EXHIBIT 4.22

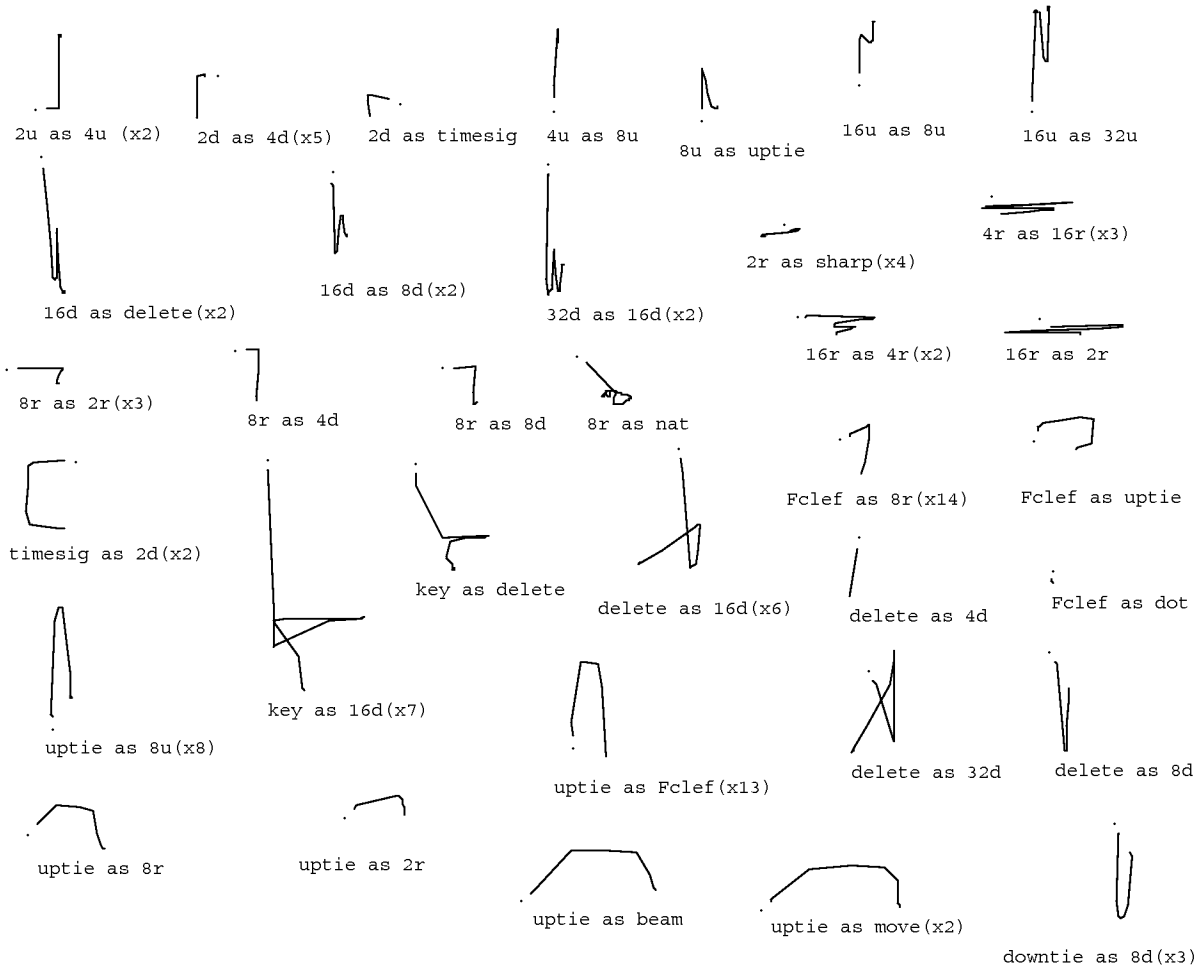


Figure 9.4: Misclassified GSCORE gestures

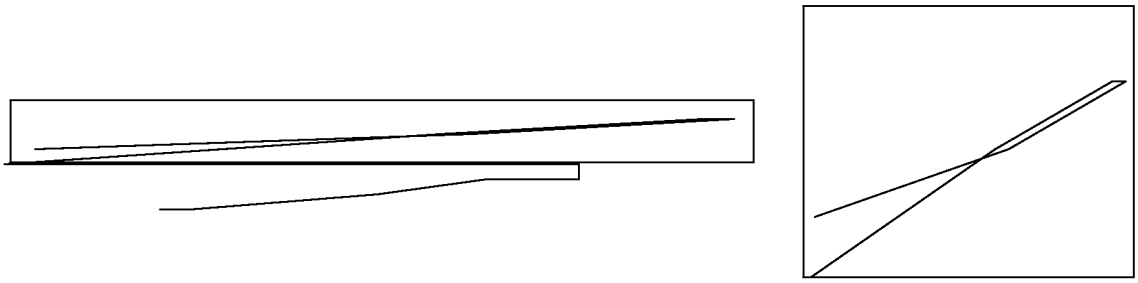


Figure 9.5: A looped corner

The left figure is a magnification of a misclassified "4r as 16r" shown in the previous figure. The portion of the gesture enclosed in the rectangle has been copied and its aspect ratio changed, resulting in the figure on the right. As can been seen, the corner, which should be a simple angle, is looped. This resulted in the angle-based features having values significantly different from the average 4r *gesture, thus the misclassification.*

In "2u as 4u," "2d as 4d," "8r as 4d," "8r as 8d," and "timesig as 2d" there is no point between the initial point and the first corner, probably due to the paging. This interacts badly with the features f_1 and f_2, the cosine and sine of initial angle. Features f_1 and f_2 are computed from the first and *third* point; this usually results in a better measurement than using the first and second point. In these cases, however, this results in a poor measurement, since the third point is after the corner.

"8r as nat" was the result of a very long page in, during which the author got impatient and jiggled the mouse.

Ambiguous classes. Some classes are very similar to each other, and are thus likely to be mistaken for each other. The 14 misclassifications of "Fclef as 8r" are an example. Actually, these may also be considered examples of poor mouse tracking, since points lost from the normally rounded top of the Fclef gesture caused the confusion. The mistakes "uptie as 8u," and "uptie as Fclef" are also examples of ambiguity.

Ideally, the gesture classes of an application should be designed so as to be as unambiguous as possible. Given nearly ambiguous classes, it is essential that the input device be as reliable and as ergonomically sound as possible, that the features be able to express the differences, and that the decider be able to discriminate between them. Without all of these properties, it is inevitable that there will be substantial confusion between the classes.

Inadequacy of the feature set. The examples where the second mouse point is the first corner show one way in which the features inadequately represent the classes. For example, the "2r as sharp" examples appear to the system as simple left strokes. Sometimes, a small error in the drawing results in a large error in a feature. This occurs most often when a stroke doubles back on itself; a small change results in a large difference in the angle features f_9, f_{10}, and f_{11} (see figure 9.5). The mistakes "4r as 16r" and "16d as delete" are in this category. "16u as 8u" and "16u as 32u" point to other places where the features may be improved.

The mapping from gestures to features is certainly not invertible; many different gestures might have the same feature vector in the current scheme. This results in ambiguities not due entirely to similarities between classes, but due to a feature set unable to represent the difference. Example "key as 16d" is an illustration of this, albeit not a great one.

Inadequacy of linear, statistical classification. Given that the differences between classes can be expressed in the feature vector, it still may be possible that the classes cannot be separated well by linear discrimination functions. This typically comes about when a class has a feature vector with a severely non-multivariate-normal distribution. In the current feature set, this most often happens in a class where the gesture folds back on itself (as discussed earlier), causing f_9, and thus the entire feature vector, to have a bimodal distribution.

The averaging of the covariance matrix in essence implies that a given feature is equally important in all classes. In the above class, the initial angle features are deemed important by the classifier. When compounded with errors in the tracking, this leads to bad performance on examples such as "uptie as beam" and "uptie as move." It is possible for a linear classifier to express the per-class importance of features in a linear classifier; in essence this is what is done by the neural-network-like training procedures (*a.k.a* back propagation, stochastic gradient, proportional increment, or perceptron training).

Inadequate training data. Drawing and tracking errors occur in the training set as well as the testing set. Given enough good examples, the effect of bad examples on the estimates of the average covariance matrix and the mean feature vectors is negligible. This is not the case when the number of examples per class is very small. Bad or insufficient training data causes bad estimates for the classifier parameters, which in turn causes classification errors. The gestures classified correctly by the $C = 30$, $E = 40$ classifier, but incorrectly by the $C = 30$, $E = 10$ classifier are examples of this.

Analyzing errors in this fashion leads to a number of suggestions for easy improvements to the classifier. Timing or distance information can be used to decide whether to compute f_1 and f_2 using the first two points or the first and third points of the gesture. Mouse events could be queued up to improve performance in the presence of paging. Some new features can be added to improve recognition even in the face of other errors; in particular, the cosine and sine of the final angle of the gesture stroke would help avoid a number of errors. These modification are left for future work, as the author, at the present time, has no desire to redo the above evaluation using 6000 examples from a different gesture set.

One error not revealed in these tests, but seen in practice, is misclassification due to a premature timeout in the two-phase interaction. This results in a gesture being classified before it is completely entered.

9.1.2 Rejection parameters

Section 3.6 considered the possibility of rejecting a gesture, *i.e.* choosing not to classify it. Two parameters potentially useful for rejection were developed. An estimate of the probability that a gesture is classified unambiguously, $\tilde{P}$, is derived from the values of the per-class evaluation

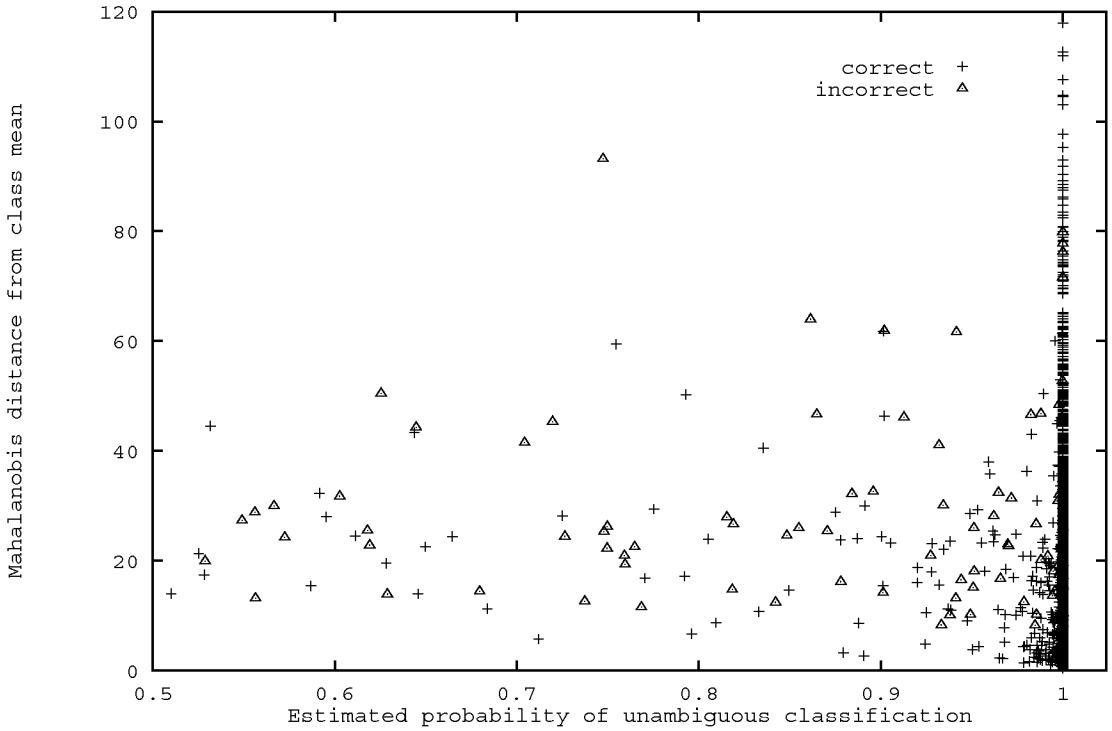


Figure 9.6: Rejection parameters

functions. An estimate of the Mahalanobis distance, d^2, is used to determine how close a gesture is to the norm of its chosen class.

It would be nice if thresholds on the rejection parameters could be used to neatly separate correctly classified example from incorrectly classified examples. It is clear that it would be impossible to do a perfect job; as "delete as 8d" illustrates, the system would need to read the user's mind. The hope is that most of the incorrectly classified gestures can be rejected, without rejecting too many correctly classified gestures.

A little thought shows that any rejection rule based solely on the ambiguity metric $\tilde{P}$ will on the average reject at least as many correctly classified gestures as incorrectly classified gestures. This follows from the reasonable conjecture that the average ambiguous gesture is at least as likely to be classified correctly as not. (This assumes that the gesture is not equally close to three or more classes. In practice, this assumption is almost always true.)

Figure 9.6 is a scatter plot that shows the value for both rejection parameters for all the gestures in the GSCORE test set. A plus sign indicates a gesture classified correctly; a triangle indicates each gesture classified incorrectly, *i.e.* those represented in figure 9.4. Most of the correctly classified examples have an estimated unambiguity probability of very close to one, thus accounting for the dark mass of points at the right of the graph. 96.3% of the correctly classified examples had

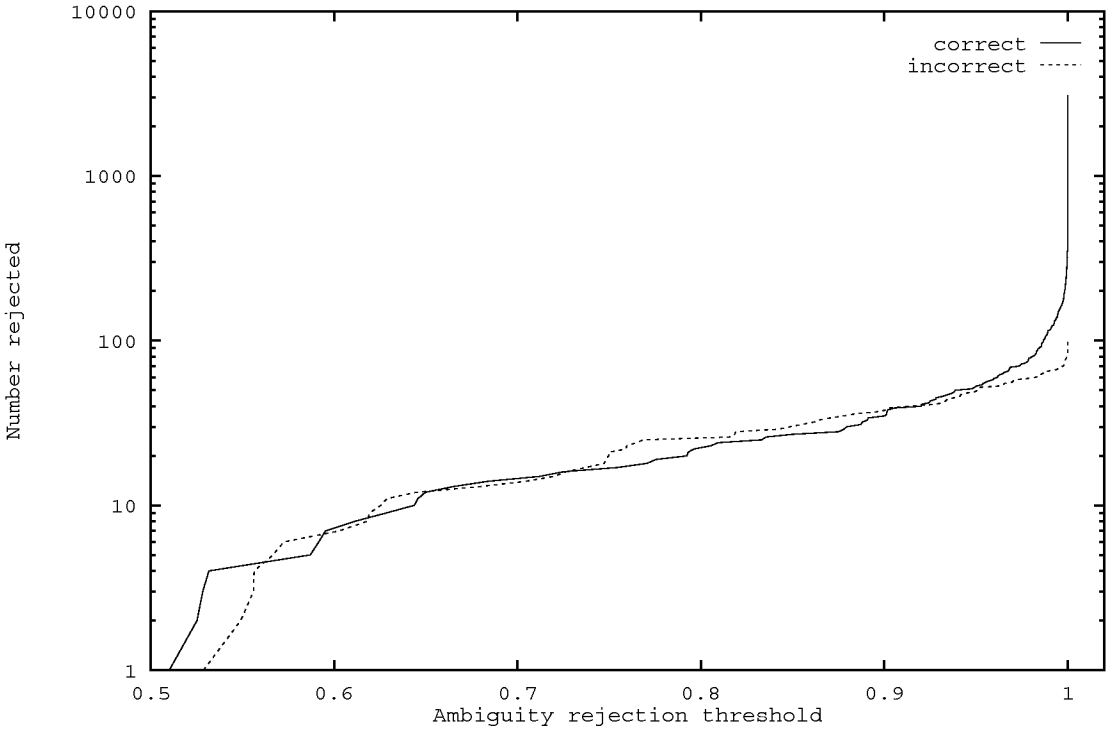


Figure 9.7: Counting correct and incorrect rejections

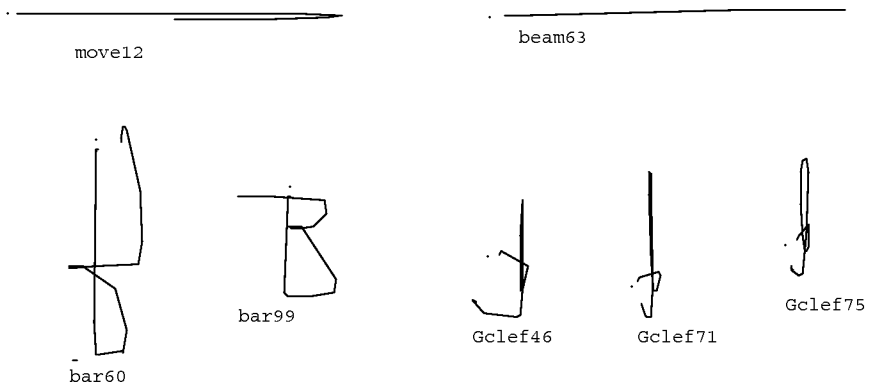


Figure 9.8: Correctly classified gestures with $d^2 \geq 90$

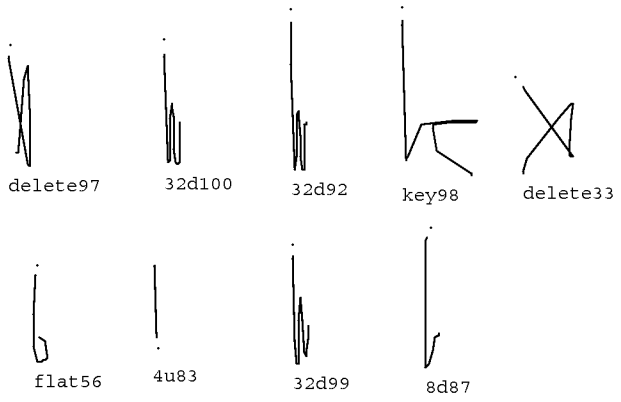


Figure 9.9: Correctly classified gestures with $\tilde{P} \leq .95$

$\tilde{P} \geq 0.99$. However, the same interval contained 33.7% of the incorrectly classified examples. Figure 9.7 shows how many correctly classified and how many incorrectly classified gestures would be rejected as a function of the threshold on $\tilde{P}$.

