EXHIBIT 3.07

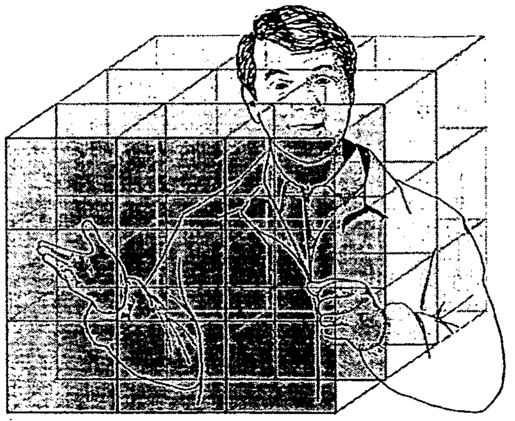


Figure 2: The discrete segmentation of the gesture space

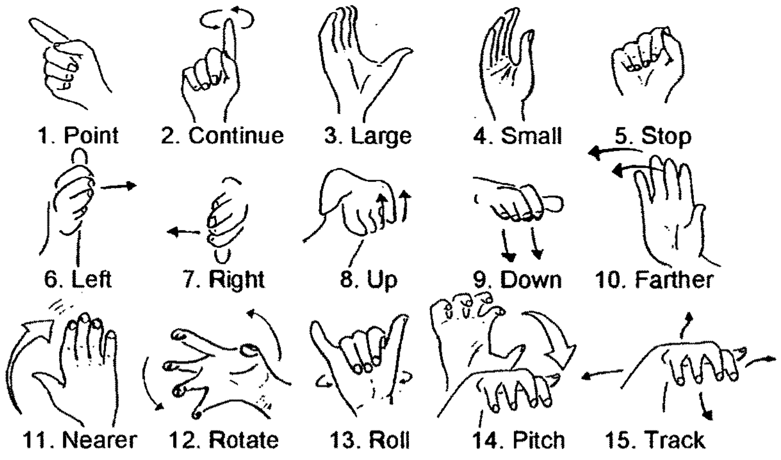
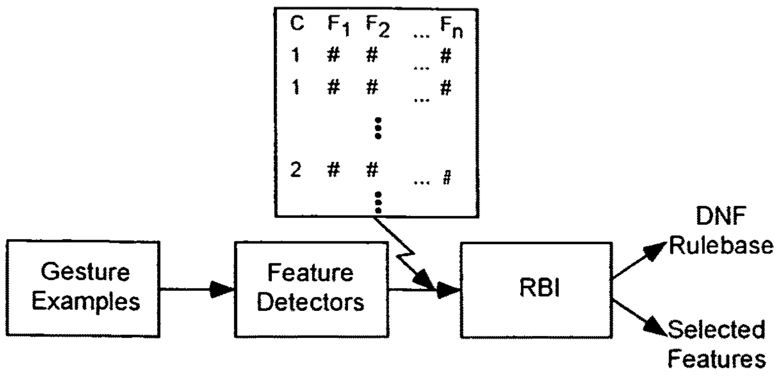


Figure 3: Gesture Vocabulary

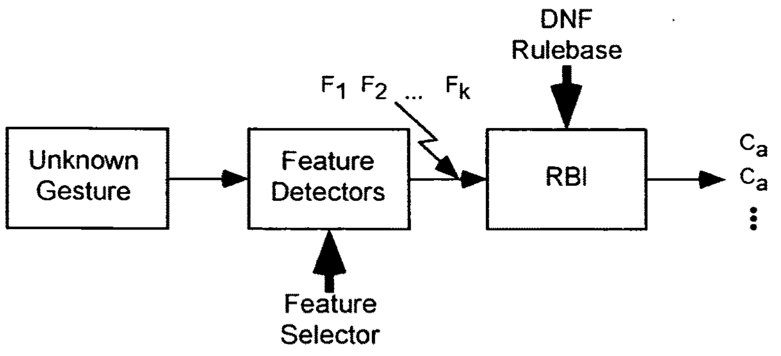
Given the inability of human interpreters to locate *stroke* extrema precisely, the hand position may be quantized into $3 \times 3 \times 3$ discrete subspaces (see figure 2).

In humans, hand gestures are dominant in communication to describe space and time [2]. Our work concentrates on such gestures. Figure 3 is a set of fifteen gestures designed for the description of space and the specification of spatial quantities. Detailed justification of this lexicon and further discussion of the psycholinguistic and semiotic basis of our work may be found in [3, 4].

In the remainder of this section, we shall discuss components of our system to achieve the criteria we have motivated for hand gesture interpretation. We shall first describe our inductive-learning-based approach to the modeling and recognition of hand configurations.



a. RBI Learning Procedure



b. Application of Selected Features and DNF Rulebase

Figure 4: RBI Learning and Rule Application

Second, we shall discuss our dynamic vision system for tracking the motion of an unadorned hand.

2.2 Gesture Pose Recognition

In deliberately communicative gestures, the chief variants in hand configuration are due to perspective effects (within a particular viewpoint as discussed earlier), the differences in user hand anatomy (larger hands, length of digits etc.), and the idiosyncracies of configuration across users for a particular gesture. Owing to these variations, it would be impractical to hand-code models of each gesture for recognition. Furthermore, the number of different hand configurations in any gesture vocabulary could conceivably be very large. Hence, we have developed and applied an inductive learning approach to acquisition of hand configuration models [6].

Figure 4a illustrates the process of generating hand pose models using our *rule-based* *induction* (RBI) system. A set of classified hand pose examples are presented to the system which operates on the images using a suite of feature detectors. The resulting table of class-feature vector associations is fed to the RBI algorithm which produces a rulebase of disjunctive normal form (DNF) formulae. Each DNF formula represents one hand pose, and each conjunct within the formula constitutes a single rule. The system also determines a subset of features that are salient to the recognition task. This reduces the number of features that need to be computed at *recognition time*. Figure 4b shows how the subset of features and DNF rulebase are used to recognize new gesture presentations.

2.2.1 Inductive Learning and Gesture Pose

Our learning system is based on an extension to the *Variable-Valued Logic* (VL1) formulation due to Michalski [7]. We have developed an *Extended Variable-Valued Logic* (EVL) system which can generalize to unobserved events for hand gesture model acquisition. The derivation and proofs of the logical system and rules for minimization of such logical expressions will not be dealt with in this paper (see [6] for such detailed treatment). In this section, we shall give a brief overview of our system.

Like VL1, EVL defines a multi-valued logic system. Each takes a input vectors of multi-valued feature values. Each vector represents an *observation instance* for a particular object (or class). Each class may be represented by multiple observations. The AQ family of learning algorithms, based on VL1, takes a training set of such inputs and generates a *disjunctive normal form* DNF formula set that may be applied to classify new observations [7]. One class is represented by one DNF. Each conjunction within that DNF formula constitutes a *rule*. Each expression within that conjunction represents one feature-value pair (or match).

The key extension in EVL is that it represents unobserved *events* (input vector combinations) explicitly. It can also represent conflicts in the training vector set (i.e. two classes having identical feature vectors). If one were to consider the universal event space as the space of all possible feature vectors, each class will divide this space into three sets: the *positive* set which consists of all observations of the class, the *negative* set which contains all observations not in the class, and the *unobserved* set which contains all vectors in the universal event space not in either the *positive* or *negative* sets. This extension, along with the theorems derived in [6] allows us to develop our *Rule-Based Induction* (RBI) algorithm.

Like the AQ algorithms, RBI generates a set of DNF formulae. The key difference between RBI and the AQ-family is that it is capable of recursive learning (i.e. RBI can learn from existing rule sets as new training observations are applied). This is important for hand gesture recognition for three reasons:

- RBI can generate more compact rules. Being able to reason with rules allows RBI to avoid local minima in the rule efficiency (measured by the number of expressions in the DNF formula). We have shown that RBI produces more compact rules than other learning systems in standard benchmarks [6]. This is important in hand pose recognition because compact rules involving fewer features result in faster recognition.
- RBI can learn incrementally. Addition of new classes and presentation of new training observations of new and old classes can be accomodated by RBI. This is important for hand pose recognition because it permits existing systems to be customized for new users. It also permits gesture libraries to be extended without having to retrain the system on all previous observations.
- RBI can accommodate hand-crafted rules. This is related to the previous point. If some rules are known to have good discriminating power, these may be hand coded and introduced to the learning system as an existing rule set. For example, the designer may deem the number of visible fingers as a key feature and code this into the rulebase by hand.

Given a set of learned DNF formulae, a new observation may fall into one of three categories: 1. It may satisfy one and only one formula in which case it is assigned to the class represented by that formula; 2. It may satisfy more than one formula, in which case it may belong to either class or to some as yet unknown class; and, 3. It may satisfy no DNF formula. RBI is capable of two matching strategies: exact match and flexible match. In the former, only the first condition is considered a match. Conditions 2 and 3 yield 'no-match' results. In flexible matching, for condition 2, we examine the conjunction (rule) within each matching DNF which yielded the match. A count is kept of the number of training instances of that class which match that rule. For example, if rule k in DNF λ matches the new observation, we count the number of training examples τ_k^λ in class λ that

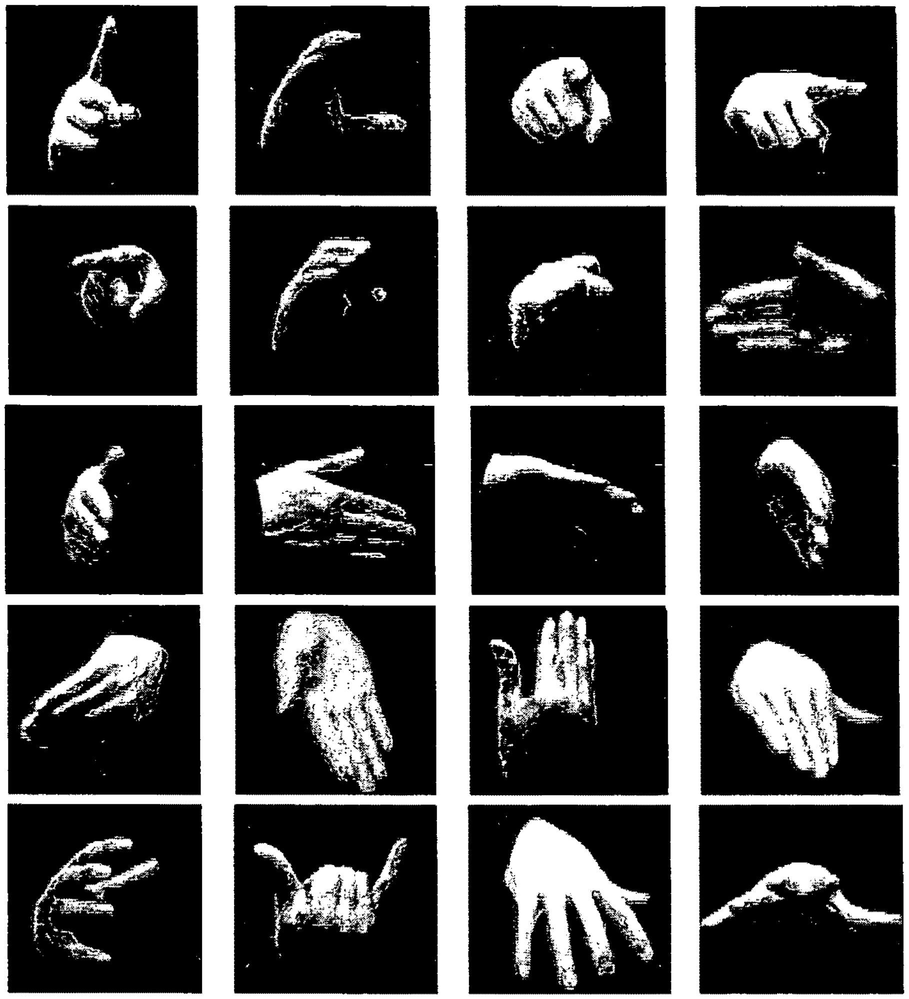


Figure 5: Samples of our Gesture Training Images

also matches rule k. The class χ with the highest similar training examples τ_k^χ is taken as the matching class. For condition 3, RBI counts the non-matching expressions within each rule in a DNF formula. The class whose DNF formula yields the lowest count is assigned to the new observation. Flexible matching is therefore able to also yield a 'match ranking' for classes based on these counts.

2.2.2 Pose Recognition Results

We evaluated our approach using a set of pose detection experiments. The gestures we used in our experiment were taken from our gesture lexicon for spatial description (figure 3).

Test	NT	NC	Accuracy	Test	NT	NC	Accuracy
EB	908	815	89.76%	EA	692	627	90.61%
FB	908	853	93.94%	FA	692	653	94.36%

Table 1: The Recognition Rate of Hand Gesture for Testing (EB, FB: Exact and Flexible matching for the Testing set Before removing the overlapping set, EA, FA: Exact and Flexible matching for the Testing set After removing the overlapping set, NT: The Number of the Testing data, NC: The Number of Correctly classified Poses)

In accordance with our gesture model described in section 2.1.3, we trained our system on 20 hand poses (classes) at the extrema of stroke of these gestures. Figure 5 shows representative instances of each of these classes. We applied the probabilistic color processing model described later in section 3.1.1 to detect and locate the hand. Our system used 28 features such as area of bounding box, compactness of hand, location of centroid in the bounding box, central moments, principle axes and normalized moments in the observation vectors. We trained our system on 934 frames (approximately 47 observations per pose). The system produced 42 rules to cover the 934 training observations.

After training, we tested the resulting rules against the training set observations and obtained a perfect recognition rate. We also tested the rules against 908 new test images and obtained a 94% recognition rate. Table 1 summarizes the results of pose recognition for the testing set before and after removing the overlapping examples (examples with identical feature vectors). Two matching approaches, i.e., exact matching and flexible matching, were used. Owing to quantization, some of the feature vectors were identical. The EB and FB tests were done with this conflicting data. EA and FA were done after the conflicting data were culled. From the table, we can see that the flexible matching can increase the recognition rate by about 4% in comparison with exact matching.

2.3 Gesture Dynamics

Gesture is a combination of hand pose and movement. We proceed to describe our system that determines hand motion directly from video input. Our system to track gestural motion involves a three-step algorithm:

1. We process the images to determine the location of moving edges.

Figure 6: Experimental Apparatus for Dynamic Gesture Acquisition

2. We compute the flow field describing the velocities of the moving edge points.

3. We cluster the vectors into coherent spatial-temporal vector flows to determine the path of the moving hand(s)

Figure 6 shows the apparatus with which we acquire dynamic visual data of hand gestures. An experimenter, who cannot see his/her hand or the workspace, pours sand into cups placed on a plexiglass tower. The subject, standing on a platform from which the entire workspace is visible, directs the experimenter to pour sand into the cups through a front-viewing camera connected to the experimenter's video monitor. The camera also records the subject's gestures onto Hi-8 videotape. A second video camera videotapes the subject from the side. A third camera videotapes both the subject and the experimenter for record keeping purposes. Although we do not currently use the data from the side-view camera, we calibrate the cameras so that we can obtain synchronized three-dimensional position information.

We designed this experiment to obtain subjects performing real gestures (as opposed to static or moving gesture poses for the sake of computer interpretation). The purpose of the experiment is to provide a sufficiently complex spatial communication task to elicit 'natural' gesturing by the subject, not to evaluate the ability of the experimenter to pour the sand accurately. We chose sand pouring as the task to eliminate tactile feedback which would be

Figure 7: Our current setup employs batch-mode data acquisition and post processing

a factor in pick-and-place or peg-in-hole tasks. The subject thus cannot infer such tactile feedback is available. The experimenter can see only the video monitor to ensure that the data available to the computer is the same as that available to the experimenter.

We use the setup shown in Figure 7 to digitize the video-tape to DAT tape. We capture a video stream of gesture interaction using a Hi-8 video camera and digitize the video using a Sony EVO-9650 computer controlled VCR. This system is able to capture full frame uncompressed data at 4 frames per second (120 $\times$ real-time). The algorithms described below were developed and tested using such data.

2.3.1 Moving Edge Detection

Our approach begins with an edge detector that detects moving edges. Details of our moving edge detector is presented in Sidebar 1.

Sidebar 1: Moving Edge Detector

Modeling the video data as a time varying image stream $I(x,y,t)$, we can estimate the partial derivative of the video data with respect to each of the variables to obtain $\frac{\partial I(x,y,t)}{\partial x}$, $\frac{\partial I(x,y,t)}{\partial y}$, and $\frac{\partial I(x,y,t)}{\partial t}$. The first two partial derivatives can be estimated using a Sobel operator, and the partial time derivative can be estimated by taking the difference image between successive video frames ($D(x,y,t) = I(x,y,t+\delta t) - I(x,y,t)$). We convert these images into spatial and temporal 'edge likelihood' images by normalizing their values to the interval [0,1], inclusive. This allows us to implement a *fuzzy-AND* operator on the images as a pixel-wise multiplication (i.e. $M_x(x,y,t) = \frac{\partial I(x,y,t)}{\partial x} \cdot \frac{\partial I(x,y,t)}{\partial t}$, $M_y(x,y,t) = \frac{\partial I(x,y,t)}{\partial y} \cdot \frac{\partial I(x,y,t)}{\partial t}$, $M_x'(x,y,t) = \frac{\partial I(x,y,t+\delta t)}{\partial x} \cdot \frac{\partial I(x,y,t)}{\partial t}$ and $M_y'(x,y,t) = \frac{\partial I(x,y,t+\delta t)}{\partial y} \cdot \frac{\partial I(x,y,t)}{\partial t}$.) Since the Sobel operator yields 'ridge-like' edges, we apply *non-maximum suppression* to obtain thin moving edge images $E_t(x,y,t)$ from $\mathbf{M}(x,y,t)$ and E_t' from $\mathbf{M}_t'(x,y,t)$.

Figure 8: Frames 1, 5, 9, and 13 of the 15 frame (0.5 second) gesture sequence

Figure 8 shows frames 1, 5, 9 and 13 of a typical video sequence processed by our system. This particular video sequence which contains 15 frames covering 0.5 seconds exhibits a fair amount of motion blur. This is typical of our data because of the speed of most hand gestures. We capture the data at 30 frames per second.

Figure 9 displays the moving edge points computed by our system after non-maximal suppression. This approach has several advantages over typical edge tracking approaches that first extract edges in each image and detect motion by correlation. First, our approach is more tolerant of noisy backgrounds. If the gesturer moves her hand against the background of a checkered shirt, our approach will suppress the edges on the stationary shirt and emphasize the boundary of the moving hand. Second, by delaying the application of a threshold until after the *fuzzy-AND* process, we reduce the introduction of non-linear thresholding errors. Finally, our approach is more efficient in that we do not have to consider non-moving edge points. Because of these features, our approach is well suited to the requirements of a gesture processing system.

Figure 9: Moving edges detected in frames 1, 5, 9, and 13 of the 15 frame (0.5 second) gesture sequence

2.3.2 Flow Field Computation

Once the moving edge points have been detected, we compute a flow field that represents the motion of these points through multiple video frames. Our approach involves selecting dominant edge points distributed across each image, computing an initial flow field, and smoothing this field by applying a field constraint. Details of our flow field computation is presented in Sidebar 2.

Sidebar 2: Flow Field Computation Algorithm

In the interest of reducing computational load, we select a subset of dominant points out of the set of moving edge points for tracking. To ensure a scattering of points, we divided the $E_t(x, y, t)$ image (we use 640×486 video streams) into 10×10 zones and selected at most one dominant point in each zone. Only points with more than 2 neighboring high gradient points are considered, and of these the one with the highest spatial-temporal gradient is chosen (some zones will have no dominant points). We denote the set of dominant points in $E_t(x, y, t)$ as $\mathbf{D}_t$. From these, we obtained an initial set of vector correspondences by performing an *absolute difference correlation* from $I(x, y, t)$ to $I(x, y, t + \delta t)$ for all $\mathbf{D}_t$. For efficiency, we performed the correlation only for points in $I(x,y,t+\delta t)$ with high spatial-temporal gradients in E'_t. This produces a list of possible correspondences $\mathbf{V}(x_e,y_e)$ for each $(x_e,y_e) \in \mathbf{D}_t$. This list is ordered by the ADC residues (which serves as a 'goodness-of-match' metric).

