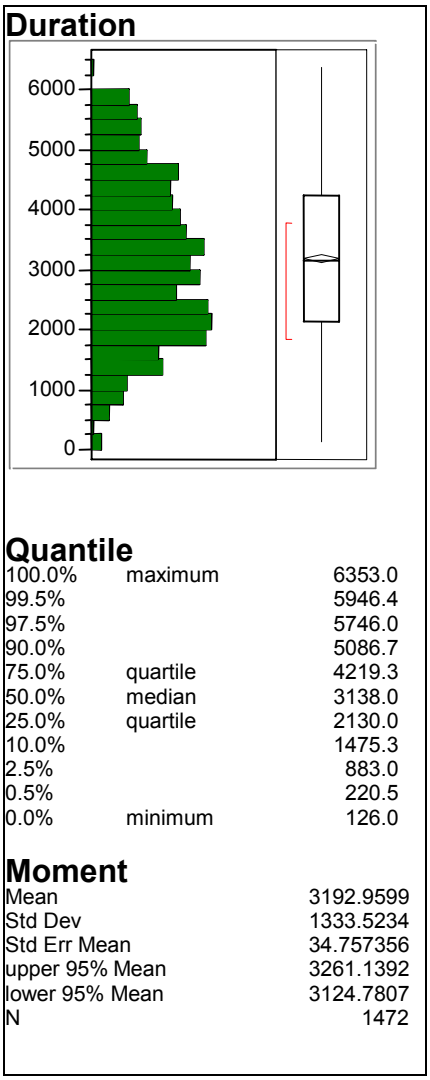


Figure 5.9: Distribution by time (ms.) of Orientation task with Touchpad across subjects

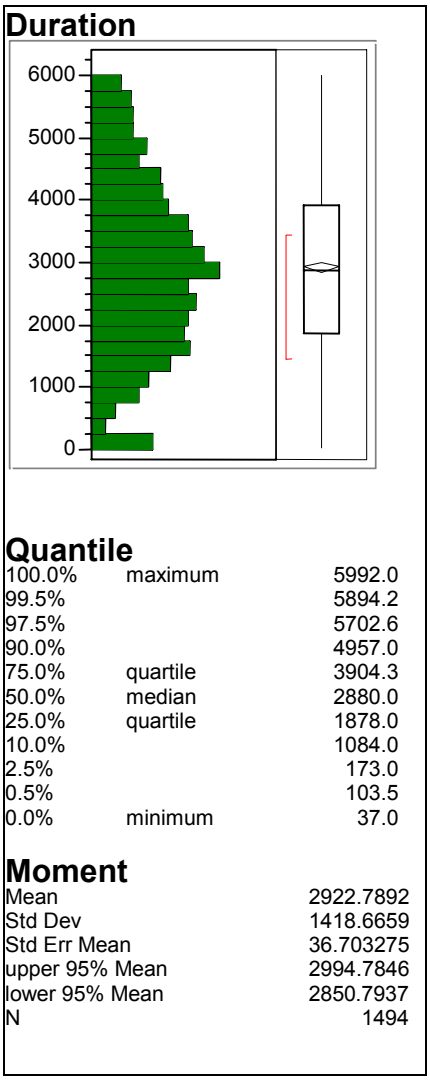
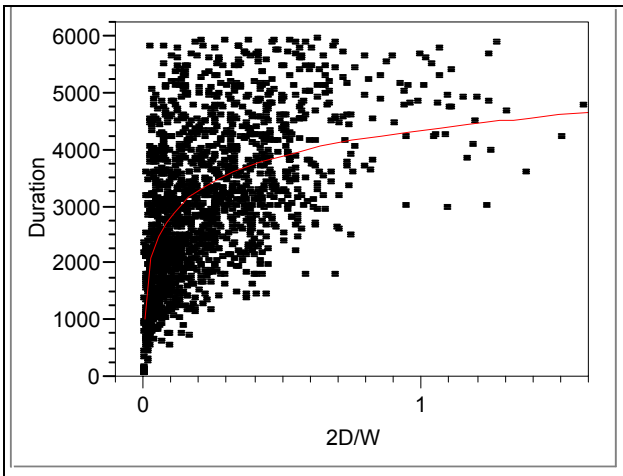


Figure 5.10: Distribution by time (ms.) of Orientation task with Trackpoint across subjects

Figures 5.11-13 depict scatter plots of all trials for all subjects of the orientation task by device. A Fitts' Law model is superimposed over the plot to fit the data.



Transformed Fit to Log

Transformed Fit to Log

Duration = 4357.2609 + 659.4835 Log(2D/W)

Summary of Fit

RSquare	0.327611
RSquare Adj	0.327172
Root Mean Square Error	1074.88
Mean of Response	3109.296
Observations (or Sum Wgts)	1534

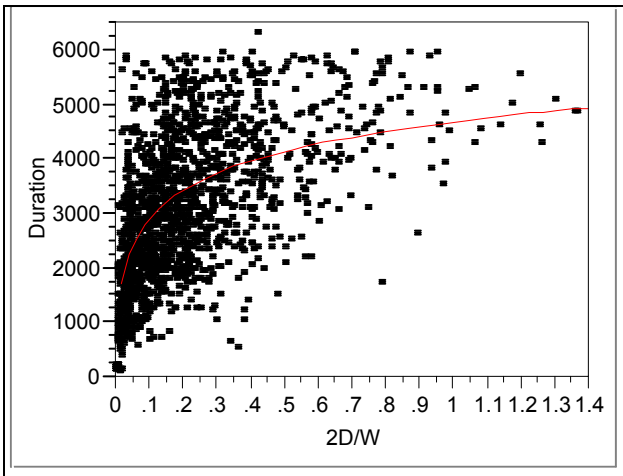
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	862416196	862416196	746.4432
Error	1532	1770023020	1155367.5	Prob > F
C. Total	1533	2632439216		<.0001

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	4357.2609	53.28812	81.77	0.0000
Log(2D/W)	659.4835	24.13824	27.32	<.0001

Figure 5.11: Scatter Plot with Fitts' Law Model for Orientation with Mouse across subjects



Transformed Fit to Log

Transformed Fit to Log

Duration = 4681.5918 + 777.71792 Log(2D/W)

Summary of Fit

RSquare	0.353176
RSquare Adj	0.352736
Root Mean Square Error	1072.856
Mean of Response	3192.96
Observations (or Sum Wgts)	1472

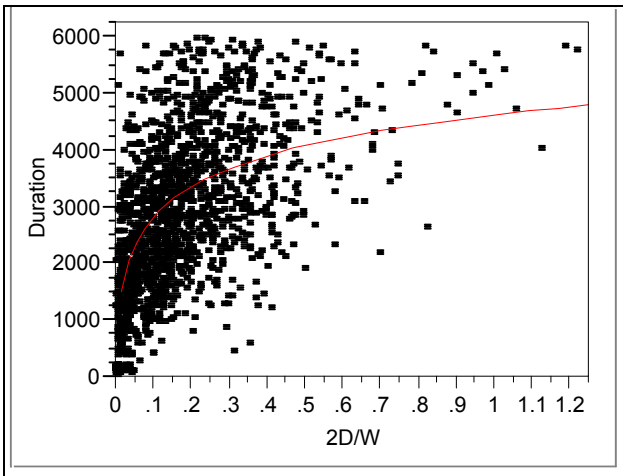
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	923857923	923857923	802.6431
Error	1470	1691998832	1151019.6	Prob > F
C. Total	1471	2615856755		<.0001

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	4681.5918	59.52186	78.65	0.0000
Log(2D/W)	777.71792	27.45117	28.33	<.0001

Figure 5.12: Scatter Plot with Fitts' Law Model for Orientation with Touchpad across subjects



Transformed Fit to Log

Transformed Fit to Log

Duration = 4626.6531 + 787.58961 Log(2D/W)

Summary of Fit

RSquare	0.390266
RSquare Adj	0.389857
Root Mean Square Error	1108.143
Mean of Response	2922.789
Observations (or Sum Wgts)	1494

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	1172683619	1.17268e9	954.9690
Error	1492	1832147372	1227980.8	Prob > F
C. Total	1493	3004830991		<.0001

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	4626.6531	62.14491	74.45	0.0000
Log(2D/W)	787.58961	25.48621	30.90	<.0001

Figure 5.13: Scatter Plot with Fitts' Law Model for Orientation with Trackpoint across subjects

The median time for completion of the position task, over all subjects, is 1688 ms. with the mouse, 2110 ms. with the touchpad and 2045 ms. with the trackpoint. Histograms in figures 5.14-16 show the distributions.

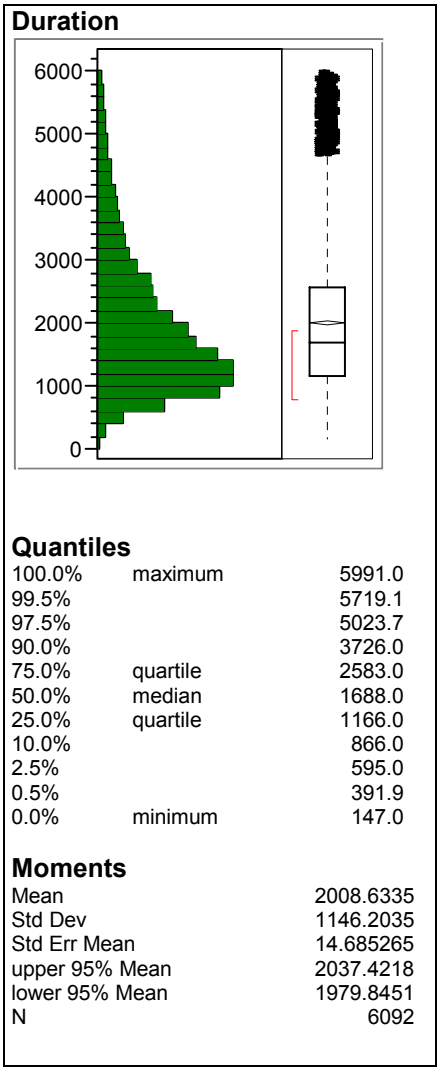


Figure 5.14: Distribution by time (ms.) of Position task with Mouse across subjects

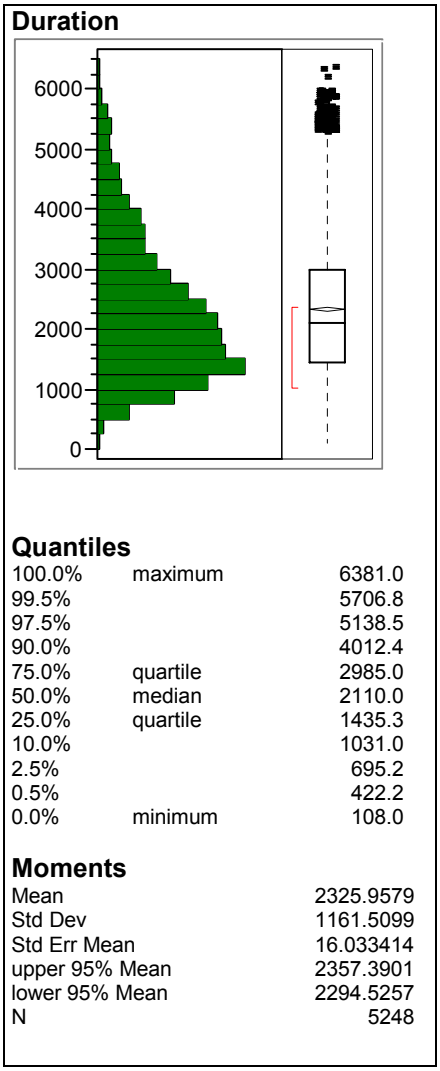


Figure 5.15: Distribution by time (ms.) of Position task with Touchpad across subjects

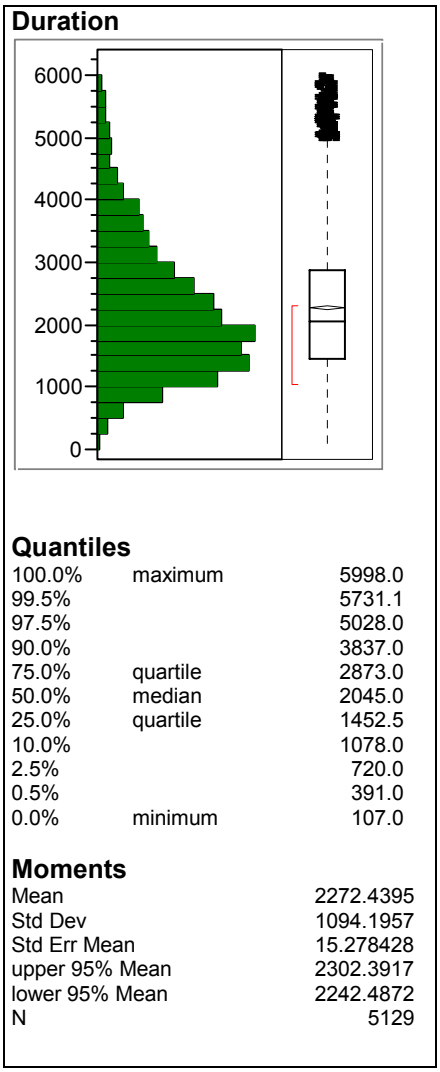
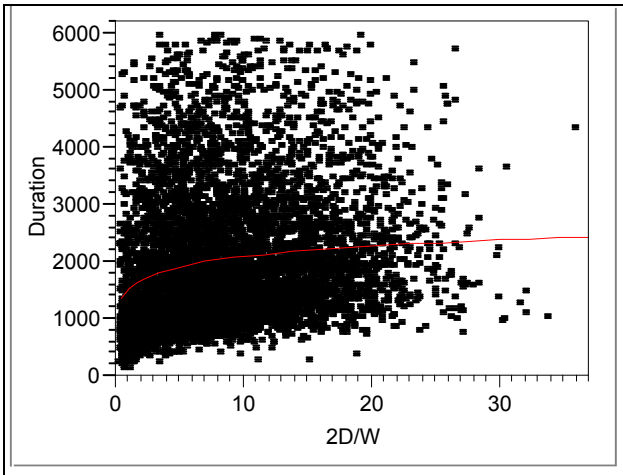


Figure 5.16: Distribution by time (ms.) of Position task with Trackpoint across subjects

Figures 5.17-19 depict scatter plots of all trials for all subjects of the position task by device. A Fitts' Law model is superimposed over the plot to fit the data.



Transformed Fit to Log

Transformed Fit to Log

Duration = 1489.9683 + 262.5289 Log(2D/W)

Summary of Fit

RSquare	0.031278
RSquare Adj	0.031119
Root Mean Square Error	1128.228
Mean of Response	2008.633
Observations (or Sum Wgts)	6092

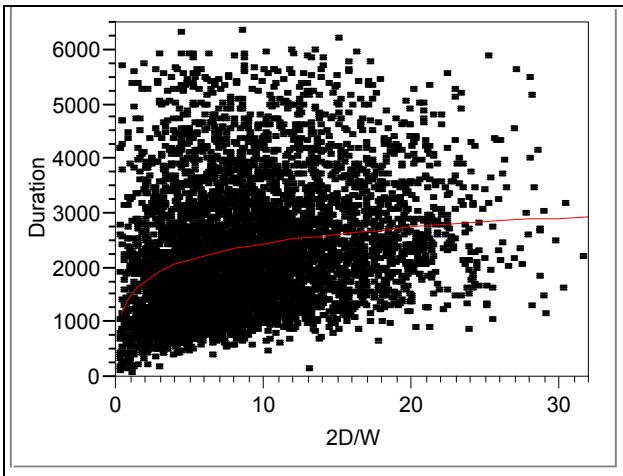
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	250297976	250297976	196.6362
Error	6090	7751951650	1272898.5	Prob > F
C. Total	6091	8002249627		<.0001

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	1489.9683	39.71175	37.52	<.0001
Log(2D/W)	262.5289	18.7217	14.02	<.0001

Figure 5.17: Scatter Plot with Fitts' Law Model for Position with Mouse across subjects



Transformed Fit to Log

Transformed Fit to Log

Duration = 1470.8271 + 426.03409 Log(2D/W)

Summary of Fit

RSquare	0.071505
RSquare Adj	0.071328
Root Mean Square Error	1119.32
Mean of Response	2325.958
Observations (or Sum Wgts)	5248

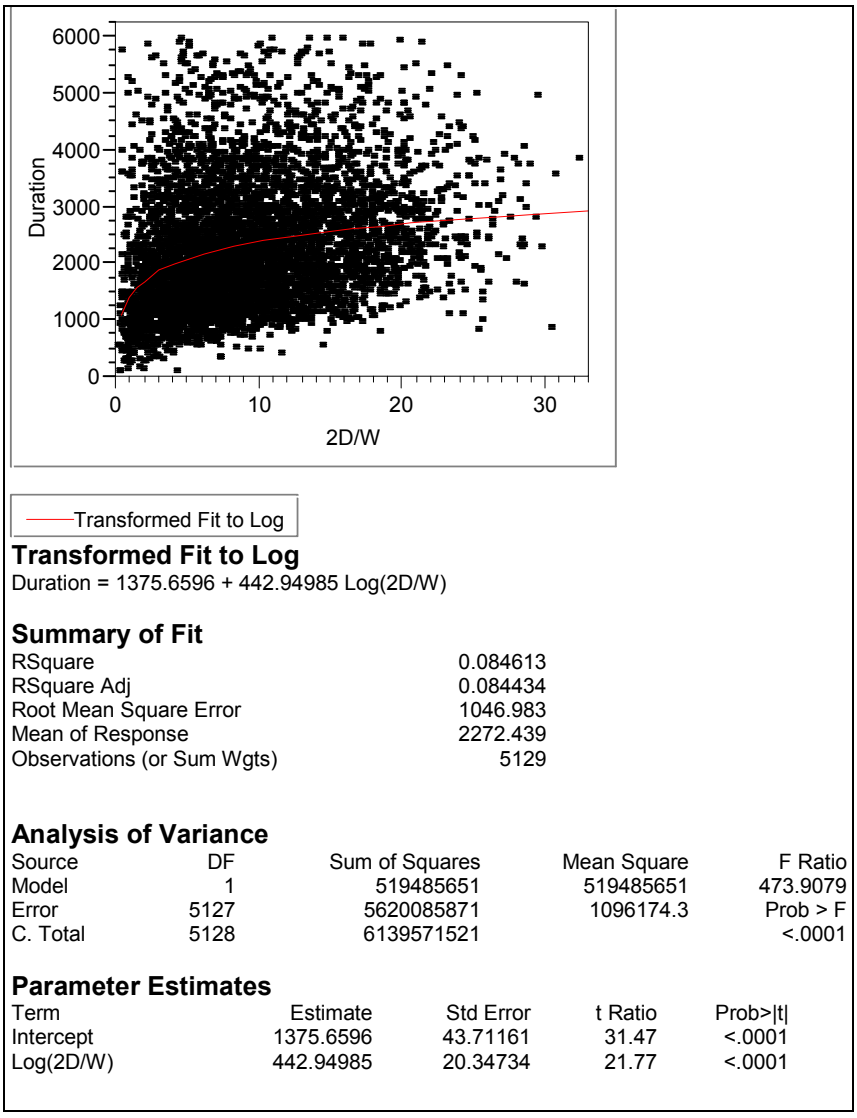
Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	506164033	506164033	404.0014
Error	5246	6572591631	1252876.8	Prob > F
C. Total	5247	7078755664		<.0001

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	1470.8271	45.26311	32.50	<.0001
Log(2D/W)	426.03409	21.19595	20.10	<.0001

Figure 5.18: Scatter Plot with Fitts' Law Model for Position with Touchpad across subjects



Transformed Fit to Log

Duration = 1375.6596 + 442.94985 Log(2D/W)

Summary of Fit

RSquare	0.084613
RSquare Adj	0.084434
Root Mean Square Error	1046.983
Mean of Response	2272.439
Observations (or Sum Wgts)	5129

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	519485651	519485651	473.9079
Error	5127	5620085871	1096174.3	Prob > F
C. Total	5128	6139571521		<.0001

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	1375.6596	43.71161	31.47	<.0001
Log(2D/W)	442.94985	20.34734	21.77	<.0001

Figure 5.19: Scatter Plot with Fitts' Law Model for Position with Trackpoint across subjects

In the aggregate task data that we have seen in figures 5.2-19, we can see that the mouse is superior to the other two devices for selection and position in terms of performance. Between the touchpad and the trackpoint, we see that the touchpad is marginally faster for

the selection task while the trackpoint is faster for the position task and fastest overall for the orientation task. All of the patterns we see in the data are accounted for by Fitts' Law models reasonably well, and yet there's considerable variance that's not accounted for. Next we examine an individual user's data in order to show where at least some of this variance comes from.