Examining exactly which of the incorrect examples have $\tilde{P} \geq 0.99$ is interesting. The garbled "8r as nat" and the left stroke "2r as sharp" have $\tilde{P} = 1.0$ within six decimal places. In retrospect this is not surprising; those gestures are far from *every* class, but happen to be unambiguously closest to a single class. This is borne out in the d^2 for those gestures, which is 380 (off the graph) for "8r as nat" and at least 70 for each "2r as sharp" gesture. Other mistakes have $\tilde{P} > .999$ but $d^2 < 20$. In this category are "Fclef as 8r," "uptie as Fclef," "delete as 8d," and "4u as 8u"; these gestures go beyond ambiguity to look like their chosen classes so could not be expected to be rejected.

Also interesting are those correctly classified test examples that are candidates for rejection based on their $\tilde{P}$ and d^2 values. Figure 9.8 shows some GSCORE gestures whose $\tilde{P} = 1$ and $d^2 \geq 90$. Examples "move12" and "beam63" are abnormal only by virtue of the fact that they are larger than

normal. The two bar examples have their endpoints in funny places, among other things, while the three Gclef examples are fairly unrecognizable. The algorithm does however classify all of these correctly; and it would be too bad to reject them. Figure 9.9 shows gestures whose ambiguity probability is less that .95. In many of the examples this is caused by at least one corner being made by two mouse points rather than one. In "delete33" one corner is looped. These gestures look so much like their prototypes it would be too bad to reject them.

The Mahalanobis estimate is mainly useful for rejecting gestures that were deliberately entered poorly. This is not as silly as it sounds; a user may decide during the course of a gesture not to go through with the operation, and at that time extend the gesture into gibberish so that it will be rejected.

One possible improvement would be to use the per-class covariance matrix of the chosen class in the Mahalanobis distance calculation. Compared to using the average covariance matrix, this would presumably result in a more accurate measurement of how much the input gesture differs for the norm of its chosen class.

9.1.3 Coverage

Figure 9.10 shows the performance of the single-path gesture recognition algorithm on five different gesture sets. The classifier for each set was trained on fifteen examples per class and tested on an additional fifteen examples per class. The first set, based on Coleman's editor [25], had a substantial amount of variation within each class, both in the training and the testing examples. The remaining sets had much less variation with each class. As the rates demonstrate, the single-path gesture recognition algorithm performs quite satisfactorily on a number of different gesture sets.

9.1.4 Varying orientation and size

One feature that distinguishes gesture from handwriting is that the orientation or size of a gesture in a given class may be used as an application parameter. For this to work, gestures of such classes must be recognized as such independent of their orientation or size. However, the recognition algorithm should not be made completely orientation and size independent, as some other classes may depend on orientation and size to distinguish themselves.

It is straightforward to indicate those classes whose gestures will vary in size or orientation: simply vary the size or orientation of the training examples. The goal of the gesture recognizer is to make irrelevant those features in classes for which they do not matter, while using those feature in classes for which they do.

Theoretically, having some classes that vary in size and orientation, while other that depend on size or orientation for correct classification should be a problem for any statistical classifier based on the assumptions of a multivariate normal distribution of features per class, with the classes having a common covariance matrix. A class whose size is variable is sure to have a different covariance matrix than one whose size remains relatively constant; the same may be said of orientation. Thus, we would suspect the classifier of Chapter 3 to perform poorly in this situation. Surprisingly, this does not seem to be the case.

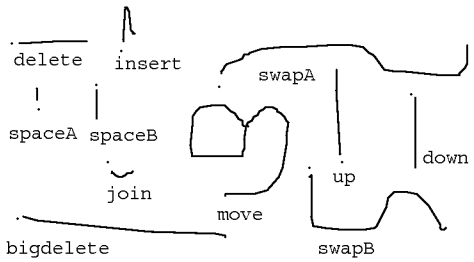
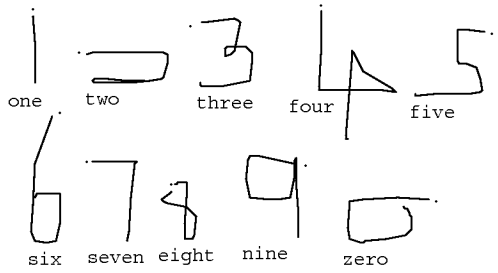
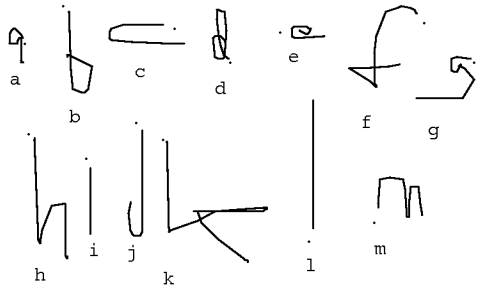
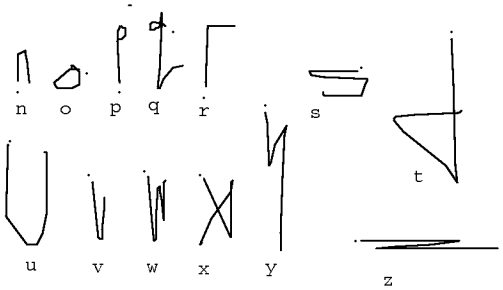
Set Name	Gesture Classes	Number of Classes	Recognition Rate
Coleman		11	100.0%
Digits		10	98.5%
Let:a-m		13	99.2%
Let:n-z		13	98.4%
Letters	Union of Let:a-m and Let:n-z	26	97.1%

Figure 9.10: Recognition rates for various gesture sets

Each set was trained with 15 examples per class and tested on an additional 15 examples per class.

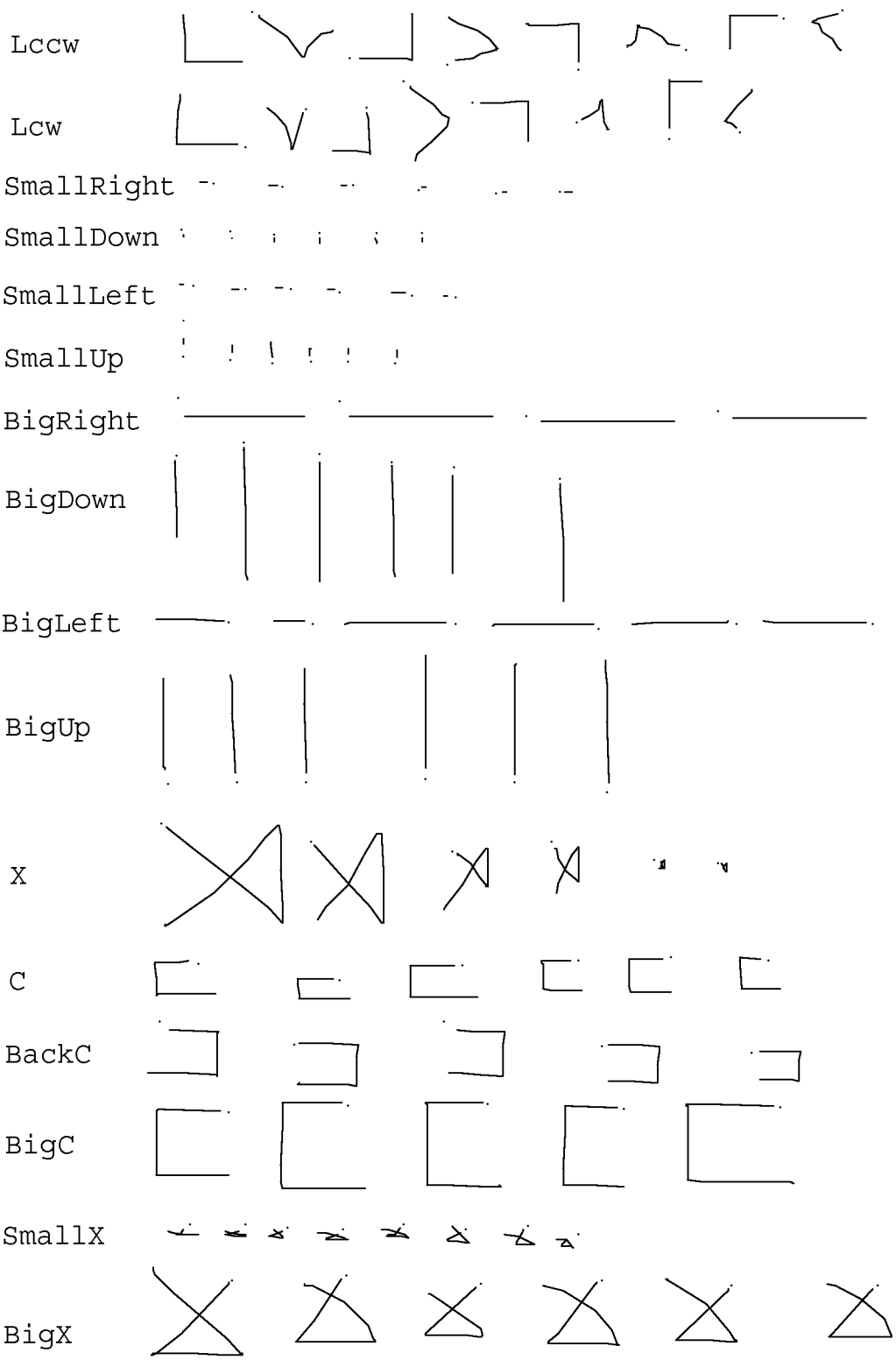


Figure 9.11: Classes used to study variable size and orientation

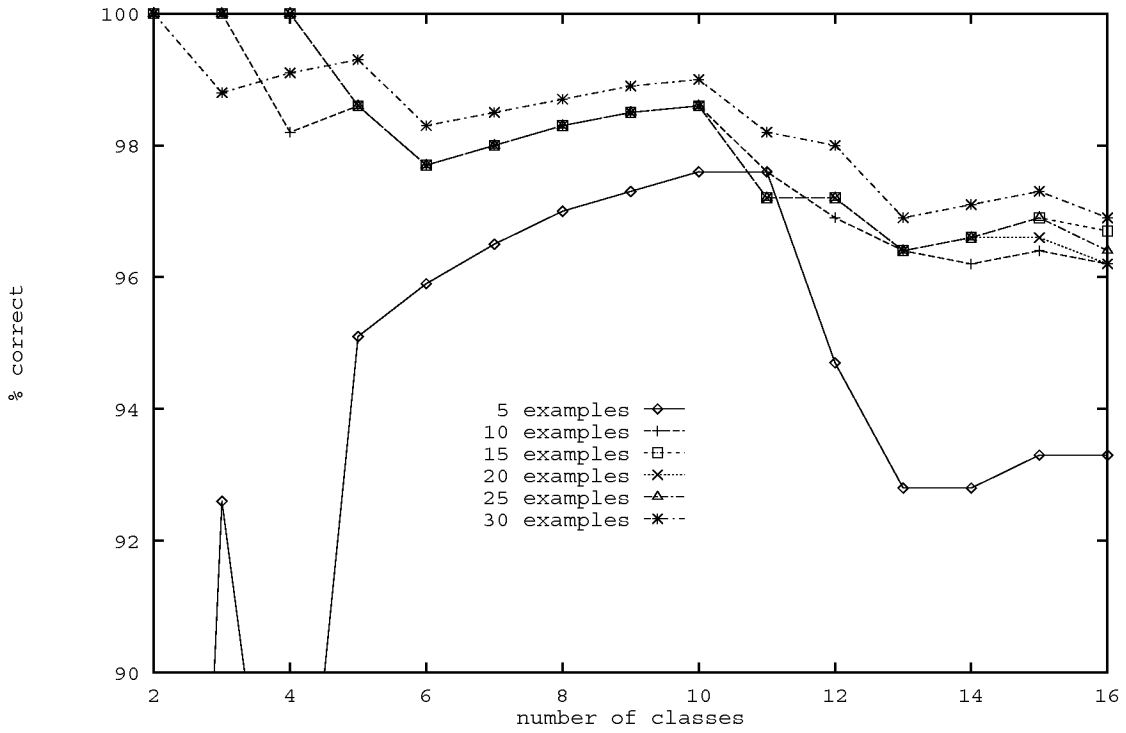


Figure 9.12: Recognition rate for set containing classes that vary

Figure 9.11 shows 16 classes, some of which vary in size, some of which vary in orientation, others of which depend on size or orientation to be distinguishable. The training set consists of thirty examples of each class; variations in size or orientation were reflected in the training examples, as shown in the figure. A testing set with thirty or so examples per class was similarly prepared.

Figure 9.12 shows the recognition rate plotted against the number of classes for various numbers of examples per class in the training set. As can be seen, the performance is good; 96.9% correct on 16 classes trained with 30 examples per class. Using only 15 examples per class results in a recognition rate of 96.7%.

Figure 9.13 shows all the mistakes made by the classifier. None of the mistakes appear to be a result of the size or orientation of a gesture being confused. Rather, the mistakes are quite similar to those seen previously. The conclusion in that the gesture classifier does surprisingly well on gesture sets in which some classes have variable size or orientation, while others are discriminated on the basis of their size or orientation.

9.1.5 Interuser variability

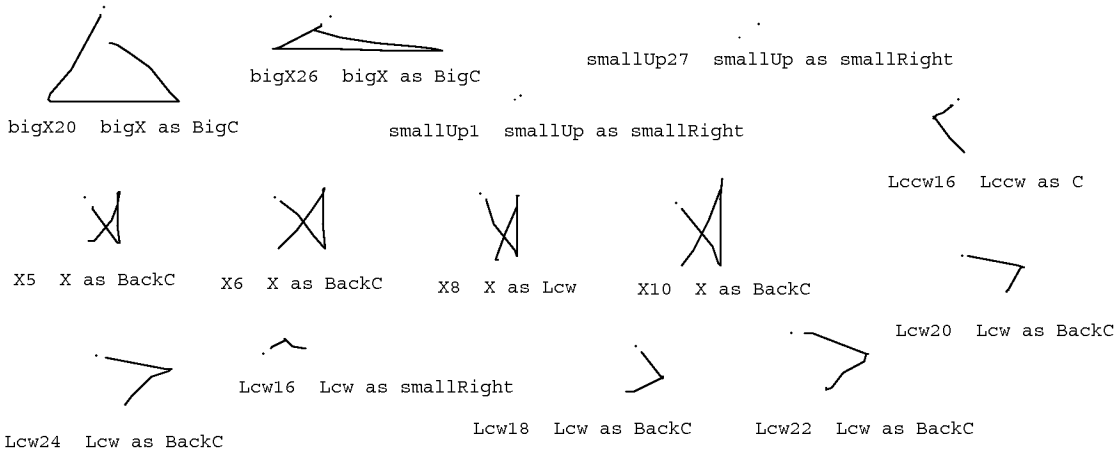


Figure 9.13: Mistakes in the variable class test

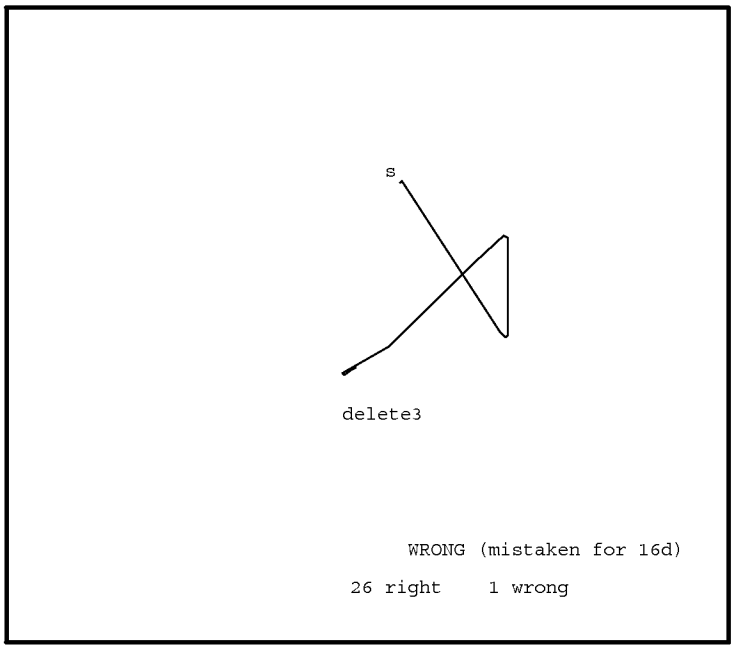


Figure 9.14: Testing program (user's gesture not shown)

All the gestures shown thus far have been those of the author. It was deemed necessary to show validity of the current work by demonstrating that the gestures of at least one other person could be recognized. Two questions come to mind: what recognition rate can be achieved when a person other than the author gestures at a classifier trained with the author's gestures, and can this rate be improved by allowing the person to train the classifier using his or her own gestures?