Taking the best $V(x_e,y_e) \in \mathbf{V}(x_e,y_e)$, we obtain a noisy initial field. We smooth out the vector field by applying the variance constraint described in equation 1 on the field.

$$\min \iint_A \frac{\frac{d\vec{V}}{dS}}{||\vec{V}||} dS = \sum_{(x_e,y_e)\in \mathbf{D}_t} \sum_{(x'_e,y'_e)\in \mathbf{D}_t} \left\{ \begin{array}{ll} \frac{||\vec{V}-\vec{V}'||}{||\vec{V}-\vec{V}'||\cdot||\vec{V}||} & \text{if } ||(x_e,y_e)-(x'_e,y'_e)|| \leq N \\ 0 & \text{otherwise} \end{array} \right\} \qquad (1)$$

where $(x'_e,y'_e) \neq (x_e,y_e)$; $\vec{V} = \vec{V}(x_e,y_e) \in \mathbf{V}(x_e,y_e)$ and $\vec{V'} = \vec{V}(x'_e,y'_e) \in \mathbf{V}(x'_e,y'_e)$ are the currently selected vectors for (x_e,y_e) and (x'_e,y'_e) respectively; and, velocity pairs outside the N-neighborhood do not affect the sum.

The goal, then, is to find the globally optimal $\vec{V}(x_e,y_e)$ at every chosen edge point from the candidates in $\mathbf{V}(x_e,y_e)$ which minimizes the global smoothness function. This general problem is NP-Complete. We use a greedy hill-climbing algorithm that works well for our data set. This approach minimizes the variance between points by switching velocity vectors at points and eliminating deviant velocity vectors.

The algorithm starts with the set of best local matches as an estimate of the field, and progressively refines it. Refinements are continued until the refinement would make a difference of less than a threshold or the maximum iteration count is exceeded. There are three refinements that can be made in one iteration. The operations in each iteration are summarized below:

1. Switch vector $\vec{V}(x_e,y_e)$ with another vector $(V_x,V_y) \in \mathbf{V}(x_e,y_e)$ that causes the greatest decrease in expression 1. The switch is only performed if the decrease is greater than a small threshold T_s. Refinements (2) and (3) are permitted if the switch decrease is less than a threshold $T_{sd} > T_s$.

2. Remove a deviant point (x_e,y_e) from $\mathbf{D}_t$. A point is deviant if removing it would decrease the sum by more than a threshold T_d. The most deviant point is always removed first.

3. Add non-deviant excluded point (x_e,y_e) back into $\mathbf{D}_t$. A point is non-deviant if adding

Figure 10: The smoothed velocity field computed in frames 1, 5, 9, and 13 of the 15 frame (0.5 second) gesture sequence

it would increase the sum by less than a threshold T_d. The most non-deviant excluded point is added first.

Figure 10 shows the smoothed velocity field computed on the video sequence shown in figure 8. In frame 13 (lower righthand image) of figure 10, the hand is coming to rest, and the cluster of short vectors labeled 'hand' describes the correct hand vectors. The set of longer vectors nearer the center of the frame actually originates at the location of the sleeve-hand boundary. Owing to the strength of the sleeve-hand edge points, and the regularity of the edge, a set of false matches were detected, giving rise to a larger-than-expected vector field in the direction of the hand movement. The determination of which vectors belong to the hand is done in a vector clustering process.

2.3.3 Field Clustering

We cluster the vectors to determine a congruent set of clusters which cohere spatially, temporally and in the direction of motion. Vectors within a frame which are close together and point in the same general direction are deemed to belong to the same cluster. The algorithm

Figure 11: Velocity plot of a left hand performing a 'right' gesture

determines the appropriate clusters in each frame in an iterative fashion. Each vector is considered in turn. If there is no existing cluster to which it is compatible according to said criteria, a new one is created to include it. If one exists, the vector is included to the cluster. Whenever a new vector is added to a cluster, the centroid of the vector origins and the average direction of the vectors in the cluster are updated.

Once the vectors in each frame are clustered, we apply a path cohesion criterion to track moving vector clusters across multiple frames. The location of centroid and average direction of each cluster is used to predict the location of the cluster in the next frame by linear projection. The fastest moving cohesive path is assumed to be the one representing the moving hand.

2.3.4 Dynamic Gesture Processing Results

Figure 11 plots the x and y components of the average velocity of the cluster. The vector velocities were computed by the absolute difference correlation and variance-based smoothing. The x-velocity shows a characteristic acceleration-deceleration curve as the hand moved from left to right. The y-velocity remained relatively constant for the horizontal gesture. Our system was able to process the full 640×486 video at a quarter of a frame per second on a 150 MHz Silicon Graphics Indigo^{II} entirely in software.

The velocity plot in figure 11 follows a typical acceleration/deceleration curve, and the hand never attains a constant velocity. Although we have not performed sufficient user

Figure 12: *FingerMouse* Setup

experimentation to confirm this observation, we believe that this is typical of communicative gestures from the datasets we have obtained thus far (extensive data acquisition on human subjects is impractical with our current setup shown in figure 7 because it takes 2 hours to digitize 1 minute of the monocular data, and puts an extraordinary amount of wear-and-tear on our VCR). Pragmatic hand movements are likely to undergo some degree of 'hand-eye-servoing' and exhibit vastly different dynamics. We are in the process of assembling a realtime data acquisition/processing system to confirm this hypothesis.

3 Two-Dimensional Pointing

FingerMouse is a two-dimensional pointing system targeted at interaction with graphical windowing systems. As the name implies, FingerMouse is intended as a freehand replacement of the regular mouse device to be used in conjunction with a keyboard. While the user operates the keyboard, her hands are monitored by a downward-looking camera. The system switches to pointing mode when it detects the user's hand in a pointing configuration (closed fist with one finger extended). The system then locks the graphical cursor to the detected finger location. The movement of the pointing finger horizontally in the plane above the keyboard is tracked and the graphical cursor moves accordingly. We chose not to use the vertical plane (the plane of the screen) for the pointing because earlier experience with touch

Figure 13: FingerMouse Processing Architecture

screens suggests that repetitive motions with the arm suspended in the air is extremely fatiguing. With horizontal plane motion, the user can support her wrist on a wristpad. The user registers a 'mouse button press' by depressing a key on the keyboard with the non-pointing hand. We chose this over having the user change the configuration of the pointing hand because it is difficult to maintain stable pointing while altering hand configuration. Figure 12 shows our *FingerMouse* system setup.

While the mouse has become the pointing device of choice for the majority of computer systems, it suffers from two notable deficiencies. First, it has been shown that the homing time (time required for the user to find the mouse and to return to the home row of the keyboard) takes up to 42% of the time in typical pointing tasks. Second, the mouse consumes valuable desk space. The FingerMouse was designed to address these deficiencies while taking advantage of the mouse's compatibility with modern windowing systems.

Several approaches have been attempted at freehand pointing and interaction on a desktop environment. Fukomoto, et.al. [8] report of a pointing system useful for applications requiring computer control from a distance, such as a slide presentation aid. In that configuration they proposed the virtual projection origin from which projectors originate and pass through the pointing fingertip toward the intended target. Wellner [9] discusses the *DigitalDesk*, a system that allows the user to interact with the computer while performing typical 'pen and paper' tasks on a desktop.

3.1 FingerMouse Processing

Figure 13 presents an overview of our runtime processing architecture. The first two steps run iteratively waiting for the predefined hand configuration. When the pointing configuration is detected, the fingertip tracking algorithm is engaged to locate the fingertip. The first time the pointing configuration is detected, the graphic cursor is locked to the fingertip location.

The fingertip displacement computed in each subsequent tracking iteration is used to update the cursor.

3.1.1 A Probabilistic Color Processing Model

For our system to operate reliably under varying conditions, it must be robust under varying lighting conditions (e.g. owing to varying illumination through the day and shadow effects). However, such robustness must be achieved efficiently at runtime to be able to provide useful interaction. We achieve this by employing a probabilistic color model which can be 'compiled' once during training and applied efficiently during regular operation.

Our approach proceeds on three basic assumptions. 1. Illumination may vary in intensity, but not significantly in spectral content. 2. A model may be customized for each user. 3. The imaging system produces 24-bit RGB outputs.

Given our first assumption, the hue-saturation-intensity (HSI) color space is an appropriate model. While RGB values vary with change in intensity, the HSI color model factors out intensity as a separate dimension. By ignoring that dimension, we can achieve a degree of intensity invariance by considering only the H-S color space. See Sidebar 3 for our HSI expression.

Sidebar 3: HSI Formulae

We apply the HSI expressions taken from [10]:

$$\begin{aligned} H' &= cos^{-1}\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}} \\ S &= 1-\frac{3}{R+G+B}[min(R,G,B)] \\ I &= \frac{1}{3}(R+G+B) \\ H &= 2\pi - H' \qquad \text{if } B > G \\ &= H' \qquad\qquad \text{otherwise} \end{aligned} \tag{2}$$

Figure 14 shows the hue and saturation histograms of a hand taken with varying camera exposures to simulate pure intensity shift. It is significant that the signals remain relatively stable even though only exposures 3–5 were in the camera's linear response range (the camera washes out at exposure 6, and had little response at exposures 1–2).

Figure 14: Hue and Saturation histogram at varying camera exposures

Our second assumption suggests that we can use a dedicated lookup table for each user. We use a training image of the user's hand on a black background and compute a training histogram $\mathcal{H}(\mathcal{S},\mathcal{I})$. Normalizing this histogram by its maximal value, we derive a table $\mathcal{L}(S,I) = \frac{\mathcal{H}(\mathcal{S},\mathcal{I})}{\max(\mathcal{H})}$ that maps a pixel's S-I values into a likelihood measure that a pixel is in the hand.

Given the third assumption we would like to have a transfer table based on RGB rather than H-S space for runtime operation. We obtain this by first passing the H-S histogram through a low-pass filter before we normalize it to obtain $\mathcal{L}(S,I)$. We then iterate through a 3D RGB likelihood table $\mathcal{L}(r,g,b)$ and fill each cell with the corresponding $\mathcal{L}(S(r,g,b),I(r,g,b))$ value.

3.1.2 Hand Pose Detection

The gesture to be detected and tracked is the typical pointing gesture with the index finger extended and other fingers closed. In this configuration, the fingertip is always the extremal point of the hand away from the body. This being the case, we need only consider the horizontal projection of the thresholded hand-likelihood image. We apply a finite state machine (FSM) detailed in [11, 12] to recognize the pointing configuration. We summarize the approach here.

The FSM takes as an input alphabet a quantization of the number of pixels likely to be in the hand in each consecutive image row. The quantization essentially classifies the likelihood that each row intersects with the finger, the closed hand, or nothing. Hidden states in the FSM encode the size of the finger and hand. When the terminal state of the FSM is reached,

Figure 15: Fingertip tracking

the hand pose is recognized.

To locate the fingertip position, we applied a rapid midpoint projection approach shown in figure 15. We experimented with more costly moments-principal axis-based approaches and found the midpoint projection more stable and less susceptible to noise. Figure 15 shows the fingertip located by the algorithm marked with cross-hairs.

3.2 User Experiments

The algorithm described in the previous section was able to track the fingertip reliably at 7 fps on a 150 MHz Silicon Graphics IndigoII entirely in software. This suggests that realtime operation is possible with parallelization and optimization. For current user experiments, however, 7 fps is far too slow. For our user experiments, we simplified the processing by placing a blue dot on the fingertip. Since the naturally distributed spectrum of a color occupies a planar parallelogram in RGB space, the blue dot can be detected by a set of three intersecting planar equations. This permitted our system to operate at 15 fps (about the same as our frame capture speed).

3.2.1 Experiment Design

Our user subjects were 24 students taken from the author's Computer Graphics and Object Oriented Programming classes. The subjects were therefore familiar with mouse usage and the windowing paradigm. The experiments involved measuring the time required by the subjects to select fields specified fields from the form shown in figure 16. We ensured that the pointing morphology using the blue dot was identical to the original formulation by

Figure 16: The form used in the experiments

requiring subjects to perform the pointing gesture when the fingertip is tracked. To minimize the effects of search time for the fields, the form was addressed by row (number) and column (letter) addresses.

Each experiment comprised 14 trials with a sequence of entries printed on a sheet constituting a *trial*. To obviate the need for the subject to take his/her eyes off the screen, the field specifications (Letter-number sequences) were printed in 36 point font and pasted to the screen next to the form. Each trial comprised a set of *entries*. For each *entry*, the subject was required to move a screen cursor into the specified field, select the field by hitting the `TAB` key with the off-hand, type '`Chicago`' into the selected field, and hitting the `RETURN` key to end the entry. The upper-case 'C' made certain that each selection began with both hands were homed on the keyboard. Since there was no hand configuration information to select pointing mode, the `RETURN` key reenabled finger tracking. The system automatically logs the time between each `RETURN` and `TAB` as the selection time of the entry. Pointing speed was computed as a ratio of time to distance between fields. Each *trial* is initiated by the experimenter hitting a key on the numeric keypad with the subject's hands on the home row of the keyboard. Each subject was briefed about the interaction methodology and permitted to familiarize herself with both the interface and the entry task.

The 14 trials were divided into three experimental groups: *random* trials, *directional* trials, and *mouse* trials. Each subject received a unique set of 5 random trials comprising 5 entries each. Each entry field specification was generated using a uniform random number. The purpose of the random trials was to estimate the rate of skill acquisition. The 8 directional trials corresponded to the 8 chessboard directions. Each trial comprised 3 entries in a directional sequence (moving right, up-right etc.) Since the subject pool was not large

Figure 17: Average Learning Curve

enough to randomize 8 trials reasonably, we accounted for ordering effects by ordering the trials in 3 different way (8 subjects performed each ordering). No systematic differences were observed among the three orderings. The purpose of the directional trials was to ascertain if the interaction approach has some directional bias. For the final trial, each subject repeated the fifth random trial in her set. The purpose of the mouse trial was to compare performance for the two input methodologies.

3.2.2 Experimental Results

Figure 17 shows the average speed for the random trials across subjects. The learning curve is saturated by the third trial. This correlated with our observations that the subjects performed awkwardly in the first trial and rapidly achieved proficiency. This was supported by our subject questionnaire in which most subjects agreed with the proposition that they were more comfortable with the *FingerMouse* as the experiment progressed (4.167 over a 1–5 agreement scale).

Figure 18 summarizes the directional trial results. The data indicates that *FingerMouse* has a bias against vertical (up-down) motion. This was not due to ordering effects since subjects performing all three orderings exhibited the same tendency. We believe that the bias, which we had not expected, is related to hand anatomy. For horizontal and diagonal motions, the user could move the shoulder, elbow, wrist and knuckle joints of the index finger, but the only joints which contribute to vertical motion of the fingertip are the shoulder and elbow joints. While this is a concern, it is probably not fatal to the interaction methodology.

Figure 18: Average Directional Speed

Figure 19: Scatter Plot of Mouse vs *FingerMouse* Speed

Most pointing schemes suffer from some directional bias – what is important is that the bias does not render the interaction scheme unusable. The anatomical constraints which contribute to the slowness of *FingerMouse* vertical motions is present in the regular mouse in all directions (elbow and shoulder motion account almost entirely for all manipulation of mouse position).

Figure 19 shows a scatter plot of mouse speed vs *FingerMouse* speed for each subject performing the same random trial. The performance was positively correlated at 69.3% indicating some degree of skill transference. The subjects performed only approximately 18% poorer on the *FingerMouse*. Three subjects even performed better on the *FingerMouse* than on the mouse. This is surprising because the *FingerMouse* operated at 15 fps at much lower resolution (320 $\times$ 200) as opposed to the ideal 60 samples-per-second of the mouse.

4 Conclusion

We have presented our work on unencumbered hand gesture interfaces. This work encompasses both three-dimensional interaction and two-dimensional pointing.

In three-dimensional gestures, we motivated our computational approach by a brief overview of human hand gesture interaction. Our model requires the determination of the gestural stroke, recognition of hand poses at the extrema of stroke and determination of the dynamics of hand motion during the stroke. We presented our work on the inductive learning of hand gesture poses using *extended variable-valued logic* and our *rule-based induction* algorithm. We were able to attain a 94% recognition rate with our system. We also discussed our work in the computation of the image flow fields representing the moving hand. Our experimental results show the efficacy of our algorithm.

In two-dimensional gestures, we discussed our freehand pointing system known as *FingerMouse*. We presented our system which is able to recognize the pointing hand pose and track the moving fingertip close to realtime in software. We presented the results of user experiments which show that *FingerMouse* is a promising mode of interaction.

References

[1] Adam Kendon, "Current issues in the study of gesture", in J-L Nespoulous, P. Peron, and A.R. Lecours, editors, *The Biological Foundations of Gestures:Motor and Semiotic Aspects*, pp. 23–47. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[2] David McNeill, *Hand and Mind*, University of Chicago Press, Chicago, 1992.

[3] Francis Quek, "Toward a vision-based hand gesture interface", *in Virtual Reality Software and Technology Conference*, pp. 17–29, Singapore, Aug. 23-26 1994.

[4] Francis Quek, "Eyes in the interface", *International Journal of Image and Vision Computing*, vol. 13, pp. 511–525, Aug. 1995.

[5] J.-L. Nespoulous and A.R. Lecours, "Gestures: Nature and function", in J-L Nespoulous, P. Peron, and A.R. Lecours, editors, *The Biological Foundations of Gestures:Motor and Semiotic Aspects*, pp. 49–62. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[6] Meide Zhao and Francis Quek, "RBI:A rule-based induction approach and its application in hand pose recognition", Technical report, Vision Interfaces and Systems Laboratory, EECS Dept., The University of Illinois at Chicago, 1996, Submitted to the IEEE Transactions on Pattern Recognition and Machine Intelligence.

[7] R.S. Michalski, "A theory and methodology of inductive learning", in Carbonell J.G. Michalski R.S. and Mitchell T.M., editors, *Machine Learning, An Artificial Intelligence Approach*, pp. 83–130. Tioga Publishing, Palo Alto, 1983.

[8] Masaaki Fukumoto, Kenji Mase, and Yasuhito Suenaga, "Real-time detection of pointing actions for a glove-free interface", *in Proceedings of IAPR Workshop on Machine Vision Applications*, Tokyo, Japan, Dec. 1992.

[9] Pierre Wellner, "Interacting with paper on the DigitalDesk", *Communications of the ACM*, vol. 36, pp. 87–95, July 1993.

[10] Rafael Gonzalez and Richard E. Woods, *Digital Image Processing*, Addison Wesley, New York, 1992.

[11] F.K.H. Quek, M.D. Zhao, and T. Mysliwiec, "Freehand pointing with fingermouse", *in submitted to:ACM Transactions on Computer-Human Interaction*, 1996.

[12] F.K.H. Quek, T. Mysliwiec, and M.D. Zhao, "FingerMouse: A freehand pointing interface", *in International Workshop on Automatic Face- and Gesture-Recognition*, pp. 372–377, Zurich, Switzerland, June 1995.