5.4. Individual performance patterns

To illustrate differences between individual and aggregate performance, it will be helpful to see the complete analysis for a single subject. Figures 5.20-22 show the performance relationship for subject six, as measured by the distance that the subject moved the pointer for the purpose of selection and the time that this action took, using the mouse, the touchpad, and the trackpoint, respectively. Similarly, figures 5.23-25 show similar data for the same devices in the orientation task. Figures 5.26-28 show similar data for the same devices in the positioning task. We are interested in seeing how closely the results of this individual match the results of the group.

Each of these data plots is shown with a superimposed line representing a least squares log model of the relationship between Duration and Distance, as dictated by Fitts' Law. Although the fit of the model varies considerably over the different input devices, the form of the model is generally acceptable, consistent with the rest of the evaluation literature.

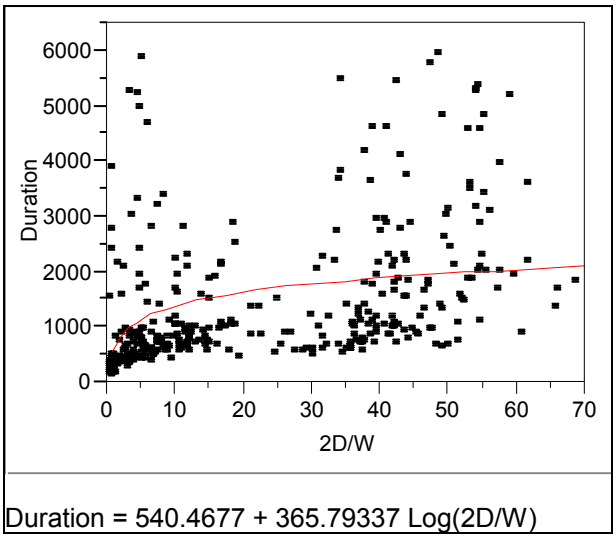


Figure 5.20: Selection task using the mouse (subject 6)

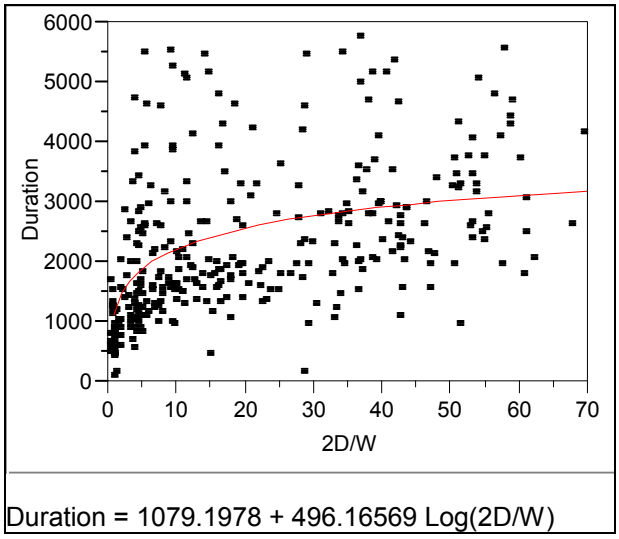


Figure 5.21: Selection task using the touchpad (subject 6)

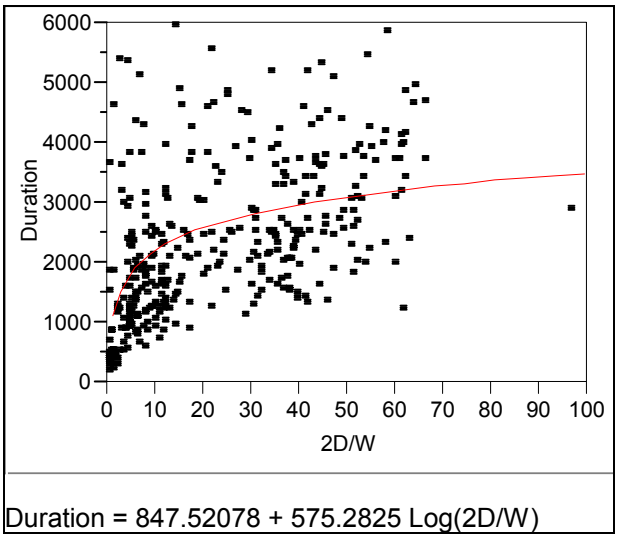


Figure 5.22: Selection task using the trackpoint (subject 6)

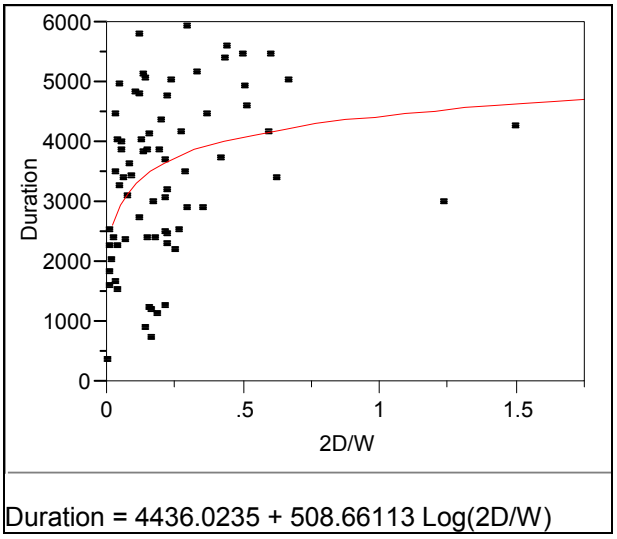


Figure 5.23: Orientation task using the mouse (subject 6)

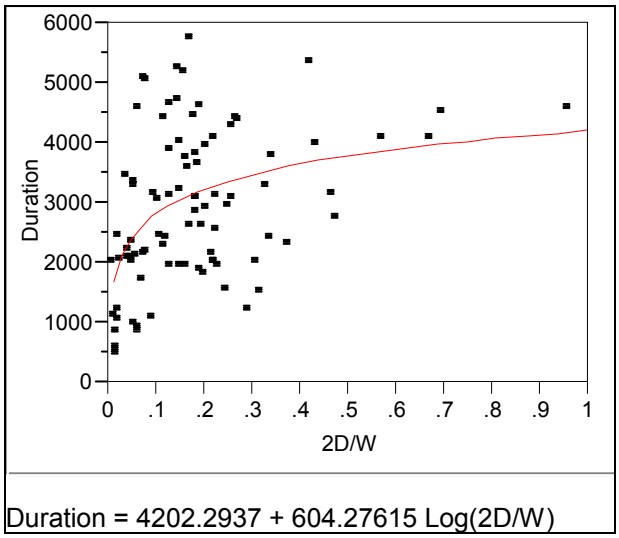


Figure 5.24: Orientation task using the touchpad (subject 6)

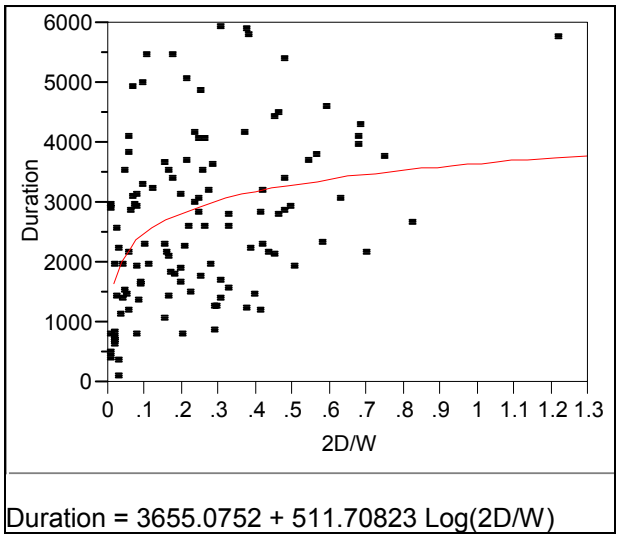


Figure 5.25: Orientation task using the trackpoint (subject 6)

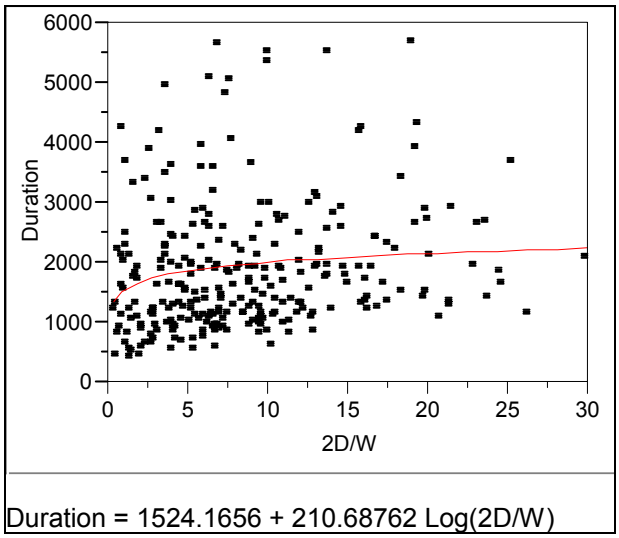


Figure 5.26: Position task using the mouse (subject 6)

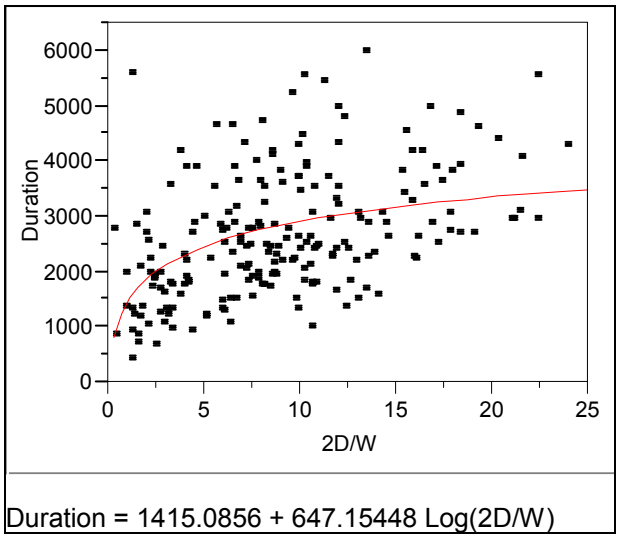


Figure 5.27: Position task using the touchpad (subject 6)

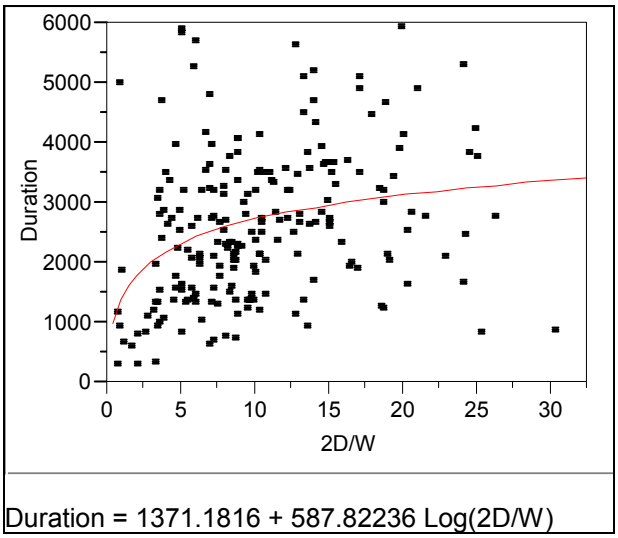


Figure 5.28: Position task using the trackpoint (subject 6)

From the figures above, we can see the overall performance of user six for the task performance times. The curve gives a reasonable approximation in most cases of the user's expected performance time over the range of the task. Table 5.4 provides detailed parameters of the quality of the curve that was fit to the data.

The performance characteristics of the selection task (figures 5.20-22) for subject six suggest that they were able to perform the task more quickly with the mouse and took the longest time with the trackpoint. While the touchpad performance was faster than the trackpoint performance, the subject did commit more errors with the touchpad (42) than the trackpoint (28). The subject committed the fewest number of errors with the mouse (25). Results were similar for the position task, the mouse was superior with regards to both performance time and errors. There was little difference, if any, in performance between the

trackpoint and touchpad for the position task. Interestingly enough, the performance dynamics of the orientation task were quite different across devices for this subject. The data indicates that the subject had the fastest time with the trackpoint and slowest time with the mouse, the performance with the touchpad being slightly better than the mouse, the exact opposite performance of the selection task. This subject had many more orientation observations when using the trackpoint than either of the other devices. It appears that they performed at least 1.54 orientations per trial with the trackpoint when only 1 was required to complete the task with the mouse and touchpad. This would seem to indicate that subject six employed a different strategy for orientation with the trackpoint than the other two devices. We speculate that the subject may have found it easier to make a series of small adjustments to the orientation when using the trackpoint and then inspecting the results, making a second corrective orientation if needed or may have been "dropping" the dial control after a short distance, requiring them to perform another orientation. This behavior is atypical.

Trans.	Device	N	R square	F ratio	Prob <f	T log(2D/W)	Prob T	T intercept	Prob.<t
Sel.	Mouse	347	.1673	69.29	<.0001	8.32	<.0001	4.35	<.0001
Sel.	T. pad	317	.2462	102.91	<.0001	10.14	<.0001	7.87	<.0001
Sel.	T.point	346	.2941	143.33	<.0001	11.97	<.0001	6.03	<.0001
Orient	Mouse	72	.1813	15.51	.0002	3.94	.0002	14.96	<.0001
Orient	T. pad	89	.2231	24.99	<.0001	5.00	<.0001	15.10	<.0001
Orient	T.point	116	.1884	26.47	<.0001	5.14	<.0001	16.65	<.0001
Pos.	Mouse	271	.0298	8.27	.0043	2.88	.0043	10.19	<.0001
Pos.	T. pad	208	.1972	50.60	<.0001	7.11	<.0001	7.36	<.0001
Pos.	T.point	209	.1069	24.78	<.0001	4.98	<.0001	5.15	<.0001

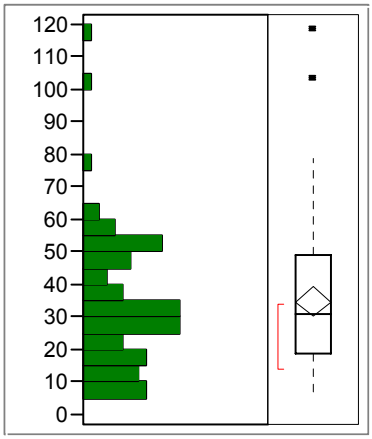
Table 5.4: Sample size and task curve fit parameters for subject six

Figures 5.29-40 show distributions of subject six's performance in ms. for the transitions between tasks. Figures 5.29-31 show the data for the selection-orientation task for the mouse, the touchpad, and the trackpoint, respectively. Figures 5.32-34 show the data for subject six related to the selection-position transition, figures 5.35-37 show the data for the orientation-selection transition and figures 5.38-40 show the data for the position-selection transition.

The transition data for subject six shows very little difference across devices for both the selection-orientation and selection-position transitions both in time and variability. This is

consistent with our definition of these transitions since they simply measure the time between the ending mouse click of a selection and the start of an orientation or position. The data for the orientation-selection and position-selection transitions indicate that subject six was consistently faster and performed with less variability using the mouse and took the longest amount of time and had the greatest variability with the touchpad while the trackpoint performance lies somewhere in-between. Subject six's performance characteristics for the orientation-selection and position-selection transitions were consistent with the group as a whole although the differential between devices was smaller for the orientation-selection transition and the touchpoint performance for the position-selection transition was much slower.