Setup

As preparation for someone besides the author actually using the GSCORE application (see Section 9.4.2 below), the GSCORE gesture set (figure 9.1) was used in the evaluation. The hardware used was the same hardware used in the majority of this work, a DEC MicroVAX II running UNIX and X10.

A simple testing program was prepared for training and evaluation (Figure 9.14). In a trial, a prototype gesture of a given class is randomly chosen and displayed on the screen, with the start point indicated. The user attempts to enter a gesture of the same class. That gesture is then classified, and the results fed back to the user. In training mode, if the system makes an error, the trial is repeated. In evaluation mode, each trial is independent.

Subject PV is a music professor, a professional musician, and an experienced music copyist. He is also an experienced computer user, familiar with Macintosh and NeXT computers, among others.

Procedure

The subject was given one half hour of practice with the testing program in training mode. He was also given a copy of figure 9.1 and instructed to take notes at his own discretion. After the half hour, the tester was put in evaluation mode, and two hundred trials run. The test was repeated one week later, without any warmup. The subject was then instructed to create his own gesture set, borrowing from the set he knew as much as he liked. Thirty examples of each gesture class were recorded, and two hundred evaluation trials run on the new set.

Results

During the initial training there was some confusion on the subject's part regarding which hand to use. The subject normally uses his right hand for mousing, but, being left handed, always writes music with his left. After about ten minutes, the subject opted to use his left hand for gesturing.

In the initial evaluation trial the system classified correctly 188 out of 200 gestures. The subject felt he could do better and was allowed a second run, during which 179 out of 200 gestures were correctly classified. By his own admission, he was more "cocky" during the second run, generally making the gestures faster than during the first. The average recognition rate is 91.8%.

After the test, the subject commented that he felt much of his difficulty was due to the fact that he was not used to using the mouse with his left hand, and that the particular mouse felt very different than the one he was used to (NeXT's). He felt his performance would further improve with additional practice.

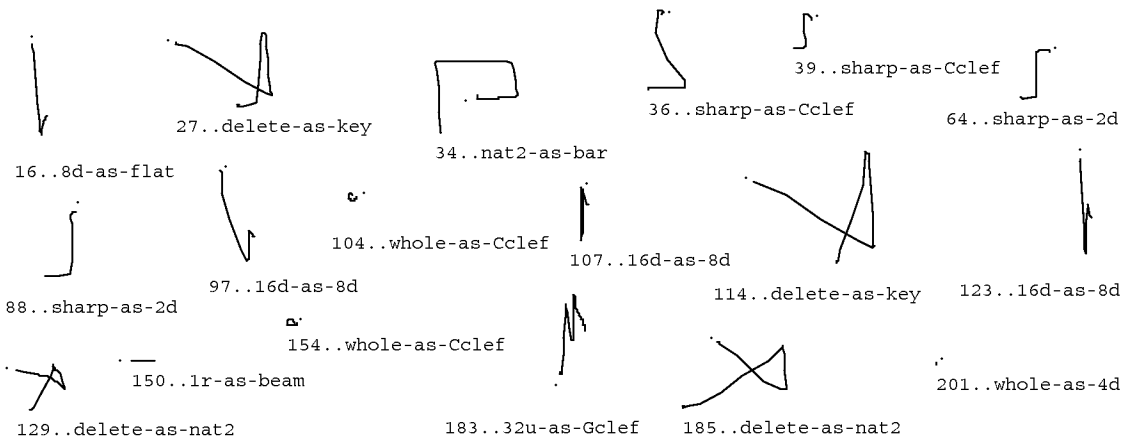


Figure 9.15: PV's misclassified gestures (author's set)

His notes are interesting. Although the subject had no particular knowledge of the recognition algorithm being used, in many cases his notes refer to the particular features used in the algorithm. For gestures whole and sharp he wrote "start up" and "don't begin too vertically" respectively, noting the importance of the correct initial angle. For 1r he wrote "make short," for bar he wrote "make large," and for delete he wrote "make quickly." For 2u and 2d he wrote "sharp angle."

The subject commented on places where the gesture classes used did not conform to standard copyist strokes. For example, he stated the loop in flat goes the wrong way. He explained that many music symbols are written with two strokes, and said that he might prefer a system that could recognize multiple-stroke symbols.

When the test was repeated a week later, the subject, without any warmup, achieved a score of 183 out of 200, 91.5%. Figure 9.15 shows the misclassified gestures. The subject was again unsure of which hand to use, but used his left hand at the urging of the author.

The subject then created his own gesture set, examples of which are shown in figure 9.16. A training set consisting of 30 examples of each class was entered. Running the training set through the resulting classifier resulted in the rather low recognition rate of 94.7% (by comparison, running the author's training set through the classifier it was used to train yielded 97.7%.) The low rate was due to the some ambiguity in the classes (*e.g.* "flat" and "16d" were frequently confused) as well as many classes where the corners were looped (as seen before in section 9.1.1), causing a bimodal distributions for f_9, f_{10}, and f_{11}

The problems in the new gesture set nonwithstanding, PV ran two hundred trials of the tester on the new set. He was able to get a score of 186 out of 200, 93%.

At the time of this writing, PV has not yet made the attempt to remove the ambiguities from the new gesture set and to be more careful on the sharp corners.

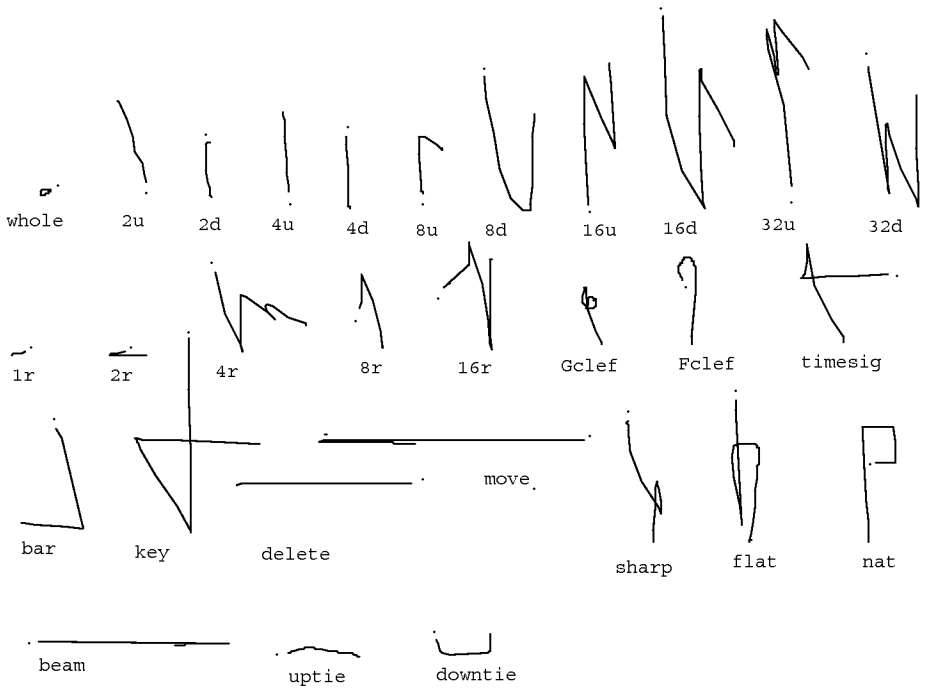


Figure 9.16: PV's gesture set

Conclusion

It is difficult to draw a conclusion given data from only one subject. The author expected the recognition rate to be higher when PV trained the system on his own gestures than when he used the author's set. The actual rate *was* slightly higher, but not enough to make a convincing argument that people do better on their own gestures (some slightly more convincing evidence is presented in section 9.4.2 below). In retrospect, PV should have created a training set that copied the author's gestures before attempting to significantly modify that gesture set. The author's gesture set turned out to be better designed than PV's, in the sense of having less inherent ambiguities; this tended to compensate for any advantage PV gained from using his own gestures.

However, PV's new gesture set is not without merit; on the contrary, it has a number of interesting gestures. The new delete gesture, a quick, long, leftward stroke, gives the user the impression of throwing objects off the side of the screen. The new move gesture is like a delete followed by a last minute change of mind. The flat gesture is much closer to the way PV writes the symbol, as are the leftward whole and half rests gestures 1r and 2r. The stylized "4" for timesig is clever, as is the way it relates to key. PV's bar gesture is much more economical that the author's.

The experiments indicate that a person can use a classifier trained on another person's gesture with moderately good results. Also indicated is that people can create interesting gesture sets on their own. Some modification to the feature set also seems desirable, mainly to make the features

less sensitive to "looped" corners. It would be useful to give more feedback to the gesture design as to which classes are confusable. This should be simple to do simply by examining the Mahalanobis distance between every pair of classes.

9.1.6 Recognition Speed

It is well known that a user interface must respond quickly in order to satisfy users; thus for gesture-based systems the speed of recognition is an important factor in the usability of the system. This section reports on measurements of the speed of the components of the recognition process.

The statistical gesture recognizer described in Chapter 3 was designed with speed in mind. Each feature is incrementally calculated in constant time; thus $O(F)$ work must be done per mouse point, where F is the number of features. Given a gesture of P mouse points, it thus takes $O(PF)$ time to compute its feature vector. The classification computes a linear evaluation function over the features for each of C classes; thus classification take $O(CF)$ time.

Feature calculation

The abstract datatype `FV` is used to encapsulate the feature calculation as follows:

`FV FvAlloc()` allocates an object of type `FV`. A classifier will generally call `FvAlloc()` only once, during program initialization.

`FvInit(fv)` initializes `fv`, an object of type `FV`. `FvInit(fv)` is called once per gesture, before any points are added.

`FvAddPoint(fv, x, y, t)` adds the point (x,y) which occurs at time t to the gesture. `FvAddPoint` performs the incremental feature calculation. It is called for every mouse point the program receives. There are thirteen features calculated ($F = 13$).

`Vector FvCalc(fv)` returns the feature vector as an array of double precision floating point numbers. It performs any necessary calculations needed to transform the incrementally calculated auxiliary features into the feature set used for classification. It is called once per gesture.

The function `CalcFeatures(g)` represents the entire work of computing the feature vector for a gesture that is in memory:

```
Fv fv;      /* allocated via FvAlloc() elsewhere */
Vector
CalcFeatures(g)
register Gesture g;
{
    register Point p;
    FvInit(fv);
    for(p = g->point; p < &g->point[g->npoints]; p++)
        FvAddPoint(fv, p->x, p->y, p->t);
```

Processor	Time(sec)	Relative Speed
MicroVAX II	227.95	0.76
VAX 11/780	172.20	1.0
MicroVAX III	60.97	2.8
PMAX-3100	11.30	15

Table 9.1: Speed of various computers used for testing

Processor	milliseconds per call		
	`FvAddPoint`	`FvCalc`	`CalcFeatures`
MicroVAX II	0.22	0.34	3.9
MicroVAX III	0.074	0.13	1.3
PMAX-3100	0.029	0.040	0.44

Table 9.2: Speed of feature calculation

```
    return FvCalc(fv);
}
```

To obtain the timings, the testing set of Section 9.1.1 was read into memory, and then each gesture was passed to `CalcFeatures`. Three processors were used: the DEC MicroVAX II that was used for the majority of the work reported in this dissertation, a DEC MicroVAX III, and a DEC PMAX-3100 (to get an idea of the performance on a more modern system). The UNIX profiling tool was used to obtain the times. In all cases, the times are virtual times, *i.e.* the time spent executing the program by the processor. All tests were run on unloaded systems, and the real times were never more than 10% more than the virtual times.

Before timing any code related to gesture recognition, the following code fragment (compiled with "cc -O") was timed on a number of processors, MicroVAX II, VAX 11/780, MicroVAX III, and PMAX-3100, in order to compare the speed of the processors used in the following tests to that of a VAX 11/780:

```
register int  i, n = 1000000;
double s, a[15], b[15];
for(i = 0; i < 15; i++) a[i] = i, b[i] = i*i;
do {
    s = 0.0;
    for(i = 0; i < 15; i++) s += a[i] * b[i];
} while(--n);
```

The times for the above fragment shown in table 9.1.

Note that on this code fragment the PMAX-3100 runs about 20 times faster than the MicroVAX II. On more typical code, it usually runs only 10-15 times faster.

The testing set averaged 13.4 points per gesture. The timings for the routines that calculate features are shown in table 9.2.

The cost per mouse point to incrementally process a mouse point is a small fraction of a millisecond, even on the slowest processor. Since mouse points typically come no faster than 40

Processor	Computation time (milliseconds)				
	$v^{\hat{c}}$ (one class)	$\max v^{\hat{c}}$ (30 classes)	$\tilde{P}$	d^2	total
MicroVAX II	0.27	8.0	0.8	3.7	12.6
MicroVAX III	0.074	2.2	0.3	1.1	3.6
PMAX-3100	0.022	0.66	0.01	0.26	0.99

Table 9.3: Speed of Classification

per second, only a small fraction of the processor is consumed incrementally calculating the feature vector. Indeed, substantially more of the processor is consumed communicating with the window manager to receive the mouse point and perform the inking.

Classification

Once the feature vector is calculated it must be classified. This involves computing a linear evaluation function $v^{\hat{c}}$ on F features ($F = 13$) for each of C classes. If the rejection parameters are desired, it takes an additional $O(C)$ work to estimate the ambiguity $\tilde{P}$ and $O(F^2)$ work to estimate the Mahalanobis distance d^2. The computation times for each of these is shown in table 9.3.

To get these times, four runs were made. Every gesture in the testing set was classified in every run. The first run did not calculate either rejection parameter. The average time to classify a gesture as one of thirty classes is reported the $maxv^{\hat{c}}$ column; the $v^{\hat{c}}$ column is computed as $\frac{1}{30}$ of that time. (The $v^{\hat{c}}$ column thus gives the time to compute the evaluation function for a single class; multiply this by the number of classes to estimate the classification time of a particular classifier.) The second run computed $\tilde{P}$ after each classification; the difference between that time and the $\max v^{\hat{c}}$ time is reported in the $\tilde{P}$ column. The third run computed d^2 and is reported similarly. The fourth run computed both $\tilde{P}$ and d^2; the average time per gesture is reported in the "total" column.

For a 30-class discrimination with both rejection parameters being used, after the last mouse point of a gesture is entered it takes a MicroVAX II 13 milliseconds to finish calculating the feature vector (`FvCalc`) and then classify it. This is acceptable, albeit not fantastic, performance. If the end of the gesture is indicated by no mouse motion for a timeout interval, the classification can begin before the timeout interval expires, and the result be ignored if the user moves the mouse before the interval is up.

Currently, all arithmetic is done using double precision floating point numbers. There is no conceptual reason that the evaluation functions could not be computed using integer arithmetic, after suitably rescaling the features so as not too lose much precision. The resulting classifier would then run much faster (on most processors). This has not been tried in the present work.

If eager recognition is running, classification must occur at every mouse point, and the number of classes is $2C$. This puts a ceiling on the number of the classes that the eager recognizer can discriminate between in real-time. On a MicroVAX II, the cost per mouse point includes `FvAddPoint` (0.22 msec) plus `FvCalc` (0.34 msec) plus the per class evaluation of $2C$ classes, $0.54C$. If mouse points come at a maximum rate of one every 25 milliseconds, $C = 45$ classes would consume the entire processor. Practically, since there is other work to do (*e.g.* inking), $C = 20$ is

probably the maximum that can be reasonably expected from an eager recognizer on a MicroVAX II. On today's processors, instead of computation time, the limiting factor will be the lower recognition rate when given a large number of classes.

One approach tried to increase the number of classes in eager recognizers was to use only a subset of the features. While this improved the response time of the system, the performance degraded significantly, so the idea was abandoned. There is no point getting the wrong answer quickly.

9.1.7 Training Time

The stated goal of the thesis work is to provide tools that allow user interface designers to experiment with gesture-based systems. One factor impacting on the usability of such tools is the amount of time it takes for gesture recognizers to retrain themselves after changes have been made to the training examples. In almost all trainable character recognizers, deleting even a single training example requires that the training be redone from scratch. For some technologies, notably neural networks, this retraining may take minutes or even hours. Such a system would not be conducive to experimenting with different gesture sets.

By contrast, statistical classifiers of the kind described in Chapter 3 can be trained very rapidly. Training the classifier from scratch requires $O(EF)$ to compute the mean feature vectors, $O(EF^2)$ time to calculate the per-class covariance matrices, $O(CF^2)$ to average them, $O(F^3)$ to invert the average, and $O(CF^2)$ to compute the weights used in the evaluation functions. If the average covariance matrix is singular, an $O(F^4)$ algorithm is run to deal with the problem.