HUMAN FACTORS, 1997, 39(1), 130–140

Activation Force and Travel Effects on Overexertion in Repetitive Key Tapping

ROBERT G. RADWIN,[1] *University of Wisconsin, Madison, Wisconsin, and* ONE-JANG JENG, *New Jersey Institute of Technology, Newark, New Jersey*

Key switch design parameters, including make force, make travel, and over travel, were investigated for minimizing operator-exerted force while maximizing key-tapping speed. A mechanical apparatus was designed, constructed, and used for independently controlling key switch parameters and for directly measuring finger exertions during repetitive key tapping using strain gauge load cells. The task for the 25 participants involved using the index finger of the dominant hand to repeatedly depress a single key as rapidly as possible. Participants received visual and auditory feedback upon a successful keystroke. Peak force exerted decreased 24% and key-tapping rate increased 2% when over travel was distended from 0.0 to 3.0 mm. Although peak force exerted was not significantly affected by make point travel, key-tapping rate increased 2% when make point travel was reduced from 4.0 to 1.0 mm. These results indicate that key switch mechanisms that provide adequate over travel might enable operators to exert less force during repetitive key tapping without inhibiting performance.

INTRODUCTION

This research is motivated by reports of upper extremity maladies among intensive keyboard users (Bernard, Sauter, Fine, Petersen, & Hales, 1994; McPhee, 1982; NIOSH, 1990; Rose, 1991). Force is one factor often cited as increasing the risk for localized fatigue and musculoskeletal disorders. Studies have shown that the force exerted by keyboard operators during keying markedly exceeds the force necessary to activate the keys (Armstrong, Foulke, Martin, Gerson, & Rempel, 1994; Feuerstein, Armstrong, & Hickey, 1994). The actual force that a typist exerts can be affected by numerous circumstances, including keyboard, workstation, and psychosocial factors, and individual factors, such as typing experience and proficiency. The current study is concerned with the effects of physical characteristics of key design on finger exertions in repetitive key tapping.

The physical characteristics of key switches are often described by their *make* and *break points*, which are measured from associated force and travel parameters. *Key force* is the force applied against the key cap. Because key force is usually opposed by a linear or nonlinear spring, when key force is applied, the key is depressed a corresponding displacement, defined as *key travel*. Make occurs when the switch makes electrical contact and the circuit is activated. Break occurs when the switch breaks electrical contact and the circuit is deactivated.

[1] Requests for reprints should be sent to Robert G. Radwin, Department of Industrial Engineering, University of Wisconsin-Madison, 1513 University Ave., Madison, WI 53706.

Because there is often hysteresis in switch mechanisms, the break point is not necessarily equivalent to the make point. *Make point force* is therefore the key force that must be applied in order to activate the key, and *break point force* is the key force that must be released in order to deactivate the key. Correspondingly, *make point travel* is the key travel necessary to activate the key. *Over travel* is defined as the maximum travel a key can be depressed beyond the make point travel until the key hits bottom. The actual key force and travel characteristics depend on both the mechanical and electrical designs of the particular key switch.

The *American National Standard for Human Factors Engineering of Visual Display Terminal Workstations* (HFES, 1988) contains requirements for the force necessary for activating keys and associated key displacement. The standard specifies key activation force ranges between 0.25 and 1.5 N with a key displacement between 1.5 and 6.0 mm and a preferred displacement between 2.0 and 4.0 mm. The force required for activating keys, however, does not necessarily reflect the actual exertions operators make when using a keyboard. Peak force during keying has been measured to be as much as 2.5 to 4.6 times the required activation force (Armstrong et al., 1994; Feuerstein et al., 1994; Martin et al., 1994).

Previous studies have shown that applied finger force increases with keyboards that have greater make forces (Armstrong et al., 1994; Rempel, Klinenberg, et al., 1994). Armstrong et al. (1994) compared three keyboards with similar layouts but different make force and key travel characteristics. Although different applied forces were observed among the keyboards, each one had specific key switch parameters, so these effects are confounded.

Rempel, Klinenberg, et al. (1994) performed a similar study and also found that applied key force was affected by make force, whereas other key switch characteristics (i.e., total travel and tactile feedback) were held constant. Neither of these investigations could account for the combined effects of keyboard force and travel characteristics on applied finger force.

The purpose of this study was to systematically investigate keyboard design factors that affect operator exertion. Key switch design parameters were investigated that maximize key-tapping speed while minimizing peak key force exerted. The make point force, make point travel, and over travel for the key switch covered a wide range of parameters currently recommended for keyboards.

METHODS

Apparatus

An experimental apparatus was designed and constructed for manipulating key force and travel characteristics while measuring the actual force exerted during key tapping. The apparatus, illustrated in Figure 1, contained a leaf spring mechanism for simulating the basic linearized force-displacement characteristics of a single key. The spring element was made from a spring steel strip that was rigidly attached as a cantilever beam at one end; force was applied against a plastic key top mounted on the other end. Different-height mechanical stops were used to control total key travel distance.

The spring force-displacement parameters were controlled using different-size spring steel strips (Blake, 1985). The linear force-displacement relationship for a simple deflected beam can be described by the following equation:

$$D = FK = F\frac{L^3}{3EI},$$

where the beam deflection (mm) is D, the applied load (g) is F, and the constant K is a function of the strip length L (in millimeters), Young's modulus of elasticity for steel, E (200 000 N/mm^2), and the moment of inertia, I (mm^4). The moment of inertia was determined using the following equation:

$$I = \frac{WH^3}{12},$$

where W is the strip width (in millimeters) and H is the strip thickness (in millimeters).

Consequently, the force-displacement parameters

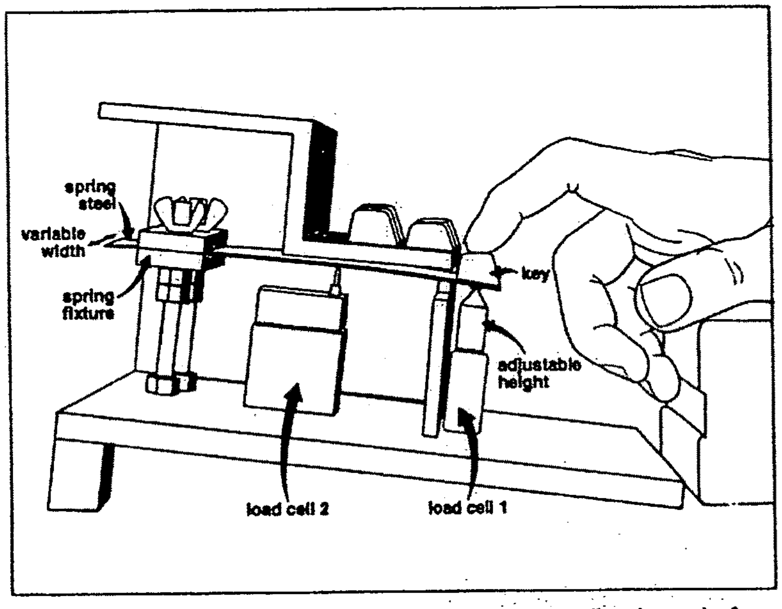


Figure 1. Experimental apparatus used for independently controlling key make force, make travel, and over travel. Key force-displacement characteristics were determined by the length, width, and thickness of the deflected spring steel strip. Load Cell 1 measured finger force after the key hit bottom. Load Cell 2 measured finger force exerted before the linear key spring static deflection limit was exceeded. Make force was controlled through software.

could be controlled simply by changing the spring steel strip thickness and width. These equations were used for estimating the spring steel strip sizes needed. Each strip width was cut slightly wider than estimated and calibrated. Calibration was accomplished by directly measuring the deflection using a digital micrometer and associated force against the strain gauge load cell and trimming each strip as necessary. Every strip had a fixed length (55 mm). Dimensions for make point force and make point travel characteristics used in this experiment are included in Table 1.

Finger force is registered through two strain gauge load cells (see Figure 1). Load Cell 1 is an 11.12 N Transducer Techniques™ (Temecula, CA) Model MDB-2.5, and Load Cell 2 is 0.5 N Transducer Techniques™ Model GS-500. When the key is depressed, finger force is initially transmitted to Load Cell 2 as the leaf spring bends. When the force is large enough to displace the leaf spring to the static limit of the mechanical stop, Load Cell 1 registers the additional force. The force exerted at the key is a weighted sum of the force measured by the two load cells. This

TABLE 1

Spring Steel Beam Dimensions and Corresponding Make Point Force and Travel Parameters

Make Point Force (N)	Make Point Travel (mm)	Thickness (mm)	Width (mm)
0.31	1.0	0.64	7.3
0.31	2.5	0.38	13.6
0.31	4.0	0.38	8.5
0.51	1.0	0.64	12.2
0.51	2.5	0.38	22.6
0.51	4.0	0.38	14.2
0.71	1.0	0.64	17.1
0.71	2.5	0.64	6.8
0.71	4.0	0.38	19.8

force measurement was calibrated against known weights using linear regression.

Daytronic™ Model 9878A strain gauge conditioning amplifiers provided excitation, amplification and filtering of the load cell signals. A MetraByte™ Model DASH-16 12-bit data acquisition board sampled the analog data from each load cell at a 200-Hz sample rate using an IBM-PC microcomputer.

A plastic keyboard cap was mounted on the spring containing the letter *M*. A cosmetic enclosure for the experimental apparatus obscured visual cues (see Figure 1). Dummy keys were mounted in proximity to the active key relative to a conventional QWERTY keyboard. The simulated keyboard was inclined toward the operator 6.2° with respect to the horizontal.

The participant sat in front of a display screen in a posture similar to that when working at a computer workstation. The simulated keyboard was located 73.5 cm from the floor. The palm rested on a rubber pad with the fingers extended, similar to a typing posture. The height and location of the palm rest was adjustable so that every participant addressed the key in the same manner using the end of the distal finger pad.

Key activation was controlled through software. A successful keystroke occurred when exerted force exceeded a specific make force. Key travel and associated force were dependent on the particular spring constant used. Successive keystrokes could not occur until the key was released below a specific break force. Break force in this study was fixed at 80% of the make force.

Participants were provided discrete visual and auditory feedback on achieving the make force. The screen displayed a letter *M* every time a successful key strike occurred, similar to keying in a word-processing program. An auditory click was simultaneously presented through headphones. White noise and headphones were used for masking apparatus noise and external distractions during the experimental trials.

Procedure

The experimental task involved using the index finger of the dominant hand to repeatedly depress the key. Participants were instructed to type the letter *M* with the index finger as fast as possible. Only one key was repeatedly pressed, so participants were instructed to fully release the key so that the index finger was horizontal each time the key was struck in order to better simulate keying.

After becoming familiar with the task, participants practiced the key-tapping task until they said they were comfortable with the instructions and felt ready to begin the experiment. Three practice trials were then provided, using three different combinations of make force, make travel, and over travel.

Every trial lasted 13 s. The key-tapping rate was displayed to participants in terms of words per minute (five characters per word) after each trial. Participants were provided a 1½-min break between experimental conditions. Every trial contained a new experimental condition. Participants were permitted to practice as long as they desired before each new experimental condition. After participants practiced each condition, each one initiated a new trial with his or her first keystroke. An experimental session typically lasted 1 h.

Participants

The 25 participants were recruited through advertisements posted on campus bulletin boards and through electronic mail broadcasts to undergraduate engineering students. All applicants completed a demographics questionnaire. Eligible applicants verified that they were free of hand conditions, disorders, or injuries that might affect typing performance, and all used computer keyboards on a regular basis.

The 15 men and 10 women selected ranged in age from 18 to 41 years (mean = 25 years, *SD* = 6 years) and were paid a nominal fee for their services. Of the 25 participants, 22 were right handed and three were left handed.

Experimental Design

The experiment consisted of a 3 × 3 × 3 repeated-measures full-factorial design. The keyboard design factors of interest included key make force, make travel, and over travel. Make point force conditions included 0.31 N (30 g), 0.51 N (50 g), and 0.71 N (70 g). Make point travel distances were 1.0, 2.5, and 4.0 mm. Over travel conditions were 0.0, 1.5, and 3.0 mm. A summary of the experimental conditions, including associated key force-displacement characteristics, is illustrated in Figure 2.

All 27 experimental conditions were counterbalanced within and between subjects. Data for the first 3 s (warm-up) of each 13-s trial were discarded, and data recorded for the next 10 s were retained for analysis.

Performance was measured as the actual force exerted and the rate that keystrokes were produced. Dependent variables in this experiment included peak key force and key-tapping rate. Peak force was the mean of the maximum force exerted for every keystroke during 10 s. Key-tapping rate was measured as the average number of keystrokes produced per second during the 10-s time window.

The static limit of deflection for a spring bounds its linear force-displacement characteristics, where applied force beyond that limit produces no corresponding displacement. This discontinuity is observed in a key switch when the key is depressed with sufficient force to displace it to its travel limit and the key contacts the bottom of the switch mechanism. Because the leaf spring has a linear force-displacement characteristic, the *bottoming rate*—the frequency that the key hits bottom—was estimated from the peak force mean and standard deviation as the probability that the key force exerted was sufficient to displace the key to its travel limit.

Repeated-measures analysis of variance was used for studying the significance of the independent variables and their interactions on keying performance. Significant contrasts were tested using the Tukey multiple contrast test.

RESULTS

Peak Exertion Force

The minimum average peak force was 1.04 N (SD = 0.61 N) and occurred when make force was 0.31 N, make travel was 4 mm, and over travel was 3 mm. The maximum average peak force was 1.79 N (SD = 0.60 N), which occurred when make force was 0.71 N, make travel was 1 mm, and over travel was 0 mm.

The relationship between average peak force exerted and key make point force is plotted in Figure 3a. Average peak force significantly decreased 0.22 N (15%), $F(2, 48) = 26.46$, $p < .01$, when the make point force was reduced from 0.71 N to 0.31 N. Multiple contrasts indicated that the average peak force exerted for pairwise differences among all three make force levels differed for a significance level of $p < .01$.

No significant peak exertion force effect was observed when key make point travel increased from 1.0 to 4.0 mm, $F(2, 48) = 2.77$, $p > .05$. Average peak force exerted for each make point travel distance is plotted in Figure 3b.

Peak force exerted was significantly affected by key over travel, $F(2, 48) = 75.21$, $p < .01$, as illustrated graphically in Figure 3c. As over travel was augmented from 0 to 3 mm, average peak force exerted decreased by as much as 0.38 N (24%). Multiple contrasts among all combinations of the three over travel conditions were significant for $p < .01$.

Although the Make Point Travel × Over Travel interaction was statistically significant, $F(4, 96) = 3.49$, $p < .05$, the effect accounted for only 0.91% of the total variance. Multiple contrasts indicated no significant ($p < .01$) peak force differences across make point travel distances for any given over travel condition. Alternatively, peak force exerted was significantly different ($p < .01$) among all over travel conditions across all make point travel conditions, except when make point travel was 1.0 mm and over travel distance was 1.5 mm or 3.0 mm.

Bottoming Rate

The frequency that the experimental apparatus was depressed with sufficient force to hit bottom was estimated as the probability that peak force exceeded the level needed to hit bottom using the average and standard deviation of the peak force measured, which was normally distributed. These probabilities are plotted respectively in

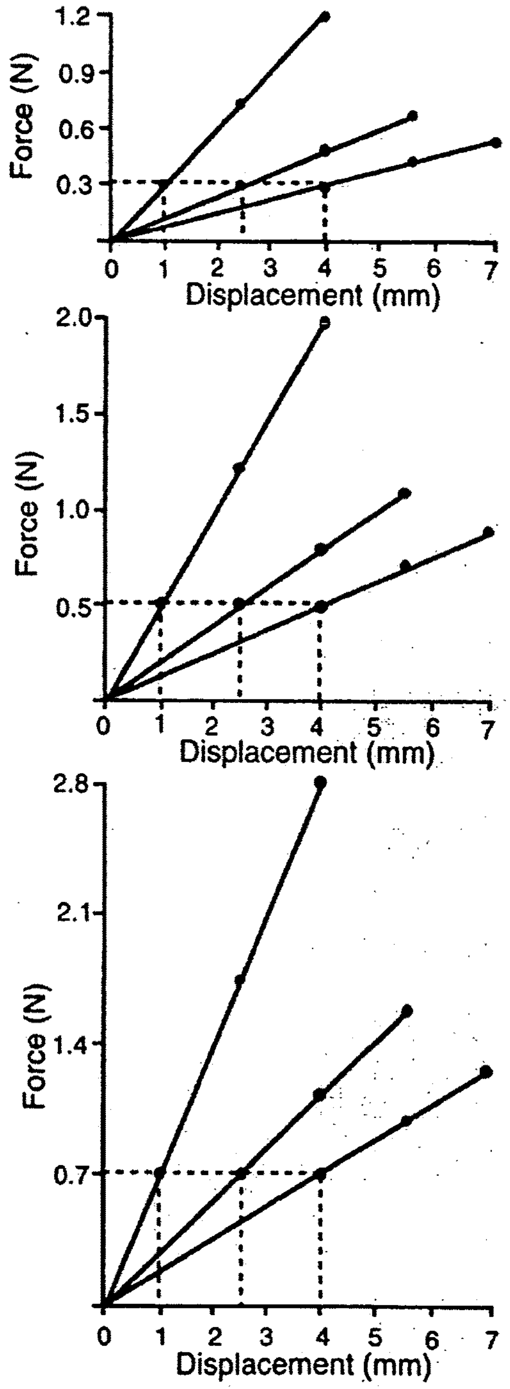


Figure 2. Key force-displacement, key travel, and over travel parameters for all 27 experimental conditions. Make point force and travel are indicated by the intersection of horizontal and vertical dotted lines, and over travel limits are indicated by dots. All graphs are plotted on the same scale.

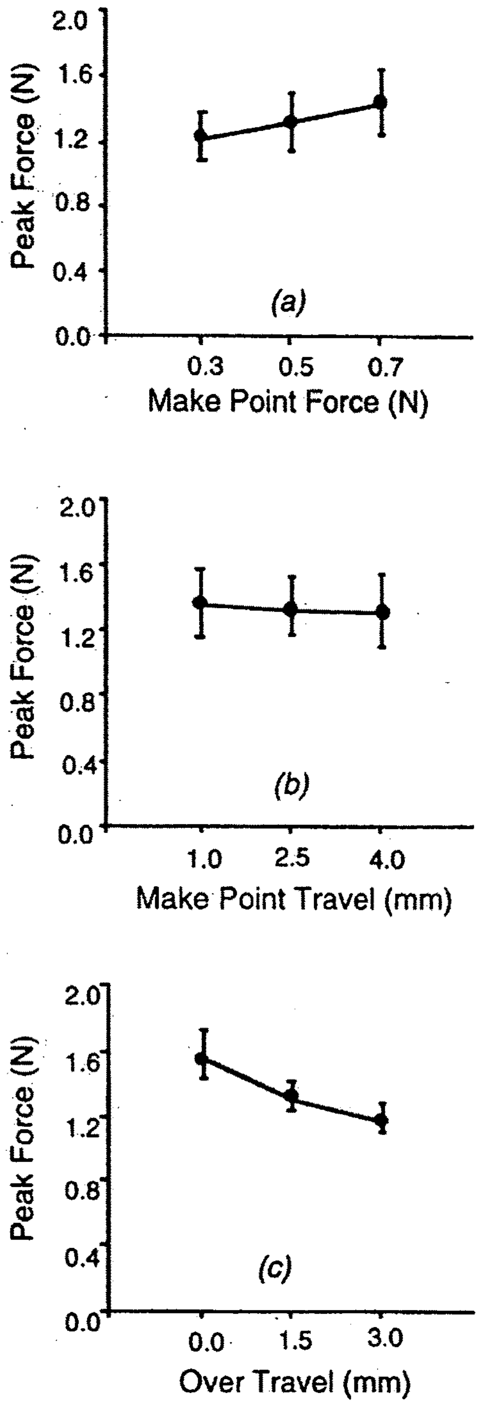


Figure 3. Average peak force exerted versus (a) make point force, (b) make point travel, and (c) over travel (25 participants).

Figures 4a, 4b, and 4c against the make point force, make point travel, and over travel conditions. The smallest average bottoming rate among all experimental conditions was 0%, which occurred when make point force was 0.51 or 0.71 N, make travel was 1 mm, and over travel was 3.0 mm.