Duration



Quantiles

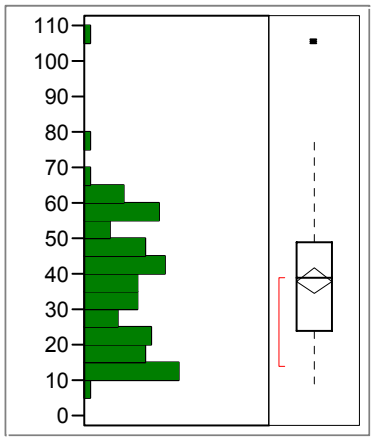
100.0%	maximum	119.00
99.5%		119.00
97.5%		100.25
90.0%		56.40
75.0%	quartile	49.00
50.0%	median	31.00
25.0%	quartile	19.00
10.0%		12.00
2.5%		8.00
0.5%		5.00
0.0%	minimum	5.00

Moments

Mean	34.8
Std Dev	20.094894
Std Err Mean	2.1795973
upper 95% Mean	39.134368
lower 95% Mean	30.465632
N	85

5.29: Distribution by time (ms.) for S/DM transition with mouse (subject 6)

Duration



Quantiles

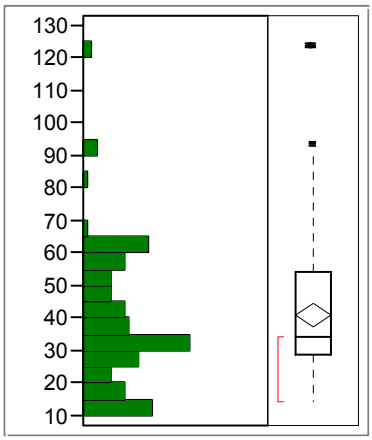
100.0%	maximum	106.00
99.5%		106.00
97.5%		72.17
90.0%		59.00
75.0%	quartile	49.00
50.0%	median	39.00
25.0%	quartile	24.00
10.0%		14.00
2.5%		12.58
0.5%		9.00
0.0%	minimum	9.00

Moments

Mean	38.04
Std Dev	18.353939
Std Err Mean	1.8353939
upper 95% Mean	41.68182
lower 95% Mean	34.39818
N	100

5.30: Distribution by time (ms.) for S/DM transition with touchpad (subject 6)

Duration



Quantiles

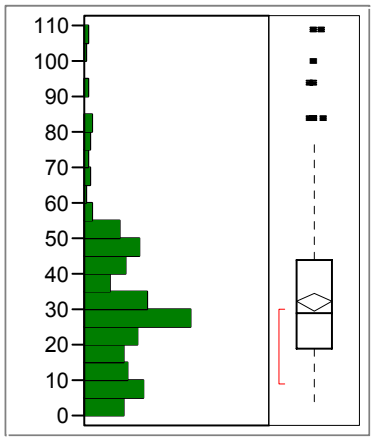
100.0%	maximum	124.00
99.5%		124.00
97.5%		93.48
90.0%		64.00
75.0%	quartile	54.00
50.0%	median	34.00
25.0%	quartile	29.00
10.0%		14.00
2.5%		14.00
0.5%		14.00
0.0%	minimum	14.00

Moments

Mean	40.714286
Std Dev	21.109565
Std Err Mean	1.8805895
upper 95% Mean	44.436206
lower 95% Mean	36.992366
N	126

5.31: Distribution by time (ms.) for S/DM transition with trackpoint (subject 6)

Duration



Quantiles

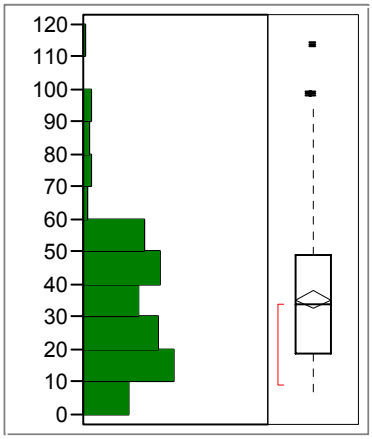
100.0%	maximum	109.00
99.5%		109.00
97.5%		84.00
90.0%		54.00
75.0%	quartile	44.00
50.0%	median	29.00
25.0%	quartile	19.00
10.0%		9.00
2.5%		4.00
0.5%		4.00
0.0%	minimum	4.00

Moments

Mean	32.057554
Std Dev	20.004429
Std Err Mean	1.1997859
upper 95% Mean	34.419411
lower 95% Mean	29.695697
N	278

5.32: Distribution by time (ms.) for S/P transition with mouse (subject 6)

Duration



Quantiles

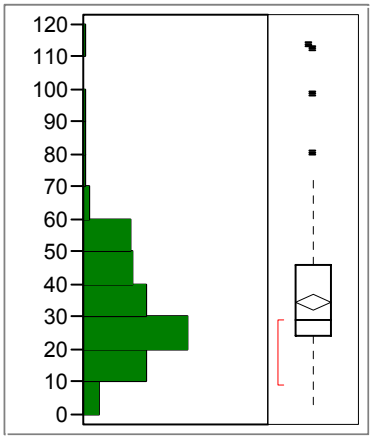
100.0%	maximum	114.00
99.5%		112.58
97.5%		87.68
90.0%		59.00
75.0%	quartile	49.00
50.0%	median	34.00
25.0%	quartile	19.00
10.0%		9.00
2.5%		9.00
0.5%		5.10
0.0%	minimum	5.00

Moments

Mean	35.279817
Std Dev	20.427903
Std Err Mean	1.3835522
upper 95% Mean	38.006737
lower 95% Mean	32.552896
N	218

5.33: Distribution by time (ms.) for S/P transition with touchpad (subject 6)

Duration



Quantiles

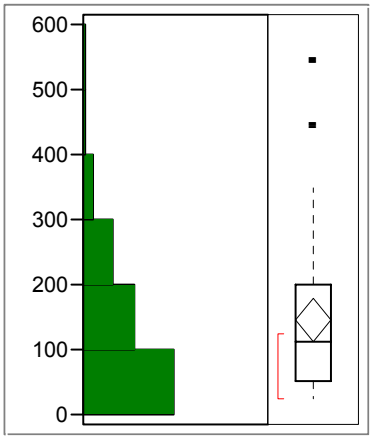
100.0%	maximum	114.00
99.5%		113.92
97.5%		70.80
90.0%		54.00
75.0%	quartile	46.00
50.0%	median	29.00
25.0%	quartile	24.00
10.0%		14.00
2.5%		9.00
0.5%		3.08
0.0%	minimum	3.00

Moments

Mean	34.488372
Std Dev	17.447347
Std Err Mean	1.1898992
upper 95% Mean	36.833796
lower 95% Mean	32.142948
N	215

5.34: Distribution by time (ms.) for S/P transition with trackpoint (subject 6)

Duration



Quantiles

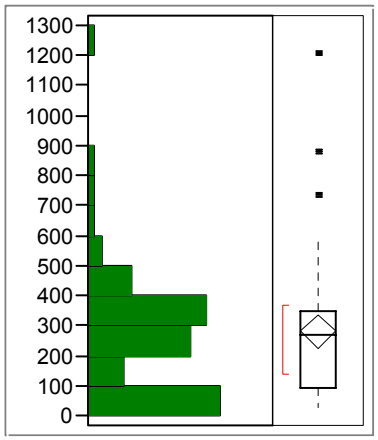
100.0%	maximum	547.00
99.5%		547.00
97.5%		514.50
90.0%		315.80
75.0%	quartile	199.75
50.0%	median	111.50
25.0%	quartile	50.00
10.0%		25.00
2.5%		24.00
0.5%		24.00
0.0%	minimum	24.00

Moments

Mean	145.36538
Std Dev	117.75282
Std Err Mean	16.329378
upper 95% Mean	178.14798
lower 95% Mean	112.58279
N	52

5.35: Distribution by time (ms.) for DM/S transition with mouse (subject 6)

Duration



Quantiles

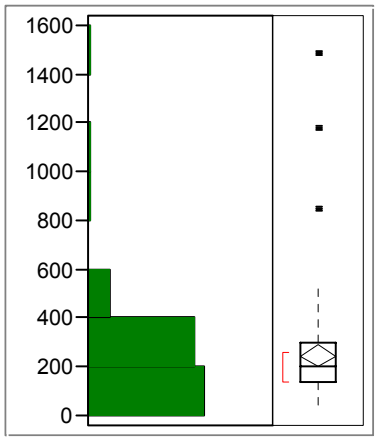
100.0%	maximum	1214.0
99.5%		1214.0
97.5%		1000.1
90.0%		512.6
75.0%	quartile	350.0
50.0%	median	270.0
25.0%	quartile	91.0
10.0%		45.0
2.5%		23.0
0.5%		23.0
0.0%	minimum	23.0

Moments

Mean	279.15385
Std Dev	215.03446
Std Err Mean	26.671742
upper 95% Mean	332.43678
lower 95% Mean	225.87092
N	65

5.36: Distribution by time (ms.) for DM/S transition with touchpad (subject 6)

Duration



Quantiles

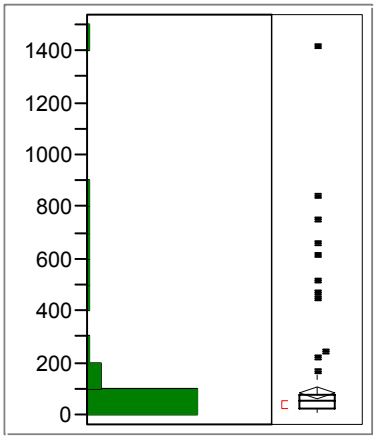
100.0%	maximum	1487.0
99.5%		1487.0
97.5%		1078.1
90.0%		443.3
75.0%	quartile	295.8
50.0%	median	205.0
25.0%	quartile	138.0
10.0%		45.9
2.5%		21.3
0.5%		21.0
0.0%	minimum	21.0

Moments

Mean	246.09783
Std Dev	215.56657
Std Err Mean	22.474369
upper 95% Mean	290.7404
lower 95% Mean	201.45525
N	92

5.37: Distribution by time (ms.) for DM/S transition with trackpoint (subject 6)

Duration



Quantiles

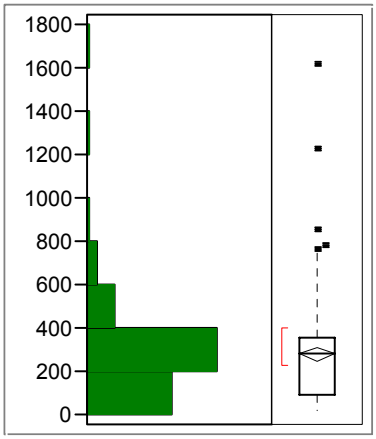
100.0%	maximum	1422.0
99.5%		1375.7
97.5%		581.4
90.0%		124.4
75.0%	quartile	75.0
50.0%	median	50.0
25.0%	quartile	25.0
10.0%		25.0
2.5%		24.0
0.5%		11.6
0.0%	minimum	11.0

Moments

Mean	83.451163
Std Dev	141.91289
Std Err Mean	9.6783786
upper 95% Mean	102.52832
lower 95% Mean	64.374002
N	215

5.38: Distribution by time (ms.) for P/S transition with mouse (subject 6)

Duration



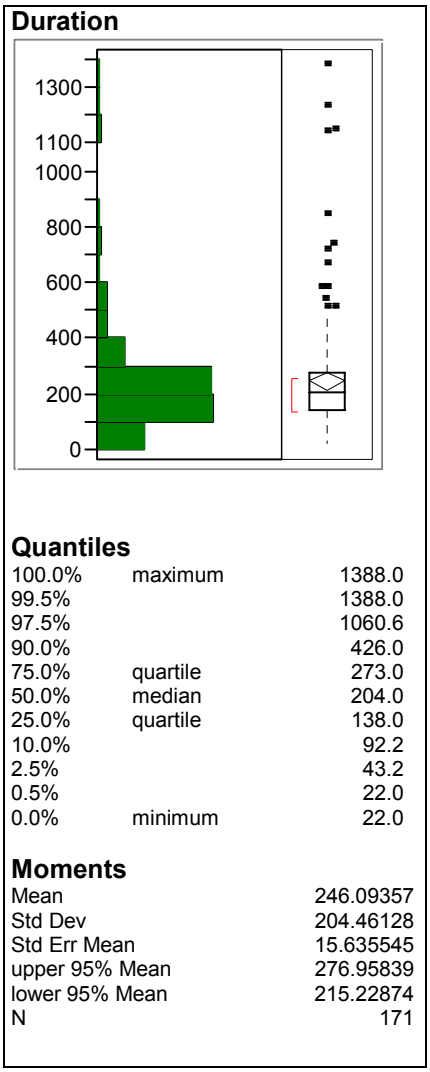
Quantiles

100.0%	maximum	1621.0
99.5%		1621.0
97.5%		779.4
90.0%		503.8
75.0%	quartile	359.0
50.0%	median	283.0
25.0%	quartile	90.0
10.0%		45.0
2.5%		23.0
0.5%		22.0
0.0%	minimum	22.0

Moments

Mean	280.57143
Std Dev	214.16455
Std Err Mean	16.189318
upper 95% Mean	312.52415
lower 95% Mean	248.61871
N	175

5.39: Distribution by time (ms.) for P/S transition with touchpad (subject 6)

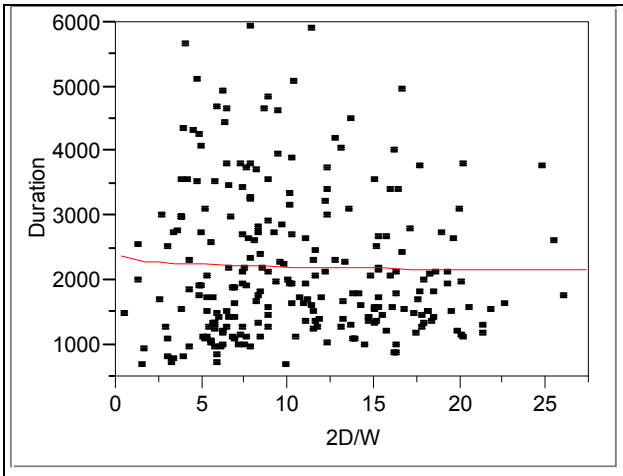


5.40: Distribution by time (ms.) for P/S transition with trackpoint (subject 6)

5.5. Individual Performance Differences

We have seen the patterns in the overall data and how these relate to individual patterns, but there are further unusual patterns we find associated with specific subjects that are worth considering, for completeness.

Two subjects were observed to have a negative log term for the Fitts' law model of their positioning task using the mouse. Figure 5.41 shows the data for subject nine.



Transformed Fit to Log

Transformed Fit to Log

Duration = 2330.3701 - 52.925814 Log(2D/W)

Summary of Fit

RSquare	0.000796
RSquare Adj	-0.0034
Root Mean Square Error	1135.079
Mean of Response	2213.5
Observations (or Sum Wgts)	240

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	244346	244346	0.1897
Error	238	306640142	1288404	Prob > F
C. Total	239	306884488		0.6636

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	2330.3701	278.1877	8.38	<.0001
Log(2D/W)	-52.92581	121.5321	-0.44	0.6636

Figure 5.41: Scatter Plot with Fitts' Law Model for Selection with Mouse (subject 9)

At first glance, data would seem to suggest that it took subject nine longer to position an object with a mouse, the shorter the distance that they were required to move. This is counter to all research in the field. Upon closer inspection, however, it would appear that with the removal of 12 data points over 4500 ms. in time, a weak Fitts' Law model would result. We speculate that the subject was trying to maximize the fidelity of their performance with little consideration for performing the task as quickly as possible.

With respect to mean transition times, we observed two interesting anomalies. Subject four had a median time of 854 ms. for the orientation-selection transition when using the trackpoint; while the mean time across all subjects was 243 ms., subject four was 611 ms. slower. Their performance was nominally below the median when using the mouse and touchpad for the same transition. Since the trackpoint was the second device they used, order effect does not explain this inconsistency. Subject four committed a large number of errors when performing the orientation task, successfully performing the operation only eight times. This leads us to conclude that their difficulty with the orientation task spilled over into the transition from orientation to selection as they attempted to confirm that they had performed the task correctly before they went on to the selection. We would, however, not try to read too much into this as there were only eight data points available for analysis.

Subjects 10 and 16 had unusual data for transitions while using the mouse. Subject 10 had an unusually short duration for the orientation-selection transition (figure 5.42), a median time of 25ms., while the median across subjects was 66.25 ms. Subject 16 had an abnormally low transition time for the position-selection transition (figure 5.43), a median

time of 25ms., compared to 63.08 ms. across subjects. We speculate that these subjects may have had a different method for performing these single transitions; for example, they may have simply started moving as soon as they completed their pre-action and performed their confirmation while they were engaged in the selection as opposed to the more normal method of pausing to confirm. What is interesting to note is that neither subject's data was abnormal for any of the other transitions or tasks.

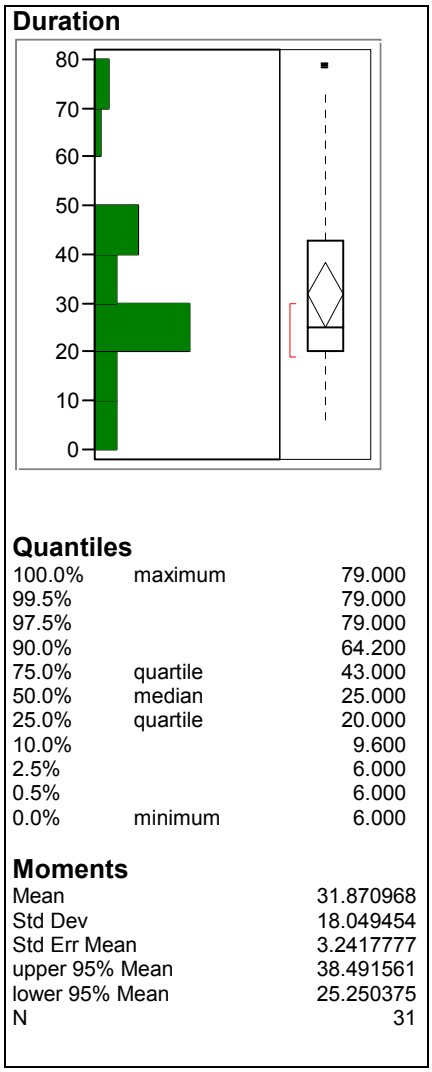


Figure 5.42: Distribution by time (ms.) for DM/S transition with mouse (subject 10)

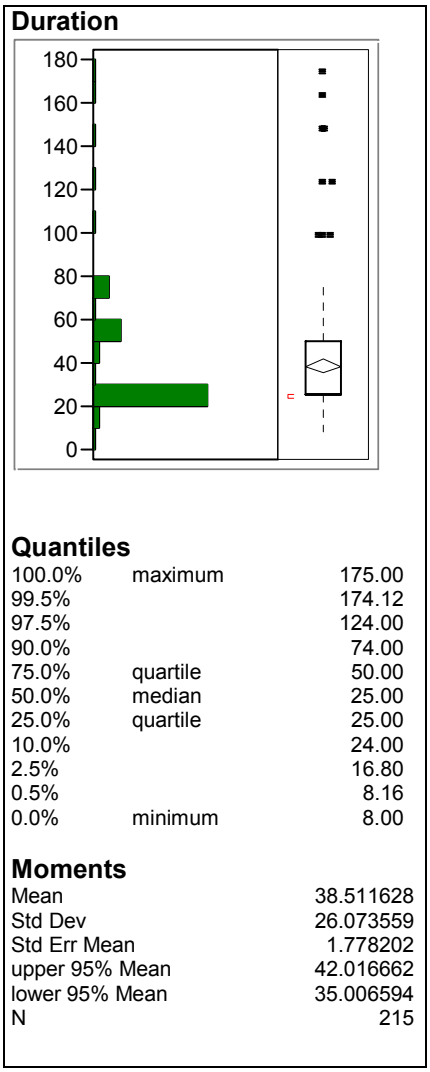


Figure 5.43: Distribution by time (ms.) for P/S transition with mouse (subject 16)

A great deal of variability between subjects was noticed in both the task performance and transition times. To illustrate this effect, figures 5.44-49 display normalized mean performance time by subject for each of the tasks and transitions. The figures were created by taking the raw data and normalizing it to the mouse being equal to 1.0. The mouse

performance therefore runs across the charts as a straight line. If all user performance was consistent in its interdevice relationships we would expect to see a straight line for both the touchpad and trackpoint data located either above or below the x axis. Instead what we see is that the relationship between performance times varies dramatically from subject to subject and from task to task.