Often, a fair amount of work can be reused in retraining after a change to some training examples. Adding or deleting an example of a class requires $O(F)$ work to incrementally update its per-class class mean vector, and $O(F^2)$ work to incrementally update its per-class covariance matrix [137]. Retraining then involves repeating the steps starting from computing the average covariance matrix. Thus, for retraining, the dependency on E, the total number of examples, is eliminated. The retraining time is instead a function of the number of examples added or deleted.

The Objective C implementation does not attempt to incrementally update the per-class covariance matrix when an example is added. Instead, only the averages are kept incrementally, and the per-class covariance matrix is recomputed from scratch. This involves $O(E^{\hat{c}}F^2)$ work for each class c changed, where $E^{\hat{c}}$ is the number of training examples for class c. This results in worse performance when a small number of examples are changed, but better performance when all the examples of a class are deleted and a new set entered. The latter operation is common when experimenting with gesture-based systems.

The author has implemented both C and Objective C versions of the single path classifier. Besides maintaining the per-class covariance matrices incrementally, the C version differs in that it does not store the list of examples that have been used to train it. (It is not necessary to store the list to add and remove examples, since the mean vector and covariance matrix are updated incrementally.) It is thus more efficient since it does not need to maintain the lists of examples. (Objective C's `Set` class, implemented via hashing, is used to maintain the lists in that version.) It also does not have the overhead of separate objects for examples, classes and classifiers that the Objective C version has (see Section 7.5).

Processor	Time (milliseconds per call)			
	`sAddExample`	`sRemoveExample`	`sDoneAdding` (10 classes)	`sDoneAdding` (30 classes)
MicroVAX II	3.7	3.8	130	234
MicroVAX III	0.90	0.90	43	78
PMAX-3100	0.024	0.026	14	22

Table 9.4: Speed of classifier training

Since only the C version could be ported to the PMAX-3100, it was used for the timings. (C versions of the feature computation and gesture recognition were used for the timings above; however in these cases the Objective C methods are straightforward translations of their corresponding C functions. In some cases, the methods merely call the corresponding C function.) The following C functions encapsulate the process of training a classifier:

`sClassifier sNewClassifier()` allocates and returns a handle to a new classifier. Initially it has no classes and no examples. The "s" at the beginning of the type and function names refers to "single-path"; there are corresponding types and functions for the multi-path classifiers.

`sAddExample(sClassifier sc, char *classname, Vector e)` adds the training example (feature vector `e`) to the named class `classname` in the passed classifier. The class is created if it has not been seen before. Linear search is used to find the class name; however, it is optimized for successive calls with the same name. The `sAddExample` function incrementally maintains the per-class mean vectors and covariance matrices.

`sRemoveExample(sClassifier sc, char *classname, Vector e)` removes example `e`, assumed to have been added earlier, from the named class. The per-class mean vector and covariance matrix are incrementally updated.

`sDoneAdding(sClassifier sc)` trains the classifier on its current set of examples. It computes the average covariance matrix, inverts it (fixing it if singular), and computes the weights.

`sClass sClassify(sClassifier sc, Vector e, double *p, *d2)` actually performs the classification of `e`. If `p` is non-`NULL` the probability of ambiguity is estimated; if `d2` is non-`NULL` the estimated Mahalanobis distance of `e` to its computed class is returned. This is the function timed in the previous section.

The functions were exercised first by adding every example in the training set, training the classifier, and then looping, removing and then re-adding 10 consecutive examples before retraining. No singular covariance matrix was encounted, due to the large number of examples. Table 9.4 shows the performance of the various routines.

Even on a MicroVAX II, training a 30 class classifier once all the examples have been entered takes less than a quarter second. Thus GRANDMA is able to produce a classifier immediately the

first time a gesture is made over a set of views whose combined gesture set has not been encountered before (see Sections 7.2.2 and 7.4). The user has to wait, but does not have to wait long.

9.2 Eager recognition

This section evaluates the effectiveness of the eager recognition algorithm on several single-stroke gesture sets. Recall that eager recognition is the recognition of a gesture while it is being made, without any explicit indication of the end of the gesture. Ideally, the eager recognizer classifies a gesture as soon as enough of it has been seen to do so unambiguously (see Chapter 4).

In order to determine how well the eager recognition algorithm works, an eager recognizer was created to classify the eight gestures classes shown in 9.17. Each class named for the direction of its two segments, e.g. ur means "up, right." Each of these gestures is ambiguous along its initial segment, and becomes unambiguous once the corner is turned and the second segment begun.

The eager recognizer was trained with ten examples of each of the eight classes, and tested on thirty examples of each class. The figure shows ten of the thirty test examples for each class, and includes all the examples that were misclassified.

Two comparisons are of interest for the gesture set: the eager recognition rate versus the recognition rate of the full classifier, and the eagerness of the recognizer versus the maximum possible eagerness. The eager recognizer classified 97.0% of the gestures correctly, compared to 99.2% correct for the full classifier. Most of the eager recognizer's errors were due to a corner looping 270 degrees rather than being a sharp 90 degrees, so it appeared to the eager recognizer the second stroke was going in the opposite direction than intended. In the figure, "E" indicates a gesture misclassified by the eager recognizer, and "F" indicates a misclassification by the full classifier.

On the average, the eager recognizer examined 67.9% of the mouse points of each gesture before deciding the gesture was unambiguous. By hand, the author determined for each gesture the number of mouse points from the start through the corner turn, and concluded that on the average 59.4% of the mouse points of each gesture needed to be seen before the gesture could be unambiguously classified. The parts of each gesture at which unambiguous classification could have occurred but did not are indicated in the figure by thick lines.

Figure 9.18 shows the performance of the eager recognizer on GDP gestures. The eager recognizer was trained with 10 examples of each of 11 gesture classes, and tested on 30 examples of each class, five of which are shown in the figure. The GDP gesture set was slightly altered to increase eagerness: the group gesture was trained clockwise because when it was counterclockwise it prevented the copy gesture from ever being eagerly recognized. For the GDP gestures, the full classifier had a 99.7% correct recognition rate as compared with 93.5% for the eager recognizer. On the average 60.5% of each gesture was examined by the eager recognizer before classification occurred. For this set no attempt was made to determine the minimum average gesture percentage that needed to be seen for unambiguous classification.

From these tests we can conclude that the trainable eager recognition algorithm performs acceptably but there is plenty of room for improvement, both in the recognition rate and the amount of eagerness.

Computationally, eager recognition is quite tractable on modest hardware. A fixed amount of

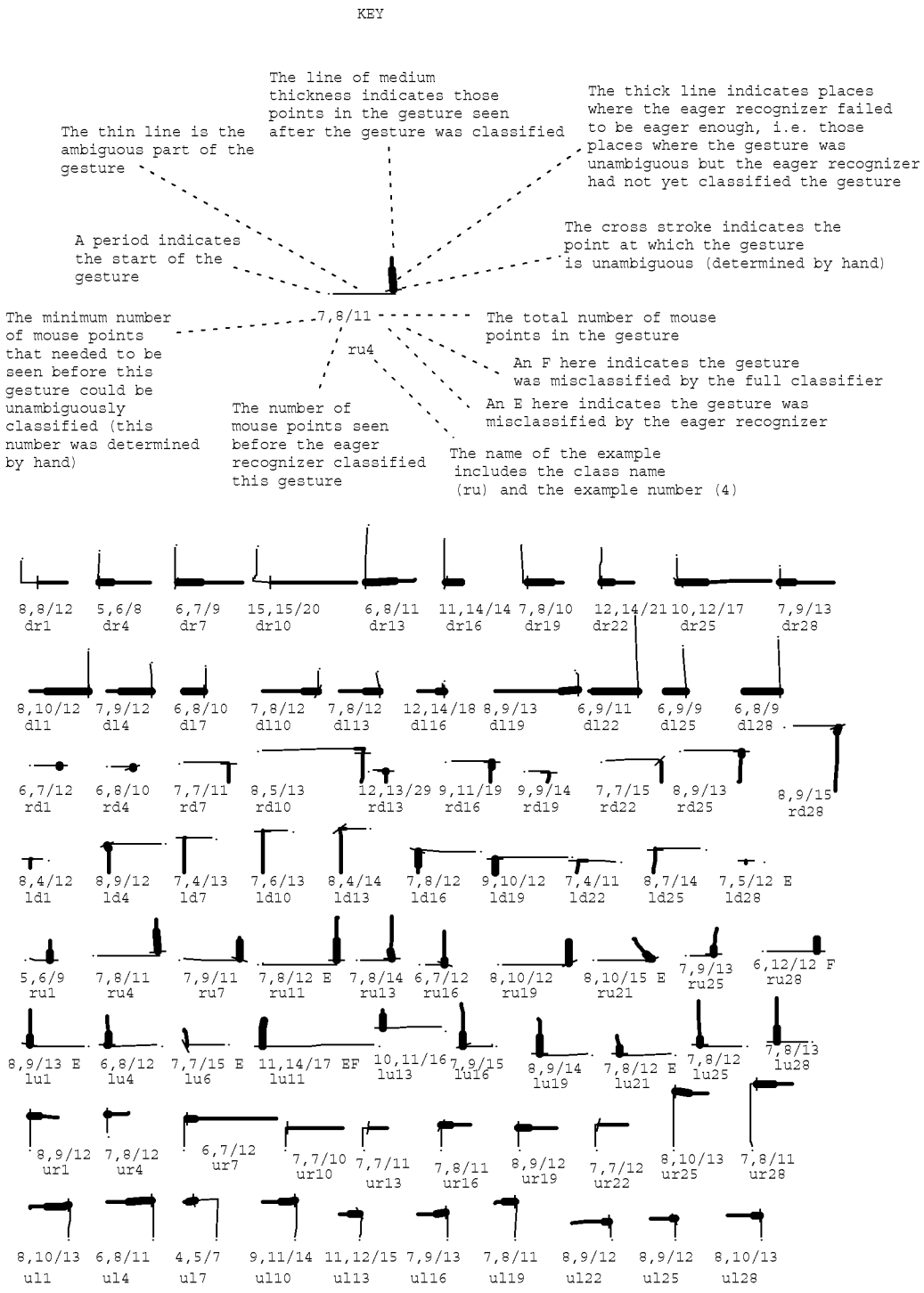


Figure 9.17: The performance of the eager recognizer on easily understood data

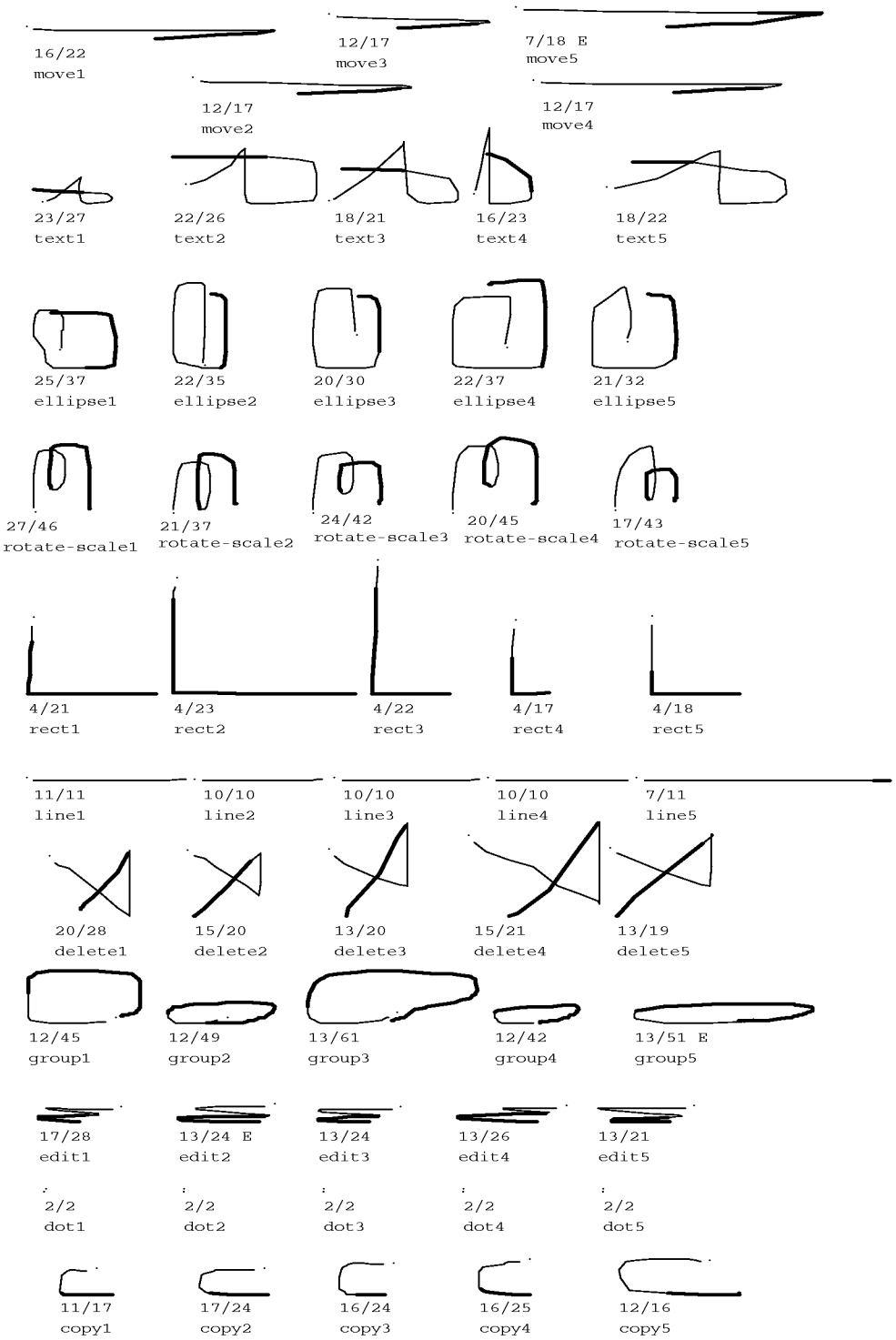


Figure 9.18: The performance of the eager recognizer on GDP gestures
The transitions from thin to thick lines indicate where eager recognition occurred.

computation needs to occur on each mouse point: first the feature vector must be updated (taking 0.5 msec on a DEC MicroVAX II), and then the vector must be classified by the AUC (taking 0.27 msec per class, or 6 msec in the case of GDP).

9.3 Multi-finger recognition

Multi-finger gestural input is a significant innovation of this work. Unfortunately, circumstances have conspired to make the evaluation of multi-finger recognition both impossible and irrelevant. The Sensor Frame is the only input device upon which the multi-finger recognition algorithm was tested. Unfortunately, there is only one functioning Sensor Frame in existence, and that was damaged sometime after the multi-finger recognition was running, but before formal testing could begin. (Fortunately, a videotape of MDP in action was made while the Sensor Frame was working.) No progress was made in repairing the Sensor Frame for over a year; testing was thus impossible. Eventually the Sensor Frame was repaired, but Sensor Frame, Inc. went out of business shortly afterward, making any detailed evaluation irrelevant. The owner of the Sensor Frame has left the country, taking the device with him.

An informal estimate of the multi-finger recognition accuracy may be estimated from ten minutes of videotape of the author using MDP. This version of MDP uses the path sorting multi-finger recognition algorithm (Section 5.2). As shown in figure 8.10, MDP recognizes 11 gestures (6 one finger gestures, 3 two finger gestures, and 2 three finger gestures). In the videotape, the author made 30 gestures, 2 of which appear to have been misclassified, and one of which was rejected, resulting in a correct recognition rate of 90%. The processing time appears to be negligible.

All three misclassifications are the result of the Sensor Frame seeing more fingers in the gesture than were intended. This was due to knuckles of fingers curled up (so as not to be used in the gesture) accidentally penetrating the sensing plane and being counted as additional fingers. As there are distinct classifiers for single finger, two finger, and three finger gestures, an incorrect number of fingers inevitably leads to a misclassification. While it is possible to imagine methods for dealing with such errors during recognition, the main cause of this problem is the ergonomics of the Sensor Frame.

For the small gesture set examined, the recognition rate is 100% once the errors due to "extra fingers" are eliminated. This is to be expected, given the small number of gestures for each number of fingers. It is expected that the multi-path classifier operating on one finger gestures would perform about as well as the single-path classifier, as the algorithms are essentially identical. The single-path classifier, when given only six classes to discriminate among, has been shown (on mouse data) to perform at 100% in almost all cases. When operating on two finger gestures, it is expected that the performance of the recognition algorithm would be similar to that of the single-path classifier on *twice* the number of classes. Actually, it is possible that some of the paths in the two-finger gestures will be similar to other paths in the set, and be merged into a single class by the training algorithm (Section 5.4). Thus, when the number of unique paths will be less than twice the number of two-finger gesture classes, performance may be expected to improve accordingly. Similarly, the three finger gesture classifier may be expected to perform as well as a single-path classifier the recognizes between one and three times the number of three finger gesture classes, depending on

the number of unique paths in the class set.

One more factor to consider is that mouse data tends to be much less noisy than Sensor Frame data. The triangulation by the Sensor Frame is erratic, especially when multiple fingers are being tracked. For example, both horizontal segments of the Parallelogram gesture of figure 8.10 should be straight lines. Until this problem can be solved, it is expected that recognition rates for Sensor Frame gesture sets will suffer.