Little correlation was observed between bottoming rate and peak exertion force. Correlation between average bottoming rate and average peak exertion force was only .36. The condition in which make point force was 0.31 N, make point travel was 4.0 mm, and over travel was 3.0 mm resulted in the least exertion force, although the bottoming rate for this condition was 88%.

Key-Tapping Rate

Key-tapping rate significantly increased 0.2 keys/s (4%), $F(2, 48) = 22.30$, $p < .01$, when the key make point force decreased from 0.71 to 0.31 N. Average key-tapping rate is plotted against make point force in Figure 5a. Multiple contrasts revealed that average key-tapping rate was significantly different among all pairwise combinations of the three make point force levels for a $p < .05$ significance level, but only contrasts between the 0.31 N and the other make point force levels were significantly different for $p < .01$.

Make point travel had a significant effect on key-tapping rate, $F(2, 48) = 9.67$, $p < .01$. Average key-tapping rate increased 0.13 keys/s (2%) as make point travel was reduced from 4.0 to 1.0 mm, as shown in Figure 5b. Pairwise contrasts between average key-tapping rate for a 4.0-mm make point travel distance and the other two make point travel conditions were significant for $p < .01$ using the Tukey test.

Key-tapping rate significantly increased 0.13 keys/s (2%) as over travel distance increased from 0.0 to 3.0 mm, $F(2, 48) = 76.54$, $p < .01$. Average key-tapping rate is plotted against over travel in Figure 5c. Multiple pairwise contrasts between average key-tapping rate for 0.0 mm and the other two over travel conditions were significant for $p < .01$.

DISCUSSION

This study provided a test paradigm for investigating operator keying behavior in terms of both keying speed and overshoot force under different key activation characteristics, including required force and key displacement, without being limited by a specific keyboard. Minimum peak exertion force and maximum key-tapping performance occurred when make force was 0.31 N and over travel was 3.0 mm. Although there was no significant make point travel effect for peak exertion force, bottoming rate and key-tapping rate were significantly greater when make point travel was 1.0 mm.

An explanation for reduced exertions when over travel is increased may come from the small increment in force from the over travel while the finger decelerates against the resistance of a spring. When a finger strikes a key, it collides with the key top in order to rapidly produce sufficient force to displace the key and to activate the switch. After the key is depressed and make force is achieved, the finger has to decelerate to reduce its velocity and reverse direction in order to release the key. The added over travel may enable the finger to decelerate and reverse direction against the incremental opposing force of the spring without colliding against the key bottom while finger velocity is high.

Greater levels of peak key force, however, were not dependent only on hitting bottom, considering that there was no correlation between bottoming rate and peak force. The least peak force was exerted when make force was small and over travel was high, although the bottoming rate was substantial. Consequently, key bottoming alone was not the only factor in overshoot force production. We have speculated that when over travel was sufficient, the finger was able to decelerate and thus collide with the bottom with less velocity. This should be investigated in future studies.

Rempel, Dennerlein, Mote, and Armstrong (1994) observed ballistic finger motion during typing and recorded peak fingertip velocities during the keystroke phase between 0.3 and 0.7 m/s.

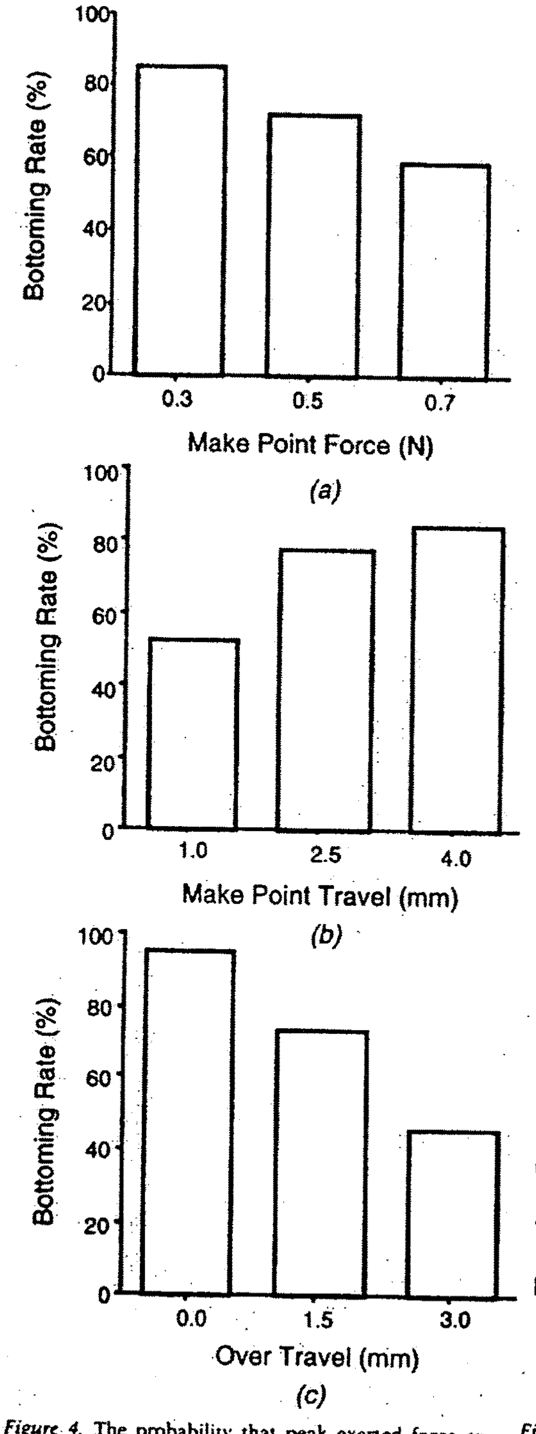


Figure 4. The probability that peak exerted force exceeds the key static limit of deflection for conditions of (a) make point force, (b) make point travel, and (c) over travel (25 participants).

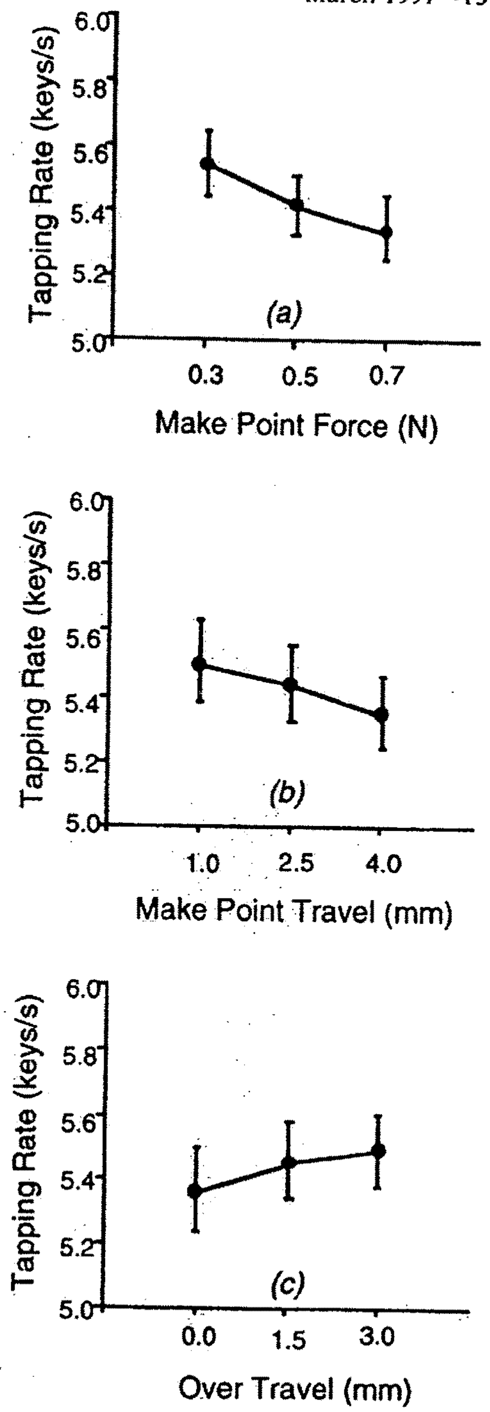


Figure 5. Average key tapping rate versus (a) make point force, (b) make point travel, and (c) over travel (25 participants).

They observed that velocity was relatively constant during initial key strike and through the make point. Key-tapping rate increased in the current study when over travel increased, but it decreased when make point travel increased (see Figure 5). This may be because increased over travel may provide additional proprioceptive feedback.

Proprioception should be enhanced with increased over travel because it increases finger articular motion when striking the key. Greater finger joint displacement and tendon travel might enrich stimulation of joint capsule receptors and muscle spindles. Consequently, finger force may be better controlled when sufficient over travel is provided because force feedback is coupled with key displacement, which is proportional to the key force applied. When a key strikes bottom, the key force can increase without additional key travel and provide less force feedback.

Although increased over travel enables the finger to decelerate before colliding with the key bottom, increasing make point travel requires the finger to displace a greater distance at a constant velocity before achieving the necessary make point travel, resulting in a smaller tapping rate. In any regard, the magnitude in key-tapping rate changes observed when make point travel was increased was only 0.1 keys/s, which would be equivalent to 1.2 words/min (assuming five characters per word). The effect on actual typing performance, however, should be investigated in the future.

The results of the current study agree with the findings of previous investigations (Armstrong et al., 1994; Rempel, Klinenberg, et al., 1994). Rempel, Klinenberg, et al. (1994) observed no difference in applied finger force when typing on keyboards with 0.28 and 0.56 N make point forces. Mean applied finger force was 0.85 and 0.86 N, respectively. A notable increase in applied finger force (1.2 N) was observed, however, when a keyboard with a 0.83 N make point force was used. The better control and isolation of key switch parameters in the current study may have provided more resolution with regard to the make point force effect, based on the significant difference in peak force observed between 0.31 and 0.51 N make point force.

All make point forces included in this study conformed with the ANSI/HFS 100-1988 standard (HFES, 1988), although they included only the lower half of the recommended range. The smallest peak force exertions in the current study occurred when total travel was 7 mm, which exceeds the ANSI recommendations for overall key travel. It may be possible to reduce the overall travel by designing a nonlinear spring that has a greater spring constant for 1 mm make travel and then decreases stiffness after make force is achieved in order to decrease resistance while increasing over travel. This will be the subject of a future study.

Similar to the findings in the current study, Loricchio (1992) reported that input speed was faster when keyboard activation force was 0.58 N than when activation force was 0.74 N for keys having the same key travel. Typists preferred the lower-force keyboard. Although the average key rate was 4.538 keys/s for a 0.58 N keyboard and 4.192 keys/s for a 0.74 N keyboard, the average numbers of words mistyped were not different. Although no difference in typing speed was observed between low-force (0.36 and 0.43 N) and high-force (0.71 N) keyboards, Akagi (1992) found that the two low-force keyboards produced 23% more errors than did the high-resistance keyboards. Maximum keying rates have also been reported when both key activation force and key displacement are small (Rose, 1991).

The results of the current study provide important information about force and displacement parameters for designing keyboards that minimize exertions. We anticipate that reducing overshoot force can ultimately lead to reduced stress from repetitive keyboard use. Stronger exertions may be associated with upper limb disorder symptoms (Feuerstein et al., 1994).

Jeng, Radwin, and Rodriquez (1994) observed a similar outcome for repetitive pinching in carpal tunnel syndrome. When rapidly pinching a strain gauge dynamometer, participants exerted an average of 0.52 N more than the required force, which was set between 5% and 50% of the participant's maximum strength. Participants diagnosed with carpal tunnel syndrome exerted 82% more overshoot force during rapid pinch and release than did control participants free of carpal tunnel syndrome. Because carpal tunnel syndrome often impairs sensory nerves, these results provide evidence supporting the role of sensory feedback in force control. It is not yet known, however, whether increased key force exerted is a symptom or a causation factor.

Although investigators agree that peak applied force is reduced when make force is reduced, there are additional constraints on this design objective. The minimum force necessary to activate a key is limited by the force necessary to prevent accidental activation. In order to prevent the fingers from accidentally activating keys, operators would encounter increased forearm extensor static muscle contraction if the activation force were set too light. Rose (1991) measured participants' effective finger force when resting against a keyboard and concluded that in order to avoid accidental key activation (which may result in static postures in which the fingers are withdrawn from the keys and extensor muscle contraction is increased), at least 0.5 N of key activation force is needed. Increasing key over travel provides an alternative to reducing make force.

Although the key apparatus in this study was designed to have characteristics corresponding to keyboards, the linear spring mechanism was a convenient approximation; most keyboards today contain nonlinear elastomer collapsible dome spring elements. Brunner and Richardson (1984) reported fewer errors and faster typing for both skilled and occasional typists on keyboards equipped with elastomer key mechanisms. Akagi (1992), however, found little difference in typing performance and preference among touch typists using linear spring keys without tactile feedback and keys that provided snap-action tactile feedback. Domes develop a rapid breakaway after a critical force (and corresponding displacement) is exceeded, causing them to collapse and produce their characteristic click.

In addition to providing kinesthetic feedback from depressing the key, the apparatus used in the current investigation provided visual and auditory feedback but no tactile snap. An investigation similar to the current study is needed to replicate these findings using a key mechanism that provides tactile feedback.

This study, however, does reveal that key switch parameters can have a remarkable effect on key-tapping performance and exertions. Because the current study used only a single key to investigate some design factors, it may not truly represent typing, considering that the biomechanics of the fingers are different when working in isolation compared with when they work in concert with the other fingers. A full-scale keyboard with the design features in the current study could be developed for follow-up in order to study actual keying behavior. Furthermore, the long-term effects of typing with keys containing different characteristics should be considered.

CONCLUSIONS

This study demonstrated that applied force during repetitive key tapping can be controlled by reducing make point force or increasing over travel. This finding is significant because it offers an alternative design objective to reduce make point force. Make point force reduction could be undesirable if it results in increased accidental key activation, and it could require additional muscular effort from antagonists in order to prevent unintentional key activation when resting the fingers. Alternatively, excessive over travel may not be practical for compact keyboards, such as laptop computers. Hence a suitable trade-off between these two design factors should be considered.

ACKNOWLEDGMENT

This research was sponsored by a grant from the Office Ergonomics Research Committee, Manchester Center, Vermont.

REFERENCES

Akagi, K. (1992). A computer keyboard key feel study in performance and preference. In *Proceedings of the Human Factors Society 36th Annual Meeting* (pp. 523-527). Santa Monica, CA: Human Factors and Ergonomics Society.

Armstrong, T. J., Foulke, J. A., Martin, B. J., Gerson, J., & Rempel, D. M. (1994). Investigation of applied forces in alphanumeric keyboard work. *American Industrial Hygiene Association Journal, 55*, 30–35.

Bernard, B., Sauter, S., Fine, L., Petersen, M., & Hales, T. (1994). Job task and psychosocial risk factors for work-related musculoskeletal disorders among newspaper employees. *Scandinavian Journal of Work Environment and Health, 20*, 417–426.

Blake, A. (1985). *Handbook of mechanics, materials, and structures*. New York: Wiley.

Brunner, H., & Richardson, R. M. (1984). Effects of keyboard design and typing skill on user keyboard preferences and throughput performance. In *Proceedings of the Human Factors Society 28th Annual Meeting* (pp. 267–271). Santa Monica, CA: Human Factors and Ergonomics Society.

Feuerstein, M., Armstrong, T. J., & Hickey, P. (1994). Keyboard force, upper extremity symptoms, perceived effort and mood in symptomatic and asymptomatic word processors. In *Proceedings of the 12th Triennial Congress of the International Ergonomics Association*. Toronto: Human Factors Association of Canada.

Human Factors and Ergonomics Society. (1988). *American national standard for human factors engineering of visual display terminal workstations* (ANSI/HFS 100-1988). Santa Monica, CA: Author.

Jeng, O. J., Radwin, R. G., & Rodriquez, A. A. (1994). Functional psychomotor deficits associated with carpal tunnel syndrome. *Ergonomics, 37*, 1055–1070.

Loricchio, D. F. (1992). Key force and typing performance. In *Proceedings of the Human Factors Society 36th Annual Meeting* (pp. 281–282). Santa Monica, CA: Human Factors and Ergonomics Society.

Martin, B. J., Armstrong, T. J., Foulke, J. A., Natarajan, S., Klinenberg, E., Serina, E., & Rempel, D. (1994). Finger force during computer keyboard work: Part I. Relation of keyboard reaction force to finger flexor muscles surface EMG. In *Proceedings of the 12th Triennial Congress of the International Ergonomics Association* (Vol. 2, pp. 198–200). Toronto: Human Factors Association of Canada.

McPhee, B. (1982). Deficiencies in the ergonomic design of keyboard work and upper limb and neck disorders in operators. *Journal of Human Ergology, 11*, 31–36.

National Institute of Occupational Safety and Health. (1990). *Health hazard evaluation report* (Tech. Report HETA 89-250-2046). Cincinnati, OH: Author.

Rempel, D., Dennerlein, J., Mote, C. D., & Armstrong, T. J. (1994). A method of measuring fingertip loading during keyboard use. *Journal of Biomechanics, 27*, 1101–1104.

Rempel, D., Klinenberg, E., Serina, E., Martin, B. J., Armstrong, T. J., Foulke, J. A., & Natarajan, S. (1994). Finger force during computer keyboard work: Part II. Relation of key switch make force to applied force and surface EMG. In *Proceedings of the 12th Triennial Congress of the International Ergonomics Association* (Vol. 2, pp. 201–203). Toronto: Human Factors Association of Canada.

Rose, M. J. (1991). Keyboard operating posture and activation force: Implications for muscle over-use. *Applied Ergonomics, 2*, 198–203.

Robert G. Radwin earned his Ph.D. in industrial and operations engineering from the University of Michigan and was a postdoctoral research fellow at the Center for Ergonomics. He is a professor in the Department of Industrial Engineering and chair of the Biomedical Engineering Program at the University of Wisconsin-Madison.

One-Jang Jeng received his Ph.D. degree in industrial engineering from the University of Wisconsin-Madison. He is an assistant professor in the Department of Industrial and Manufacturing Engineering at New Jersey Institute of Technology.

Date received: October 20, 1995
Date accepted: March 25, 1996

SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces

Jun Rekimoto
Interaction Laboratory
Sony Computer Science Laboratories, Inc.
3-14-13 Higashigotanda
Shinagawa-ku, Tokyo 141-0022, Japan
Phone: +81 3 5448 4380
Fax: +81 3 5448 4273
Mail: rekimoto@acm.org
http://www.csl.sony.co.jp/person/rekimoto.html

ABSTRACT
This paper introduces a new sensor architecture for making interactive surfaces that are sensitive to human hand and finger gestures. This sensor recognizes multiple hand positions and shapes and calculates the distance between the hand and the surface by using capacitive sensing and a mesh-shaped antenna. In contrast to camera-based gesture recognition systems, all sensing elements can be integrated within the surface, and this method does not suffer from lighting and occlusion problems. This paper describes the sensor architecture, as well as two working prototype systems: a table-size system and a tablet-size system. It also describes several interaction techniques that would be difficult to perform without using this architecture.

Keywords
Interactive surfaces, gesture recognition, augmented tables, two-handed interfaces, touch-sensitive interfaces.

INTRODUCTION
Many methods for extending computerized workspace beyond the computer screen have been developed. One goal of this research has been to turn real-world surfaces, such as tabletops or walls, into interactive surfaces [23, 21, 16, 20, 9]. The user of such a system can manipulate, share, and transfer digital information in situations not associated with PCs. For these systems to work, the user's hand positions often must be tracked and the user's gestures must be recognizable to the system. Hand-based interaction offers several advantages over traditional mouse-based interfaces, especially when it is used in conjunction with physical interactive surfaces.

While camera-based gesture recognition methods are the most commonly used (such as [24, 13, 9]), they often suffer from occlusion and lighting condition problems. To correctly capture hand images on a surface, a camera must be mounted above the table or in front of the wall. As a result, the system configuration becomes complex, making it difficult to implement the system as a portable (integrated) unit. The use of magneto-electric sensors (e.g., Polhemus [15]) is another possible sensing method, but it requires attaching a tethered magneto-electric sensor to each object being tracked.