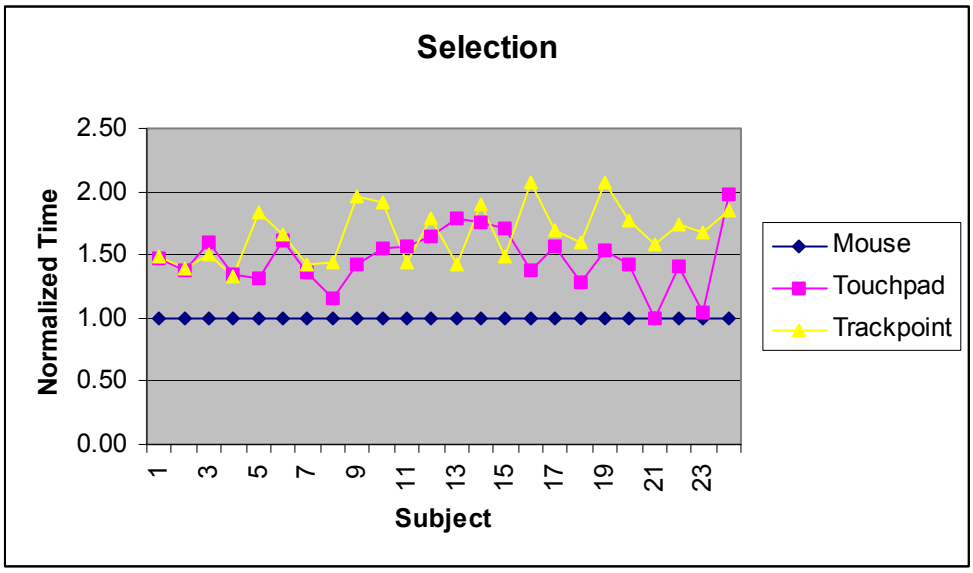


Figure 5.44: Normalized performance time (ms.) for selection task (mouse=1.0)

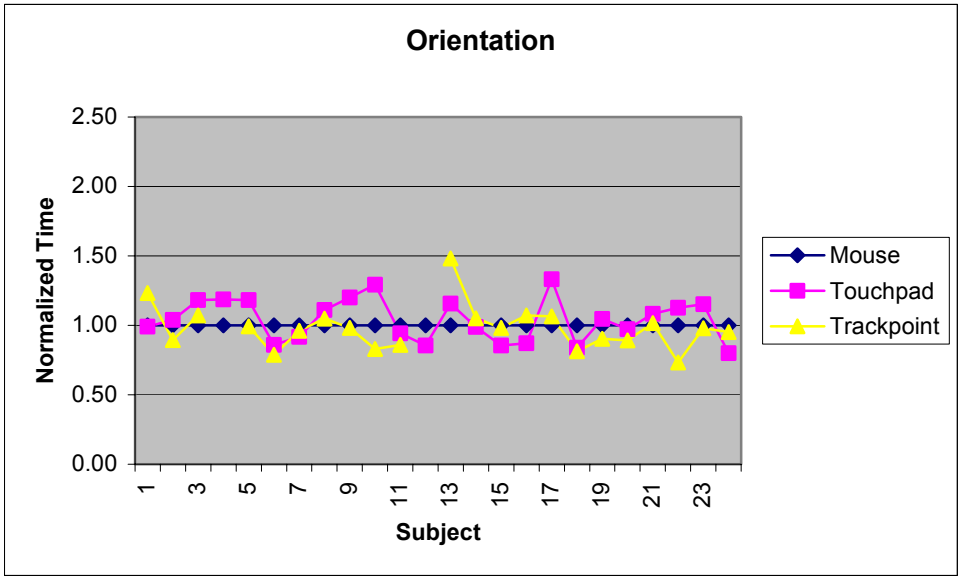


Figure 5.45: Normalized performance time (ms.) for orientation task (mouse=1.0)

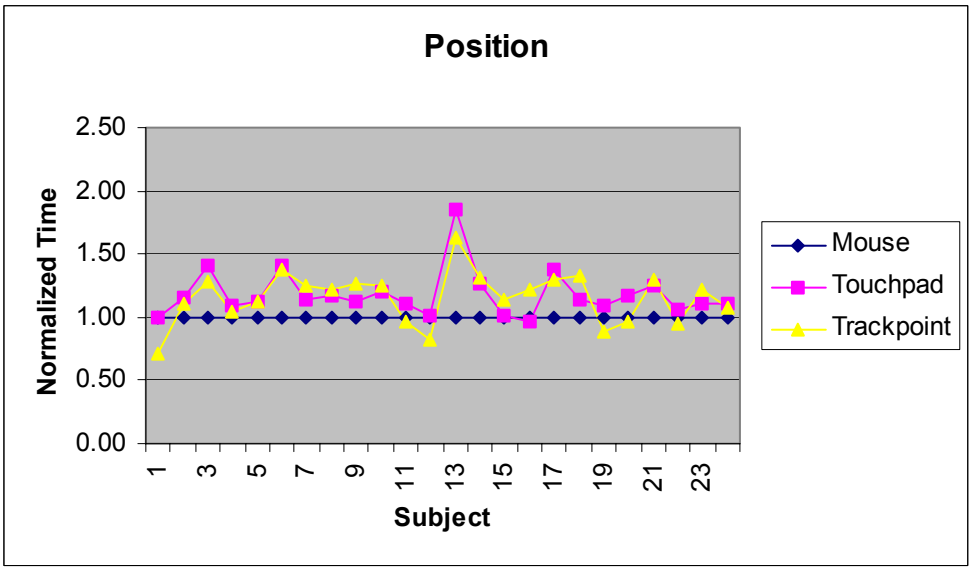


Figure 5.46: Normalized performance time (ms.) for position task (mouse=1.0)

In the selection task (figure 5.44) we see that 14 users had faster mean performance with the touchpad as opposed to the trackpoint. In no case was either of those two devices faster than the mouse. Most users' performance with the trackpoint and touchpad were between 1.5 and 2 times slower than when using the mouse. In the orientation task data (figure 5.45), we see an entirely different picture. While the relationship or general trends of performance are less clear, 13 of the subjects are performing fastest with the trackpoint as opposed to the mouse (subjects 4 and 12 had insufficient trackpoint data to reach any meaningful comparisons, hence the 0 values). For the position task (figure 5.46) we see that between the touchpad and trackpoint, there is no clear consensus with regards to superiority; 14 users were faster with the trackpoint. In six cases, the user was faster with the trackpoint as opposed to the mouse.

These results provide a much clearer understanding of the wide range of differences in performance between subjects that simply isn't captured in previous work where devices are evaluated across groups with no consideration for individual differences.

When looking at the normalized transition data in figures 5.47-50, we see an entirely different trend. As we would expect, there is little if any difference between any of the devices for the S/DM and S/P transitions (figures 5.47-48). As you may recall, these two transitions are just the time required to activate the pointing device, so they are more reflective of a hardware timing interval than anything else.

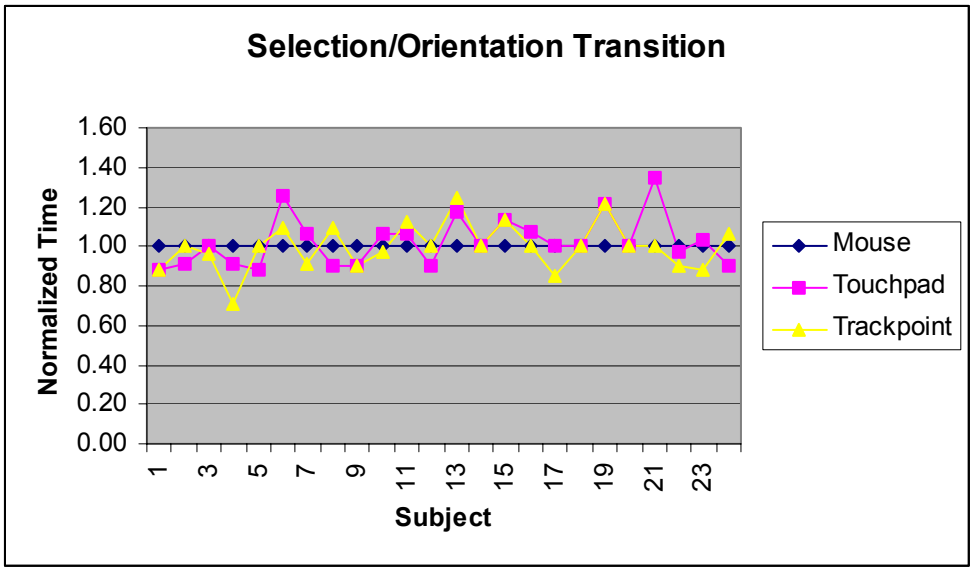


Figure 5.47: Normalized performance time (ms.) for S/DM transition (mouse=1.0)

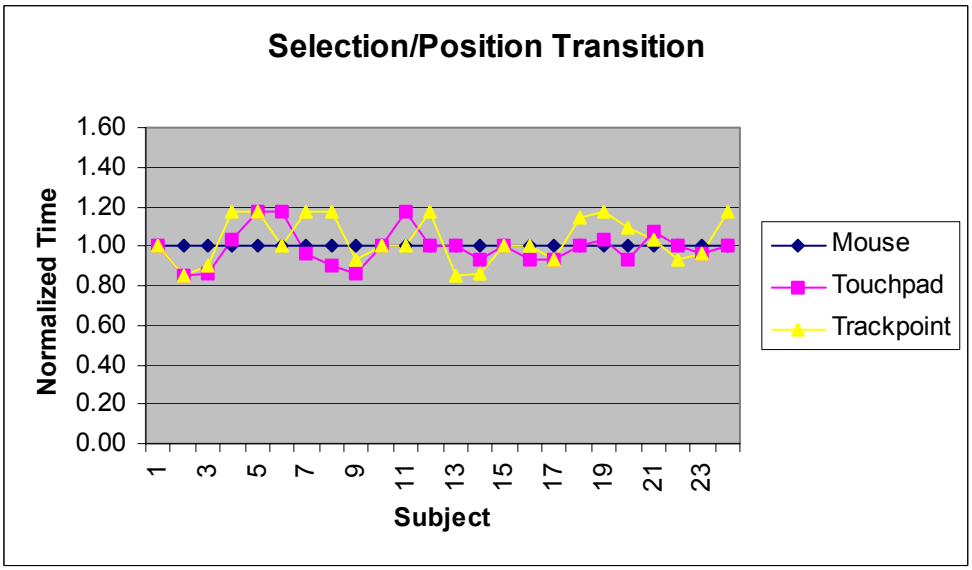


Figure 5.48: Normalized performance time (ms.) for S/P transition (mouse=1.0)

For the DM/S and P/S transitions (figures 5.49-50), we see a trend of both greater magnitudes of difference and the mouse's clear superiority, while there is no clear trend between the trackpoint and touchpad. While the task completion times typically didn't vary by more than a factor of two, we see that for the DM/S and P/S transitions, the range is from two to six times slower than the mouse.

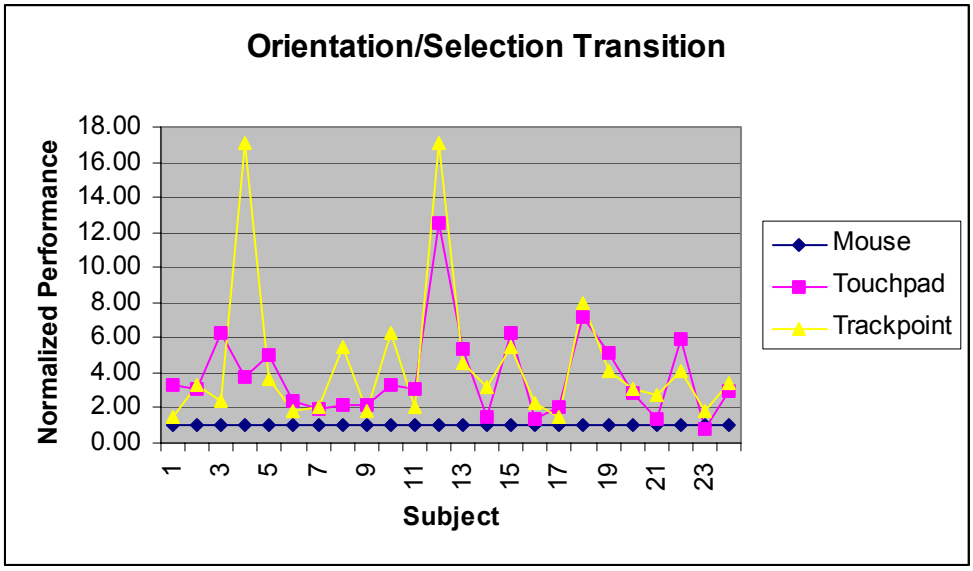


Figure 5.49: Normalized performance time (ms.) for DM/S transition (mouse=1.0)

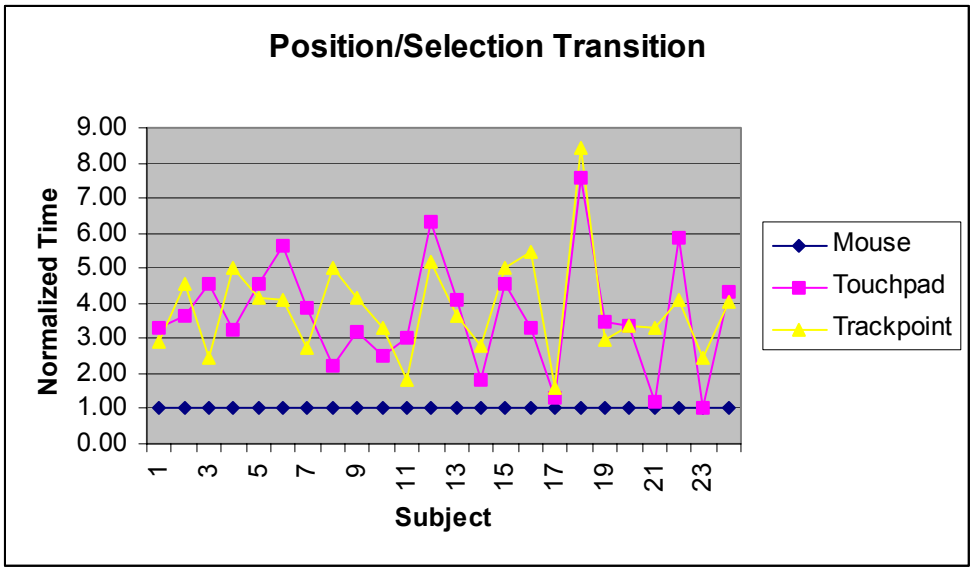


Figure 5.50: Normalized performance time (ms.) for P/S transition (mouse=1.0)

5.6 Aggregate Error Rates

In our simulation, an error is defined solely as clicking on a control in the wrong task ordering or a non-selectable region of the screen. We did not try to take user intent into account because of the inherent difficulty of determining that intent. In table 5.5 we provide a summary of the number of trials of the curve matching task that the user completed and the number of selection errors that they committed. Bear in mind that these errors are reported on a per trial basis when there can be a minimum of 4 selections in a successful trial. On a per trial basis, there were .385, .455 and .449 errors per trial with the mouse, touchpad and trackpoint respectively. Looking at the error rate on a per task basis we get .084, .111 and .108 errors per selection with the mouse, touchpad and trackpoint respectively. At first glance, these error rates look rather high but keep in mind our

	Mouse		Touchpad		Trackpoint	
Subject	Trials	Errors	Trials	Errors	Trials	Errors
1	75	15	75	41	73	57
2	75	10	75	17	75	26
3	75	41	72	48	74	51
4	75	19	74	40	73	45
5	73	25	74	27	75	19
6	75	25	75	42	74	28
7	73	24	73	12	75	19
8	75	17	74	18	75	35
9	75	40	74	20	75	40
10	75	51	75	43	71	54
11	75	16	74	52	73	40
12	75	20	72	16	73	29
13	75	10	71	67	68	24
14	75	48	74	40	75	23
15	75	31	74	26	73	29
16	75	43	75	35	71	48
17	75	34	75	30	75	15
18	75	19	75	19	75	28
19	74	51	75	34	75	17
20	75	16	73	24	71	20
21	75	10	73	21	73	27
22	74	52	74	48	72	42
23	75	23	73	25	71	37
24	74	51	75	63	74	36

Table 5.5: Number of trials and selection errors by device

definition of an error. More than half of the errors we logged were simply participants attempting to activate a control out of order, not missing the controls as one might define an error.

While it is beyond the scope of our work, an interesting question was raised. How "well" did the participants perform the curve-matching task? Defining the quality of a curve match is a subjective judgment. There are many different approaches that can be taken, from simply measuring the linear and angular discrepancies to overlaying the two curves and determining the area between them. We approached this question by coming up with three simple characteristics for the curve: center, angle and points. Center is defined as the distance (in pixels) between the reference curve center and the user curve center with respect to the origin of the region of the screen that they appear in. The center measure corresponds to the positioning of the entire curve. Angle is the angle between the line segments formed by the reference curve endpoints and the user curve endpoints. There are two possible angles that can be calculated; we calculate both and select the smallest. The angle measure corresponds to the orientation task. Finally, the points measure is derived by translating the user curve so that the user curve center overlays the reference curve center and then calculating the distance between the reference curve interior control points and the user curve interior control points in a pair-wise fashion. Since this can be done in two different ways (R1 to U1 and R2 to U2 as well as R1 to U2 and R2 to U1), we compute both distances and select the minimum. This methodology was not meant to be a definitive treatment of curve matching quality. It merely provided us with a little insight into the

problem. We observed a median center value of 48.0 pixels, median angle of .787 radians and median point value of 84.67 pixels.

In appendix 8.1 we provide the median data by subject for those interested in this problem.