9.4 GRANDMA

Evaluating GRANDMA is much more subjective than evaluating the low-level recognition rates. GRANDMA may be evaluated on several levels: the effort required to build new interaction techniques, to build new applications, to add gestures to an application, to change an application's gestures, or to use an application to perform a task.

No attempt was made to formally evaluate any of these. In order to get statistically valid results, it would have been necessary to run carefully designed experiments on a number of users, something the author had neither the time, space, inclination, or qualifications to do. Furthermore, the author does not wish to claim that GRANDMA is superior to existing object-oriented toolkits for any particular task. GRANDMA is simply the platform through which some ideas for input processing in object-oriented toolkits were explored. GRANDMA's significance, if any, will be its influence on future toolkits, rather than any more direct results.

Nonetheless, this section informally reports on the author's experience building gesture-based systems with GRANDMA. (No one besides the author tried to program with GRANMDA.) Also, in order to confirm that GRANDMA can be used by someone other than the author, this section also reports on observations of a subject trying to use GSCORE and GRANDMA to due some tasks.

9.4.1 The author's experience with GRANDMA

GRANDMA took approximately seven months to design and develop. It consists of approximately 12000 lines of Objective C code. There are an additional 5000 lines of C code which implement a graphics layer as well as the feature vector calculation. GDP took an additional 2000 lines of Objective C code to implement. GDP was developed at the same time as GRANDMA, as it was the primary application used to test GRANDMA.

Initially, only two GDP gestures were used to test GRANDMA's gesture handler and associated utilities. Once these were working well, it took four days to add the remaining gestures to GDP. Most of this time was spent writing Objective C methods to use in semantic expressions. These were methods that were not needed for the existing direct manipulation interface.

GSCORE consists of 6000 lines of Objective C code. It took six weeks to design and implement GSCORE, including its palette-based interface. Much of this time was spent on the details of representing common music notation, mechanisms for displaying music notation, and producing usable music fonts. The palette, an interaction technique that did not as yet exist in GRANDMA, took about eight hours to implement. It took two weeks to add the gestural interface to GSCORE,

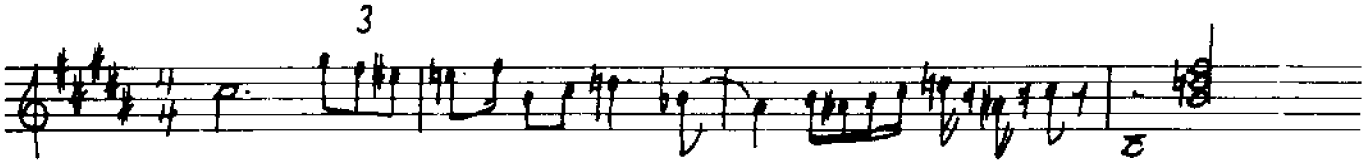


Figure 9.19: PV's task

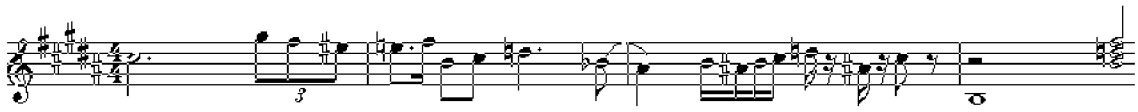


Figure 9.20: PV's result

including writing some additional methods. Much of this time was spent experimenting with different semantics for the gestures.

Section 10.1.3 lists features of GRANDMA that will be important to incorporate into future toolkits that support gestures.

9.4.2 A user uses GSCORE and GRANDMA

This sections informally reports on subject PV's (see Section 9.1.5) attempts to use the GSCORE program.

The task was to enter the music shown in figure 9.19. The music was chosen to exercise many of the GSCORE gestures. PV is an experienced music copyist, and it took him 100 seconds to write out the music as shown, copying it from an earlier attempt.

Using gestures, the author was able to enter the above score in 280 seconds (almost five minutes). He made a total of 53 gestures, four of which did not give the desired results and were immediately undone. Only two of those were misclassifications; the other two were notes gestures where the note was created having the wrong pitch, due to misplacement of the cursor at the start of the gesture. Turning off gestures and using only the palette interface, it took the author 670 seconds (eleven minutes). No mistakes needed to be undone in the latter trial.

PV's first attempt was at using the GSCORE program trained with the author's gestures. PV had already gained experienced with this set of gestures during the study of interuser variation. PV practiced for one half hour with the GSCORE program before attempting the task. The author coached PV during this time, as no other documentation or help was available.

PV took 600 seconds (10 minutes) to complete the task. He made a total of 73 gestures, 16 of which were immediately undone. It appeared to the author, who was silently observing, that each undo was used to recover from a misclassification. Figure 9.20 shows the product of his labor. PV then turned off gestures, and used the palette interface to enter the example. He completed the task in 680 seconds (11 minutes).

PV then entered his own gestures in place of some of the author's. In particular, he substituted his own gestures for the nine classes: delete, move, beam, 1r, 2r, 8r, 16r, keysig, and bar, entering 15 or more examples of each. The total time to do this, including incremental testing of the new gestures and periodic saves to disk, was 25 minutes. PV did not attempt to emulate the author's gestures; instead, he used the forms he had created earlier (see Section 9.1.5).

Once done, PV repeated the experiment. It took him 310 seconds (5 minutes) to enter the music. He made 58 gestures, 4 of which were undone.

PV was interviewed after the tests, and made the following comments: The biggest problem, he stated, was that mouse tracking in the GSCORE program was much more sluggish than in the recorder and tester. (This is accurate, as the time required for GSCORE events to be created and consumed adds significant overhead to the mouse tracking. Much of this is overhead due to GRANDMA.) PV characterized the system as "sluggish." Bad tracking, especially at the start of the gesture, contributed significantly to the number of misclassifications.

PV stated that he thought the system "intuitive" for entering notes. He described the gesture-based interface as "excellent" compared to the palette-based system, but when asked how the gesture-based interface compared to writing on paper, he replied "it sucks." He did not like using the mouse for gesturing, and believed that a stylus and tablet would be much better.

It is again difficult to draw conclusions from an informal study of one user. Did PV's performance improve because he tailored the gestures to his liking, or because he had been practicing? This is unknown. Some things are clear: GRANDMA makes it easy to experiment with new gesture sets, and, in GSCORE, with moderate practice the gesture-based interface improved task performance by a factor of two over the palette-based interface. Whether gesture-based interfaces generally improve task performance over non-gesture-based interfaces is a question that requires *much* further study.

Chapter 10

Conclusion and Future Directions

This chapter summarizes the contributions of this thesis and indicates some directions for future work.

10.1 Contributions

This thesis makes contributions in four areas:

- New interaction techniques
- New recognition-related algorithms
- Integrating gestures into interfaces
- Input in object-oriented toolkits

Each of these will be discussed in turn.

10.1.1 New interactions techniques

A major contributions of this work has been the invention and exploration of three new interaction techniques.

The two-phase single-stroke interaction The two-phase interaction enables gesture and direct manipulation to be integrated in a single interaction that combines the power of each. The first phase is collection, during which the points that make up the gesture are collected. In the simplest case, the end of the collection phase is indicated by a motion timeout, classification occurs, and the second phase, manipulation, is entered. In the manipulation phase, the user moves the mouse to manipulate some parameters in the application. The particular parameters manipulated depend on the classification of the collected gesture. The collection phase is like character entry in handwriting interfaces; the manipulation phase is like a drag interaction in direct-manipulation interfaces. Generally, the operation, operands, and some parameters are

determined at the phase transition (when the gesture is recognized), and then the manipulation phase allows additional parameters to be set in the presence of application feedback.

Eager recognition Eager recognition is a modification of the two-phase single-stroke interaction in which the phase transition from gesturing to manipulation occurs as soon as enough of the gesture has been seen so that it may be classified unambiguously. The result is an interaction that combines gesturing and direct manipulation in a single, smooth interaction.

The two-phase multiple-finger interaction Gesture and direct manipulation may be combined for multiple path inputs in a way similar to the two-phase single-stroke interaction. With multiple finger input, opportunities exist for expanding the power of each phase of the interaction. By allowing multiple fingers in the collection phase, the repertoire of possible gestures is greatly increased, and a multiple finger gesture allows many parameters to be specified simultaneously when the gesture is recognized. Similarly, even when only one finger is used for the gesture, additional fingers may be brought in during the manipulation phase. Thus, the two-phase multiple-finger interaction allows a large number of parameters to be specified and interactively manipulated.

10.1.2 Recognition Technology

This thesis discloses five new algorithms of general utility in the construction and use of gesture recognizers.

Automatic generation of single-stroke gesture recognizers from training examples A practical and efficient algorithm for generating gesture recognizers has been developed and tested. In it, a gesture is represented as a vector of real-valued features, and a standard pattern recognition technique is used to generate a linear classifier that discriminates between the vectors of different gesture classes. The training algorithm depends on aggregate statistics of each gesture class, and empirically it has been shown that usually only fifteen examples of each class are needed to produce accurate recognizers. It is simple to incorporate dynamic attributes, such as the average speed of the gesture, into the feature set. The algorithm has been shown to work even when some classes vary in size and orientation while others depend on size or orientation to be recognized. The recognizer size is independent of the number of training examples, and both the recognition and training times have been shown to be small. A features set that is both meaningful and extensible potentially allows the algorithm to be adapted to future input devices and requirements.

Incremental feature calculation The calculation used to generate features from the input points of a gesture is incremental, meaning that it takes constant time to update the features given a new input point. This allows arbitrarily large gestures to be processed with no delay in processing.

Rejection algorithms Two algorithms for rejecting ill-formed gestures have been developed and tested. One estimates the probability of correct classification, enabling input gestures that are ambiguous with respect to a set of gesture classes to be rejected. The other uses a normalized

distance metric to determine how close an input gesture is to the typical gesture of the its computed class, allowing outliers to be rejected.

Automatic generation of eager recognizers from training examples An eager recognizer classifies a gesture as soon as it is unambiguous, alleviating the need for the end of the gesture to be explicitly indicated. An algorithm for generating eager recognizers from training examples has been developed and tested. The algorithm produces a two-class classifier which is run on every input point and used to determine if the gesture being entered is unambiguous.

Automatic generation of multi-path gesture recognizers The single-stroke recognition work has been extended so that a number of single-stroke recognizers may be combined into a multi-finger gesture recognizer. The described algorithm produces a multi-path recognizer given training examples. Relative path timing information is considered during the recognition, and global classification is attempted when the individual path classifications do not uniquely determine the class of the multi-path gesture. For dealing with the problems that arise from multi-path input devices that do not *a priori* determine "which path is which," two approaches, path sorting and path clustering, have been explored. The resulting algorithm has been demonstrated using the Sensor Frame as a multi-finger input device.

10.1.3 Integrating gestures into interfaces

A paradigm for integrating gestures into object-oriented interfaces has been developed and demonstrated. The key points are:

A gesture set is associated with a view or view class. Each class of object in the user interface potentially responds to a different set of gestures. Thus, for example, notes respond to a different set of gestures than staves in the GSCORE music editor.

The gesture set is dynamically determined. From the first point of a gesture, the system dynamically determines the set of gestures possible. The first point determines the possible views at which the gesture is directed. For each of these views, inheritance up the class hierarchy determines the set of gestures it handles. These sets are combined, and if need be, a classifier for the resulting union is dynamically created.

The gesture class and attributes map to an application operation, operands, and parameters. Gestures are powerful because they contain additional information beyond the class of the gesture. The attributes of a gesture, such as size, orientation, length, speed, first point, and enclosed area, can all be mapped to parameters (including operands) of application routines. In the two-phase interaction, after the gesture is recognized there is an opportunity to map subsequent input to application parameters in the presence of application feedback.

Gesture handlers may be manipulated at runtime. In order to encourage exploration of gesture-based systems, all aspects of the gestural interface can be specified while the application is running. A new gesture handler may be created at runtime and associated with one or more views or view classes. Gesture classes may be added, deleted, or copied from other handlers.

Examples of each gesture class can be entered and modified at runtime. Finally, the semantics of the gesture class can be entered and modified at runtime. Three semantic expressions are specifiable: one evaluated when the gesture is first recognized, one evaluated on each subsequent mouse point, and one evaluated when the interaction completes.

10.1.4 Input in Object-Oriented User Interface Toolkits

A number of new ideas in the area of input in object-oriented user interface toolkits arose in the course of this work.

Passive and active event handlers A single passive event handler may be associated with multiple views. When input occurs on one such view, the handler usually activates a copy of itself. Thus, the active/passive dichotomy eliminates the need to have a controller object instantiated for each view that expects input, a major expense in many MVC systems.

Event handlers may be associated with view classes Instead of having to associate a handler with every instance of a view, the handler may be associated with one or more view classes. A view may have multiple handlers associated with it, and handlers are queried in a specific order to determine which handler will handle particular input.

Unified mouse input and virtual tools All input devices are tools, but when desired a single input device may at times be different tools, one way to implement modes in the interface. Tools may also be software objects, and some views are indeed such virtual tools. Tools often have an action, which allows them to operate on any views that respond to that action. The test of whether a given view responds to a given tool is made by an event handler associated with every view; this allows semantic feedback to occur automatically without any explicit action on the part of the view or the tool.

Automatic semantic feedback As just mentioned, the feedback as to whether a given tool operates upon a view over which it is has been dragged happens automatically. For example, objects that respond to the delete message will automatically highlight when a delete tool is dragged over them. If desired, an object can do more elaborate processing to determine if it truly responds to a given tool, e.g. an object may check that the user has permission to delete it before indicating it responds to the delete tool.

Runtime creation and manipulation of event handlers Event handlers may be created and associated with views or view classes at runtime. For example, a drag handler may be associated with an object, allowing that object to be dragged (i.e. have its position changed). In addition, such handlers may be modified at runtime, for example, to change the predicate that activates the handler.

10.2 Future Directions

In this section, directions for future work are discussed. These directions include remedies for deficiencies of the current work as well as extensions.

The single-stroke training and recognition algorithm is the most robust and well-tested part of the current work, and even in its current form it is probably suitable for commercial applications. However, a number of simple modifications should improve performance. Sections 9.1.1 and 9.1.5 contain suggestions for additional features as well as modifications to existing features; these should be implemented. Tracking the mouse in the presence of paging has proved to be a problem, and a significant improvement in recognition rate would be achieved if real-time response to mouse events could be guaranteed.

It should be simple to extend the algorithm to three dimensional gestures. All that would be required would be to add several more features to capture motion in the extra dimension. The training algorithm and linear classifier would be untouched by this extension.

Alternatives for rejection should be explored further. The estimated probability of ambiguity is useful, though using it will always result in rejections of about as many gestures that would have been correctly classified as not. The estimated Mahalanobis distance based on the common covariance matrix is really only useful for rejecting deliberately garbled gestures. The Mahalanobis distance based on the per-class covariance matrix fares somewhat better, but requires significantly more training examples to work well.

Given the obvious false assumption of equal per-class covariance matrices, it seems that the statistical classifier should not perform well on gesture sets, some classes of which vary in size and orientation, others of which do not. In practice, when the gesture classes are unambiguous, the classifiers have tended to perform admirably. Presumably this would not be the case for all such gesture sets. One area for exploration is a method for calculating the common covariance matrix differently, in particular, by not weighing the per class contributions by the number of examples of that class.

Another challenge would be to handle such gesture sets without giving up linear classification with a closed form training formula. There seems to be only one candidate, which relies on the multiclass minimum squared error and the pseudoinverse of the matrix of examples [30]. It should be explored as a potential alternative to classifiers that rely on estimates of a common covariance matrix.

It would be interesting to explore the possibility of allowing the user to indicate declaratively that a given gesture classes will vary in size and/or orientation. This might be handled simply by generating additional training examples by varying the user-supplied examples accordingly. Alternatively, it may be possible to augment the training algorithm so that the evaluation functions for certain classes are constrained to ignore certain features.

Relaxing the requirement that a closed form exist for the per-class feature weights allows iterative training methods to be considered. They have been ignored in this dissertation since they are expensive in training time and tend to require many training examples. However, as processor speed increases, iterative methods become more practical for use in a tool for experimenting with gesture-based interfaces.

Similarly, relaxing the requirement that the classifier be a linear discriminator opens the door for many other possibilities. Quadratic discrimination, and various non-parametric discrimination algorithms are but a few. These too are expensive and require many training examples.

Perhaps recognition technologies that require expensive training may be used in a production

system while the cheaper technology developed here used for prototyping. This is analogous to using a fast compiler for development and an optimizing compiler for production. At the time of this writing it seems likely that neural networks will soon be in common use, and gesture recognition is but one application.

Additional attention should be given to the problem of detecting ambiguous sets of gesture classes and useless features. The triangular matrix of Mahalanobis distance between each pair of gesture classes is a useful starting point for determining similar gesture classes. Multivariate analysis of variance techniques [74] can determine which features contribute to the classification and which features are irrelevant. These techniques can be used to support the design of new features.

Eager recognition needs to be explored further. The classifiers generated by the algorithm of Chapter 4 are less eager than they could possibly be, due to the conservative choices being made. Hand labeling of ambiguous and unambiguous subgestures should be explored more fully; it is not difficult to imagine an interface that makes such labeling relatively painless, and it is likely to give better results than the current automatic labeling. Another possible improvement comes from the observation that, during eager recognition, the full classifier is being used to classify subgestures, upon which it was not trained. It might be worth trying to retrain the full classifier on the complete subgestures. Even better, perhaps a new classifier, trained on the *newly* complete subgestures (*i.e.* those made complete by their last point), should be substituted for the full classifier. Also, eager recognition needs to be extended to multi-path gestures.