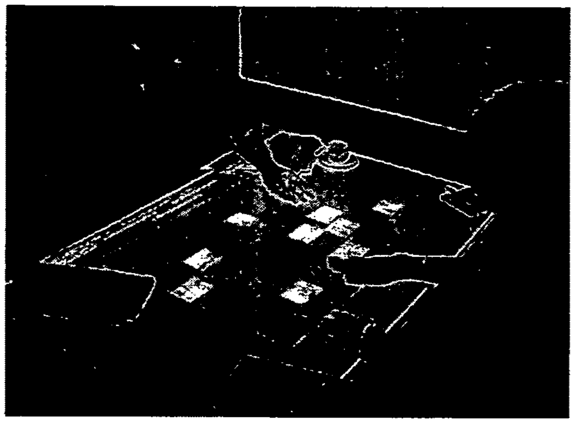


Figure 1: An interactive surface system based on the SmartSkin sensor.

This paper introduces a new sensing architecture, called SmartSkin, which is based on capacitive sensing (Figure 1). Our sensor accurately tracks the position of the user's hands (in two dimensions) and also calculates the distance from the hands to the surface. It is constructed by laying a mesh of transmitter/receiver electrodes (such as copper wires) on the surface. As a result, the interactive surface can be large, thin, or even flexible. The surface does not need to be flat – i.e., virtually any physical surface can interactive. By increasing the density of the sensor mesh, we can accurately determine the shape of the hand and detect the different positions of the fingers. These features enable interaction techniques that are beyond the scope of normal mouse-based interactions.

BEST AVAILABLE COPY

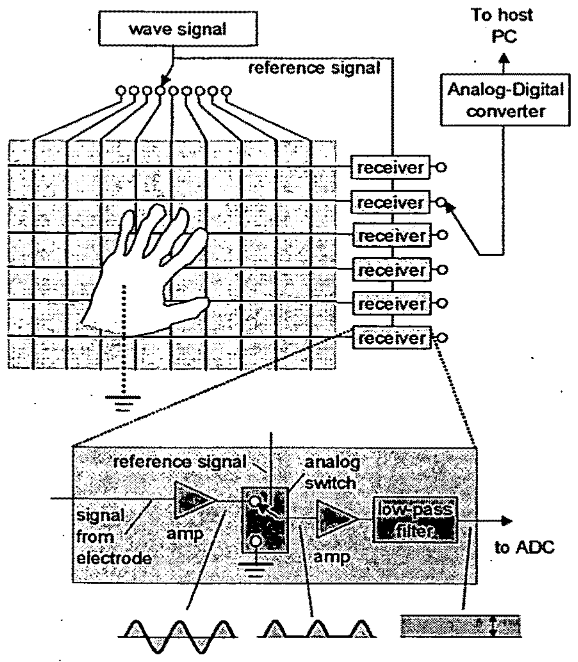


Figure 2: The SmartSkin sensor configuration: A mesh-shaped sensor grid is used to determine the hand's position and shape.

We describe the sensing principle of SmartSkin and two working systems: an interactive table system and a hand-gesture sensing tablet. We also describe new interaction techniques of these systems.

SMARTSKIN SENSOR ARCHITECTURE

Figure 2 shows the principle of operation of the SmartSkin sensor. The sensor consists of grid-shaped transmitter and receiver electrodes (copper wires). The vertical wires are transmitter electrodes, and the horizontal wires are receiver electrodes. When one of the transmitters is excited by a wave signal (of typically several hundred kilohertz), the receiver receives this wave signal because each crossing point (transmitter/receiver pairs) acts as a (very weak) capacitor. The magnitude of the received signal is proportional to the frequency and voltage of the transmitted signal, as well as to the capacitance between the two electrodes. When a conductive and grounded object approaches a crossing point, it capacitively couples to the electrodes, and drains the wave signal. As a result, the received signal amplitude becomes weak. By measuring this effect, it is possible to detect the proximity of a conductive object, such as a human hand.

The system time-dividing transmitting signal sent to each of vertical electrodes and the system independently measures values from each of receiver electrodes. These values are integrated to form two-dimensional sensor values, which we called "proximity pixels". Once these values are obtained, algorithms similar to those used in image processing, such as peak detection, connected region analysis, and template matching, can be applied to recognize gestures. As a result, the system can recognize multiple objects (e.g., hands). If the granularity of the mesh is dense, the system can recognize the shape of the objects.

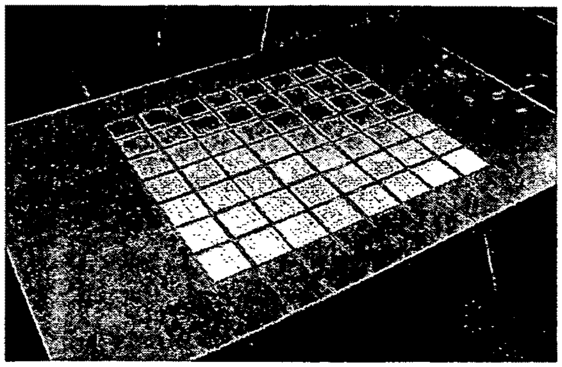


Figure 3: Interactive table with an 8 × 9 SmartSkin sensor: A sheet of plywood covers the antennas. The white squares are spacers to protect the wires from the weight of the plywood cover.

The received signal may contain noise from nearby electric circuits. To accurately measure signals only from the transmitter electrode, a technique called "lock-in amplifier" is used. This technique uses an analogue switch as a phase-sensitive detector. The transmitter signal is used as a reference signal for switching this analog switch, to enable the system to select signals that have the synchronized frequency and the phase of the transmitted signal. Normally, a control signal needs to be created by phase-locking the incoming signal, but in our case, the system can simply use the transmitted signal, because the transmitter and the receiver are both on the same circuit board. This feature greatly simplifies the entire sensor design.

We chose a mesh-shaped electrode design for our initial experiment because of its simplicity and suitability for sensing hand shapes as pixel patterns. Other layouts are possible, depending on the application requirements. For example, the density of the mesh can be adjusted. In addition, since the electrodes are simply thin copper wires, it is possible to create a very thin interactive surface such as interactive paper, which can even be flexible.

PROTOTYPE 1: AN INTERACTIVE TABLE

Based on the principle described above, we developed two interactive surfaces: a table-size system that can track multiple hand positions, and a smaller (and more accurate) system that uses a finer electrode layout.

The table system is constructed by attaching sensor elements to a wooden table. A mesh-like antenna, made of polyurethane-coated 0.5 mm-thick copper wire, is laid on the tabletop. The number of grid cells is 8×9, and each grid cell is 10×10 cm. The entire mesh covers an 80×90 cm area of the tabletop

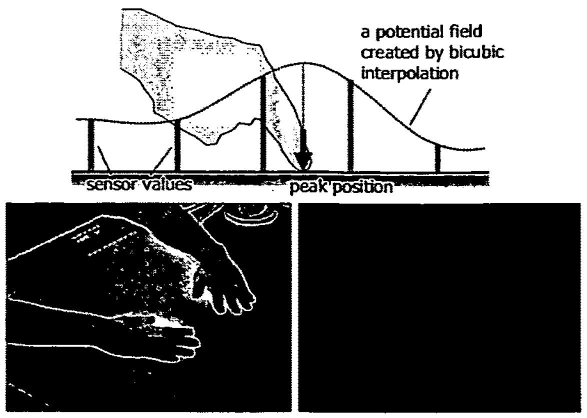


Figure 4: top: A bicubic interpolation method is used to detect the peak of the potential field created by hand proximity. bottom: arms on a table and a corresponding potential field.

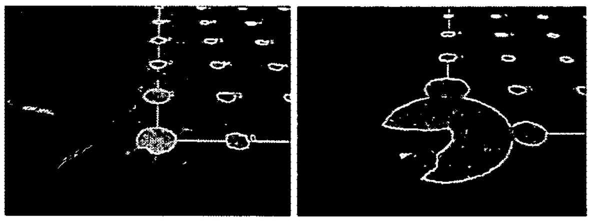


Figure 5: Relationship between distance between hand and sensor and sensed values. The diameter of the circle represents the amplitude of the sensed value.

(Figure 3). A plywood board covers the antennas. A signal transmitter / receiver circuit is attached to the side of the table. Two Atmel AVR microprocessors control this circuit. One microprocessor generates square-wave signals (400 KHz) with firmware that directly controls the I/O port, and the other microprocessor with a built-in A/D converter measures the values of the received signals and transmits them to the host computer. A projector is used to display information on the table. The current implementation is capable of processing 8×9 sensor values 30 times per second.

When the user's hand is placed within 5-10 cm from the table, the system recognizes the effect of the capacitance change. A potential field is created when the hand is in the proximity to the table surface. To accurately determine the hand position, which is the peak of the potential field, a bicubic interpolation method is used to analyze the sensed data (Figure 4). By using this interpolation, the position of the hand can be determined by finding the peak on the interpolated curve. The precision of this calculated position is much finer than the size of a grid cell. The current implementation has an accuracy of 1 cm, while the size of a grid cell is 10 cm.

As for the distance estimation, although there is no way to directly measure the precise distance between the hand and the table surface, we can estimate relative distance. Figure 5 shows the relationship between the hand position and obtained A/D-converted values. Our system enables detecting various levels of hand proximity, which is difficult to do with other technologies such as computer vision.

Figure 6: Mouse emulation by using calculated hand position. The distance between the hand and the surface is used to determine button-press and button-release states.

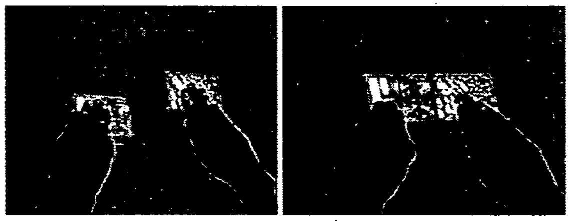


Figure 7: Two-handed operation is used to concatenate two objects.

Since each point on the grid can independently measure the proximity of an object, the system can simultaneously track more than one hand. This feature is important because many table-based applications are used by more than one user.

Interaction techniques

We studied two types of basic interaction techniques for this platform. One is 2D-position control with distance measurement, and the other uses a sensor potential field as input.

Mouse emulation with distance measurement The first interaction technique is the simple emulation of a mouse-like interface. The estimated 2D position is used to emulate moving the mouse cursor, and the hand-surface distance is used to emulate pressing the mouse button. A threshold value of the distance is used to distinguish between pressed and released states that the user can activate "mouse press" by touching the table surface with the palm, and move the cursor without pressing the mouse button by touching the table surface with the fingers. Normally, touch-sensitive panels cannot distinguish between these two states, and many interaction techniques developed for the mouse (such as "mouse over") cannot be used. In contrast, an interactive table with a SmartSkin sensor can emulate most mouse-based interfaces. Figure 6 shows how the user "drags" a displayed object.

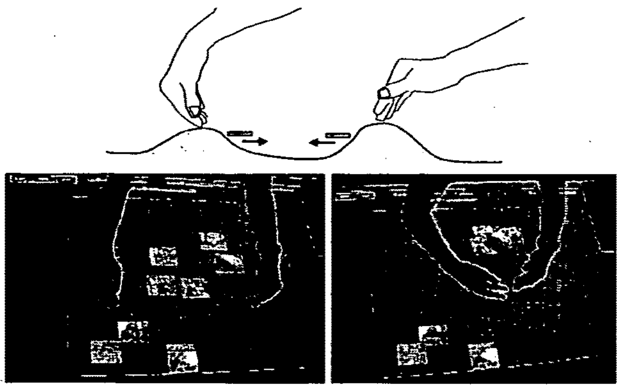


Figure 8: Shape-based object manipulation. The potential field created by the hand's proximity to the table is used to move objects. The user can use both hands or even entire arms to manipulate objects.

A notable advantage of SmartSkin over traditional mouse-based systems is its natural support for multiple-hand, multiple-user operations. Two or more users can simultaneously interact with the surface at the same time. The multiple-hand capability can also be used to enhance object manipulation. For example, a user can independently move objects with one hand. He or she can also "concatenate" two objects by using both hands, as shown in Figure 7, or can take objects apart in the same manner.

Shape-based manipulation The other interaction technique, which we call "shape-based manipulatio", does not explicitly use the 2-D position of the hand. Instead, a potential field created by sensor inputs is used to move objects. As the hand approaches the table surface, each intersection of the sensor grid measures the capacitance between itself and the hand. By using this field, various rules of object manipulation can be defined. For example, an object that "descend" to a lower potential area repels from the human hand. By changing the hand's position around the object, the direction and speed of the object's motion can be controlled.

We implemented this interface and observed how users tried to control objects. Overall, the reaction to the interface was quite encouraging. The people were quickly able to use this interface even though they did not fully understand the underlying dynamics. Many users naturally used two hands, or even arms. For example, to move a group of objects, one can sweep the table surface with one's arm. Two arms can be used to "trap" and move objects (Figure 8).

PROTOTYPE 2: A GESTURE-RECOGNITION PAD

The table prototype demonstrates that this sensor configuration can be used to create interactive surfaces for manipulating virtual objects. Using a sensor with a finer grid pitch we should be able to determine the position and shape of the hand more accurately. In addition, if the sensor can sense more than one finger position, several new interaction techniques are possible. For example, a 3D-modeling system often requires manipulation of multiple control points such as curve control points. Normally, a user of traditional mouse-based interfaces has to sequentially change these control points one by one. However, it would be more efficient and more intuitive if the user could control many points simultaneously.

The second prototype uses a finer mesh pitch compared to that of the table version (the number of grid cells is 32 × 24, and each grid is 1 × 1 cm). A printed circuit board is used for the grid electrodes (Figure 9). The prototype uses the bicubic interpolation algorithm of the interactive table sys-

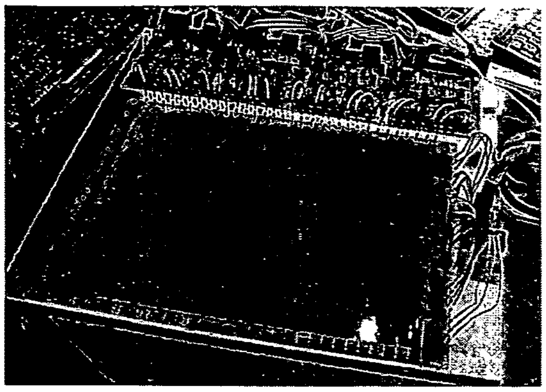


Figure 9: A gesture-recognition pad made up of a 32×24 grid mesh. A sheet of plastic insulating film covers Sensor electrodes.

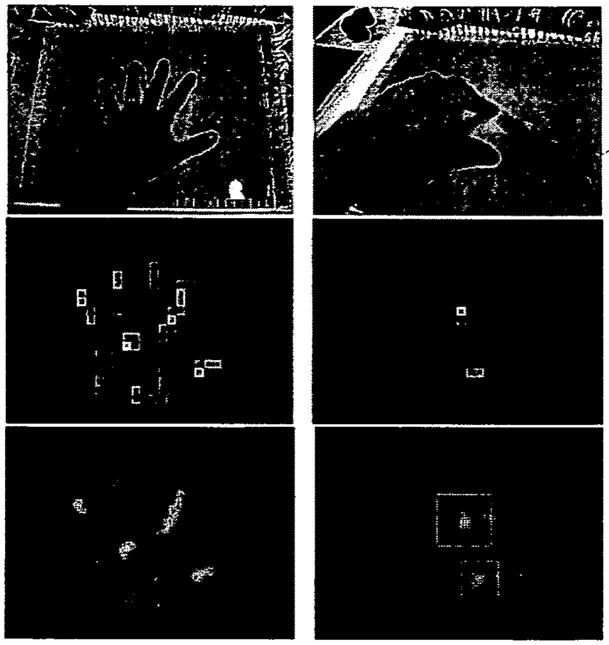


Figure 10: Gestures and corresponding sensor values. (top: a hand on the sensor mesh, middle: raw input values, bottom: after bicubic interpolation)

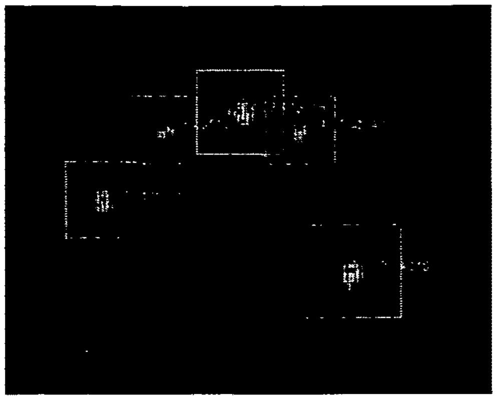


Figure 11: Fingertip detection.

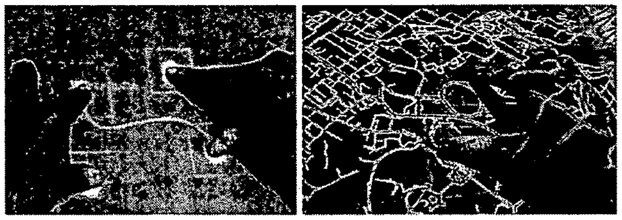


Figure 12: Examples of uses of multiple-finger interfaces: left: curve editing. right: a map browsing system. The user can use one finger for panning, or two or more fingers for simultaneous panning and scaling.

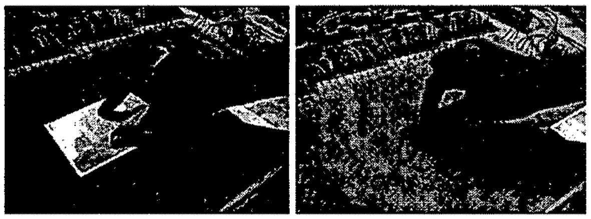


Figure 13: Two-finger gestures can be used to "pick-up" objects.

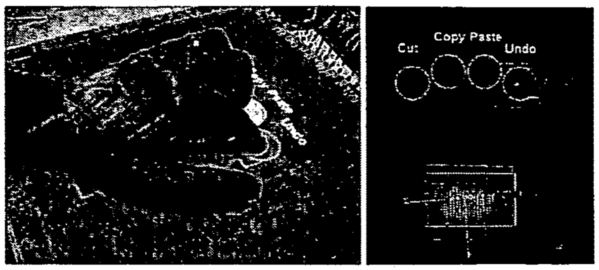


Figure 14: A palm is used to trigger a corresponding action (opening menu items). The user then taps on one of these menu items.

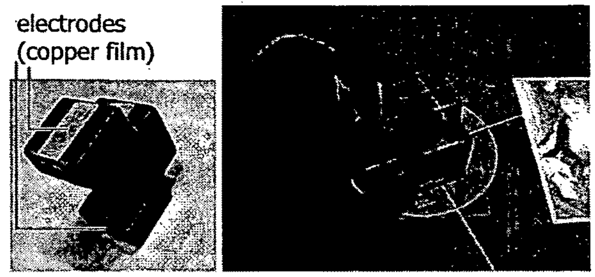


Figure 15: The "capacitance tag": a conductive pattern attached at the bottom of an object is used to identify this object.

tem, and it can determine the human hand shape as shown in Figure 10. The peak detection algorithm can also be used, and in this case, the algorithm can track multiple positions of the fingertips, not just one position of the hand (Figure 11).

Interactions by using fingers and hand gestures

We studied three possible types of interaction for this platform. The first one is (multiple) finger tracking. Here, the user simultaneously controls several points by moving his or her fingertips. The second is using hand or finger shape as input, and the third is identifying and tracking physical objects other than the user's hands.

A typical example of a situation in which the multi-finger interface is useful is diagram manipulation. A user can simultaneously move and rotate a displayed pictogram on the surface with two fingers. If three or more fingers are used, the system automatically uses a least-squares method to find the motion (consisting of moving, rotating, and scaling) that best satisfies the constraints given by the fingers.