It should also be noted that while we can't guarantee that all users were operating at expert level performance in all devices tested, we did check median performance time across trial blocks two through four and did not find a significant difference in performance times.

6. Discussion

This thesis makes several important contributions to the body of HCI literature. Methodological contributions have been made in the form of an extensible generic simulation testbed which permits an efficient evaluation of user performance. The need for the consideration of individual performance characteristics is strongly indicated based on the results we have obtained. A number of theoretical contributions are made, specifically: a new taxonomy of tasks has been derived which can be directly used for evaluation, task execution graphs have been introduced which permit the representation of realistic tasks and corresponding performance metrics, and a methodology for representing device differences within this framework. Finally we provide some insight into potentially rewarding areas that this work can be applied to.

6.1 Methodological Contributions

One of the primary focuses of this work is to provide a generic test bed for the evaluation of devices. A testbed was created that allows an experimenter to evaluate multiple pairings of devices with an extensible set of tasks that might be performed in a given interface. A natural follow-on, once these results are obtained, is verification in the deliverable environment.

6.1.1 Implications of user differences to generality of device testing

The results point to a deficiency in previous device comparison studies. Prior multi-device studies have looked at the differences between devices across a user population. While this approach yields a convenient general trend, it does not allow us to see that individual differences in performance do not fit so neatly into the between-subject approach. While all of our users were able to perform most tasks and transitions faster and with fewer errors using a mouse, the differences between the trackpoint and touchpad were not as clear-cut. As we saw in section 5.5, there was quite a bit of variability in the relative performance characteristics of the devices. These results would indicate that a one-size-fits-all approach to the reporting of performance data inaccurately characterizes the performance of a substantial percentage of the population. It is clear from the results that individual users may perform substantially better with a particular device than the general population. What is not clear is why this is the case. One possible explanation is that users are not pursuing an optimal strategy of device usage, having developed an approach to using a particular device that while it appears to be intuitively optimal may actually not be [Andre 1995].

A look at the mean transition times of subjects in figure 5.1 indicates the wide range of performance characteristics between users for the DM/S and P/S transitions. It is convenient to look at one simple indication of performance over a large group, but there is clearly a large spread between the performance of individual users.

While it may be useful to have an overall measure of the performance characteristics of a device, the differences in individual performance demonstrate the need for calibrating a model to an individual's performance data to yield higher quality predictive results.

6.2 Theoretical Contributions

A task taxonomy was created by decomposing common user tasks in a graphical user interface into a set of primitive tasks that appear to be able to adequately describe any conventional interaction in a graphical user interface. While many studies have been previously undertaken to look at the performance characteristics of elemental actions, our contribution is to look at the combinations of primitive tasks and the time it takes for a user to transition from one to another in an ecologically valid environment.

6.2.1 Task Execution Graphs

A novel method for the depiction of tasks was developed in the form of task execution graphs. This depiction of elemental tasks and the transitions between them allows one to structure realistic tasks in a simple but meaningful way. Our model is composed of three actions or tasks, and four transitions that connect the actions. Based on our data analysis, each of the actions was modeled as a Fitts' Law task while the transitions were modeled as empirically derived constants. A model for a single subject on a single device therefore consists of a set of eight coefficients or weights for the graph: 4 transitions, 3 tasks, and a selection error rate. A model must be generated for each device used by each subject. To

derive a prediction for performance time of a task, one simply traces a shortest path through the state graph and sums the weights of each node and transition in the path.

6.2.2 Creating and Upgrading Models

One of the purposes of this work is to give researchers the ability to easily compare device performance characteristics over a variety of environments. With this goal in mind, it is necessary to be able to do the following three things with respect to modeling:

1. Generating models for new users: a procedure is outlined for generating the models that will enable a comparison of the user's performance across devices.

2. Extending a given model by the addition of a new task.

3. Extending a set of models with the addition of a new device.

A performance model can be generated for a new user by running the user on the experimental test bed, collecting the relevant data as outlined in the methods section, and then analyzing and summarizing the data according to our data analysis procedure. (The code appears in Appendices 8.4-8.7). A set of performance parameters will be generated that can then be attached to a model diagram. This procedure would be repeated for each device of interest.

A performance model can be generated for a new device by simply following the procedure above, assuming that performance data for the subject on the other comparators is already

available. It is important to make sure that the subject has had sufficient experience with the device before data collection begins.

To extend a model with a new task it is necessary to modify the experimental test bed (unless the addition is simply a redefinition of the current primitives) to add the additional task, modifying the states and order of presentation appropriately. The data analysis software must be updated as well to encompass the new task definition and timing requirements. Then, models can be generated as for a new user.

6.2.3 Limitation of model

To lend some ecological validity to the experiment, the study used a curve-matching task, which required judgment on the part of the participant and involved some uncertainty or qualitative judgment rather than a more simplistic set of tasks like simple target acquisition and line orientation. Our model, like Fitts' Law, does not take into account intervals where a user is involved with cognitive activities like goal formation. We structured the timing of user events to log when a user started and completed an action. This eliminated most exploratory behaviors (activities where the goal was not immediately clear to the user). We therefore assumed that any user action taking longer than six seconds was non-goal directed and removed it from consideration.

Fitts' Law was chosen as the dominant means of expressing the predicted times for tasks because of its dominance in the literature and its ability to express the relationship between performance and distance so well. While there are several variations of Fitts' Law, we selected a very general form that is well regarded in the literature, $T= I \log_2(2D/W)$, where

T is time, I is the index of difficulty, D is the distance traveled and W is the width of the target. Other variations of Fitts' Law have been shown to provide nominally better predictive accuracy over short distances but we use only the general form to enhance the generality and applicability of our findings across a wide range of situations. [Card 1983, MacKenzie 1991a]

While there may be multiple or even infinite acceptable paths through the model graph to complete a task, this work assumes that a goal directed path (where each action leads closer to a goal) will be taken.

6.3 Future Work

A number of compelling areas for future research are opened by this work. A more standardized and effective mode of device evaluation can be pursued. In complex application domains, critical sections of the application can be analyzed for appropriateness of device selection with good generality for performance in the whole application. The methodology described herein could be used to create a recommendation system for device appropriateness for a given user, based on monitoring actual user performance. Finally, in areas where individual performance varies greatly, our techniques will be helpful in evaluating assistive technologies.

6.3.1 Evaluation of novel input devices

By adopting the environment and representations of user actions described herein, it should be possible to advance the state of device evaluation by creating a set of consistent and

coherent procedures across the field. This would allow for more effective testing of devices and the ability to generalize across studies if all other variables were kept constant. A common set of procedures would minimize the occurrence of methodological flaws that can arise in this type of research.

6.3.2 Analysis of Time/Error critical regions of applications

In complex application domains, it may not be possible to apply our approach to device evaluation. It may, however, be quite feasible to profile the application and look at the most frequently used bottlenecks of user interaction, where a bottleneck might be defined as a time *intensive* or error *intensive* area of the application. Time or error *critical* areas of the application could be studied as well.

6.3.3 Device recommendation for specific users

Using our approach, it should be possible to optimize a particular user's performance given a specific set of tasks and set of devices. Given that a user might perform different tasks more efficiently with different devices, it is essential to have a good understanding of what a representative workload might be. The user would be tested on the task set with each of the devices or simple logging software could be installed that observed what the user was doing as they went about their normal workflow. Performance metrics would be derived from the tests and then a recommendation could be made. In emerging areas of computational applications like mobile computing, these recommendations might be particularly helpful in understanding these more demanding interaction environments.

6.3.4 Analysis of disabled user's performance

As computers become more ubiquitous, it becomes increasingly more important to assess how disabled users make use of different input devices. Our approach can allow a researcher to analyze the performance characteristics of a particular type of disability and make appropriate recommendations for device usage.

Within a specific disability it is likely that the individual differences that we observed in our study will be even more enhanced, illustrating that a one size fits all approach simply won't work when attempting to tailor computational devices to disabled users. Our model could enable an approach where computational devices could be customized by way of input devices to the differently-abled user.

7. References

Accot, J. & Zhai, S. (1997). Beyond Fitts' Law: Models for Trajectory-Based HCI Tasks. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems,* p.295-302.

Accot, J. & Zhai, S. (2001). Scale Effects in Sterring Law Tasks. In *Proceedings of ACM CHI 2001 Conference on Human Factors in Computing Systems,* p.1-8.

Albert, A.E. (1982). The effect of graphic input devices on performance in a cursor positioning task, In *Proceedings of the Human Factors Society - 26th annual meeting*, **26**, p.54-58.

Anderson, J.R., Matessa, M. & Lebiere, C. (1997). ACT-R: A Theory of Higher Level Cognition and Its Relation to Visual Attention. *Human Computer Interaction*, **12**(4), p. 439-462

Andre, A.D. & Wickens, C.D. (1995). When Users Want What's Not Best For Them. *Ergonomics in Design*, **3**(10), p.10-13.

Baber, C. (1997). *Beyond the desktop, designing and using interaction devices*. Academic Press, New York.

Barham, P. & McAllister, D.F. (1991). A Comparison of Stereoscopic Cursors for the Interactive Manipulation of B-Splines. *SPIE Proceedings, Stereoscopic Displays and Applications II*, **1457**, p. 18-26

Buxton, W. (1983). Lexical and pragmatic considerations of input structures. *Computer Graphics*, **17**, p.31-37

Buxton, W. (1986). There's more to interaction than meets the eye: some issues in manual input. In D.A. Norman and S.W. Draper (Eds), *User Centred System Design*. LEA, Hillsdale, N.J.

Buxton, W. (1990). A three state model of graphical input. In *Proceedings of INTERACT '90*, p. 449-456.

Card, S.K., English, W.K. & Burr, B.J. (1978). Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT. *Ergonomics*, **21**(8), p.601-613.

Card, S.K., Moran, T., & Newell, A. (1980). The keystroke level model for user performance time with interactive systems. *Communications of the ACM*, **23**, p.396-410.

Card, S.K., Moran, T., & Newell, A. (1983). *The Psychology of Human Computer Interaction*. LEA, Hillsdale, N.J.

Card, S.K., MacKinlay, J.D., & Robertson, G.G. (1991). A morphological analysis of the design space of input devices. *ACM Transactions on Information Systems*, **9**, p.99-122.

Card, S.K., MacKinlay, J.D., & Robertson, G.G. (1992). The design space of input devices. In M.M. Blattner and R.B. Dannenberg (Eds), *Multimedia Interface Design*. New York, ACM Press.

Dix, A.J., Finlay, J.E., Abowd, G.D. & Beale, R. (1998). *Human-Computer Interaction*. Prentice Hall, New York.

Douglas, S.A., & Mithal, A.K. (1994). The Effect of Reducing Homing Time on the Speed of a Finger-Controlled Isometric Pointing Device. In *Proceedings of ACM CHI '94*, p.411-416

Douglas, S.A., Kirkpatrick, A.E. & MacKenzie, I.S. (1999). Testing Pointing Device Performance and User Assessment with the ISO 9241, Part 9 Standard. In *Proceedings of ACM CHI '99 Conference on Human Factors in Computing Systems*, p.215-222

Dulberg, M.S., St. Amant, R. & Zettlemoyer, L. (1999). An Imprecise Mouse Gesture for the Fast Activation of Controls. In *Proceedings of INTERACT '99,* p.375-382

Epps, B.W. (1986). Comparison of six cursor control devices based on Fitts' law models. In *Proceedings of the Human Factors Society - 30th annual meeting*, 30, p.327-331.

Fitts, P.M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, **47**, p.381-391.

Foley, J.D., Wallace, V.L., & Chan, P. (1984). The human factors of graphics interaction techniques. *IEEE Computer Graphics and Applications*, **4**(11), p.13-48.

Graham, E.D. & MacKenzie, C.L. (1996). Physical versus virtual pointing. In P*roceedings on Human factors in computing systems*, p.292-299

Guimbretière,F. & Winograd, T. (2000). FlowMenu: Combining Command, Text, and Data Entry Selection. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, p.213-216.

Hornof, A.J. & Kieras, D.E. (1999). Cognitive modeling demonstrates how people use anticipated location of menu items. In *Proceedings of CHI '99, The CHI is the Limit*, p.410-417.

ISO (1999). ISO/DIS 9241-9 Ergonomic Requirements for Office Work with Visual Display Terminals, Nonkeyboard Input Device Requirements, Draft International Standard, International Organization for Standardization.

Jagacinski, R.J. & Monk, D.L. (1985). Fitts' Law in Two Dimensions with Hand and Head Movements. *Journal of Motor Behavior*, 17(1), 77-95.

Kieras, D. & Polson, P.G. (1986). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, **22**, p.365-394

Knowles, C. (1988). Can CCT produce a measure of system usability? In D.M. Jones and R. Winder (eds) *People and Computers IV*. Cambridge University Press, Cambridge.

Kurtenbach, G. & Buxton, W. (1993). The limits of expert performance using hierarchic marking menus. In *Proceedings of ACM INTERCHI '93*, p.482-487.

Langolf, G.D., Chaffin, D.B., & Foulke, J.A. (1976). An investigation of Fitts' Law using a wide range of movement amplitudes. *Journal of Motor Behavior*, **8**, 113-128.

MacKenzie, I.S., Sellen, A. & Buxton, W. (1991). A comparison of input devices in elemental pointing and dragging tasks. In *Proceedings of ACM CHI '91 Conference on Human Factors in Computing Systems,* p.161-166.

MacKenzie, I.S. (1991a). Fitts' law as a performance model in human-computer interaction. Doctoral dissertation, University of Toronto

MacKenzie, I.S., Buxton, W. (1992). Extending Fitts' Law to Two-Dimensional Tasks *Proceedings of ACM CHI '92 Conference on Human Factors in Computing Systems,* p.219-226

MacKenzie, I.S., Kauppinen, T. & Silfverberg, M. (2001). Accuracy Measures for Evaluating Computer Pointing Devices Human Performance Points. In *Proceedings of ACM CHI 2001 Conference on Human Factors in Computing Systems*, p.9-16

Meyer, D.E., Abrams, R.A., Kornblum, S., Wright, C.E., and Smith, J.E.K. (1988). Optimality in human motor performance: Ideal control of rapid aimed movements. *Psychological Review*, **95**, 340-370.

Mithal, A.K. & Douglas, S.A. (1996) Differences in movement microstructure of the mouse and the finger-controlled isometric joystick. *In Proceedings of CHI '96*, p.300-307.

Murata, Atsuo (1991). An experimental evaluation of mouse, joystick, joycard, lightpen, trackball and touchscreen for pointing - Basic study on human interface design. In H.J. Bullinger (Ed.), *Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals*, 123-127, Elsevier: B.V.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.

Norman, D.A. & Draper, S.W. (1986). *User Centered System Design: New Perspectives on Human-Computer Interaction,* Lawrence Erlbaum Associates, Hillsdale, NJ.

Olsen, J.R., & Olson, G.M. (1990) The growth in cognitive modeling in human-computer interaction since GOMS. *Human Computer Interaction*, **5**, p.221-265.

Poupyrev, I., Billinghurst, M, Weghorst, S. & Ichikawa, T. (1996). The go-go interaction technique: non-linear mapping for direct manipulation in VR. *Proceedings of the ACM symposium on User interface software and technology,* p.79-80.

Rudnicky, A.I., & Hauptmann, A.G. (1991). Models for evaluating interaction protocols in speech recognition. *In Proceedings of CHI '91*, p.285-291.

Schmitt, A. & Oel, P. (1999). Calculation of totally optimized button configurations using fitts' law. In *Proceedings of the Eighth International Conference on Human-Computer Interaction*, **1**, p.392-396

Shneiderman, B. (1983). Direct manipulation: a step beyond programming languages. *IEEE Computer*, **16**, p.57-69.

Simpson, C.A. & St. Amant, R. (2003). Search for Efficient Device-Dependent Action Sequences in the User Interface. *In Proceedings of IUI*. To appear.

Tognazzini, B. (1999). A Quiz Designed to Give You Fitts. *On the World Wide Web*: http://www.asktog.com/columns/022DesignedToGiveFitts.html

van de Pol, R. Ribarsky, W., Hodges, L., & Post, F. (1998). Interaction in Semi-Immersive Large Display Environments. *GVU Technical Report* 98-30.

Walker, N., Meyer, D.E., & Smelcer, J.B. (1993). Spatial and temporal characteristics of rapid cursor-positioning movements with electromechanical mice in human-computer interaction. *Human Factors*, **35**, p.431-458.

Worden, A., Walker, N., Bharat, K. and Hudson, S. (1997). Making Computers Easier for Older Adults to Use: Area Cursors and Sticky Icons. *In Proceedings of the ACM Conference on Computer Human Interaction*, March 1997, pp. 266-271.