Algorithms for automatically determining the start of a gesture would also be useful, especially for devices without any discrete signaling capability (most notably the DataGlove). In the current work, gestures are considered atomic, essentially having no discernible structure. It is easy to imagine separate gestures such as select, copy, move, and delete that are concatenated to make single interactions: select and move, select and delete. This raises the segmentation question: when does one gesture end and the next begin? Specifying allowable combinations of gestures opens up the possibility of gesture grammars, an interesting area for future study.

This dissertation has concentrated on single-path gestures that are restricted to be single strokes, for reasons explained previously. The utility of multiple-stroke gestures needs to be examined more thoroughly. In a multiple-stroke gesture, does the relaxation between strokes ruin the correspondence between mental and physical tension that makes for good interaction? Does the need for segmentation make the system less responsive than it otherwise might be? Can a manipulation phase and eager recognition be incorporated into a system based on multiple-stroke gestures? These questions require further research.

Due to the interest in multiple stroke recognition, the question arises as to whether the single-stroke algorithm can be extended to handle multiple stroke gestures. First, the segmentation problem (grouping strokes into gestures) needs to be addressed. One way this might be done is to add a large timeout to determine the end of a gesture. The distance of a stroke from the previous stroke might also be used. A sequence of strokes determined to be a single gesture might then be treated as a single stroke, with the exception of an additional feature which records the number of strokes in the gesture. The single-stroke recognition algorithm may then be applied.

Multi-path recognition is really still in its infancy. While the recognition algorithms of Chapter 5 seem to work well, there is not much to compare them against. Many others methods for multi-path

recognition need to be explored. That said, the author is somewhat wary that multiple finger input devices are so seductive that gesture research will concentrate on such devices to the exclusion of single-path devices. This would be unfortunate, as it seems likely that single-path devices will be much more prevalent for the foreseeable future, and thus more users will potentially benefit from the availability of single-path gesturing. Also, a thorough understanding of the issues involved in single-path gesturing will likely be of use in solving the more difficult problems encountered in the multi-path case.

The advent of pen-based computers leads to the question of how the single-stroke recognition described here may be combined with handwriting recognition. One approach is to pass input to the gesture recognizer after it has been rejected by the handwriting recognizer. The context in which the stroke has been made (e.g. drawing window or text window) can also be used to determine whether to invoke handwriting recognition or stroke recognition first.

The start of a single-stroke gesture is used to determine the set of possible gestures by looking at possible objects at which the gesture is directed. It may be desirable to explore the possibility that the gesture is directed at an object other than one indicated by the first point, e.g. an object may be indicated by a hot point of the gesture (e.g. the intersection point of the delete gesture). A similar ambiguity occurs when the input is a multiple-finger gesture; which of the fingers should be used to determine the object(s) at which the gesture is directed? In this case, a union of the gestures recognized by objects indicated by each finger could be used, but the possibility of conflict remains.

One problem with gesture-based systems is that there is usually no indication of the possible gestures accepted by the system.[1] This is a difficulty that will potentially prevent novices from using the system. One approach would be to use animation[6] to indicate the possible gestures and their effects, although how the user asks to see the animation remains an open question.

Also daunting to beginners is the timeout interval, where "stillness" is used to indicate that collection is over and manipulation is to begin. Typically, a beginner presses a mouse button and then thinks about what to do next; by that time the system has already classified the gesture as a dot. The timeout cannot be totally disabled, since it is the only way to enter the manipulation phase for some gestures. Perhaps some scheme where the timeouts are long (0.75 seconds) for novices and decrease with use is desirable. Another possibility is eliminating the timeout totally at the beginning of the gestures, thus disallowing dot gestures.

The current work suffers from a lack of formal user evaluation. Additional studies are needed to determine classifier performance as a function of training examples, and whether one user can use a classifier trained by another. In general, the costs and the benefits of fixed verses trainable recognition strategies need to be studied. The usability of eager recognizers is also of interest.

Recognizers that gradually adapt to users need to be studied as well. Such a recognizer requires the user to somehow indicate when a gesture is misclassified by the system. Lerner [78] demonstrated a potentially applicable scheme in which the system monitored subsequent actions to see if the user was satisfied with the result of an applied heuristic. There are dangers inherent in doubly-adaptive systems–if the system adapts to the user and the user to the system, both are aiming at moving targets, and thrashing is possible. The current approach requires the user explicitly to replace the

[1] Kurtenbach et. al. [75] say that gesture-based interfaces are "non-revealing," and present an interesting solution that unifies gesturing and pie-menu selection.

existing training examples with his own–a workable, if not glamorous, solution.

The low-level recognition work in this thesis is quite usable in its current state, and may be directly incorporated into systems as warranted. GRANDMA, however, is not useful as a base for future development. It is purely a research system, built as a platform for experimenting with input in user interface toolkits. Its output facilities are totally inadequate for real applications. GRANDMA was built solely by and for the author, who has no plans to maintain it. Nonetheless, GRANDMA embodies some important concepts of how gestures are to be integrated into object-oriented user interface tools.

The obvious next step is to integrate gestures into some existing user interface construction tools. Issues of technical suitability are important, but not paramount, in deciding which system to work on. Any chosen system must be well supported and maintained, so that there is a reasonable assurance that the system will survive. Furthermore, any chosen system must be widely distributed, in order to make the technology of gesture recognition available to as many experimenters as possible.

A number of existing systems are candidates for the incorporation of gestures. The NeXT Application Kit is technically the ideal platform–it is even programmed in Objective C. The appropriate hooks seem to be there to capture input at the right level in order to associate gestures with view classes. It is probably not worth the effort to implement an entire interpreter for entering gesture semantics at runtime, as this is not something a user will typically manipulate. A graphical interface to control semantics, based on constraints, would be an interesting addition. In general, a simpler way for mapping gestural attributes to application parameters needs to be determined.

The Andrew Toolkit (ATK) is another system into which gestures may be incorporated. ATK uses its own object-oriented programming language on top of C, so runtime representation of the class hierarchy, if not already present, should be straightforward to add. ATK has implemented dynamic loading of objects into running programs–this should make it possible to compile gesture semantics and load them into a running program without restarting the program. Unfortunately, due to their overhead, views tend to be large objects in ATK (e.g. individual notes in a score editor would not be separate views in ATK) making it difficult to associate different gestures with the smaller objects of interest in the interface. Scott Hassan, in a different approach, has added the author's gesture recognizer to the ATK text object, creating an interface that allows text editing via proofreader's marks.

Integrating gestures into Garnet is another possibility. What would be required is a gesture interactor, analogous to the gesture event handler in GRANDMA. Garnet interactors routinely specify their semantics via constraints, with an escape into Lisp available for unusual cases. Specifying gesture semantics should therefore be no problem in Garnet. James Landay has begun work integrating the author's recognizer into Garnet.

Gestures could also be added to MacApp. Besides being widely used, MacApp has the advantage that it runs on a Macintosh, which historically has run only one process at a time and has no virtual memory (this has changed with a recent system software release). While these points sound like disadvantages, the real-time operation needed to track the mouse reliably should be easy to achieve because of them. Because MacApp is implemented in Object Pascal, minimal meta-information about objects is available at runtime. In particular, message selectors are not first class objects in Object Pascal, it is not possible to ask if a given object responds to a message at runtime, and there

is no runtime representation of the class hierarchy. Many things that happen automatically because GRANDMA is written in Objective C will need to be explicitly coded in MacApp.

It would be desirable to have additional attributes of the gesture available for use in gesture semantics. Notably missing from the current set are locations where the path intersects itself and locations of sharp corners of the stroke. Both kinds of attributes can be used for pointing with a gesture, and allow for multiple points to be indicated with one single-path gesture. Also, having the numerical attributes also available in a scaled form (*e.g.* between zero and one) would simplify their use as parameters to application functions.

10.3 Final Remarks

The utility of gesture-based interfaces derives from the ability to communicate an entire primitive application transaction with a single gesture. For this to be possible, the gesture needs to be classified to determine the operation to be performed, and attributes of the gesture must be mapped to the parameters of the operation. Some parameters may be culled at the time the gesture is recognized, while others are best manipulated in the presence of feedback from the application. This is the justification for the two-phase approach, where gesture recognition is followed by a manipulation phase, which allows for the continuous adjustment of parameters in the presence of application feedback.

From the outset, the goal of this work was to provide tools to allow the easy creation of gesture-based applications. This research has led to prototypes of such tools, and has thus laid much of the groundwork for building such tools in the future. However, the goal will not have been achieved until gestures are integrated into existing user interface construction tools that are both well maintained and highly available. This involves more development and marketing than it does research, but it is vitally important to the future of gesture-based systems.

Appendix A

Code for Single-Stroke Gesture Recognition and Training

This appendix contains the actual C code used to recognize single-stroke gestures. The feature vector calculation, classifier training algorithm, and the linear classifier are all presented. The code may be obtained free of charge via anonymous ftp to emsworth.andrew.cmu.edu (subdirectory gestures) and is also available as part of the Andrew contribution to the X11R5 distribution.

A.1 Feature Calculation

The lowest level of the code deals with computing a feature vector from a sequence of mouse points that make up a gesture. Type `FV` is a pointer to a structure that holds a feature vector as well as intermediate results used in the calculation of the features. The function `FvAlloc` allocates an `FV`, which is initialized before processing the points of a gesture via `FvInit`. `FvAddPoint` is called for each input point of the gesture, and `FvCalc` returns the feature vector for the gesture once all the points have been entered.

The following is a sample code fragment demonstrating the use of these functions:

```
#include "matrix.h"
#include "fv.h"

Vector
InputAGesture()
{
    static FV fv;
    int x, y; long t; Vector v;

    /* FvAlloc() is typically called only once per program invocation. */
    if(fv == NULL) fv = FvAlloc();
```

```
    /* A prototypical loop to compute a feature vector from a gesture
       being read from a window manager: */
    FvInit(fv);
    while(GetNextPoint(&x, &y, &t) != END_OF_GESTURE)
        FvAddPoint(fv, x, y, t);
    v = FvCalc(fv);
    return v;

}
```

The returned vector `v` might now be passed to `sClassify` to classify the gesture.

The remainder of this section shows the header file, fv.h, which defines the `FV` type and the feature vector interface. This interface is implemented in fv.c, shown next.

```
/*****************************************************************
fv.h -- Create a feature vector, useful for gesture classification,
   from a sequence of points (e.g. mouse points).
*****************************************************************/

/* ---------- compile time settable parameters ---------- */
/* some of these can also be set at runtime, see fv.c */

#undef  USE_TIME
                    /* Define USE_TIME to enable the duration and maximum */
                    /* velocity features.  When not defined, 0 may be passed */
                    /* as the time to FvAddPoint. */

#define DIST_SQ_THRESHOLD  (3*3)
                    /* points within sqrt(DIST_SQ_THRESHOLD) */
                    /* will be ignored to eliminate mouse jitter */

#define SE_TH_ROLLOFF     (4*4)
                    /* The SE_THETA features (cos and sin of */
                    /* angle between first and last point) will */
                    /* be scaled down if the distance between the */
                    /* points is less than sqrt(SE_TH_ROLLOFF) */

/* ---------- Interface ---------- */

typedef struct fv *FV;
                    /* During gesture collection, an FV holds */
                    /* all intermediate results used in the */
                    /* calculation of a single feature vector */
```

```
FV      FvAlloc();      /* */
void    FvFree();       /* Fv fv */
void    FvInit();       /* FV fv */
void    FvAddPoint();   /* FV fv; int x, y; long time; */
Vector  FvCalc();       /* FV fv; */

/*------ internal data structure ------*/
#define MAXFEATURES 32
        /* maximum number of features, occasionally useful as an array bound */

/* indices into the feature Vector returned by FvCalc */

#define PF_INIT_COS    0 /* initial angle (cos) */
#define PF_INIT_SIN    1 /* initial angle (sin) */
#define PF_BB_LEN      2 /* length of bounding box diagonal */
#define PF_BB_TH       3 /* angle of bounding box diagonal */
#define PF_SE_LEN      4 /* length between start and end points */
#define PF_SE_COS      5 /* cos of angle between start and end points */
#define PF_SE_SIN      6 /* sin of angle between start and end points */
#define PF_LEN         7 /* arc length of path */
#define PF_TH          8 /* total angle traversed */
#define PF_ATH         9 /* sum of abs vals of angles traversed */
#define PF_SQTH       10 /* sum of squares of angles traversed */

#ifndef USE_TIME
#   define NFEATURES      11
#else
#   define PF_DUR         11     /* duration of path */
#   define PF_MAXV        12     /* maximum speed */
#   define NFEATURES      13
#endif

/* structure which holds intermediate results during feature vector calculation */

struct fv {

    /* the following are used in calculating the features */
    double      startx, starty; /* starting point */
    long        starttime;      /* starting time */

    /* these are set after a few points and then left alone */
```

```
    double      initial_sin, initial_cos;  /* initial angle to x axis */

    /* these are updated incrementally upon every point */
    int     npoints;            /* number of points in path */

    double      dx2, dy2;       /* differences: endx-prevx, endy-prevy */
    double      magsq2;         /* dx2*dx2 + dy2*dy2 */

    double      endx, endy;     /* last point added */
    long        endtime;

    double      minx, maxx, miny, maxy;    /* bounding box */

    double      path_r, path_th;  /* total length and rotation (in radians) */
    double      abs_th;           /* sum of absolute values of path angles */
    double      sharpness;        /* sum of squares of path angles */
    double      maxv;             /* maximum velocity */


    Vector      y;                /* Actual feature vector */
};


/*****************************************************************
fv.c - Creates a feature vector, useful for gesture classification,
   from a sequence of points (e.g. mouse points).
*****************************************************************/

#include <stdio.h>
#include <math.h>
#include "matrix.h"  /* contains Vector and associated functions */
#include "fv.h"

/* runtime settable parameters */
double dist_sq_threshold = DIST_SQ_THRESHOLD;
double se_th_rolloff = SE_TH_ROLLOFF;

#define EPS (1.0e-4)

/* allocate an FV struct including feature vector */

FV
```

```
FvAlloc()
{
    register FV fv = (FV) mallocOrDie(sizeof(struct fv));
    fv->y = NewVector(NFEATURES);
    FvInit(fv);
    return fv;
}

/* free memory associated with an FV struct */
void
FvFree(fv)
FV fv;
{
    FreeVector(fv->y);
    free((char *) fv);
}

/* initialize an FV struct to prepare for incoming gesture points */
void
FvInit(fv)
register FV fv;
{
    register int i;

    fv->npoints = 0;
    fv->initial_sin = fv->initial_cos = 0.0;
    fv->maxv = 0;
    fv->path_r = 0;
    fv->path_th = 0;
    fv->abs_th = 0;
    fv->sharpness = 0;
    fv->maxv = 0;
    for(i = 0; i < NFEATURES; i++)
        fv->y[i] = 0.0;
}

/* update an FV struct to reflect a new input point */
void
FvAddPoint(fv, x, y, t)
register FV fv; int x, y; long t;
{
    double dx1, dy1, magsq1;
```

```
    double th, absth, d;
#ifdef PF_MAXV
    long lasttime;
#endif

    ++fv->npoints;
    if(fv->npoints == 1) {      /* first point, initialize some vars */
        fv->starttime = fv->endtime = t;
        fv->startx = fv->endx = fv->minx = fv->maxx = x;
        fv->starty = fv->endy = fv->miny = fv->maxy = y;
        fv->endx = x; fv->endy = y;
        return;
    }

    dx1 = x - fv->endx; dy1 = y - fv->endy;
    magsq1 = dx1 * dx1 + dy1 * dy1;

    if(magsq1 <= dist_sq_threshold) {
        fv->npoints--;
        return;       /* ignore a point close to the last point */
    }

    if(x < fv->minx) fv->minx = x;
    if(x > fv->maxx) fv->maxx = x;
    if(y < fv->miny) fv->miny = y;
    if(y > fv->maxy) fv->maxy = y;

#ifdef PF_MAXV
    lasttime = fv->endtime;
#endif
    fv->endtime = t;

    d = sqrt(magsq1);
    fv->path_r += d;       /* update path length feature */

    /* calculate initial theta when the third point is seen */
    if(fv->npoints == 3) {
        double magsq, dx, dy, recip;
        dx = x - fv->startx; dy = y - fv->starty;
        magsq = dx * dx + dy * dy;
        if(magsq > dist_sq_threshold)  {
            /* find angle w.r.t. positive x axis e.g. (1,0) */
```

```
            recip = 1 / sqrt(magsq);
            fv->initial_cos = dx * recip;
            fv->initial_sin = dy * recip;
        }
    }

    if(fv->npoints >= 3) {   /* update angle-based features */
        th = absth = atan2(dx1 * fv->dy2 - fv->dx2 * dy1,
                           dx1 * fv->dx2 + dy1 * fv->dy2);
        if(absth < 0) absth = -absth;
        fv->path_th += th;
        fv->abs_th += absth;
        fv->sharpness += th*th;

#ifdef PF_MAXV   /* compute max velocity */
        if(fv->endtime > lasttime &&
           (v = d / (fv->endtime - lasttime)) > fv->maxv)
            fv->maxv = v;
#endif
    }

    /* prepare for next iteration */
    fv->endx = x; fv->endy = y;
    fv->dx2 = dx1; fv->dy2 = dy1;
    fv->magsq2 = magsq1;

    return;
}

/* calculate and return a feature vector */
Vector
FvCalc(fv)
register FV fv;
{
    double bblen, selen, factor;

    if(fv->npoints <= 1)
        return fv->y;          /* a feature vector of all zeros */

    fv->y[PF_INIT_COS] = fv->initial_cos;
    fv->y[PF_INIT_SIN] = fv->initial_sin;
```

```
    /* compute the length of the bounding box diagonal */
    bblen = hypot(fv->maxx - fv->minx, fv->maxy - fv->miny);

    fv->y[PF_BB_LEN] = bblen;

    /* the bounding box angle defaults to 0 for small gestures */
    if(bblen * bblen > dist_sq_threshold)
        fv->y[PF_BB_TH] = atan2(fv->maxy - fv->miny,
                                fv->maxx - fv->minx);

    /* compute the length and angle between the first and last points */
    selen = hypot(fv->endx - fv->startx,
                  fv->endy - fv->starty);
    fv->y[PF_SE_LEN] = selen;

    /* when the first and last points are very close, the angle features
       are muted so that they satisfy the stability criterion */
    factor = selen * selen / se_th_rolloff;
    if(factor > 1.0) factor = 1.0;
    factor = selen > EPS ? factor/selen :  0.0;
    fv->y[PF_SE_COS] = (fv->endx - fv->startx) * factor;
    fv->y[PF_SE_SIN] = (fv->endy - fv->starty) * factor;

    /* the remaining features have already been computed */
    fv->y[PF_LEN] =  fv->path_r;
    fv->y[PF_TH] = fv->path_th;
    fv->y[PF_ATH] = fv->abs_th;
    fv->y[PF_SQTH] = fv->sharpness;

#ifdef PF_DUR
    fv->y[PF_DUR] = (fv->endtime - fv->starttime)*.01;
#endif

#ifdef PF_MAXV
    fv->y[PF_MAXV] = fv->maxv * 10000;
#endif

    return fv->y;
}
```