Another simple example is the expression of attributes during manipulation. For example, the user normally drags a scroll bar with one finger, but to increase the scrolling ratio, he or she could use two or more fingers.

Figure 12 shows a map browsing system. The user scrolls the map by sliding a finger along the sensor surface. The scrolling speed increases with the number of fingers in contact with the surface. If the user touches the surface with two or more fingers, by changing the distance from the fingers to the surface, he/she can control the map scale. Simultaneous control of scrolling and zooming is intuitive, because the user feels as if his or her fingers are fixed to the map's surface.

Other possibilities we have explored include gesture commands. For example, two fingers moving toward the center of an object represent a "picking up" action (Figure 13), while a similar outward motion represents a "dropping" action. There are probably many other actions or functions representable by multi-finger gestures, for example, those based on the geographical relations between tapped fingers.

An example of using a hand shape as input is shown in Figure 14. In this example, the user places a hand on the surface, its shape is recognized by the system, and a corresponding action, in this case, "showing menu item", is triggered. The action is selected by template matching. The system first lists up connected regions (a group of sensor values that are connected), and then calculates the values of correlation between the stored templates. The system selects the region with the highest correlation value, and if this value exceeds a predetermined threshold value, the corresponding action is activated. In Figure 14, the user first touches the surface with his/her palm, then selects one of the displayed menu items.

Capacitance tags

While exploring several hand-based interaction techniques, we also found a way to make the SmartSkin sensor interact with objects besides than the hand. This feature can support graspable / tangible user interfaces [2, 8].

The principle of this method, called "capacitance tags", is shown in Figure 15. The capacitance tag block is made of a dielectric material such as wood or plastic. Some parts of this tag block are coated with a conductive material such as copper film. These conductive areas are connected to each other (by a copper wire, for example). This wire also connects the conductive areas at the bottom and at the top of the block.

When this block is placed on the SmartSkin surface, the sensor does not detect the capacitance change because the block is ungrounded. However, when a user grasps it (and touches the conductive area at the top), all the conductive areas become grounded, and areas corresponding to the conductive parts coated at the bottom of the block can be detected. Since the geometrical relationship (e.g., the distance between conductive areas) is predetermined, the system can distinguish these patterns from other patterns created when the user moves his/her hands or fingers. Essentially, the combination of conductive areas works like a barcode. In addition, the geometry of the patterns indicates the position and orientation of the tag block. Simultaneous object identification and position tracking is a key technology for many post-GUI user interface systems (such as [21, 22, 16]), and this method should be a new solution for such systems.

Another advantage of this capacitance tag method is its ability to support simultaneous hand gestures. For example, a user places a capacitance tag block on an interactive surface, and then issues a "data transfer" command by hand-dragging the displayed object toward the block.

DISCUSSIONS

Design issues

Most computers now use mice as input devices. With a mouse, the user controls one 2D position on the screen, and uses various interaction techniques, such as clicking or dragging. Although the mouse is a popular input device, its 'way' of interaction is different from the way manipulate objects in our daily lives. In the real world, we often use multiple fingers or both hands to manipulate a physical object. We control several points on the object's surface by touching, not by using "one position of the cursor" as in GUI systems. Consequently, with mouse-based interfaces, we have to unnaturally decompose some tasks into primitive operations.

In addition, our ability to interact with the physical environment is not limited to the control of multiple points. Hands and fingers can also create various phenomena, such as pressure. As a result, interaction becomes more subtle and analogue.

Related work

Capacitive sensing for human-computer interaction The idea of using capacitive sensing in the field of human-computer interfaces has a long history. Probably the earliest example is a musical instrument invented by Theremin in the early 20th century, on which a player can control the pitch and volume by changing the distance between the hand and the antenna. Other examples include a "radio drum" [11], which is also an electric musical instrument, and Lee et al.'s multi-finger touch panel, which has a sub-divided touch-sensitive surface [10].

Zimerman et al.'s work [26] pioneered the sensing of an electric field as a method for hand tracking and data communication (e.g., "personal area network" [25]). Although there has been a lot of research in this area, interaction techniques, like the ones described in this paper, have not been studied extensively. Our other contributions to this field are the new electrode design that enables accurate and scalable interactive surfaces, and the creation of tagged physical objects that can be used in combination with hand gestures.

Hinkely et al. showed how a simple touch sensor (which is also based on a simple capacitive sensor) can enhance existing input devices such as a mouse or a trackball [6].

Vision-based gesture recognition There have been a number of studies on using computer vision for human gesture recognition [7]. However, achieving robust and accurate gesture recognition in unconditioned environments, such as the home or office, is still difficult. The EnhancedDesk [9] uses a thermo-infrared camera mounted above the table to extract the shape of the hand from the background. In contrast to these vision-based approaches, our solution does not rely on the use of external cameras, and all the necessary sensors are embedded in the surface. As a result, our technology offers more design flexibility when we implement systems.

Other types of vision-based systems include HoloWall [13] and Motion Processor [14]. Both systems use a video camera with an optical infrared filter for recognition, and infrared lights are used to illuminate objects in front of the camera. While Motion Processor directly uses this infrared reflection, HoloWall uses a diffuser surface to eliminate the background image. "Barehand" [19] is an interaction technique for a large interactive wall. It enables recognizing hand shapes by using a sensor similar to that of HoloWall, and it uses the shapes to trigger corresponding actions. Using infrared reflection, the system can detect not only the shape of the hand, but also its distance from the camera. As a result, gestures that cannot be recognized by other vision-based systems, such as moving a finger vertically over a surface (i.e., tapping), can be detected. However, like other vision-based systems, these systems also require the use of external cameras and lights, and thus they cannot be integrated into a single unit.

Bimanual interfaces Various types of bimanual (two-handed) interfaces (for example, see [1, 5, 17] and [4] for physiological analysis of these interfaces) have been studied. With such an interface, the user normally holds two input devices (e.g., a trackball and a mouse), and controls two positions on the screen. For example, the user of ToolGlasses [1] controls the tool-palette location with his/her non-dominant hand, while the cursor position is controlled by the user's dominant hand. Some bimanual systems [5, 17] provide higher-degree-of-freedom control by using motion- or rotation-sensitive input devices. With the SmartSkin sensor, the user can also control more than two points at the same time, and the shape of the arm or hand can be used as input. This is another approach to achieving higher-degree-of-freedom manipulation.

In contrast to two-handed interfaces, interaction techniques that are based on the use of multiple fingers have not been well explored. DualTouch [12] uses a normal touch panel to detect the position of tow fingers. Its resistive touch panel gives the middle position between two fingers when two positions are pressed, and assuming that the position of one finger is known (i.e., fixed to the initial position), the position of the other finger can be calculated. DualTouch can perform various interaction techniques such as "tapping and dragging", but due to this assumption of the initial position, most multiple-finger interfaces described in this paper are not possible.

CONCLUSION AND DIRECTIONS FOR FUTURE WORK

Our new sensing architecture can turn a wide variety of physical surfaces into interactive surfaces. It can track the position and shape of hands and fingers, as well as measure their distance from the surface. We have developed two working interactive surface systems based on this technology: a table and a tablet, and have studied various interaction techniques for them.

This work is still at an early stage and may develop in several directions. For example, interaction using multiple fingers and shapes is a very new area of human-computer interaction, and the interaction techniques described in this paper are just a few examples. More research is needed, in particular, focusing on careful usability evaluation.

Apart from investigating different types of interaction techniques, we are also interested in the following research directions.

Using a non-flat surface as an interaction medium: Places of interaction are not limited to a tabletop. Armrests or table edges, for example, can be good places for interaction, but have not been studied well as places for input devices. Placing SmartSkin sensors on the surface of 'pet' robots, such as Sony's AIBO, is another possibility. The robot would behave more naturally when interacting with humans. Similarly, if a game pad were "aware" of how the user grasps it, the game software could infer the user's emotions from this information.

Combination with tactile feedback: Currently, a SmartSkin user can receive only visual feedback, but if SmartSkin could make the surface vibrate by using a transducer or a piezo actuator, the user could "feel" as if he/she were manipulating a real object (the combination of a touch panel and tactile feedback is also described by Fukumoto [3]). [1]

Use of transparent electrodes: A transparent SmartSkin sensor can be obtained by using Indium-Tin Oxide (ITO) or a conductive polymer. This sensor can be mounted in front of a flat panel display or on a rear-projection screen. Because most of today's flat panel displays rely on active-matrix and transparent electrodes, they can be integrated with SmartSkin electrodes. This possibility suggests that in the future, display devices that will be interactive from the beginning, and will not require "retrofitting" sensor elements into them.

We also want to make transparent tagged objects by combining transparent conductive materials with the use of capacitance tags as shown in Figure 15. This technology will enable creating interface systems such as "DataTiles" [18], a user can interact with the computer via the use of tagged physical objects and hand gestures.

Data communication between the sensor surface and other objects: Because the SmartSkin sensor uses a wave signal controlled by software, it is possible to encode this signal with data. For example, location information can be transmitted from a SmartSkin table, and a digital device such as a PDA or a cellular phone on the table can recognize this information and trigger various context-aware applications. The table could also encode and transmit a "secret key" to mobile devices on the table, and these devices can establish a secure network with this key.

ACKNOWLEDGEMENTS

We thank our colleagues at Sony Computer Science Laboratories for the initial exploration of ideas described in this paper. We also thank Shigeru Tajima for the valuable technical advice, Takaaki Ishizawa and Asako Toda for their contribution to the implementation of the prototype system. We also would like to thank Toshi Doi and Mario Tokoro for their continuing support of our research.

REFERENCES

1. Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony DeRose. Toolglass and Magic Lenses: The see-through interface. In James T. Kajiya, ed-

[1] One interesting but unasked question is "Is it possible to provide tactile or similar feedback to a user whose hand is in the proximity of the surface, but not directly touching the surface?".

itor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 73–80, August 1993.

2. George W. Fitzmaurice, Hiroshi Ishii, and William Buxton. Bricks: laying the foundations for graspable user interfaces. In *CHI'95 Conference*, pages 442–449, 1995.

3. Masaaki Fukumoto and Toshiaki Sugimura. ActiveClick: Tactile feedback for touch panels. In *CHI 2001 summary*, pages 121–122, 2001.

4. Y. Guiard. Asymmetric divisoin of labor in human skilled bimanual action: the kinematic chain as a model. *Journal of Motor Behavior*, pages 485–517, 1987.

5. Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive real-world interface props for neurosurgical visualization. In *CHI'94 Proceedings*, pages 452–458, 1994.

6. Ken Hinckley and Mike Sinclair. Touch-sensing input devices. In *CHI'99 Proceedings*, pages 223–230, 1999.

7. IEEE. Proceedings of the fourth ieee international conference on automatic face and gesture recognition, 2000.

8. Hiroshi Ishii and Brygg Ullmer. Tangible Bits: Towards seamless interfaces between people, bits and atoms. In *CHI'97 Proceedings*, pages 234–241, 1997.

9. Hideki Koike, Yoichi Sato, Yoshinori Kobayashi, Hiroaki Tobita, and Motoki Kobayashi. Interactive textbook and interactive venn diagram: natural and intuitive interfaces on augmented desk system. In *CHI 2000 Proceedings*, pages 121–128, 2000.

10. S.K. Lee, William Buxton, and K. C. Smith. A multi-touch three dimensional touch-sensitive tablet. In *CHI '85 Proceedings*, pages 21 – 25, 1985.

11. M. Mathews and W. Schloss. The radiodrum as a synthesis controller. In *Proceedings international computer music conference*, 1989.

12. Nobuyuki Matsushita, Yuji Ayatsuka, and Jun Rekimoto. Dual Touch: a two-handed interface for pen-based PDAs. In *ACM UIST 2000 Proceedings*, pages 211–212, 2000.

13. Nobuyuki Matsushita and Jun Rekimoto. HoloWall: Designing a Finger, Hand, Body, and Object Sensitive Wall. In *Proceedings of UIST'97*, October 1997.

14. Shunichi Numazaki, Akira Morshita, Naoko Umeki, Minoru Ishikawa, and Miwako Doi. A kinetic and 3D image input device. In *Proceedings of the conference on CHI 98 summary*, pages 237–238, 1998.

15. Polhemus, Inc., Colchester, Vermont. *3SPACE ISOTRAK User's Manual*, 1987.

16. Jun Rekimoto and Masanori Saitoh. Augmented Surfaces: A spatially continuous workspace for hybrid computing environments. In *Proceedings of ACM CHI'99*, pages 378–385, May 1999.

17. Jun Rekimoto and Eduardo Sciammarella. ToolStone: Effective use of the physical manipulation vocabularies of input devices. In *Proc. of UIST 2000*, 2000.

18. Jun Rekimoto, Brygg Ullmer, and Haruo Oba. DataTiles: a modular platform for mixed physical and graphical interactions. In *CHI 2001 proceedings*, pages 269–276, 2001.

19. Meredith Ringel, Henry Berg, Yuhui Jin, and Terry Winograd. Barehands: implement-free interaction with a wall-mounted display. In *CHI 2001 summary*, pages 367–368, 2001.

20. Norbert A. Streitz, Jorg Geisler, Torsten Holmer, Shin'ichi Konomi, Christian Muller-Tomfelde and-Wolfgang Reischl, Petr Rexroth, Peter Seitz, and Ralf Steinmetz. i-LAND: An interactive landscape for creativity and innovation. In *CHI'99 Proceedings*, pages 120–127, 1999.

21. Brygg Ullmer and Hiroshi Ishii. The metaDESK: models and prototypes for tangible user interfaces. In *UIST'97 Proceedings*, pages 223–232, 1997.

22. John Underkoffler and Hiroshi Ishii. Illuminating Light: An optical design tool with a luminous-tangible interface. In *CHI'98 Proceedings*, pages 542–549, 1998.

23. Pierre Wellner. The DigitalDesk calculator: Tangible manipulation on a desk top display. In *ACM UIST'91 Proceedings*, pages 27–34, November 1991.

24. Pierre Wellner. Interacting with paper on the DigitalDesk. *Communication of the ACM*, 36(7):87–96, August 1993.

25. Thomas Zimmerman. Personal area networks: Near-field intrabody communication. *IBM Systems Journal*, 35(3-4):609–617, 1996.

26. Thomas G. Zimmerman, Joshua R. Smith, Joseph A. Paradiso, David Allport, and Neil Gershenfeld. Applying electric field sensing to human-computer interfaces. In *CHI'85 Proceedings*, pages 280–287, 1995.

ToolStone: Effective Use of the Physical Manipulation Vocabularies of Input Devices

Jun Rekimoto and Eduardo Sciammarella
Interaction Laboratory
Sony Computer Science Laboratories, Inc.
3-14-13 Higashigotanda
Shinagawa-ku, Tokyo 141-0022 JAPAN
Phone: +81 3 5448 4380
Fax +81 3 5448 4273
rekimoto@acm.org, http://www.csl.sony.co.jp/person/rekimoto.html

ABSTRACT
The ToolStone is a cordless, multiple degree-of-freedom (MDOF) input device that senses physical manipulation of itself, such as rotating, flipping, or tilting. As an input device for the non-dominant hand when a bimanual interface is used, the ToolStone provides several interaction techniques including a toolpalette selector, and MDOF interactors such as zooming, 3D rotation, and virtual camera control. In this paper, we discuss the design principles of input devices that effectively use a human's physical manipulation skills, and describe the system architecture and applications of the ToolStone input device.

KEYWORDS: Interaction techniques, input devices, physical user interfaces, multiple function inputs, multiple-degree-of-freedom input, two-handed input

INTRODUCTION
Although the mouse is the most successful input device in the history of computer interfaces, its limits are becoming frustrating as the complexity of software increases. The mouse is a generic input device, so a user must first be able to see a command (such as a menu item or tool button) on a screen, and then select it before the command is actually put into effect. For example, when a user wishes to draw a circle in a drawing editor, the user would open a toolpalette containing a circle tool, select the tool, then start drawing. These command objects are spatially deployed around the application (e.g., on tool bars or scroll bars), or appear according to the user's operations (e.g., pop-up menus or toolpalettes). Selection of these commands requires both physical (manipulation of an input device) and visual (recognizing a tool button and a cursor on a screen) efforts.

While this operating style has been effective for relatively simple software applications, an increasing number of functions, makes it more cumbersome to specify appropriate functions (menus, tool palettes, or property sheets). Many modern (and feature rich) applications use several toolpalettes and tool bars, and these take up screen space leaving less space for actual use. It has become impossible to lay out all available tools on a screen, so users have to frequently open and close tool palettes according to the task. This trend is forcing us to use bigger computer displays, but moving the mouse cursor between tool areas and application areas (such as a drawing canvas) becomes more time-consuming as the screen size increases.

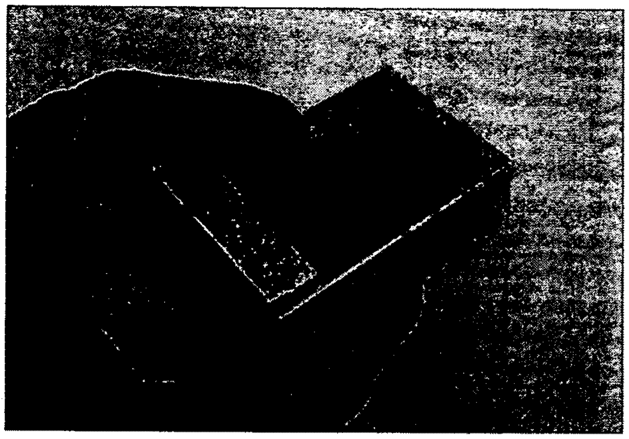


Figure 1: The ToolStone: a cordless semi-6DOF input device. Coils embedded in the device are used to measure position, orientation, tilt angle, and contacting face when it is placed on a tablet surface.

In contrast, physical tools allow effective use of a human's rich manipulation skills, and a single physical tool can often be used in many different ways. To illustrate this difference, Gershenfled compared the mouse with the violin bow [12]; while the mouse only provides a limited set of manipulation vocabularies (such as clicking or dragging), the violin bow has hundreds of ways in which it can be used. Trained violin players can easily change between tones (i.e., interaction modes) quite rapidly, and this selection relies heavily

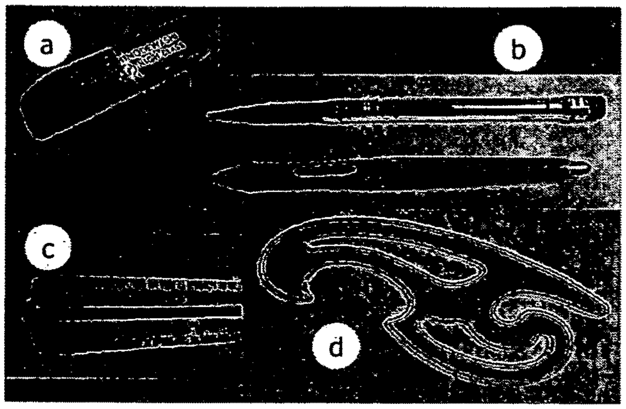


Figure 2: Multifunction physical tools: (a) a two-way rubber eraser, (b) a pencil with an eraser at one end and, its digital adaptation (the WACOM stylus), (c) a scale with six different divisions, and (d) a French curve with which a user can draw several different curves.

on motor skills: visual attention directed to the tool is not required.

Although directly comparing the mouse and the bow might be extreme, there are many daily tools that also provide multiple function through a single physical object. Figure 2 shows some examples of these. One thing these examples have in common is that we change the way we holding them to perform different functions. With a pencil that has a rubber eraser at one end, for example, we can easily change from a draw mode to eraser mode by simply reversing our grip.

In this paper, we explore the idea of expanding the functionalities of a single input device, and enabling users to select functions by changing the way they hold the device. Although this technique is related to multiple-degree-of-freedom (MDOF) input devices, we are also interested in developing interaction techniques that are not limited to the manipulation of 3D objects. We refer to such a physically enriched input style as a rich-action input.