Appendices

8.1 Median trial completion times and curve error factors

Subject	N	Time (ms)	Center (pixels)	Angle (radians)	Points (pixels)
1	75	21615	53.1507	0.812059	119.351
2	75	23931	39.6232	0.772066	64.884
3	75	16409	53.1413	0.805395	72.3506
4	75	21487	47.1699	0.71883	80.3083
5	73	27236	48.7955	0.815389	119.276
6	75	21024	42.72	0.748748	65.9433
7	73	24925	50.0899	0.574305	84.7159
8	75	17938	39.9625	0.902312	79.1206
9	75	17915	53.6004	0.691713	68.7549
10	75	18931	46.8615	0.745419	104.569
11	75	20655	58.8218	0.882086	61.3799
12	75	22168	48.2597	0.782779	81.6827
13	75	16553	43.382	0.798731	93.2998
14	75	20368	41.8808	0.738765	78.5296
15	75	19699	38.833	0.785398	84.7859
16	75	13154	47.634	0.738765	86.133
17	75	16818	52.1536	0.72547	85.3799
18	75	19845	46.4004	0.785398	84.8539
19	74	21115	40.66175	0.768734	72.29325
20	75	14217	45.1774	0.91223	83.536
21	75	14554	51.6624	0.872137	102.047
22	74	24357	66.6286	0.97526	85.0003
23	75	13648	45.2217	0.73544	73.0472
24	74	25588	50.2352	0.798997	100.7346

8.2 Simulation code: Form1.frm (Visual Basic)

```
VERSION 5.00
Begin VB.Form Form1
   BackColor       =   &H00FFFFFF&
   BorderStyle     =   0  'None
   Caption         =   "Form1"
   ClientHeight    =   11520
   ClientLeft      =   0
   ClientTop       =   0
   ClientWidth     =   15360
   LinkTopic       =   "Form1"
   Moveable        =   0   'False
   ScaleHeight     =   768
   ScaleMode       =   3  'Pixel
   ScaleWidth      =   1024
   ShowInTaskbar   =   0   'False
   Begin VB.Timer Timer3
      Interval        =   55
      Left            =   1440
      Top             =   0
   End
   Begin VB.Timer Timer2
      Interval        =   1000
      Left            =   720
      Top             =   0
   End
   Begin VB.CommandButton Command1
      Caption         =   "Next"
      Height          =   615
      Left            =   5400
      TabIndex        =   0
      Top             =   9360
      Width           =   1215
   End
   Begin VB.Timer Timer1
      Interval        =   55
      Left            =   120
      Top             =   0
   End
   Begin VB.Shape Shape3
      BorderColor     =   &H00404040&
      FillColor       =   &H00404040&
      FillStyle       =   0  'Solid
      Height          =   300
      Left            =   11370
      Top             =   3690
      Width           =   300
   End
   Begin VB.Shape box2
      FillColor       =   &H00FFFFFF&
      Height          =   345
      Left            =   12720
      Top             =   1200
      Width           =   345
   End
   Begin VB.Shape box1
      FillColor       =   &H00FFFFFF&
```

```
   Height          =   345
   Left            =   10080
   Top             =   1320
   Width           =   345
End
Begin VB.Shape Shape2
   BackColor       =   &H00FFFFFF&
   BorderColor     =   &H00FFFFFF&
   FillColor       =   &H00FFFFFF&
   Height          =   3375
   Left            =   9840
   Top             =   8040
   Width           =   3735
End
Begin VB.Line Line1
   BorderColor     =   &H00C0C0C0&
   BorderWidth     =   15
   X1              =   776
   X2              =   776
   Y1              =   650
   Y2              =   550
End
Begin VB.Shape Shape7
   BorderColor     =   &H00000000&
   FillColor       =   &H00C00000&
   FillStyle       =   0  'Solid
   Height          =   1515
   Left            =   10890
   Shape           =   3  'Circle
   Top             =   9000
   Width           =   1515
End
Begin VB.Shape Shape6
   BorderColor     =   &H00C00000&
   BorderWidth     =   3
   FillColor       =   &H00C00000&
   Height          =   3015
   Left            =   10140
   Shape           =   3  'Circle
   Top             =   8250
   Width           =   3015
End
Begin VB.Shape Shape5
   FillStyle       =   0  'Solid
   Height          =   195
   Left            =   0
   Shape           =   3  'Circle
   Top             =   1080
   Width           =   195
End
Begin VB.Shape Shape4
   BackStyle       =   1  'Opaque
   FillStyle       =   0  'Solid
   Height          =   195
   Left            =   0
   Shape           =   3  'Circle
   Top             =   720
   Width           =   195
End
```

```
Begin VB.Label Label2
   BackColor       =   &H8000000E&
   Caption         =   "Time Left"
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   24
      Charset         =   0
      Weight          =   400
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   615
   Left            =   840
   TabIndex        =   2
   Top             =   9360
   Width           =   2415
End
Begin VB.Label Label1
   BackColor       =   &H8000000E&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   24
      Charset         =   0
      Weight          =   400
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   615
   Left            =   3360
   TabIndex        =   1
   Top             =   9360
   Width           =   1815
End
Begin VB.Shape Shape1
   BackStyle       =   1  'Opaque
   BorderColor     =   &H00C00000&
   FillColor       =   &H00C00000&
   FillStyle       =   0  'Solid
   Height          =   255
   Index           =   3
   Left            =   14400
   Shape           =   3  'Circle
   Top             =   3690
   Width           =   255
End
Begin VB.Shape Shape1
   BackStyle       =   1  'Opaque
   BorderColor     =   &H00808080&
   FillColor       =   &H00808080&
   FillStyle       =   0  'Solid
   Height          =   255
   Index           =   2
   Left            =   12390
   Shape           =   3  'Circle
   Top             =   3690
   Width           =   255
End
```

```
   Begin VB.Shape Shape1
      BackStyle       =   1  'Opaque
      BorderColor     =   &H00808080&
      FillColor       =   &H00808080&
      FillStyle       =   0  'Solid
      Height          =   255
      Index           =   1
      Left            =   10395
      Shape           =   3  'Circle
      Top             =   3690
      Width           =   255
   End
   Begin VB.Shape Shape1
      BackStyle       =   1  'Opaque
      BorderColor     =   &H00C00000&
      FillColor       =   &H00C00000&
      FillStyle       =   0  'Solid
      Height          =   255
      Index           =   0
      Left            =   8400
      Shape           =   3  'Circle
      Top             =   3690
      Width           =   255
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
Dim shapeindex As Integer
Dim INTERVAL As Integer
Dim trial As Integer
Dim MAXTRIAL As Integer
Dim time As Integer

Dim state As Integer

Private Sub Command1_Click()
Timer1.Enabled = True
time = INTERVAL
End Sub


Private Sub Form_Load()
DrawWidth = 3
state = 0
filenum = FreeFile
Open "c:\test.txt" For Output As filenum
fileiter = FreeFile
Open "c:\trials.txt" For Output As fileiter
'order(1)= dial
'order(2)= points
'order(3)= position
setorder
usercenter.X = 768
usercenter.y = 256
referencecenter.X = 256 'replace w/random nums!
```

```
referencecenter.y = 256
trial = 0
MAXTRIAL = 25
Randomize  'remove argument to get system seed
shapeindex = -1
radius = 9
INTERVAL = 30 ' MAX LENGTH OF TRIAL SET HERE
time = INTERVAL
Dialradius = 100
Dialcenter.X = 776
Dialcenter.y = 650
'setboxes
R1.X = 106
R1.y = 256
R4.X = 406
R4.y = 256
'need to set P's to something so that they dont intefere
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, y As
Single)
Dim counter As Single
Dim dist As Double
Dim cursor As VECTOR
Dim previous As VECTOR
shapeindex = isinside(X, y)
message = "X" 'assume error, correct below
   If shapeindex > -1 Then  'move point
      If OrderCheck(order(2)) = 1 Then
         Shape1(shapeindex).Left = X - 6
         Shape1(shapeindex).Top = y - 6
         DrawBez P1, P2, P3, P4, 15
         LoadPoints P1, P2, P3, P4
         DrawBez P1, P2, P3, P4, 2
         For counter = 0 To 3
            Shape1(counter).Refresh
         Next
         Shape3.Refresh
         message = "S" & shapeindex
         refreshbox
      End If
   ElseIf shapeindex = -3 Then  ' move whole user figure
      If (OrderCheck(order(3)) = 1) Then
         moveusercurve X, y
         message = "SM"
      End If
   Else
      shapeindex = -2
      previous.X = CDbl(Line1.X2)
      previous.y = Line1.Y2
      vectsub2 previous, Dialcenter, previous
      vectunit2 previous, previous
      cursor.X = X
      cursor.y = y
      dist = Distance(cursor, Dialcenter)
      If dist < Dialradius And dist > 2 Then  'move needle to new point
         If (OrderCheck(order(1)) = 1) Then
            movedial previous, cursor
            message = "SD"
```

```
         End If
      End If
End If
Logdata trial
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, y As
Single)
Dim counter
Dim dist As Double
Dim magnitude As Double
Dim cursor As VECTOR
Dim previous As VECTOR
'0-3 for points
'-2 for dial
'-3 for positioner
If shapeindex > -1 Then
   If (OrderCheck(order(2)) = 1) Then
      message = ""
      Shape1(shapeindex).Left = X - 6
      Shape1(shapeindex).Top = y - 6
      DrawBez P1, P2, P3, P4, 15
      For counter = 0 To 3
         Shape1(counter).Refresh
      Next
      Shape3.Refresh
      refreshbox
      LoadPoints P1, P2, P3, P4
      DrawBez P1, P2, P3, P4, 2
   End If
ElseIf shapeindex = -3 Then
   If (OrderCheck(order(3)) = 1) Then
      ' move whole user figure
      moveusercurve X, y
      message = ""
   End If
ElseIf shapeindex = -2 Then
   message = "" 'NOT COUNTING AS ERROR IF YOU MOVE OUT
   refreshdial
   previous.X = CDbl(Line1.X2)
   previous.y = Line1.Y2
   vectsub2 previous, Dialcenter, previous
   vectunit2 previous, previous
   cursor.X = X
   cursor.y = y
   dist = Distance(cursor, Dialcenter)
   If dist < Dialradius And dist > 2 Then  'move needle to new point
      If (OrderCheck(order(1)) = 1) Then
         movedial previous, cursor
      End If
   End If
End If
Logdata trial
End Sub

Private Sub Form_MouseUp(Button As Integer, Shift As Integer, X As Single, y As
Single)
shapeindex = -1
message = "ED"
```

```
Logdata trial
message = ""
End Sub

Private Sub Timer1_Timer()
Dim randAngle As Double
randAngle = 0
Line (512, 0)-(512, 512), QBColor(1)
Line (0, 512)-(1024, 512), QBColor(1)
DrawBez R1, R2, R3, R4, 15 ' erase previous reference curve
Randompoints R2, R3 'get new midpoints
R1.X = 106
R1.y = 256
R4.X = 406
R4.y = 256

DrawBez P1, P2, P3, P4, 15 'erase previous user curve
resetuser 'reset position of user shapes
LoadPoints P1, P2, P3, P4 'reset user curve
DrawBez P1, P2, P3, P4, 2
Line1.X2 = 776
Line1.Y2 = 556


setboxes ' puts boxes in correct unrotated position
randAngle = (2 * 3.14127 * Rnd)

rotatereference R1, randAngle
rotatereference R2, randAngle
rotatereference R3, randAngle
rotatereference R4, randAngle

'Take out for debug
RandomRefCenter referencecenter

TranslateRef referencecenter, R1
TranslateRef referencecenter, R2
TranslateRef referencecenter, R3
TranslateRef referencecenter, R4

DrawBez R1, R2, R3, R4, 1 'draw new reference curve
Shape4.Left = R2.X - 7
Shape4.Top = R2.y - 7
Shape5.Left = R3.X - 7
Shape5.Top = R3.y - 7

trial = trial + 1
setorder
If trial > MAXTRIAL Then
   End
End If
state = 0
message = ""
Write #fileiter, trial, R1.X, R1.y, R2.X, R2.y, R3.X, R3.y, R4.X, R4.y, P1.X,
P1.y, P2.X, P2.y, P3.X, P3.y, P4.X, P4.y, Line1.X1, Line1.Y1, Line1.X2, Line1.Y2,
referencecenter.X, referencecenter.y, order(1), order(2), order(3)

Timer1.Enabled = False
End Sub
```

```
Private Sub Timer2_Timer()
time = time - 1
Label1.Caption = time
If time = 0 Then
   time = INTERVAL
   shapeindex = -1 ' stop current drag if one is occuring
   Timer1.Enabled = True
End If
End Sub

Private Sub Timer3_Timer()
Logdata trial
End Sub
```

8.3 Simulation code: Modheaderbas.bas (Visual Basic)

```
Attribute VB_Name = "Module2"
Option Explicit
Type POINTAPI
        X As Long
        y As Long
End Type

'added for timer
Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Declare Function SetCursorPos Lib "user32" (ByVal X As Long, ByVal y As Long) As
Long
Declare Function GetTickCount Lib "kernel32" () As Long

Public dl&
Public filenum As Integer
Public fileiter As Integer
Public message As String
Public cursorspositionapi As POINTAPI
Public cursorsposition As POINTAPI
Public prevcursorsposition As POINTAPI
Public pull As VECTOR
Public state As Integer
Public P1 As POINTAPI 'user points
Public P2 As POINTAPI
Public P3 As POINTAPI
Public P4 As POINTAPI
Public R1 As POINTAPI 'reference points
Public R2 As POINTAPI
Public R3 As POINTAPI
Public R4 As POINTAPI
Public Dialcenter As VECTOR
Public usercenter As VECTOR
Public referencecenter As VECTOR
Public Dialradius As Double
Public radius As Integer
Public t As Double
Public currpt As POINTAPI
Public prevpt As POINTAPI
Public refreshcounter As Integer
Public order(3) As Integer
'order 1=dial, order 2=points, order 3=position

Public Function LoadPoints(P1 As POINTAPI, P2 As POINTAPI, P3 As POINTAPI, P4 As
POINTAPI)
P1.X = Form1.Shape1(0).Left + radius
P1.y = Form1.Shape1(0).Top + radius
P2.X = Form1.Shape1(1).Left + radius
P2.y = Form1.Shape1(1).Top + radius
P3.X = Form1.Shape1(2).Left + radius
P3.y = Form1.Shape1(2).Top + radius
P4.X = Form1.Shape1(3).Left + radius
P4.y = Form1.Shape1(3).Top + radius
End Function

Public Function isinside(X As Single, y As Single) As Integer
'returns -1 if not inside, else returns index of point
```

```
'returns -2 if dial ???
'returns -3 if shape3 (positioner)
Dim center As VECTOR
Dim current As VECTOR
Dim cursordist As Double
isinside = -1
current.X = X
current.y = y
center.X = Form1.Shape1(1).Left + radius
center.y = Form1.Shape1(1).Top + radius
cursordist = Distance(current, center)
If cursordist <= radius Then
   isinside = 1
End If
center.X = Form1.Shape1(2).Left + radius
center.y = Form1.Shape1(2).Top + radius
cursordist = Distance(current, center)
If cursordist <= radius Then
   isinside = 2
End If
center.X = Form1.Shape3.Left + 10
center.y = Form1.Shape3.Top + 10
cursordist = Distance(current, center)
If cursordist <= 12 Then
   isinside = -3
End If
End Function

Public Sub Randompoints(a As POINTAPI, b As POINTAPI)
  Dim randAngle As Double
  Dim randDistance As Double
  randAngle = (2 * 3.14127 * Rnd)
  randDistance = 150 * Rnd
  a.y = CInt(Sin(randAngle) * randDistance) + 256
  a.X = CInt(Cos(randAngle) * randDistance) + 256
  randAngle = (2 * 3.14127 * Rnd)
  randDistance = 150 * Rnd
  b.y = CInt(Sin(randAngle) * randDistance) + 256
  b.X = CInt(Cos(randAngle) * randDistance) + 256
End Sub
Public Sub resetuser()
   Form1.Shape1(0).Top = 247
   Form1.Shape1(1).Top = 247
   Form1.Shape1(2).Top = 247
   Form1.Shape1(3).Top = 247
   Form1.Shape1(0).Left = 611
   Form1.Shape1(1).Left = 719
   Form1.Shape1(2).Left = 802
   Form1.Shape1(3).Left = 911
   usercenter.X = 768
   usercenter.y = 256
   Form1.Shape3.Left = usercenter.X - 9
   Form1.Shape3.Top = usercenter.y - 9

End Sub
Public Sub rotatereference(a As POINTAPI, theta As Double)
Dim newpoint As VECTOR
Dim newpoint2 As VECTOR
'translate to origin
```

```
newpoint.X = a.X - referencecenter.X
newpoint.y = a.y - referencecenter.y
'rotate
newpoint2.X = newpoint.X * Cos(theta) - newpoint.y * Sin(theta)
newpoint2.y = newpoint.X * Sin(theta) + newpoint.y * Cos(theta)
a.X = newpoint2.X + referencecenter.X
a.y = newpoint2.y + referencecenter.y
End Sub

Public Sub rotatepoint(a As Shape, c As VECTOR, theta As Double)
Dim newpoint As VECTOR
Dim newpoint2 As VECTOR
'translate to origin
newpoint.X = a.Left + radius - c.X
newpoint.y = a.Top + radius - c.y
'rotate
newpoint2.X = newpoint.X * Cos(theta) - newpoint.y * Sin(theta)
newpoint2.y = newpoint.X * Sin(theta) + newpoint.y * Cos(theta)
'translate back
a.Left = newpoint2.X + c.X - radius
a.Top = newpoint2.y + c.y - radius
End Sub
Public Sub rotatebox(a As Shape, c As VECTOR, theta As Double)
Dim newpoint As VECTOR
Dim newpoint2 As VECTOR
'translate to origin
newpoint.X = a.Left + (a.Width / 2 + 1) - c.X
newpoint.y = a.Top + (a.Height / 2 + 1) - c.y
'rotate
newpoint2.X = newpoint.X * Cos(theta) - newpoint.y * Sin(theta)
newpoint2.y = newpoint.X * Sin(theta) + newpoint.y * Cos(theta)
'translate back
a.Left = newpoint2.X + c.X - (a.Width / 2 + 1)
a.Top = newpoint2.y + c.y - (a.Height / 2 + 1)
End Sub

Public Sub setboxes()
Form1.box1.Left = usercenter.X + (R2.X - referencecenter.X) - 12
Form1.box1.Top = usercenter.y + (R2.y - referencecenter.y) - 12
Form1.box2.Left = usercenter.X + (R3.X - referencecenter.X) - 12
Form1.box2.Top = usercenter.y + (R3.y - referencecenter.y) - 12

End Sub

Public Sub movedial(previous As VECTOR, cursor As VECTOR)
Dim counter
Dim dist As Double
Dim magnitude As Double
Dim tempvect As VECTOR
Dim temp2vect As VECTOR
Dim scalefactor As Double
Dim zdirect As Double
Dim theta As Double
Dim costheta As Double
Dim temp1 As Double
Dim temp2 As Double
Dim temp3 As Double
Dim temp4 As Double
```

```
 '  message = "dialdrag"
   vectsub2 cursor, Dialcenter, tempvect
   zdirect = previous.X * tempvect.y - previous.y * tempvect.X
   scalefactor = Dialradius / vectmag2(tempvect)
   vectmult2 tempvect, scalefactor, tempvect
   vectadd2 tempvect, Dialcenter, temp2vect
   Form1.Line1.X2 = CInt(temp2vect.X)
   Form1.Line1.Y2 = CInt(temp2vect.y)
   'calculate angle of rotation
   vectunit2 tempvect, tempvect
   costheta = vectdot2(previous, tempvect)
   If costheta = 1 Then 'to prevent divide by 0
      theta = 0
   ElseIf costheta < 1 Then
      temp1 = Sqr(-costheta * costheta + 1)
      If temp1 = 0 Then  'force no divide by 0
         temp1 = 0.000001
      End If
      temp2 = -costheta / temp1
      temp3 = Atn(temp2)
      temp4 = 2 * Atn(1)
      theta = temp3 + temp4
      'theta = Atn(-costheta / Sqr(-costheta * costheta + 1)) + 2 * Atn(1)
   End If
   If zdirect < 0 Then
      theta = theta * -1#
   End If
'   sintheta = Sin(theta)
   'rotate points P
   tempvect.X = 768 ' center of user circle
   tempvect.y = 256
   DrawBez P1, P2, P3, P4, 15 'erase previous user curve
   For counter = 0 To 3
      rotatepoint Form1.Shape1(counter), usercenter, theta
      Form1.Shape1(counter).Refresh
   Next
   LoadPoints P1, P2, P3, P4 'reset user curve
   DrawBez P1, P2, P3, P4, 2
      rotatebox Form1.box1, usercenter, theta
      rotatebox Form1.box2, usercenter, theta
      LoadPoints P1, P2, P3, P4 'reset user curve
      DrawBez P1, P2, P3, P4, 2
   Form1.Shape3.Refresh
End Sub

Public Sub moveusercurve(X As Single, y As Single)
Dim translation As VECTOR
Dim counter As Integer
 Form1.Shape3.Left = X - 10
 Form1.Shape3.Top = y - 10
 translation.X = usercenter.X - X
 translation.y = usercenter.y - y
 usercenter.X = X
 usercenter.y = y
'Erase current user curve
 DrawBez P1, P2, P3, P4, 15
   For counter = 0 To 3
      translatepoint Form1.Shape1(counter), translation
      Form1.Shape1(counter).Refresh
```

```
   Next
 translatebox Form1.box1, translation
 translatebox Form1.box2, translation
 'redraw user curve
 LoadPoints P1, P2, P3, P4 'reset user curve
 DrawBez P1, P2, P3, P4, 2
End Sub

Public Sub translatepoint(a As Shape, c As VECTOR)
'translate to origin
a.Left = a.Left - c.X
a.Top = a.Top - c.y
End Sub

Public Sub translatebox(a As Shape, c As VECTOR)
'translate to origin
a.Left = a.Left - c.X
a.Top = a.Top - c.y
End Sub

Public Function OrderCheck(pos As Integer) As Integer
   If pos = state Then
      OrderCheck = 1
   ElseIf pos = (state + 1) Then
      state = state + 1
      OrderCheck = 1
   Else
      OrderCheck = 0
   End If
End Function

Public Sub RandomRefCenter(a As VECTOR)
  Dim randAngle As Double
  Dim randDistance As Double
  randAngle = (2 * 3.14127 * Rnd)
  randDistance = 75 * Rnd
  a.y = CInt(Sin(randAngle) * randDistance) + 256
  a.X = CInt(Cos(randAngle) * randDistance) + 256
End Sub
Public Sub TranslateRef(refcenter As VECTOR, a As POINTAPI)
   a.X = a.X + refcenter.X - 256
   a.y = a.y + refcenter.y - 256
End Sub

Public Sub Logdata(trial As Integer)
Dim time As Long
Dim curpos As POINTAPI
time = GetTickCount
GetCursorPos curpos

Write #filenum, time, trial, message, state, curpos.X, curpos.y, P1.X, P1.y,
P2.X, P2.y, P3.X, P3.y, P4.X, P4.y, Form1.Line1.X2, Form1.Line1.Y2, usercenter.X,
usercenter.y
message = ""
End Sub
```