A.2 Deriving and Using the Linear Classifier

Type `sClassifier` points at an object that represents a classifier able to discriminate between a set of gesture classes. Each gesture class is represented by an `sClassDope` type. The functions `sRead` and `sWrite` read and write a classifier to a file. The function `sNewClassifer` creates a new (empty) classifier. A training example is added using `sAddExample`. There is no function to explicitly add a new class to a classifier. When an example of a new class is added, the new class is created automatically. To train the classifier based on the added examples, call `sDoneAdding`. Once trained, `sClassify` and `sClassifyAD` are used to classify a feature vector as one of the classes; `sClassifyAD` optionally computes the rejection information.

Here is an example fragment for creating a new classifier, entering new training examples, and writing the resulting classifier out to a file. Some of these functions are timed (and further described) in section 9.1.7.

```
#include <stdio.h>
#include <math.h>
#include "bitvector.h"
#include "matrix.h"
#include "sc.h"

#define NEXAMPLES   15

sClassifier
MakeAClassifier()
{
    sClassifier sc = sNewClassifier();
    Vector InputAGesture();
    char name[100];
    int i;

    for(;;) {
        printf("Enter class name, newline to exit: ");
        if(gets(name) == NULL || name[0] == '\0')
            break;
        for(i = 1; i <= NEXAMPLES; i++) {
            printf("Enter %s example %d\n", name, i);
            sAddExample(sc, name, InputAGesture());
        }
    }
    sDoneAdding(sc);
    sWrite(fopen("classifier.out", "w"), sc);
    return sc;
```

```
}
```

Once a classifier has been created it can be used to classifier gestures as follows:

```
TestAClassifier(sc)
sClassifier sc;
{
    Vector v;
    sClassDope scd;
    double punambig, distance;

    for(;;) {
        printf("Enter a gesture\n");
        v = InputAGesture();
        scd = sClassifyAD(sc, v, &punambig, &distance);
        printf("Gesture classified as %s ", scd->name);
        printf("Probability of unambiguous classification: %g\n",
                punambig);
        printf("Distance from class mean: %g\n", distance);
    }
}
```

What follows is the header file and code to implement the statistical classifier.

```
/*************************************************************************
sc.h - create single path classifiers from feature vectors of examples,
  as well as classifying example feature vectors.
*************************************************************************/

#define MAXSCLASSES 100 /* maximum number of classes */

typedef struct sclassifier *sClassifier;   /* classifier */
typedef int sClassIndex;                   /* per-class index */
typedef struct sclassdope  *sClassDope;    /* per-class information */

struct sclassdope { /* per gesture class information within a classifier */
    char        *name;          /* name of a class */
    sClassIndex number;         /* unique index (small integer) of a class */
    int         nexamples;      /* number of training examples */
    Vector      average;        /* average of training examples */
    Matrix      sumcov;         /* covariance matrix of examples */
};
```

```
struct sclassifier { /* a classifier */
    int      nfeatures;      /* number of features in feature vector */
    int      nclasses;       /* number of classes known by this classifier */
    sClassDope  *classdope;  /* array of pointers to per class data */

    Vector       cnst;       /* constant term of discrimination function */
    Vector       *w;         /* array of coefficient weights */
    Matrix       invavgcov;  /* inverse covariance matrix */
};

sClassifier sNewClassifier();       /* */
sClassifier sRead();                /* FILE *f */
void        sWrite();               /* FILE *f; sClassifier sc; */
void        sFreeClassifier();      /* sc */
void        sAddExample();          /* sc, char *classname; Vector y */
void        sDoneAdding();          /* sc */
sClassDope  sClassify();            /* sc, y */
sClassDope  sClassifyAD();          /* sc, y, double *ap; double *dp */
sClassDope  sClassNameLookup();     /* sc, classname */
double      MahalanobisDistance();  /* Vector v, u; Matrix sigma */

/*****************************************************************
sc.c - creates classifiers from feature vectors of examples, as well as
  classifying example feature vectors.
*****************************************************************/


#include <stdio.h>
#include <math.h>
#include "bitvector.h"
#include "matrix.h"
#include "sc.h"

#define EPS (1.0e-6)      /* for singular matrix check */

/* allocate memory associated with a new classifier */

sClassifier
sNewClassifier()
{
    register sClassifier sc =
```

```
        (sClassifier) mallocOrDie(sizeof(struct sclassifier));
    sc->nfeatures = -1;
    sc->nclasses = 0;
    sc->classdope = (sClassDope *)
        mallocOrDie(MAXSCLASSES * sizeof(sClassDope));
    sc->w = NULL;
    return sc;
}

/* free memory associated with a new classifier */
void
sFreeClassifier(sc)
register sClassifier sc;
{
    register int i;
    register sClassDope scd;

    for(i = 0; i < sc->nclasses; i++) {
        scd = sc->classdope[i];
        if(scd->name) free(scd->name);
        free(scd);
        if(sc->w && sc->w[i]) FreeVector(sc->w[i]);
        if(scd->sumcov) FreeMatrix(scd->sumcov);
        if(scd->average) FreeVector(scd->average);
    }
    free(sc->classdope);
    if(sc->w) free(sc->w);
    if(sc->cnst) FreeVector(sc->cnst);
    if(sc->invavgcov) FreeMatrix(sc->invavgcov);
    free(sc);
}

/* given a string name of a class, return its per-class information */
sClassDope
sClassNameLookup(sc, classname)
register sClassifier sc;
register char *classname;
{
    register int i;
    register sClassDope scd;
    static sClassifier lastsc;
    static sClassDope lastscd;
```

```
    /* quick check for last class name */
    if(lastsc == sc && STREQ(lastscd->name, classname))
        return lastscd;

    /* linear search through all classes for name */
    for(i = 0; i < sc->nclasses; i++) {
        scd = sc->classdope[i];
        if(STREQ(scd->name, classname))
            return lastsc = sc, lastscd = scd;
    }
    return NULL;
}

/* add a new gesture class to a classifier */
static sClassDope
sAddClass(sc, classname)
register sClassifier sc;
char *classname;
{
    register sClassDope scd;

    sc->classdope[sc->nclasses] = scd = (sClassDope)
                mallocOrDie(sizeof(struct sclassdope));
    scd->name = scopy(classname);
    scd->number = sc->nclasses;
    scd->nexamples = 0;
    scd->sumcov = NULL;
    ++sc->nclasses;
    return scd;
}

/* add a new training example to a classifier */
void
sAddExample(sc, classname, y)
register sClassifier sc;
char *classname;
Vector y;
{
    register sClassDope scd;
    register int i, j;
    double nfv[50];
```

```
    double nm1on, recipn;

    scd = sClassNameLookup(sc, classname);
    if(scd == NULL)
        scd = sAddClass(sc, classname);

    if(sc->nfeatures == -1)
        sc->nfeatures = NROWS(y);

    if(scd->nexamples == 0) {
        scd->average = NewVector(sc->nfeatures);
        ZeroVector(scd->average);
        scd->sumcov = NewMatrix(sc->nfeatures,sc->nfeatures);
        ZeroMatrix(scd->sumcov);

    }

    if(sc->nfeatures != NROWS(y)) {
        PrintVector(y, "sAddExample: funny vector nrows!=%d",
            sc->nfeatures);
        return;
    }

    scd->nexamples++;
    nm1on = ((double) scd->nexamples-1)/scd->nexamples;
    recipn = 1.0/scd->nexamples;

    /* incrementally update covariance matrix */
        for(i = 0; i < sc->nfeatures; i++)
        nfv[i] = y[i] - scd->average[i];

    /* only upper triangular part computed */
        for(i = 0; i < sc->nfeatures; i++)
       for(j = i; j < sc->nfeatures; j++)
        scd->sumcov[i][j] += nm1on * nfv[i] * nfv[j];

    /* incrementally update mean vector */
    for(i = 0; i < sc->nfeatures; i++)
        scd->average[i] =
             nm1on * scd->average[i] + recipn * y[i];

}
```

```
/* run the training algorithm on the classifier */
void
sDoneAdding(sc)
register sClassifier sc;
{
    register int i, j;
    int c;
    int ne, denom;
    double oneoverdenom;
    register Matrix s;
    register Matrix avgcov;
    double det;
    register sClassDope scd;

    if(sc->nclasses == 0)
        error("sDoneAdding: No classes\n");

    /* Given covariance matrices for each class (* number of examples - 1)
       compute the average (common) covariance matrix */

    avgcov = NewMatrix(sc->nfeatures, sc->nfeatures);
    ZeroMatrix(avgcov);
    ne = 0;
    for(c = 0; c < sc->nclasses; c++) {
        scd = sc->classdope[c];
        ne += scd->nexamples;
        s = scd->sumcov;
        for(i = 0; i < sc->nfeatures; i++)
            for(j = i; j < sc->nfeatures; j++)
                avgcov[i][j] += s[i][j];
    }

    denom = ne - sc->nclasses;
    if(denom <= 0) {
        printf("no examples, denom=%d\n", denom);
        return;
    }

    oneoverdenom = 1.0 / denom;
    for(i = 0; i < sc->nfeatures; i++)
        for(j = i; j < sc->nfeatures; j++)
```

```
            avgcov[j][i] = avgcov[i][j] *= oneoverdenom;

    /* invert the avg covariance matrix */

    sc->invavgcov = NewMatrix(sc->nfeatures, sc->nfeatures);
    det = InvertMatrix(avgcov, sc->invavgcov);
    if(fabs(det) <= EPS)
        FixClassifier(sc, avgcov);

    /* now compute discrimination functions */
    sc->w = (Vector *)
              mallocOrDie(sc->nclasses * sizeof(Vector));
    sc->cnst = NewVector(sc->nclasses);
    for(c = 0; c < sc->nclasses; c++) {
        scd = sc->classdope[c];
        sc->w[c] = NewVector(sc->nfeatures);
        VectorTimesMatrix(scd->average, sc->invavgcov,
                          /* product = */ sc->w[c]);
        sc->cnst[c] = -0.5 *
                  InnerProduct(sc->w[c], scd->average);
        /* could add log(priorprob class c) to cnst[c] */
    }

    FreeMatrix(avgcov);
    return;
}

/* classify a feature vector */
sClassDope
sClassify(sc, fv) {
    return sClassifyAD(sc, fv, NULL, NULL);
}

/* classify a feature vector, possibly computing rejection metrics */
sClassDope
sClassifyAD(sc, fv, ap, dp)
sClassifier sc;
Vector fv;
double *ap;
double *dp;
{
    double disc[MAXSCLASSES];
```

```
    register int i, maxclass;
    double denom, exp();
    register sClassDope scd;
    double d;

    if(sc->w == NULL)
        error("sClassifyAD: %x no trained classifier", sc);

    for(i = 0; i < sc->nclasses; i++)
        disc[i] = InnerProduct(sc->w[i], fv) + sc->cnst[i];

    maxclass = 0;
    for(i = 1; i < sc->nclasses; i++)
        if(disc[i] > disc[maxclass])
            maxclass = i;

    scd = sc->classdope[maxclass];

    if(ap) {      /* calculate probability of non-ambiguity */
        for(denom = 0, i = 0; i < sc->nclasses; i++)
            /* quick check to avoid computing negligible term */
            if((d = disc[i] - disc[maxclass]) > -7.0)
                denom += exp(d);
        *ap = 1.0 / denom;
    }

    if(dp)    /* calculate distance to mean of chosen class */
        *dp = MahalanobisDistance(fv, scd->average,
                                  sc->invavgcov);

    return scd;
}

/* Compute the Mahalanobis distance between two vectors v and u */
double
MahalanobisDistance(v, u, sigma)
register Vector v, u;
register Matrix sigma;
{
    register i;
    static Vector space;
    double result;
```

```
    if(space == NULL || NROWS(space) != NROWS(v)) {
        if(space) FreeVector(space);
        space = NewVector(NROWS(v));
    }
    for(i = 0; i < NROWS(v); i++)
        space[i] = v[i] - u[i];
    result =  QuadraticForm(space, sigma);
    return result;
}

/* handle the case of a singular average covariance matrix by removing features */
FixClassifier(sc, avgcov)
register sClassifier sc;
Matrix avgcov;
{
    int i;
    double det;
    BitVector bv;
    Matrix m, r;

    /* just add the features one by one, discarding any that cause
       the matrix to be non-invertible */

    CLEAR_BIT_VECTOR(bv);
    for(i = 0; i < sc->nfeatures; i++) {
        BIT_SET(i, bv);
        m = SliceMatrix(avgcov, bv, bv);
        r = NewMatrix(NROWS(m), NCOLS(m));
        det = InvertMatrix(m, r);
        if(fabs(det) <= EPS)
            BIT_CLEAR(i, bv);
        FreeMatrix(m);
        FreeMatrix(r);
    }

    m = SliceMatrix(avgcov, bv, bv);
    r = NewMatrix(NROWS(m), NCOLS(m));
    det = InvertMatrix(m, r);
    if(fabs(det) <= EPS)
        error("Can't fix classifier!");
    DeSliceMatrix(r, 0.0, bv, bv, sc->invavgcov);
```

```
    FreeMatrix(m);
    FreeMatrix(r);

}

/* write a classifier to a file */
void
sWrite(outfile, sc)
FILE *outfile;
sClassifier sc;
{
    int i;
    register sClassDope scd;

    fprintf(outfile, "%d classes\n", sc->nclasses);
    for(i = 0; i < sc->nclasses; i++) {
        scd = sc->classdope[i];
        fprintf(outfile, "%s\n", scd->name);
    }
    for(i = 0; i < sc->nclasses; i++) {
        scd = sc->classdope[i];
        OutputVector(outfile, scd->average);
        OutputVector(outfile, sc->w[i]);

    }
    OutputVector(outfile, sc->cnst);
    OutputMatrix(outfile, sc->invavgcov);
}

/* read a classifier from a file */
sClassifier
sRead(infile)
FILE *infile;
{
    int i, n;
    register sClassifier sc;
    register sClassDope scd;
    char buf[100];


    printf("Reading classifier "), fflush(stdout);
```

```
    sc = sNewClassifier();
    fgets(buf, 100, infile);
    if(sscanf(buf, "%d", &n) != 1) error("sRead 1");
    printf("%d classes ", n), fflush(stdout);
    for(i = 0; i < n; i++) {
        fscanf(infile, "%s", buf);
        scd = sAddClass(sc, buf);
        scd->name = scopy(buf);
        printf("%s ", scd->name), fflush(stdout);
    }
    sc->w = allocate(sc->nclasses, Vector);
    for(i = 0; i < sc->nclasses; i++) {
        scd = sc->classdope[i];
        scd->average = InputVector(infile);
        sc->w[i] = InputVector(infile);
    }
    sc->cnst = InputVector(infile);
    sc->invavgcov = InputMatrix(infile);
    printf("\n");
    return sc;
}
```

/ compute pairwise distances between classes, and print the closest ones, as a clue as to which gesture classes are confusable */*

```
sDistances(sc, nclosest)
register sClassifier sc;
{
    register Matrix d = NewMatrix(sc->nclasses, sc->nclasses);
    register int i, j;
    double min, max = 0;
    int n, mi, mj;

    printf("------------\n");
    printf("%d closest pairs of classes\n", nclosest);
    for(i = 0; i < NROWS(d); i++) {
        for(j = i+1; j < NCOLS(d); j++) {
            d[i][j] = MahalanobisDistance(
                        sc->classdope[i]->average,
                        sc->classdope[j]->average,
                        sc->invavgcov);
            if(d[i][j] > max) max = d[i][j];
```

```
            }
        }

        for(n = 1; n <= nclosest; n++) {
            min = max;
            mi = mj = -1;
            for(i = 0; i < NROWS(d); i++) {
                for(j = i+1; j < NCOLS(d); j++) {
                    if(d[i][j] < min)
                        min = d[mi=i][mj=j];
                }
            }
            if(mi == -1)
                break;

            printf("%2d) %10.10s to %10.10s d=%g nstd=%g\n",
                n,
                sc->classdope[mi]->name,
                sc->classdope[mj]->name,
                d[mi][mj],
                sqrt(d[mi][mj]));

            d[mi][mj] = max+1;
        }
        printf("------------\n");
        FreeMatrix(d);
    }
```

A.3 Undefined functions

The above code uses some functions whose definitions are not included in this appendix. These fall into four classes: standard library functions (including the math library), utility functions, bitvector functions, and vector/matrix functions, The standard library calls will not be discussed.