In this paper, we discuss the design principles for rich-action input devices, then describe our ToolStone input device (Figure 1). The ToolStone is a cordless, MDOF interaction device that is designed to be easily rotated and flipped to activate several different functions.

RELATED WORK

Much research has been aimed at enhancing the "richness" of input devices. Embodied User Interfaces [13] attach several sensors to increase the usability of PDA. Tagged Handles allow a user to attach different handles to a rotational rod [17]. Users can differentiate between the functions of tag handles both visually and physically. The Cubic Mouse is a 6DOF input device with pushbuttons and movable penetrating shafts [11]. These shafts are used to provide additional operation modes; such as changing a cross-sectional plane of a 3D object.

There are also several examples of using the movements of input devices. Tilting user interfaces [19] use the tilt of portable devices as input. For example, a tilt sensor embedded in a hand-held computer can be used to scroll for menu selection or map scrolling. Embodied User Interfaces [13, 9] and Rock'n'Scroll [4] also use tilt interfaces.

The Rockin'Mouse is a mouse with a tilt sensor that can be used to manipulate a 3D object [3]. Kuroki and Kawai proposed the use of tilt information for pen interfaces [15]. They observed that people hold three physical tools (a pencil, a knife, and a syringe) differently, and they built a pen interface that allows a user to select different functions by changing its tilt on a tablet based on this observation.

We have also explored several interaction techniques that can be used when motion sensing becomes available in hand-held devices [21, 2]. For example, when a user places a PDA near an object displayed on a digital whiteboard, the PDA becomes a toolpalette for that object and the user can 'click through' a command by tapping on the PDA. Likewise, sweeping the surface of a digital table with a PDA enables data transfer between them just as we sweep breadcrumbs from a table into a dustpan.

Some researchers have also proposed associating multiple functions with a single object. PadMouse is a mouse with a touch-pad instead of a button. A user can make a finger gesture on a pad to select different functions. Fitzmaurice described the concept of flipbricks as part of his graspable user interfaces [10]. On each face of a flipbrick device, different commands, such as "cut" or "copy" are associated and users can activate one of them by flipping the device. Want et al. proposed an augmented photo-cube, a block with six wireless tags attached to its faces [23]. Up to six different digital contents can be associated with these tags, and can be retrieved by touching a face with a tag reader.

Our ToolStone uses multiple faces for different functions, and further increases the number of selectable functionalities by combining other manipulation vocabularies such as rotation or tilting. The ToolStone also uses several interaction techniques based on the physical movement of the ToolStone itself during operations.

DESIGN PRINCIPLES FOR RICH-ACTION INPUT DEVICES

In designing input devices that allow a user to select an appropriate function through physical actions, the following principles are important.

The device's state should be perceived through touch

Well-designed physical devices often reveal their operation mode without relying on visual information. While the user is concentrating on a task, the physical device's feel implicitly shows its state. Figure 3 shows two examples. The first example (a) is the three-state button of a video camcorder. While this camcorder is held by hand, the user's thumb always touches this button so the user can perceive the camcorder's state (camera-mode, playback-mode, and off) through the tactile impression. In contrast, a user of the camera in the second example (b) has to look to see the setting of a dial

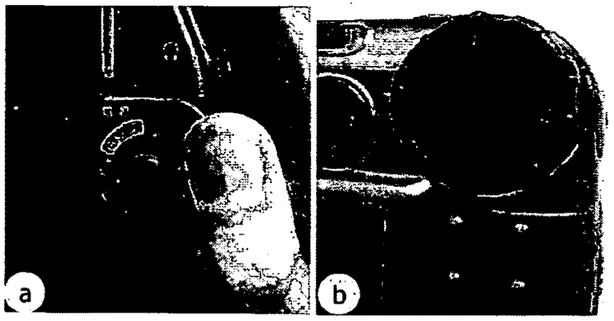


Figure 3: Tactile impressions reveal the state: (a) A camcorder switch with three states. A user can physically perceive the current state from the thumb position on the switch during operation. (b) A dial of a digital camera. In this example, the physical shape does not change so visual labels (hence, visual attention) are needed to know the current state.

because its shape does not indicate the selected mode. Labels on the dial are necessary, and a user must read these.

If we assign multiple functions to one input device, its state should be perceivable from tactile impressions; as well as through visual feedback, so that determining the currently selected state does not distract a user's visual attention.

This is one reason why a physical dial, such as the rotating dial in the WACOM 4D mouse, is not an effective way to select a function. With such a device, a user needs to find out the current state through visual feedback which may distract the user's visual attention. A second reason is a dial's sequential feature; instead of selecting a function in one operation, a user has to change the states one at at time by rotating the dial until the desired function becomes available.

The interaction space should be easily understandable
Although it is technically possible to implement a number of functionalities in a single input device, it is useless unless users can find them. Thus, visual appearance of the device is still important, in that it can help users visually recognize available functions at a glance. For example, a camcorder user (Figure 3(a)) would first understand the function of the switch from its visual appearance, and would then gradually learn to manipulate it by touch.

THE TOOLSTONE

To explore the benefits of input devices that support richer physical manipulations, we built the ToolStone device (Figure 1). The ToolStone is a cordless, rectangular object that is designed mainly as an input device for the user's non-dominant hand with bimanual interfaces (Figure 4). While the dominant hand manipulates a pointing device such as a mouse or a stylus, a ToolStone held by the non-dominant hand is used to select appropriate functions, or to provide more flexibility in operations.

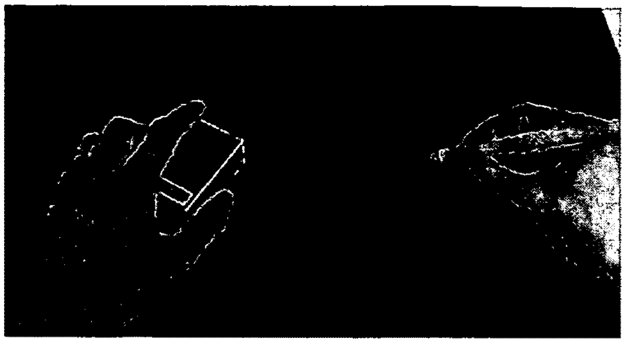


Figure 4: Bimanual interaction with the ToolStone.

The ToolStone is a semi-6DOF input device. When placed on a tablet, its x-y positions and orientation are measured. The tablet also detects which face of the ToolStone is touching the tablet surface. When one of the edges is touching the tablet, the tilt angle can also be measured.

A small projection (a bar) is attached to the lower edge of one face. By feeling this projection, a user can perceive the device's orientation and face direction without visual/audio

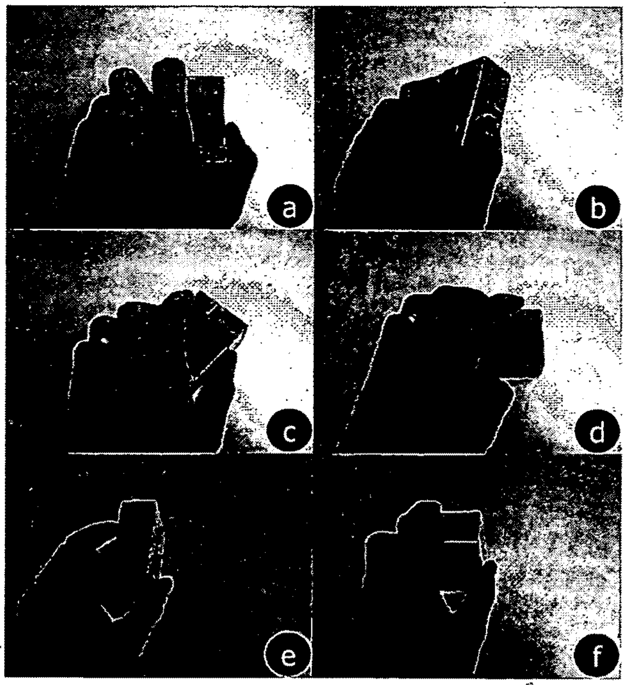


Figure 5: Several possible ways of holding the ToolStone: (a) Normal mode (Note: a projection attached near the lower edge of the upper face can be felt by the hand). (b) Tilting while one edge is contacting the tablet (c, d) Rotating, and (e, f) Flipping to select other faces.

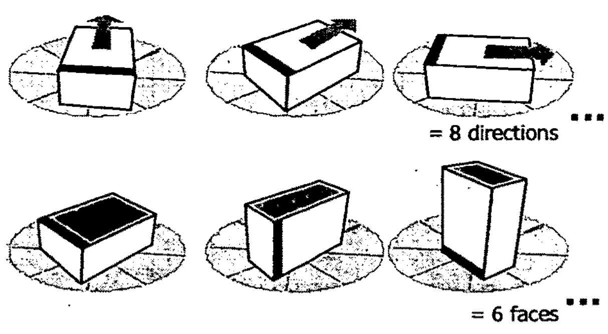


Figure 6: Selecting multiple functions by rotating and flipping the ToolStone: The combination of eight directions and six faces allows a user to quickly select 48 different functions (e.g., toolpalettes) with a single physical action.

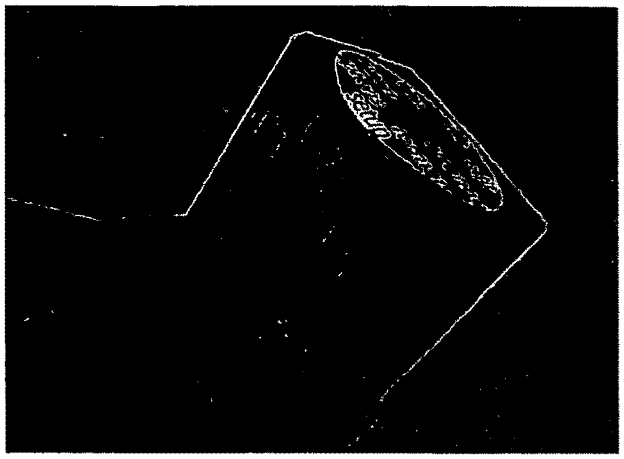


Figure 8: A ToolStone device with labels on each face. A (novice) user would be able to visually inspect available commands by physically turning the device.

feedback (Figure 5).

INTERACTION TECHNIQUES

Although the ToolStone is not a complete 6DOF input device (at least one face or edge has to be touching the tablet during operation), several new interaction techniques can be realized. This section briefly describes the typical use of the ToolStone.

Tool selection

When used as a non-dominant hand device for bimanual interfaces, the ToolStone can be used as a tool selector for the dominant-hand input device. For example, eight different toolpalettes, each with several different command items, can be assigned to eight directions separated by 45 degrees, and a user can quickly select an appropriate palette by rotating the

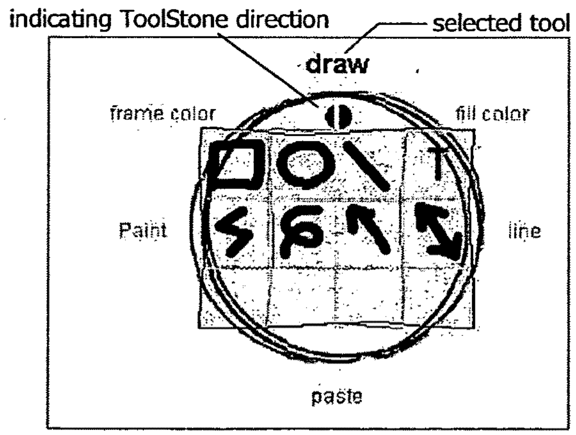


Figure 7: Example of a selected toolpalette: A dial and labels around the tool palette indicate available functionalities attached to the same face. The currently selected one is shown in bold. The selected toolpalette acts as a ToolGlass sheet.

ToolStone. Furthermore, the user can switch to a different set of tools by flipping the ToolStone to select another face. If a set of eight toolpalettes are attached to each face, and the six faces have different sets, 48 different toolpalettes can be selected by through a single physical action (Figure 6). This would meet the requirements of most real-world applications.

This feature is particularly suitable for selecting toolpalettes in ToolGlass or MagicLense interfaces [6, 5]. In the original ToolGlass design, the non-dominant hand is only used to control the location of a ToolGlass sheet. With the ToolStone, it can also be used to switch between several toolpalettes (ToolGlass sheets) with a quick physical action. Since only one toolpalette appears on the screen at a time, the screen would not be cluttered by a number of floating palettes, as is often the case with today's application software.

The form-factor of the ToolStone is designed to enable comfortable manipulation. The width, height, and depth of the ToolStone are all different; combined with the attached projection, this allows the user to easily distinguish the physical state.

In addition, it is useful to add labels to the ToolStone faces, so that (novice) users can visually inspect the available functionalities by physically turning the device in their hands (Figure 8).

An interesting feature of the ToolStone is that we can organize the command space physically. For example, when we assign related functionalities (such as tools for picture element creation and tools for giving a color to an element) to adjacent positions (i.e., adjacent angles), the ToolStone's physical manipulation distance (the time required for switching between two functions) would also represent the logical distance between tools. Currently we assign a color selection tool and picture creation tool to adjacent angles of the same face, and file manipulation commands to another face. After creating a picture element, a user can slightly rotate the Tool-

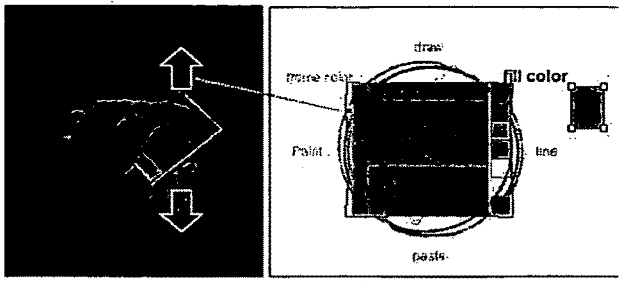


Figure 9: A color selection tool example: ToolStone's vertical motion controls the brightness parameter of the color space, while two other parameters (hue and saturation) are mapped according to the x and y axes of a 2D palette. A user can dynamically navigate though the color space before selecting a color instance. Note that the direction of the ToolStone is used to select the color selection tool.

Stone (45 degrees) to get the color selection tool. As a user repeatedly performs this sequence, we expect that it would become a chunk of physical operations.

MDOF interaction techniques

When one tool is selected, the ToolStone's x-y positions are still available for other manipulation. We can use these values to control the position of the selected toolpalette in ToolGlass-type interfaces.

The other possibility is to use them for controlling parameters during toolpalette operations. For example, for a color selection toolpalette, the forward/backward movement of the ToolStone can be used to control the brightness parameter (Figure 9). Since color space is a 3D space (e.g., hue-saturation-brightness), color selection requires control of three parameters. Existing color selection tools often force unintuitive operations because of the bad mappings between the 3D color space and the 2D toolpalette space. Our solution allows a user to simultaneously control the third parameter (e.g., brightness) by moving the ToolStone device, while the dominant-hand pointing device selects a point on a toolpalette.

It should also be possible to apply this idea for various kinds of interactions that require more than 2D parameter control. For example, a World within a World interface [8] for exploring up to a 4D information space can be implemented as a 3D graph, and remaining 2D parameters can be manipulated by forward/backward and sideways movement of the ToolStone.

MDOF control

In addition to the MDOF interaction techniques described above, it is possible to simultaneously control parameters of more than two degrees of freedom.

For example, one face of the ToolStone can be assigned to

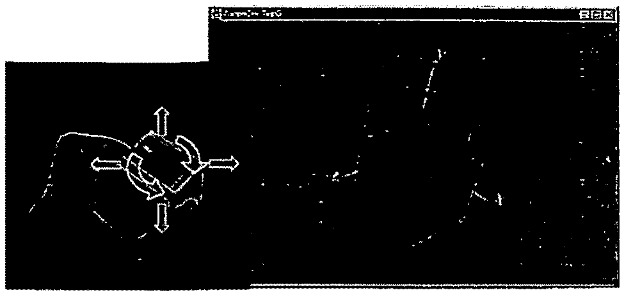


Figure 10: MDOF movement of the ToolStone can be mapped for 3D object control.

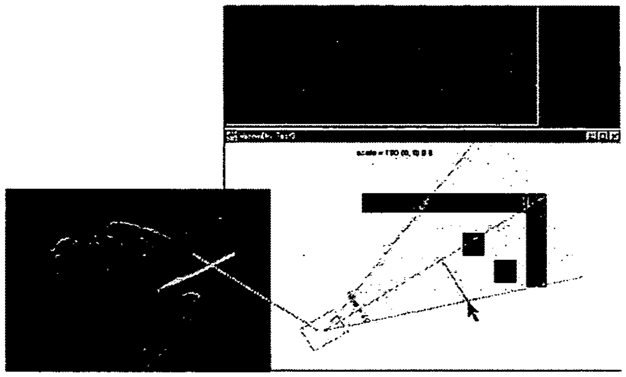


Figure 11: A user is manipulating a virtual camera of a 3D world. While the non-dominant hand is used to control the camera's position and orientation, the user can also change the field of view by dragging a viewing area (projected as a filled arc) with the dominant-hand's pointing device. Note that the pointing device is also used to change the viewing angle of the camera.

zooming and panning of the workspace. Without moving the cursor to the scrollbars at the edges of a window, a user can select a zooming tool by flipping the ToolStone. The ToolStone's forward/backward and sideways motions are mapped to scrolling, while its rotation controls scaling. For example, rotating the ToolStone clockwise can be mapped to increasing the scale (i.e., zooming in).

Another example is 3D rotation of an object. When a user selects an object on a screen and holds it with the dominant-hand's pointing device, the ToolStone becomes a rotation tool. For example, the horizontal and vertical motions of the ToolStone control the direction of the rotation axis, and its rotation controls the angle of object rotation (Figure 10).

Figure 11 shows another example of combining tool selection and MDOF control. When a user flips the ToolStone to select a virtual camera tool in this 3D scene-building appli-

APLNDC00025907

cation, a 3D view window appears and the user can control the viewpoint of the camera by manipulating the ToolStone as a physical camera on a floor plan. During this operation, the dominant-hand pointing device can also be used to alter interaction parameters. For example, the field of view of the camera can be directly manipulated by dragging an edge of a view frustum that is projected onto a floor plan.

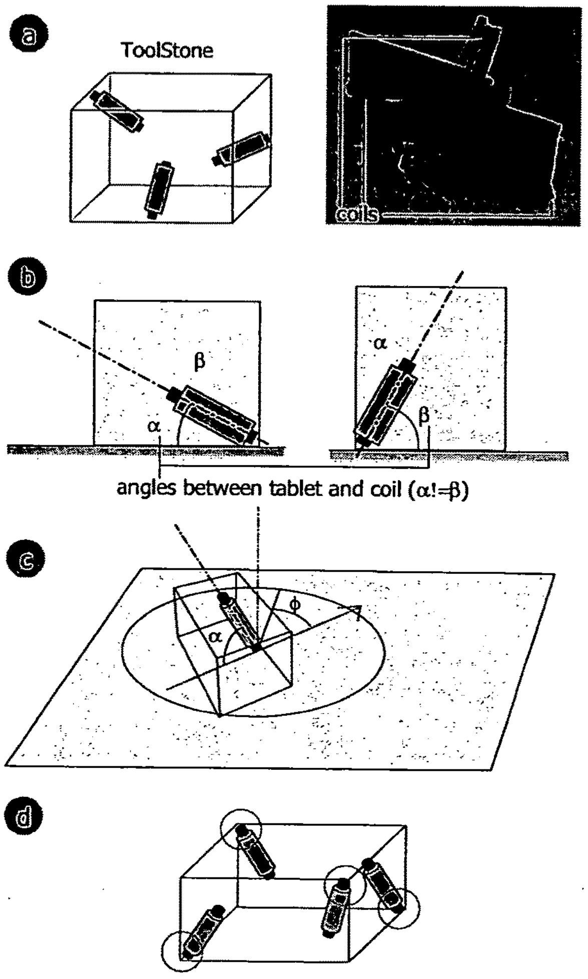


Figure 12: Detection of the touching face and orientation: (a) Inside the ToolStone: Three WACOM coils are embedded, and only one of them will be close enough to the tablet surface when the ToolStone is placed on the tablet. (b) When a coil touches the tablet, it can be identified by its unique resonance value. Two faces that share the same coil can be distinguished by comparing the tilt values (α and β). (c) Once the touching face is known, the orientation of the ToolStone can be determined from the orientation angle of the coil (ϕ). (d) An alternative sensor configuration with coils at the four corners of the device. Two of these coils are in contact with the surface when one face is placed on the tablet.