8.4 Simulation code: vectorlib.bas (Visual Basic)

```
Attribute VB_Name = "Module1"
Type VECTOR
   X As Double
   y As Double
End Type

Type VECTOR3
   X As Double
   y As Double
   z As Double
End Type

Public Function vectmag2(v1 As VECTOR) As Double
vectmag2 = Sqr((v1.X * v1.X) + (v1.y * v1.y))
End Function

Public Function vectmag3(v1 As VECTOR3) As Double
vectmag3 = Sqr((v1.X * v1.X) + (v1.y * v1.y) + (v1.z * v1.z))
End Function
'v1 and v2 must be normalized
Public Function vectdot2(v1 As VECTOR, v2 As VECTOR) As Double
vectdot2 = (v1.X * v2.X) + (v1.y * v2.y)
End Function
'v1 and v2 must be normalized
Public Function vectdot3(v1 As VECTOR3, v2 As VECTOR3) As Double
vectdot3 = (v1.X * v2.X) + (v1.y * v2.y) + (v1.z * v2.z)
End Function
'adds v1 and v2 and returns in v3
Public Sub vectadd2(v1 As VECTOR, v2 As VECTOR, v3 As VECTOR)
v3.X = v1.X + v2.X
v3.y = v1.y + v2.y
End Sub
'adds v1 and v2 and returns in v3
Public Sub vectadd3(v1 As VECTOR3, v2 As VECTOR3, v3 As VECTOR3)
v3.X = v1.X + v2.X
v3.y = v1.y + v2.y
v3.z = v1.z + v2.z
End Sub
'v1 - v2 = v3
Public Sub vectsub2(v1 As VECTOR, v2 As VECTOR, v3 As VECTOR)
v3.X = v1.X - v2.X
v3.y = v1.y - v2.y
End Sub
'v1 - v2 = v3
Public Sub vectsub3(v1 As VECTOR3, v2 As VECTOR3, v3 As VECTOR3)
v3.X = v1.X - v2.X
v3.y = v1.y - v2.y
v3.z = v1.z - v2.z
End Sub
'v1*c=v2
Public Sub vectmult2(v1 As VECTOR, c As Double, v2 As VECTOR)
v2.X = v1.X * c
v2.y = v1.y * c
End Sub
'v1*c=v2
Public Sub vectmult3(v1 As VECTOR3, c As Double, v2 As VECTOR3)
```

```
v2.X = v1.X * c
v2.y = v1.y * c
v2.z = v1.z * c
End Sub
'v2 is normalized from v1
Public Sub vectunit2(v1 As VECTOR, v2 As VECTOR)
Dim magnitude As Double
magnitude = vectmag2(v1)
If magnitude = 0 Then   'GIGO keeps from divide by 0 if
   magnitude = 0.00001  ' vector = 0
End If
Call vectmult2(v1, (1 / magnitude), v2)
End Sub
'v2 is normalized from v1
Public Sub vectunit3(v1 As VECTOR3, v2 As VECTOR3)
Dim magnitude As Double
magnitude = vectmag3(v1)
If magnitude = 0 Then   'GIGO keeps from divide by 0 if
   magnitude = 0.00001  ' vector = 0
End If
vectmult3 v1, (1 / magnitude), v2
End Sub

Public Function Distance(v1 As VECTOR, v2 As VECTOR) As Double
Dim temp As VECTOR
temp.X = v1.X - v2.X
temp.y = v1.y - v2.y
Distance = vectmag2(temp)
End Function
```

8.5 Simulation code: staticstuff.bas (Visual Basic)

```
Attribute VB_Name = "Module3"
Public Sub refreshbox()
   Form1.box1.Refresh
   Form1.box2.Refresh
End Sub

Public Sub refreshdial()
refreshcounter = refreshcounter + 1
If (refreshcounter Mod 5) = 0 Then
   Form1.Shape2.Refresh
End If
End Sub

Public Function DrawBez(P1 As POINTAPI, P2 As POINTAPI, P3 As POINTAPI, P4 As
POINTAPI, color As Single)
Dim co1 As Double
Dim co2 As Double
Dim co3 As Double
Dim co4 As Double
t = 0
prevpt.X = P1.X
prevpt.y = P1.y
Do
   t = t + 0.05
   co1 = (1 - t) * (1 - t) * (1 - t)
   co2 = (1 - t) * (1 - t) * 3 * t
   co3 = (1 - t) * 3 * t * t
   co4 = t * t * t
   currpt.X = CDbl(P1.X * co1 + P2.X * co2 + P3.X * co3 + P4.X * co4)
   currpt.y = P1.y * co1 + P2.y * co2 + P3.y * co3 + P4.y * co4
  Form1.Line (prevpt.X, prevpt.y)-(currpt.X, currpt.y), QBColor(color)
   prevpt.X = currpt.X
   prevpt.y = currpt.y
Loop While t < 1
End Function
Public Function getvector(start As POINTAPI, ending As POINTAPI, result As
VECTOR)
result.X = CDbl(ending.X - start.X)
result.y = CDbl(ending.y - start.y)
End Function

Public Sub setcolors()
If order(1) = 1 Then
   Form1.Line1.BorderColor = &HD0D0D0
ElseIf order(1) = 2 Then
   Form1.Line1.BorderColor = &H909090
Else
   Form1.Line1.BorderColor = &H0
End If

If order(2) = 1 Then
   Form1.Shape1(1).BorderColor = &HD0D0D0
   Form1.Shape1(1).FillColor = &HD0D0D0
   Form1.Shape1(2).BorderColor = &HD0D0D0
   Form1.Shape1(2).FillColor = &HD0D0D0
ElseIf order(2) = 2 Then
```

```
   Form1.Shape1(1).BorderColor = &H808080
   Form1.Shape1(1).FillColor = &H808080
   Form1.Shape1(2).BorderColor = &H808080
   Form1.Shape1(2).FillColor = &H808080
Else
   Form1.Shape1(1).BorderColor = &H0
   Form1.Shape1(1).FillColor = &H0
   Form1.Shape1(2).BorderColor = &H0
   Form1.Shape1(2).FillColor = &H0
End If
If order(3) = 1 Then
   Form1.Shape3.BorderColor = &HD0D0D0
   Form1.Shape3.FillColor = &HD0D0D0
ElseIf order(3) = 2 Then
   Form1.Shape3.BorderColor = &H808080
   Form1.Shape3.FillColor = &H808080
Else
   Form1.Shape3.BorderColor = &H0
   Form1.Shape3.FillColor = &H0
End If

End Sub
Public Sub setorder()
Dim perm As Integer
perm = Int(6 * Rnd + 1)
If perm = 1 Then
   order(1) = 1
   order(2) = 2
   order(3) = 3
ElseIf perm = 2 Then
   order(1) = 1
   order(2) = 3
   order(3) = 2
ElseIf perm = 3 Then
   order(1) = 2
   order(2) = 1
   order(3) = 3
ElseIf perm = 4 Then
   order(1) = 2
   order(2) = 3
   order(3) = 1
ElseIf perm = 5 Then
   order(1) = 3
   order(2) = 1
   order(3) = 2
Else
   order(1) = 3
   order(2) = 2
   order(3) = 1
End If
state = 0
setcolors
End Sub
```

8.6 Data Analysis Code: dataanal.cpp

```
/************************************************
 * FILE: dataanal.cpp
 *
 * AUTHOR: Martin Dulberg, Angelina Talley
 *
 * DATE: 27-Nov-2001

 * Last Modified 18-Sep-2002
 *
 ************************************************/


#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "dataanal.h"
void dodial(int & j, int size, int state);
void doposition(int & j, int size, int state);
void dopoints(int & j, int size, int state);
void doselection(int & start, int & end);
double compute_radians(int start, int end);
void do_selection(int &start_move, int &end_move, int end);
double VectMag (double v1[]);
double VectDot (double v1[], double v2[]);
double VectCos (double v1[], double v2[]);
void VectUnit (double v1[], double result[]);
int num_datapoints;



/*************************************
 * Name: main
 *
 * Description:
 *      main function of program - coordinates
 *      reading in and processing
 *
 * Date: 27-Nov-2001
 *************************************/
int main()
{
   datapoint curdatapoint;
   int trialnum = 0;
   int end_move = 0;

   /* prompt user for file names and open data
      and trial file streams */
   if (!prompt_openFiles())
   {
      return 0; //error could not open files
   }
   /* read in first data point */
   readDatapoint(curdatapoint);
```

```
   //trim off first few lines where trial = 0
   while(curdatapoint.trial == 0)
   {
      readDatapoint(curdatapoint);
   }

   /* while we still have trials to complete, iterate through them */
   while(readTrial(curtrial, trialnum))
   {
      int start_move=0;  //current datapoint
      int t = 0;

      num_datapoints = 0; //reset count of datapoints for this trial

      /* copy first datapoint into array manually */
      trial_datapoints[0] = curdatapoint; //CHANGED

      /* read in all datapoints for this trial */
      while(trialnum==readDatapoint(curdatapoint))
      {
          trial_datapoints[num_datapoints]= curdatapoint; //CHANGED
          num_datapoints++;
      }

      /* get the initial selection */
      int end_move = start_move;
      do_selection(start_move,end_move,num_datapoints);

      /* based on order user must go in, assume what we see in data is actually
         a certain type of action, and go look for it */
      for(int k=0;k<3;k++)
      {
         if(curtrial.o[k]==1)
         {
            dodial(end_move, num_datapoints, k+1); //check args and fix state
         }
         else if (curtrial.o[k]==2)
         {
            dopoints(end_move,   num_datapoints,   k+1);   //must   fix,   curently
hardcoded
         }
         else if (curtrial.o[k]==3)
         {
            doposition(end_move,   num_datapoints,   k+1);   //must   fix,   curently
hardcoded
         }
         else
         {
            //cout << "error k=" << k<< '\t'<<curtrial.o[k]<<endl;
         }
         if (end_move >= (num_datapoints-1) && k < 2)
         {
             /* watch for us hitting end without getting to all three tasks */
             //cout << "TIMEOUT detected before completing all tasks! Last state:
" << k << endl;
             break;
         }
      }
      //cout << "\nEND - trial " << trialnum <<endl;
```

```
   }//while there is another trial
   return 0;
}

/*************************************
 * Name: dodial
 *
 * Description:
 *      process movement of the dial
 *
 * Date: 27-Nov-2001
 *************************************/
void dodial(int & i, int size, int state)
{
   int j=0;
   int end=i;
   int count=0;
   int front=0;
   bool error = false;
   point dial_start_pos;
   int start = 0;
   int ED_loc = -1; // save location of last ED we find before end

   while(state==trial_datapoints[end].state && end<size-1)
   {
      end++; // find the end of this part of the trial
   }
   do
   {
     /* find start of dial */
     while(strcmp(trial_datapoints[i].message,"SD")!=0 && i<end)
     {
        i++;
     }

     dial_start_pos = trial_datapoints[i].l2;
     start = i;
     i+=1; // move to point past SD

     /* find last ED of this round on the dial */
     while (i < end && strcmp(trial_datapoints[i].message,"SD")!=0)
     {
         while(strcmp(trial_datapoints[i].message,"ED")!=0 && i<end)
         {
            if (trial_datapoints[i].message[0]=='X')
            {
                cout << "ERROR IN DIAL" << endl;
                error = true;
                break;
            }
            i++;
         }

         if(i>start+1 && strcmp(trial_datapoints[i].message,"ED")==0)
         {
            ED_loc = i;
            i++; // now that we've marked ED, keep moving to end of this round
```

```
            // if we don't have an error, this is the ED we're looking for -
abort
            if (!error)
            {
                break;
            }
        }
    }

    if (!error && ED_loc != -1)
    {
        cout << "dialmove \t" << compute_radians(start,ED_loc-1)
             << '\t' << trial_datapoints[start].time << '\t'
             <<trial_datapoints[ED_loc].time <<endl;
    }
    else
    {
        // reset the error variable
        error = false;
    }

    /* If ED_loc is == -1 or if it as the end, don't do selection here - go
       back out to main loop and start over */
    if (ED_loc == -1)
    {
        //cout << "TIMEOUT in dial" << endl;
        break;
    }
    else if (ED_loc == end)
    {
       break;
    }

    //i is at start, j is at end of selection
    i=ED_loc+1;
    j=i+1;

    /* find the next selection */
    do_selection(i,j,end);

    i = j; // set i to the start of the next trial

  } while(i<end);
}

/*************************************
 * Name: doposition
 *
 * Description:
 *      process drag of curve into position
 *
 * Date: 27-Nov-2001
 *************************************/
void doposition(int & i, int size, int state)
{
   int j=0;
   int end=i;
   int count=0;
   int front=0;
```

```
   bool error = false;
   int start = 0;
   int ED_loc = -1; // save location of last ED we find before end

   while(state==trial_datapoints[end].state && end<size-1)
   {
      end++; // find the end of this part of the trial
   }
   do
   {
     /* find start of position */
     while(strcmp(trial_datapoints[i].message,"SM")!=0 && i<end)
     {
        i++;
     }

     start = i;
     i+=1; // move to point past SM
     /* find last ED of this round on the dial */
     while (i < end && strcmp(trial_datapoints[i].message,"SM")!=0)
     {
         while(strcmp(trial_datapoints[i].message,"ED")!=0 && i<end)
         {
            if (trial_datapoints[i].message[0]=='X')
            {
                cout << "ERROR IN POSITION" << endl;
                error = true;
                break;
            }
            i++;
         }

         if(i>start+1 && strcmp(trial_datapoints[i].message,"ED")==0)
         {
            ED_loc = i;
            i++; // now that we've marked ED, keep moving to end of this round
            // if we don't have an error, this is the ED we're looking for -
abort
            if (!error)
            {
                break;
            }
         }
     }

     if (!error && ED_loc != -1)
     {
        cout                              <<                              "position\t"
<<distance(trial_datapoints[start].cur,trial_datapoints[ED_loc].cur)<< '\t'
             <<           trial_datapoints[start].time           <<           '\t'
<<trial_datapoints[ED_loc].time <<endl;
     }
     else
     {
         // reset the error variable
         error = false;
     }

     /* If ED_loc is == -1 or if it as the end, don't do selection here - go
```

```
         back out to main loop and start over */
      if (ED_loc == -1)
      {
          //cout << "TIMEOUT in position" << endl;
          break;
      }
      else if (ED_loc == end)
      {
         break;
      }

      //i is at start, j is at end of selection
      i=ED_loc+1;
      j=i+1;

      /* find the next selection */
      do_selection(i,j,end);

      i = j; // set i to the start of the next trial

   } while(i<end);
}

/*************************************
 * Name: dopoints
 *
 * Description:
 *      process moving control points to boxes
 *
 * Date: 27-Nov-2001
 *************************************/
void dopoints(int & i, int size, int state)
{
   int j=0;
   int end=i;
   int count=0;
   int front=0;
   bool error = false;
   int start = 0;
   int ED_loc = -1; // save location of last ED we find before end

   while(state==trial_datapoints[end].state && end<size-1)
   {
      end++; // find the end of this part of the trial
   }
   do
   {
     /* find start of first point */
     while((strcmp(trial_datapoints[i].message,"S1")!=0 &&
            strcmp(trial_datapoints[i].message,"S2")!=0) &&
           i<end)
     {
        i++;
     }

     start = i;
     i+=1; // move to point past S1/S2
     /* find last ED of this point's move */
     while (i < end && strcmp(trial_datapoints[i].message,"S1")!=0
```

```
            && strcmp(trial_datapoints[i].message,"S2")!=0)
     {
         while(strcmp(trial_datapoints[i].message,"ED")!=0 && i<end)
         {
            /* if hit second point, stop */
            if (strcmp(trial_datapoints[i].message,"S1")==0 ||
                strcmp(trial_datapoints[i].message,"S2")==0)
            {
                break;
            }
            if (trial_datapoints[i].message[0]=='X')
            {
                cout << "ERROR IN POINTMOVE" << endl;
                error = true;
                break;
            }
            i++;
         }

         if(i>start+1 && strcmp(trial_datapoints[i].message,"ED")==0)
         {
            ED_loc = i;
            i++; // now that we've marked ED, keep moving to end of this round
            // if we don't have an error, this is the ED we're looking for -
abort
            if (!error)
            {
                break;
            }
         }
     }

     if (!error && ED_loc != -1)
     {
        cout                              <<                              "pointmove\t"
<<distance(trial_datapoints[start].cur,trial_datapoints[ED_loc].cur)<< '\t'
             <<           trial_datapoints[start].time           <<           '\t'
<<trial_datapoints[ED_loc].time <<endl;

     }
     else
     {
         // reset the error variable
         error = false;
     }

     /* If ED_loc is == -1 or if it as the end, don't do selection here - go
        back out to main loop and start over */
     if (ED_loc == -1)
     {
         //cout << "TIMEOUT in pointmove" << endl;
         break;
     }
     else if (ED_loc == end)
     {
        break;
     }

     //i is at start, j is at end of selection
```

```
     i=ED_loc+1;
     j=i+1;
     ED_loc=-1; //re-initialize ED_loc, since we'll need it again

     /* find the next selection */
     do_selection(i,j,end);

     i = j; // set i to the start of the next trial

   } while(i<end);
}

/*************************************
 * Name: computeRadians
 *
 * Description:
 *      compute angular distance moved for dial
 *      NOTE: Start and end are exact indecies
 *      of start and end points.
 *
 * Date: 05-Dec-2001
 *************************************/
double compute_radians(int data_start, int data_end)
{
/* create vector for l2-l1 at start and end of
   each move. Sum up the radians */
    int i = 0;
    double radians = 0;
    double cosine = 0;
    double vec1[2];
    double vec2[2];

    for (i = data_start; i < data_end-1; i++)
    {
        /* get vectors for the line made from these points */
        vec1[0] = trial_datapoints[i].l2.x-curtrial.l1.x;
        vec1[1] = trial_datapoints[i].l2.y-curtrial.l1.y;
        vec2[0] = trial_datapoints[i+1].l2.x-curtrial.l1.x;
        vec2[1] = trial_datapoints[i+1].l2.y-curtrial.l1.y;

        /* if vectors are not equal, we had movement */
        if ((vec1[0] != vec2[0]) ||
            (vec1[1] != vec2[1]))
        {
            cosine = VectCos(vec1,vec2);
       //     cosine = ((vec1[0]*vec2[0])+(vec1[1]*vec2[1]))/
        //
(sqrt(pow(vec1[0],2)*pow(vec1[1],2))*sqrt(pow(vec2[0],2)*pow(vec2[1],2)));
            radians += acos(cosine);
        }

    }
    return radians;
}

/*************************************
 * Name: do_selection
 *
 * Description:
```

```
 *      compute next selection
 *
 * Date: 28-Nov-2001
 ************************************/
void do_selection(int &start_move, int &end_move, int end)
{
    int state = trial_datapoints[end_move].state;
    int ED_loc = -1;
    bool error = false;
    int true_start = 0;

    //throw away movement until have minimum 3 pixels
    while(trial_datapoints[start_move].state  ==  trial_datapoints[end_move].state
&&
          distance(trial_datapoints[start_move].cur,
trial_datapoints[end_move].cur) < 3
          && end_move < end)
    {
        /* watch for ED - throw out this selection if
           don't move 3 pixels. Probably just mouse jitter
        */
        if (trial_datapoints[end_move].message[0] == 'E')
        {
            ED_loc = end_move;
            error = true; // by setting error here, we can
                          // still use loop below to find our
                          // way to start of next task
            break;
        }
        /* we'll handle error once we start the loop below */
        else if (trial_datapoints[end_move].message[0] == 'X')
        {

            error = true;
            break;
        }
        end_move++;
    }
    /* if broke because distance < 3 and have no error, we have a timeout */
    if                                  (distance(trial_datapoints[start_move].cur,
trial_datapoints[end_move].cur) < 3
        && error == false)
    {
        //cout << "TIMEOUT in selection" << endl;
        return;
    }

    start_move=end_move;

    /* see how far we moved in initial selection before we start first task*/
    while(trial_datapoints[end_move].state == state &&
          end_move < end)
    {
        if (strcmp(trial_datapoints[end_move].message,"X") == 0)
        {
            /* go ahead and move forward to next slot inside error section */
            cout << "ERROR IN SELECTION" << endl;
            end_move++;
            /* move forward spot last ED and start selection over */
```

```
            while (trial_datapoints[end_move].state == state &&
                   end_move < end)
            {
                if (strcmp(trial_datapoints[end_move].message,"ED")==0)
                {
                    ED_loc = end_move;
                    end_move+=1; // move to spot after ED
                    start_move=end_move;
                }
                /* we've hit another error, report it and go back up to
                   the if to handle this new one
                */
                else
                {
                    if (strcmp(trial_datapoints[end_move].message,"X") == 0)
                    {
                        break;
                    }
                    end_move++;
                }
            }
        }
        else if (trial_datapoints[end_move].message[0]=='S')
        {
            break;
        }
        else
        {
            end_move++;
        }
    }
    /* we get here, we know our valid start point (either start or just after
       the last ED. If was after an ED, check for three pixels movement to get
       true start point and watch for a timeout */
    if (ED_loc >= 0)
    {
        true_start = start_move;
        while(trial_datapoints[start_move].state                                    ==
trial_datapoints[true_start].state &&
             distance(trial_datapoints[start_move].cur,
trial_datapoints[true_start].cur) < 3)
        {
            true_start++;
        }
        start_move = true_start;
        if (true_start == end)
        {
            //cout << "TIMEOUT in selection" << endl;
            return;
        }
    }
    /* if selection timing out is TIMEOUT, uncomment this code */
    else if (ED_loc == -1 && end_move == num_datapoints-1)
    {
        //cout << TIMEOUT in selection" << endl;
        return;
    }
```

```
cout<<"Selection\t"<<distance(trial_datapoints[start_move].cur,trial_datapoints[e
nd_move-1].cur) <<'\t'<< trial_datapoints[start_move].time<<'\t'
<< trial_datapoints[end_move-1].time <<endl;

}


#define VectAdd(a,b,c) {c[0]=a[0]+b[0];c[1]=a[1]+b[1];c[2]=a[2]+b[2];}
#define VectMult(a,b,c) {c[0]=a[0]*b;c[1]=a[1]*b;}
double VectMag (double v1[])
{
   double result;
   result=(pow(v1[0],2))+(pow(v1[1],2));
   result=sqrt(result);
   return result;
}

double VectDot (double v1[], double v2[])
{
   double result;
   result=(v1[0]*v2[0])+(v1[1]*v2[1]);
   return result;
}
double VectCos (double v1[], double v2[])
{
   double norm1[2], norm2[2], dot;
   VectUnit(v1,norm1);
   VectUnit(v2,norm2);
   dot=VectDot(norm1,norm2);
   return dot;
}

void VectUnit (double v1[], double result[])
{
   double magnitude;
   magnitude=VectMag(v1);
   VectMult(v1,(1/magnitude),result);
}
```