SYSTEM IMPLEMENTATION

Sensor architecture

To enable the interactions described in the previous sections, we needed a means of sensing ToolStone's orientation, which of it's faces is in contact with the tablet surface, and its position. The ability to measure these parameters "untethered", freeing the user from the bother of a wire during operations, was also desirable. Since most MDOF input devices (such as the Poluhemus isotrak [18]) are tethered devices, we decided to design our own sensor architecture.

Our first implementation was based on visual sensing. We attached six different visual patterns to the ToolStone faces and placed it on a semi-transparent acrylic board that acted as a tablet surface. A camera below the acrylic board was used to determine the position and orientation of the ToolStone, and to detect the contacting face. As a prototype, this architecture worked reasonably well, but the tablet was too thick. We thus looked for an alternative solution based on the widely used electro-magnetic pen tablet.

We used the WACOM tablet(WACOM UD-II series) [7] as our next platform and developed a ToolStone with a three-coils architecture (Figure 12). We embedded three coils, taken from WACOM styluses, at three different edges of the ToolStone. The WACOM tablet emits magneto-electric signals to the nearby area, and a coil with a specific resonance parameter responds to this signal. By analyzing this response pattern, we can measure the coil's position on a tablet, as well as its angle and orientation.

When one of the ToolStone surfaces touches the tablet, only one coil is in contact with the tablet (Figure 12). Although this coil is shared by two faces, the system determines which face it is by measuring the angle of the coil. The orientation of the ToolStone can also be calculated from the coil orientation when contacting face is known.

Each of the three coils can be identified through its unique resonance parameter. The original WACOM stylus consists of a coil and a small ferrite core that is combined with a small spring. This mechanism is used to measure the pen pressure. When a user changes pen pressure, the system measures the resonance parameter of the pen which will vary according to the distance between the coil and the ferrite core. To use this value to determine which coil touches the tablet surface, we attached small ferrite cores to the three coils, each at a slightly different distance. The coils can thus be detected in the same way as three pens with different pen pressures.

Combining these features made it possible to determine the touching surface of a ToolStone, as well as its position and orientation in relation to the tablet, without using wires or batteries. In our prototype design, the ToolStone is 2.5 × 4 × 5 cm, and it weighs 22 g. This is much smaller and lighter than a conventional mouse. The weight and form-factor make it easy to manipulate in a user's hand.

Since only one coil (out of three) needs to be sensed at one time, a tablet that supports simultaneous sensing of two objects can be used as a bimanual manipulation tablet (most commercially available tablets can simultaneously sense only two objects).

We are also planning to attach a button to one face of the ToolStone, so that it can also operate as a normal mouse.

Software architecture

For application programmers, we have developed a ToolStone device driver interface of 'raw' tablet driver [16]. This layer hides the internal recognition algorithm, and provides an event-driven interface to applications. For example, a ToolStone-aware application is programmed to receive a "stone" event, as well as mouse events. The stone event contains information concerning the ToolStone status, including the currently selected face, position, and orientation.

This driver interface was written in C on Windows 98, and all example applications described in the previous sections were written in Java. These Java applications communicate to the ToolStone driver layer through the Java Native Interface (JNI). Applications that support 3D object manipulation were built with Java 3D.

PROTOTYPE TRIAL

To date, we have implemented a simple drawing tool and interaction techniques based on the ToolStone. Five pilot users (all were expert with GUI tools, but not familiar with two-handed interfaces) have tried the system after a minimal demonstration. Although a formal user study is still being planned, we obtained some interesting feedback during this trial.

All users instantly understood the ToolStone concept and could easily select different tools. Some users preferred to keep the same face down and rotate the ToolStone, rather than to flip it, mainly because these was less physical motion and sound generated than when it was flipped.

To provide visual cues, the current implementation used labels around a currently used tool to indicate other available functionalities (Figure 7), but this information was limited to the functions that belonged to the same face. Many user required similar labels for other faces.

Some users told us that they felt there was a strong relationship between the spatial manipulation and the tool space. One user compared the ToolStone to an analog clock, and explained his image of all the tools being assigned on a dial of a clock. Another user mentioned that he could easily remember the assignment of the functions when he imagined he was manipulating a small doll instead of a rectangular shape.

Some users explained that they could remember a sequence of hand actions in the same way we remember word spellings when touch-typing. We observed that one user, who was quite accustomed to the prototype application, had difficulty when he tried to recall the assignment of tools without actually manipulating the ToolStone. In our daily lives, we often use physical skills that we can apply but cannot explain in words. Whether the ToolStone requires the use of similar physical skills is an interesting question.

Figure 13: An object with several different ways of holding

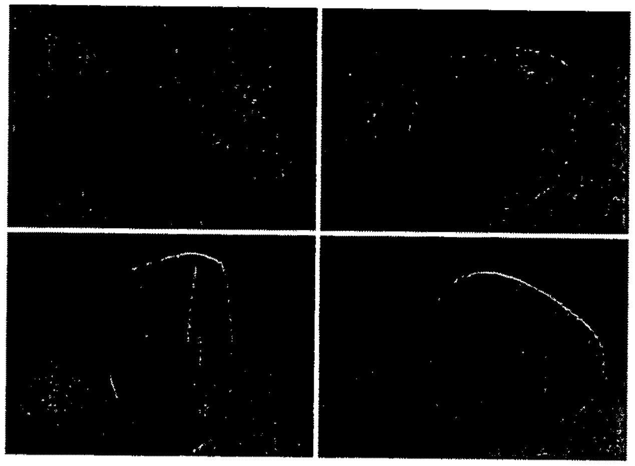


Figure 14: Several design variations of the ToolStone shapes.

CONCLUSION AND FUTURE DIRECTIONS

A rich-action input device allows users to interact with computer functionalities by physically changing the way they hold the input device. The ToolStone, a cordless multiple-degree-of-freedom (MDOF) device is such an input device. A ToolStone's unique sensor architecture allows the system to sense physical manipulation of the device itself; for example, rotating, flipping, or tilting. As an input device for the non-dominant hand with bimanual interfaces, the ToolStone can be used, for example, as a tool selector, for MDOF interactions such as zooming, 3D rotation, or virtual camera control.

ToolStone is still at an early stage of development, though, and there are several directions for further study. Our research topics for the immediate future are explained in the following.

Evaluation of other physical shapes

Our initial prototype was rectangular but other shapes are also worth considering. One possibility is a polygonal (e.g., hexagonal) pyramid with its top cut off. With this shape,

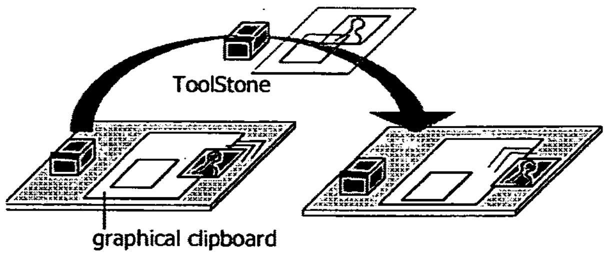


Figure 15: The ToolStone is used to transfer data from one computer to another: A user can carry the contents of the graphical clipboard with the ToolStone.

a user can select a face with less physical motion than is required with the present shape. We may also add distinctive physical textures such as small holes or grooves to every surface, or round edges or faces to make tilting motions easier.

As an alternative to moving the device, it may also be possible to select functions by detecting the way the user touches the device. By extending the idea of the touch-sensitive mouse [14], we can attach touch sensors to the faces of the input device. Thus, users may be able to switch between operating modes by changing their grip on the device. For example, the physical object shown in Figure 13 can be held in several different ways.

Also, other shapes may be more esthetically appealing than a simple rectangle. Figure 14 shows some shapes that create stronger positive impressions, and we expect that future computer applications will be symbolized by their own unique 'stone' shapes.

Multiple ToolStones

For highly complicated tools, such as a high-end graphic tool, it is also possible to use more than one ToolStone device. Each stone object represents a category of operations, such as 3D modeling tools or photo retouching tools. A user can switch the operating mode by physically exchanging one ToolStone for another. Another idea is to provide a different ToolStone set for different categories of users. A software application may behave differently with different kinds of ToolStone. For example, a ToolStone for children might provide only basic functions, while a ToolStone for adults would also provide more complicated functions.

The ToolStone may also act as a physical information carrier between computers by using techniques similar to Pick-and-Drop [20] or mediaBlocks [22]. In this scenario, a user can copy data from one computer to the ToolStone, and retrieve it when using another computer. For example, a user could display a graphical clipboard panel on one computer then drag an object onto the clipboard (Figure 15). When the ToolStone is removed from the tablet, the clipboard panel is removed with it and the user virtually carries it with the ToolStone. When the user places the ToolStone on a different computer, the same clipboard re-appears and the user can drag-out any of the carried objects. (We assume that only the ID would be stored in the ToolStone and the actual data transfer would be done through the network).

Study of human memory and computer input

Finally, we would also like to study the human memory skills required to deal with computer systems. According to cognitive psychology theory (such as [1]), a human's long-term memory can be classified into two major categories: declarative memory (i.e., knowledge that can be explained by words), and procedural memory (i.e., learned skills). In our daily lives, we rely heavily on the latter so that we can concentrate on tasks requires active use of knowledge. It seems, however, that today's computer systems still relay too much on users' declarative memory, and do not effectively utilize learned skills. Our experience with the ToolStone suggests that input devices would be more effective if they made better use of human motor skills.

ACKNOWLEDGEMENTS

We thank Haruo Oba and the members of the Sony CSL Interaction Laboratory for helping us to explore the ideas described in this paper. We also thank Brygg Ullmer and George Fitzmaurice for introducing us to flipbricks. Finally, we are indebted to Mario Tokoro and Toshi Doi for their continuing support of our research.

REFERENCES

1. Mark H. Ashcraft. *Human Memory and Cognition*. Addison-Wesley Publishing Company, 1994.
2. Yuji Ayatsuka, Nobuyuki Matsushita, and Jun Rekimoto. HyperPalette: a hybrid computing environment for small computing devices. In *CHI 2000 Extended Abstracts*, pages 133–134, 2000.
3. Ravin Balakrishnan, Thomas Baudel, Gordon Kurtenbach, and George Fitzmaurice. The Rockin'Mouse: Integral 3D manipulation on a plane. In *CHI'97 Proceedings*, pages 311–318, 1997.
4. Joel F. Bartlett. Rock'n'Scroll is here to stay. *IEEE Computer Graphics and Applications*, 20(3):40–45, 2000.
5. Eric A. Bier, Maureen C. Stone, Ken Fishkin, William Buxton, and Thomas Baudel. A taxonomy of see-through tools. In *Proceedings of CHI'94*, pages 358–364, 1994.
6. Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony DeRose. Toolglass and Magic Lenses: The see-through interface. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 73–80, August 1993.
7. Wacom Technology Co. Product information. http://www.wacom.com.

8. Steven Feiner and Clifford Beshers. Worlds within worlds: Metaphors for exploring n-dimensional virtual worlds. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 76–83, Snowbird, Utah, October 1990.

9. Kenneth P. Fishkin, Thomas P. Moran, and Beverly L. Harrison. Embodied User Interfaces: Towards invisible user interfaces. In *Engineering for Human-Computer Interaction: Seventh Working Conference on Engineering for Human-Computer Interaction*, pages 1–18, 1998.

10. George W. Fitzmaurice. Graspable user interfaces. *Ph.D thesis, University of Toronto*, 1996.

11. Bernd Frohlich and John Plate. The Cubic Mouse: A new device for three-dimensional input. In *CHI'2000 Proceedings*, pages 526–531, 2000.

12. Neil A. Gershenfeld. *When things start to think*. Henry Hold & Company Inc., 1999.

13. Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. In *Conference proceedings on Human factors in computing systems (CHI'99)*, pages 17–24, 1998.

14. Ken Hinckley and Mike Sinclair. Touch-sensing input devices. In *CHI'99 Proceedings*, pages 223–230, 1999.

15. Tsuyoshi Kuroki and Satoru Kawai. An interface technology for pen devices using tilt information. In *Interactive Systems and Software VII*, pages 1–6. Kindai Kagaku Publishing Inc., 1999.

16. LCS/Telegraphics. Wintab: Advanced pointing device management for windowed environments. http://www.pointing.com.

17. Karon E. MacLean, Scott S. Snibbe, and Golan Levin. Tagged Handles: Merging discrete and continuous manual control. In *CHI'2000 Proceedings*, pages 225–232, 2000.

18. Polhemus, Inc., Colchester, Vermont. *3SPACE ISOTRAK User's Manual*, 1987.

19. Jun Rekimoto. Tilting operations for small screen interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '96)*, pages 167–168, 1996.

20. Jun Rekimoto. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. In *Proceedings of UIST'97*, pages 31–39, October 1997.

21. Jun Rekimoto. A multiple-device approach for supporting whiteboard-based interactions. In *Proceedings of ACM CHI'98*, pages 344–351, February 1998.

22. Brygg Ullmer, Hiroshi Ishii, and Dylan Glas. mediaBlocks: Physical containers, transports, and controls for online media. In *SIGGRAPH'98 Proceedings*, pages 379–386, 1998.

23. Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging physical and virtual worlds with electronic tags. In *CHI'99 Proceedings*, pages 370–377, 1999.

Dean Rubine and Paul McAvinney
School of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213 USA
Rubine@cs.cmu.edu
McAvinney@cs.cmu.edu

Programmable Finger-tracking Instrument Controllers

Introduction

Traditionally, the controls of a musical instrument have been mechanically connected to its sound generators. It is well known that electrical couplings may be used instead of mechanical ones, and that digital instead of analog connections may be used. The freedom of not having the controls mechanically constrained to the means of sound production brings a corresponding burden. What form should the controls now take? Knowledge of the practice of music, psychoacoustics, ergonomics, sensing technologies, design, computer science, and economics can all be brought to bear on the question; common sense and personal taste also play a large role.

Our interest in instrument control has led us to create the VideoHarp, a controller that optically tracks a performer's fingertips. The finger-tracking data is mapped to MIDI commands, which are sent to a synthesizer to make sound. The mapping is fully programmable; the VideoHarp can act like many different controllers and is able to switch between various modes instantaneously. The VideoHarp may be played with strumming or bowing motions, with keyboard or drum-like gestures, or with gestures having no analog in existing instruments.

In this paper, we discuss some aspects of instrument control and controllers. We begin with reasons for wanting new, programmable instrument controllers. Next, we introduce the idea that an instrument controller is both a measurer of gestural parameters and a mapper from gestural to sound control parameters. We then attempt to use results from psychoacoustics and ergonomics to determine requirements for sensing and representing some common gestural and sound control parameters. A discussion of some existing sensing technologies follows. The design and implementation of the VideoHarp is then presented, followed by several concluding remarks.

Computer Music Journal, Vol. 14, No. 1, Spring 1990,
© 1990 Massachusetts Institute of Technology.

Why Create New Instrument Controllers?

Often, a new instrument is derived directly from an existing instrument. This has the advantage that much of the playing technique of the existing instrument may be transferred to the new instrument. We claim this transfer will often result in similar music from the two instruments, however. Conversely, one method for making truly new music might be to create an instrument that allows performers to experiment with new playing techniques. This is the approach we have taken with the VideoHarp.

Creating a new instrument that responds to gesture in a single fixed way might result in truly new music, but this originality could not be sustained indefinitely. The novelty would fade as the instrumental technique became standardized. Avoiding such ossification was our motivation for exploring programmable finger-tracking instrument controllers. We felt that an instrument able to track the paths of multiple fingers could be programmed to respond to many different types of gesture; in effect, such an instrument could act like many different instruments.

A programmable finger-tracking controller enables the composer to "compose" new controllers in a manner similar to the way in which programmable synthesis has given composers detailed control over timbre. Such a controller also allows broad classes of gestures, such as bowing, strumming, and keyboarding, to be applied to instrument classes for which they have traditionally been inappropriate (e.g., to strum a trumpet). By allowing instrument control mechanisms to be decoupled from sound generators, MIDI has become the "crossbar switch" between gesture and timbre. The VideoHarp generates MIDI commands as its output, freeing us from the details of sound generation and allowing us to concentrate solely upon instrument control.

Sound Control Parameters

To make music, a performer gestures at a musical instrument. This results in sound, which is perceived by the listener. We are concerned with the components of the gesture that affect the perceived sound. We believe that the essence of performance is contained in those gestures, or rather in the effect those gestures have on the parameters of the sound.

There is a distinction to be made between the aspects of the gesture that affect the sound, and the aspects of the sound that are affected. The former, which we term *gestural parameters*, are data such as the positions and forces of the performer's fingers on the instrument. The latter, which we term *sound control parameters*, are perceptible parameters of the sound—pitch, amplitude, timing, and the many dimensions of timbre. A musical instrument may be thought of as a device that maps gestural parameters to sound control parameters and then maps the sound control parameters to sound. A musical instrument controller does not itself have a sound generating mechanism; it is, however, a device that detects gestural parameters and maps these to sound control parameters.

One characteristic of the mapping from gestural parameters to sound control parameters, in the case of most traditional instruments, is its simplicity and directness. The position of a finger on the piano keyboard maps directly to pitch; the downward velocity of the finger maps directly to amplitude. In these and many other examples, a single gestural parameter maps directly onto a single sound control parameter. (The timbre of the piano also changes with velocity, but since the changes are linked inseparably with the amplitude changes, we consider the pair to be a single sound control parameter of the piano.) This directness is not lost on the listener, who can often infer the form of the gesture from the sound generated. Cadoz (1988) states that "the sound phenomenon produced by a natural object or instrument is an indelible trace of the gesture" and that the perceptual system of the brain "infers possible causes," i.e., possible gestures that made the sound.

We divide control parameters into two major classes that we call *instantaneous* and *envelope* parameters. An instantaneous parameter is one whose value is determined at an instant of time; the initial amplitude of a piano note is a fine example. Although the amplitude of the note changes over the course of the note's sounding, the course of the decay is dictated by the initial amplitude, so it is considered an instantaneous control parameter. An envelope parameter is one whose value may be considered a function of time over an interval; the amplitude of a saxophone note is an example. In nonmusical settings, switches are often used to control instantaneous parameters while envelope parameters are controlled by knobs or sliders.

The control of many parameters simultaneously is limited by both the performer and the instrument. Instruments capable of a large degree of polyphony, such as the piano, have for the most part allowed the player to control only instantaneous parameters. In contrast, monophonic instruments generally allow the performer control over one or more envelope parameters; the saxophonist, for example, can continually control the amplitude and pitch (bend) of a single note. Some polyphonic instruments, such as the violin and the guitar, allow control over some envelope parameters. The guitar player often bends strings—demonstrating continuous control over pitch—when playing monophonic lines, but usually does not bend strings when playing polyphonic parts. It seems that in these instruments, the limitations of the performer are such that he or she can only attend to a subset of the available parameters at any one time. The lack of envelope parameters on polyphonic instruments seems to be a special case of this phenomenon; by concentrating on producing many notes simultaneously, the performer pays less attention to controlling details of each note.

While a programmable controller allows an unlimited number of mappings from gestural to sound control parameters, it should be clear that not every such mapping is musically useful. The limits of human hearing and gesturing, as well as the limits of