8.7 Data Analysis Code: dataanal.h

```
/************************************************
 * FILE: dataanal.h
 *
 * AUTHOR: Martin Dulberg, Angelina Talley
 *
 * DATE: 27-Nov-2001
 *
 ************************************************/

/**************************************
 * Name: point
 *
 * Description:
 *      a simple x and y coordinate
 *
 * Date: 27-Nov-2001
 **************************************/
struct point
{
   unsigned short x;
   unsigned short y;
};

/**************************************
 * Name: datapoint
 *
 * Description:
 *      data point
 *
 * Date: 27-Nov-2001
 **************************************/
struct datapoint
{
   unsigned long time;
   short trial;
   char message[10];
   short state;
   point cur;
   point p1; point p2; point p3; point p4;
   point l2;
   point uc;
};

/**************************************
 * Name: trials
 *
 * Description:
 *      hold information about a trial,
 *      including number and setup values
 *
 * Date: 27-Nov-2001
 **************************************/
struct trials
{
   int trialnum;
   point r1; point r2; point r3; point r4;
```

```
   point p1; point p2; point p3; point p4;
   point l1; point l2; point ref;
   short o[3];
};

/*
 global variables to hold input streams
 from files
*/
ifstream trial_stream;
ifstream data_stream;

//global array to handle 1 trials worth of info
datapoint trial_datapoints[1500];
trials curtrial;


/*************************************
 * Name: distance
 *
 * Description:
 *      simple function to compute distance
 *
 * Date: 27-Nov-2001
 *************************************/
double distance(point start, point end)
{
   double tempx=(double)abs((start.x-end.x));
   double tempy=(double)abs((start.y-end.y));
   return sqrt(tempx*tempx + tempy*tempy);
}

/*************************************
 * Name: readTrial
 *
 * Description:
 *      simple function to output one trial
 *
 * Date: 27-Nov-2001
 *************************************/
bool readTrial(trials& c, int& trialnum)
{
   char t;
   /* read trial number */
   if(trial_stream >> c.trialnum)
   {
       /* read in comma, x coord, comma, y coord, etc. */
      trial_stream >> t >> c.r1.x>> t >> c.r1.y >> t >> c.r2.x >> t >> c.r2.y
                   >> t >> c.r3.x >> t >> c.r3.y >> t >> c.r4.x
                   >> t >> c.r4.y >> t >> c.p1.x >> t >> c.p1.y
                   >> t >> c.p2.x >> t >> c.p2.y >> t >> c.p3.x
                   >> t >> c.p3.y >> t >> c.p4.x >> t >> c.p4.y
                   >> t >> c.l1.x >> t >> c.l1.y >> t >> c.l2.x
                   >> t >> c.l2.y >> t >> c.ref.x >> t >> c.ref.y
                   >> t >> c.o[0] >> t >> c.o[1] >> t >> c.o[2]; // last three
things
                                                          // read are the order parts
                                                           // should be completed in
      trialnum++;
```

```
      if (trial_stream.eof())
      {
          trial_stream.close();
      }

      return true;
   }
   return false;
}


/*************************************
 * Name: prompt_openFiles
 *
 * Description:
 *      simple function to prompt for trial
 *      and data input file names and open them
 *      for later reading
 *
 * Date: 27-Nov-2001
 *************************************/
bool prompt_openFiles(void)
{
    char filename[20];
    //cerr << "Enter Trial File Name: ";
    //cin >> filename;
    strcpy(filename,"trials.txt");
    trial_stream.open(filename);
    if(!trial_stream){
       cerr << "Error in Trial File Name" << endl;
       return false;
       }
    //cerr << "Enter Data File Name: ";
    //cin >> filename;
    strcpy(filename, "test.txt");
    data_stream.open(filename);
    if(!data_stream)
    {
       cerr << "Error in Data File Name" << endl;
       return false;
    }
    return true;
}

/*************************************
 * Name: readDatapoint
 *
 * Description:
 *      Read in the next data point from the
 *      data file.
 *      Returns the current trial number which
 *      can be used to check to see if you have
 *      moved to the next trial.
 *      Returns 0 if no more data
 *
 * Date: 27-Nov-2001
 *************************************/
int readDatapoint(datapoint& d)
{
```

```
   char t;
   int i=0;

  /* first read in time, continue
      if not zero */
   if (!data_stream || !data_stream.is_open() || data_stream.eof())
   {
       return 0;
   }
   if(data_stream >>d.time)
   {
       /* read in comma, trial #, comma, "" */
      data_stream >> t >> d.trial >> t >> t;
      do
      {
        /* read in message, if any, about what user's doing */
          data_stream >> t;
          d.message[i]=t;
          i++;
        /* stop when hit next " */
      } while(t != '\"');
      /* null terminate the message */
      d.message[i-1]='\0';

      /* read in comma, then state, comma, x, comma, y, etc. */
      data_stream >> t >> d.state >> t >> d.cur.x >> t >> d.cur.y
                  >> t >> d.p1.x >> t >> d.p1.y >> t >> d.p2.x
                  >> t >> d.p2.y >> t >> d.p3.x >> t >> d.p3.y
                  >> t >> d.p4.x >> t >> d.p4.y >> t >> d.l2.x
                  >> t >> d.l2.y >> t >> d.uc.x >> t >>d.uc.y;
      if (data_stream.eof() || data_stream.peek() == EOF)
      {
          data_stream.close();
      }
      return d.trial;
   }
   return 0; //error condition no more records
}
```

8.8 Data Analysis Code: transitions.cpp

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(){
char name[50];
char name2[50];
char trash[80];
float junk;
long int start[2];
long int end[2];
cout << "Enter File name : ";
cin >> name;
//cout <<"got to 1\n";
ifstream in(name);
cout <<"got to 2\n";

if(!in){
   cout << "ERROR" << endl;
   return 0;
}
in >> name;

in >>junk>>start[0] >> end[0];
in.getline( trash,80,'\n');

while(in>>name2){//cout << "here too";
   cout << name << ' '<<name2;
   if ( name2[0]=='E'){
      cout << endl;
      }
   else if ( name[0]=='E'){
      cout << endl;
      in >>junk>>start[1] >> end[1];
     }

   else{
      in >>junk>>start[1] >> end[1];
      cout <<'\t'<< start[1]-end[0]<<endl;
      }
   in.getline(trash,80, '\n');
   strcpy(name,name2);
   start[0]=start[1];
   end[0]=end[1];
   }

return 0;

}
```

8.9 Data Analysis Code: trialanalysis.cpp

```
/************************************************
 * FILE: trialanalysis.cpp
 *
 * AUTHOR: Martin Dulberg
 * Based on code from dataanal.cpp with Angelina Talley
 *
 * DATE: 18-Nov-2002
 * Last Modified
 *
 ************************************************/
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "trialanalysis.h"  //same as dataanal.h
double compute_radians( int end);
int num_datapoints;
double computeinteriordist(int data_end);



/*************************************
 * Name: main
 *
 * Description:
 *      main function of program - coordinates
 *      reading in and processing
 *
 * Date: 27-Nov-2001
 *************************************/
int main()
{
   datapoint curdatapoint;
   int trialnum = 0;
   int end_move = 0;

   if (!prompt_openFiles())
      return 0; //error could not open files
   cout << "Order,Time,Center,Angle,Points,Selection,Position,Orientation"<<endl;
   readDatapoint(curdatapoint);// read in first data point
   while(curdatapoint.trial == 0)//trim off first few lines where trial = 0
      readDatapoint(curdatapoint);
   while(readTrial(curtrial, trialnum))
   {
      int start_move=0;  //current datapoint
      int t = 0;
      num_datapoints = 0; //reset count of datapoints for this trial
      trial_datapoints[0] = curdatapoint;
      while(trialnum==readDatapoint(curdatapoint)){
          trial_datapoints[num_datapoints]= curdatapoint; //CHANGED
          num_datapoints++;
          }
//Process Trial
//correct bug in vb software logs 3-1-2 as 2-3-1 and vice versa
      if (curtrial.o[0]==3 && curtrial.o[1]== 1 && curtrial.o[2]== 2 ){
```

```
         curtrial.o[0]=2;curtrial.o[1]=3; curtrial.o[2]=1;}
      else if (curtrial.o[0]==2 && curtrial.o[1]== 3 && curtrial.o[2]== 1 ){
         curtrial.o[0]=3;curtrial.o[1]=1; curtrial.o[2]=2;}


//output ordering as single integer
      cout << curtrial.o[0]*100+curtrial.o[1]*10+curtrial.o[2]<<',';
//figure out where end of user movement in trial was.
      int i=num_datapoints-1;
      while(strcmp(trial_datapoints[i].message,"")==0)
         i--;
      int elapsed=trial_datapoints[i].time-trial_datapoints[0].time;
      if (elapsed>30000)
         elapsed=30000;
      cout << elapsed <<',';
//error for center positioning
      point temp = trial_datapoints[i].uc;
      temp.x-=512;  //translate to reference
      float centerdist=distance(curtrial.ref, temp);
      cout << centerdist<<',';

      double angle=compute_radians(i);
      cout << angle << ',';

//figure out which user points distance is shorter
//don't want to take into account displacement of entire curve)

      double pointdist=computeinteriordist(i);
      cout <<pointdist<<',';

//compute errors for trial
int selection=0, orientation=0, position=0;
      while(i>0){
         while(i>0 && trial_datapoints[i].message[0]!='X')
            i--;
         if(i--){
            while(i>0 && trial_datapoints[i].message[0]!='\0')
               i--;
            if(strcmp(trial_datapoints[i].message, "SD")==0)
               orientation++;
            else if(trial_datapoints[i].message[0]=='S')
               position++;
            else
               selection++;
         }//end if i--
      }  //while i>0
   cout << selection<<','<<position<<','<<orientation<<endl;

   }//while there is another trial
   return 0;
}

/*************************************
 * Name: computeRadians
 *
 * Description:
 *      compute angular distance moved for dial
 *      NOTE: Start and end are exact indecies
 *      of start and end points.
```

```
 *
 * Date: 05-Dec-2001
 ************************************/
double compute_radians(int data_end)
{
/* create vector for l2-l1 at start and end of
   each move. Sum up the radians */
    int i = 0;
    double radians = 0;
    double radians2 = 0;

    double cosine = 0;
    double vec1[2];
    double vec2[2];

        /* get vectors for the line made from these points */
        vec1[0]= trial_datapoints[data_end].p1.x-trial_datapoints[data_end].p4.x;
        vec1[1]= trial_datapoints[data_end].p1.y-trial_datapoints[data_end].p4.y;
        vec2[0] = curtrial.r1.x-curtrial.r4.x;
        vec2[1] = curtrial.r1.x-curtrial.r4.x;
        cosine = VectCos(vec1,vec2);
        radians = acos(cosine);
        //now try the other way
        vec1[0]= trial_datapoints[data_end].p4.x-trial_datapoints[data_end].p1.x;
        vec1[1]= trial_datapoints[data_end].p4.y-trial_datapoints[data_end].p1.y;
        cosine = VectCos(vec1,vec2);
        radians2 = acos(cosine);
        if (radians2<radians)
           radians=radians2;
    return radians;  //return smallest
}


double computeinteriordist(int data_end){
point u2, u3, translation;
double dist1=0, dist2=0;
translation.x=trial_datapoints[data_end].uc.x-curtrial.ref.x;
translation.y=trial_datapoints[data_end].uc.y-curtrial.ref.y;

u2=trial_datapoints[data_end].p2;
u3=trial_datapoints[data_end].p3;
u2.x-=translation.x;
u2.y-=translation.y;
u3.x-=translation.x;
u3.y-=translation.y;
dist1=distance(u2,curtrial.r2)+distance(u3,curtrial.r3);
dist2=distance(u3,curtrial.r2)+distance(u2,curtrial.r3);
if (dist2<dist1)
   dist1=dist2;
return dist1;
}
```