The utility functions used are

`STREQ(s1, s2)` returns FALSE iff strings `s1` and `s2` are equal.

`scopy(s)` returns a copy of the string `s`.

`error(format, arg1...)` prints a message and causes the program to exit.

`mallocOrDie(nbytes)` calls malloc, dying with an error message if the memory cannot be obtained.

The bit vector operations are an efficient set of functions for accessing an array of bits.

`CLEAR_BIT_VECTOR(bv)` resets an entire bit vector `bv` to all zeros,

`BIT_SET(i, bv)` sets the i^{th} bit of `bv` to one, and

`BIT_CLEAR(i, bv)` sets the i^{th} bit of `bv` to zero.

The vector/matrix functions are declared in matrix.h. Objects of type `Vector` and `Matrix` may be accessed like one and two dimensional arrays, respectively, but also contain additional information as to the size and dimensionality of the object (accessible via macros `NROWS`, `NCOLS`, and `NDIM`. It should be obvious from the names and the use of most of the functions (`NewVector`, `NewMatrix`, `FreeVector`, `FreeMatrix`, `ZeroVector`, `ZeroMatrix`, `PrintVector`, `PrintMatrix`, `InvertMatrix`, `InputVector`, `InputMatrix`, `OutputVector`, `OutputMatrix`, `VectorTimesMatrix`, and `InnerProduct`) what they do. As for the remaining functions,

`double QuadraticForm(Vector V, Matrix M)` computes the quantity $V'MV$, where the prime denotes the transpose operation.

`Matrix SliceMatrix(Matrix m, BitVector rowmask, BitVector colmask)` creates a new matrix, consisting only of those rows and columns in `m` whose corresponding bits are set it `rowmask` and `colmask`, respectively.

`Matrix DeSliceMatrix(Matrix m, double fill, BitVector rowmask; BitVector colmask; Matrix result)` first sets every element in `result` to `fill`, and then, every element in `result` whose row number is on in `rowmask` and whose column number is on in `colmask`, is set from the corresponding element in the input matrix `m`, which is smaller than `r`. The result of `SliceMatrix(DeSliceMatrix(m, fill, rowmask, colmask, result), rowmask, colmask)` is a copy of `m`, given legal values for all parameters.

These auxiliary functions, as well as a C-based X11R5 version of GDP, are all available as part of the ftp distribution mentioned above.

Bibliography

[1] Apple. *Inside Macintosh*. Addison-Wesley, 1985.

[2] Apple. *Macintosh System Software User's Guide, Version 6.0*. Apple Computer, 1988.

[3] H. Arakawa, K. Okada, and J. Masuda. On-line recognition of handwritten characters: Alphanumerics, Hiragana, Katakana, Kanji. In *Proceedings of the 4th International Joint Conference on Pattern Recognition*, pages 810–812, 1978.

[4] R. Baecker. Towards a characterization of graphical interaction. In Guedj, R. A., et. al., editor, *Methodology of Interaction*, pages 127–147. North Holland, 1980.

[5] R. Baecker and W. A. S. Buxton. *Readings in Human-Computer Interaction - A Multidisciplinary Approach*. Morgan Kaugmann Readings Series. Morgan Kaufmann, Los Altos, California, 1987.

[6] Ronald Baecker, Ian Small, and Richard Mander. Bringing icons to life. In *CHI'91 Conference Proceedings*, pages 1–6. ACM, April 1991.

[7] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), 1975.

[8] M. Berthod and J. P. Maroy. Learning in syntactic recognition of symbols drawn on a graphic tablet. *Computer Graphics Image Processing*, 9:166–182, 1979.

[9] J. Block and R. Dannenberg. Polyphonic accompaniment in real time. In *International Computer Music Conference*, Cambridge, Mass., 1985. Computer Music Association.

[10] R. Boie. Personnel communication. 1987.

[11] R. Boie, M. Mathews, and A. Schloss. The Radio Drum as a synthesizer controller. In *1989 International Computer Music Proceedings*, pages 42–45. Computer Music Association, November 1989.

[12] R. A. Bolt. *The Human Interface: where people and computers meet*. Lifetime Learning Publications, 1984.

[13] Radmilo M. Bozinovic. *Recognition of Off-line Cursive Handwriting: A Case of Multi-level Machine Perception*. PhD thesis, State University of New York at Buffalo, March 1985.

[14] W. A. S. Buxton. Chunking and phrasing and the design of human-computer dialogues. In *Information Processing 86*, North Holland, 1986. Elsevier Science Publishers B.V.

[15] W. A. S. Buxton. There's more to interaction than meets the eye: Some issues in manual input. In D. A. Norman and S. W. Draper, editors, *User Centered Systems Design: New Perspectives on Human-Computer Interaction*, pages 319–337. Lawrence Erlbaum Associates, Hillsdale, N.J., 1986.

[16] W. A. S. Buxton. Smoke and mirrors. *Byte*, 15(7):205–210, July 1990.

[17] W. A. S. Buxton. A three-state model of graphical input. *Proceedings of Interact 90*, August 1990.

[18] W. A. S. Buxton, R. Hill, and P. Rowley. Issues and techniques in touch-sensitive tablet input. *Computer Graphics*, 19(3):215–224, 1985.

[19] W. A. S. Buxton and B. Myers. A study in two-handed input. In *Proceedings of CHI '86*, pages 321–326. ACM, 1986.

[20] W. A. S. Buxton, W. Reeves, R. Baecker, and L. Mezei. The user of hierarchy and instance in a data structure for computer music. In Curtis Roads and John Strawn, editors, *Foundations of Computer Music*, chapter 24, pages 443–466. MIT Press, Cambridge, Massachusetts, 1985.

[21] W. A. S. Buxton, R. Sniderman, W. Reeves, S. Patel, and R. Baecker. The evolution of the SSSP score-editing tools. In Curtis Roads and John Strawn, editors, *Foundations of Computer Music*, chapter 22, pages 387–392. MIT Press, Cambridge, Massachusetts, 1985.

[22] S. K. Card, Moran, T. P., and A. Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23(7):601–613, 1980.

[23] L. Cardelli and R. Pike. Squeak: A language for communicating with mice. *SIGGRAPH '85 Proceedings*, 19(3), April 1985.

[24] R. M. Carr. The point of the pen. *BYTE*, 16(2):211–221, February 1991.

[25] Michael L. Coleman. Text editing on a graphic display device using hand-drawn proofreader's symbols. In M. Faiman and J. Nievergelt, editors, *Pertinent Concepts in Computer Graphics, Proceedings of the Second University of Illinois Conference on Computer Graphics*, pages 283–290. University of Illinois Press, Urbana, Chicago, London, 1969.

[26] P. W. Cooper. Hyperplanes, hyperspheres, and hyperquadrics as decision boundaries. In J. T. Tou and R. H. Wilcox, editors, *Computer and Information Sciences*, pages 111–138. Spartan, Washington, D.C., 1964.

[27] Brad J. Cox. Message/object programming: An evolutionary change in programming technology. *IEEE Software*, 1(1):50–61, January 1984.

[28] Brad J. Cox. *Object Oriented Programming: An Evolutionary Approach*. Addison-Wesley, 1986.

[29] R. B. Dannenberg. A structure for representing, displaying, and editing music. In *Proceedings of the 1986 International Computer Music Conference*, pages 153–160, San Francisco, 1986. Computer Music Association.

[30] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley Interscience, 1973.

[31] The Economist. Digital quill. *The Economist*, 316(7672):88, September 15 1990.

[32] H. Eglowstein. Reach out and touch your data. *Byte*, 15(7):283–290, July 1990.

[33] W. English, D. Engelbart, and M. L. Berman. Display-selection techniques for text manipulation. *IEEE Transactions on Human Factors in Electronics*, HFE-8(1):21–31, 1967.

[34] S. S. Fels and Geoffrey E. Hinton. Building adaptive interfaces with neural networks: The glove-talk pilot study. Technical Report CRG-TR-90-1, University of Toronto, Toronto, Canada, 1990.

[35] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[36] Flecchia and Nergeron. Specifying complex dialogs in ALGAE. *SIGCHI+GI 87 Proceedings*, April 1987.

[37] K. S. Fu. *Syntactic Recognition in Character Recognition*, volume 112 of *Mathematics in Science and Engineering*. Academic Press, 1974.

[38] K. S. Fu. Hybrid aproaches to pattern recognition. In K. S. Fu J. Kittler and L. F. Pau, editors, *Pattern Recognition Theory and Applications*, NATO Advanced Study Institute, pages 139–155. D. Reidel, 1981.

[39] K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice Hall, 1981.

[40] K. S. Fu and T. S. Yu. *Statistical Pattern Classification using Contextual Information*. Pattern Recognition and Image Processing Series. Research Studies Press, New York, 1980.

[41] J. Gettys, R. Newman, and R. W. Schiefler. *Xlib - C Language Interface X11R2*. Massachusetts Institute of Technology, 1988.

[42] Dario Giuse. DP command set. Technical Report CMU-RI-TR-82-11, Carnegie Mellon University Robotics Institute, October 1982.

[43] R. Glitman. Startup readies 4-pound stylus pc. *PC Week*, 7(34):17–18, August 27 1990.

[44] Adele Goldberg and David Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley series in Computer Science. Addison-Wesley, 1983.

[45] D. Goodman. *The complete HyperCard handbook*. Bantam Books, 1988.

[46] G. H. Granlund. Fourier preprocessing for hand print character recognition. *IEEE Transactions on Computers*, 21:195–201, February 1972.

[47] I. Guyon, P. Albrecht, Y. Le Cun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for a touch terminal. *Pattern Recognition*, 24(2):105–119, 1991.

[48] D. J. Hand. *Kernel Discriminant Analysis*. Pattern Recognition and Image Processing Research Studies Series. Research Studies Press (A Division of John Wiley and Sons, Ltd.), New York, 1982.

[49] A. G. Hauptmann. Speech and gestures for graphic image manipulation. In *CHI '89 Proceedings*, pages 241–245. ACM, May 1989.

[50] Frank Hayes. True notebook computing arrives. *Byte*, pages 94–95, December 1989.

[51] P. J. Hayes, P. A. Szekely, and R. A. Lerner. Design alternatives for user interface management systems based on experience with COUSIN. In *CHI '85 Proceedings*, pages 169–175. ACM, April 1985.

[52] T. R. Henry, S. E. Hudson, and G. L. Newell. Integrating gesture and snapping into a user interface toolkit. In *UIST '90*, pages 112–122. ACM, 1990.

[53] C. A. Higgins and R. Whitrow. On-line cursive script recognition. In B. Shackel, editor, *Human-Computer Interaction - Interact '84, IFIP*, pages 139–143, North-Holland, 1985. Elsevier Science Publishers B.V.

[54] R. Hill. Supporting concurrency, communication, and synchronization in human-computer interaction. *ACM Transactions on Graphics*, 5(3):179–210, July 1986.

[55] J. Hollan, E. Rich, W. Hill, D. Wroblewski, W. Wilner, K. Wittenberg, J. Grudin, and Members of the Human Interface Laboratory. An introduction to HITS: Human interface tool suite. Technical Report ACA-HI-406-88, Microelectronics and Computer Technology Corporation, Austin, Texas, 1988.

[56] Bruce L. Horn. An introduction to object oriented programming, inheritance and method combination. Technical Report CMU-CS-87-127, Carnegie Mellon University Computer Science Department, 1988.

[57] A. B. S. Hussain, G. T. Toussaint, and R. W. Donaldson. Results obtained using a simple character recognition procedure on Munson's handprinted data. *IEEE Transactions on Computers*, 21:201–205, February 1972.

[58] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. In D. A. Norman and S. W. Draper, editors, *User Centered System Design*, pages 118–123. Lawrence Erlbaum Associates, Hillsdale, N.J., 1986.

[59] Pencept Inc. Software control at the stroke of a pen. In *SIGGRAPH Video Review*, volume Issue 18: Edited Compilations from CHI '85. ACM, 1985.

[60] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Trans. Acoustics, Speech, Signal Processing*, ASSP-23(67), 1975.

[61] J. C. Jackson and Renate J. Roske-Hofstrand. Circling: A method of mouse-based selection without button presses. In *CHI '89 Proceedings*, pages 161–166. ACM, May 1989.

[62] Mike James. *Classification Algorithms*. Wiley-Interscience. John Wiley and Sons, Inc., New York, 1985.

[63] R. E. Johnson. Model/View/Controller. November 1987 (unpublished manuscript).

[64] Dan R. Olsen Jr. Syngraph: A graphical user interface generator. *Computer Graphics*, 17(3):43–50, July 1983.

[65] K. G. Morse Jr. In an upscale world. *Byte*, 14(8), August 1989.

[66] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice-Hall, New Jersey, 1978.

[67] Joonki Kim. Gesture recognition by feature analysis. Technical Report RC12472, IBM Research, December 1986.

[68] Nancy T. Knolle. Variations of model-view-controller. *Journal of Object-Oriented Programming*, 2:42–46, September/October 1989.

[69] D. Kolzay. Feature extraction in an optical character recognition machine. *IEEE Transactions on Computers*, 20:1063–1067, 1971.

[70] Glenn E. Krasner and Stephen T. Pope. A description of the Model-View-Controller user interface paradigm in the Smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, August 1988.

[71] M. W. Kreuger. *Artificial Reality*. Addison-Wesley, Reading, MA., 1983.

[72] M. W. Kreuger, T. Gionfriddo, and K. Hinrichsen. Videoplace: An artificial reality. In *Proceedings of CHI'85*, pages 35–40. ACM, 1985.

[73] E. Kreyszig. *Advanced Engineering Mathematics*. Wiley, 1979. Fourth Edition.

[74] W. J. Krzanowski. *Principles of Multivariate Analysis: A User's Perspective*. Oxford Statistical Science Series. Clarendon Press, Oxford, 1988.

[75] G. Kurtenbach and W. A. S. Buxton. GEdit: A test bed for editing by contiguous gestures. *SIGCHI Bulletin*, pages 22–26, 1991.

[76] Martin Lamb and Veronica Buckley. New techniques for gesture-based dialog. In B. Shackel, editor, *Human-Computer Interaction - Interact '84, IFIP*, pages 135–138, North-Holland, 1985. Elsevier Science Publishers B.V.

[77] C. G. Leedham, A. C. Downton, C. P. Brooks, and A. F. Newell. On-line acquisition of pitman's handwritten shorthand as a means of rapid data entry. In B. Shackel, editor, *Human-Computer Interaction - Interact '84, IFIP*, pages 145–150, North-Holland, 1985. Elsevier Science Publishers B.V.

[78] Barbara Staudt Lerner. *Automated Customization of User Interfaces*. PhD thesis, Carnegie Mellon University, 1989.

[79] M. A. Linton, J. M. Vlissides, and P. R. Calder. Composing user interfaces with InterViews. *IEEE Computer*, 22(2):8–22, February 1989.

[80] James S. Lipscomb. A trainable gesture recognizer. *Pattern Recognition*, 1991. Also available as IBM Tech Report RC 16448 (#73078).

[81] D. J. Lyons. Go Corp. gains ground in pen-software race. *PC Week*, 7(29):135, July 23 1990.

[82] Gale Martin, James Pittman, Kent Wittenburg, Richard Cohen, and Tome Parish. Sign here, please. *Byte*, 15(7):243–251, July 1990.

[83] J. T. Maxwell. Mockingbird: An interactive composer's aid. Master's thesis, MIT, 1981.

[84] P. McAvinney. The Sensor Frame–a gesture-based device for the manipulation of graphic objects. Available from Sensor Frame, Inc., Pittsburgh, Pa., December 1986.

[85] P. McAvinney. Telltale gestures. *Byte*, 15(7):237–240, July 1990.

[86] Margaret R. Minsky. Manipulating simulated objects with real-world gestures using a force and position sensitive screen. *Computer Graphics*, 18(3):195–203, July 1984.

[87] P. Morrel-Samuels. Clarifying the distinction between lexical and gestural commands. *International Journal of Man-Machine Studies*, 32:581–590, 1990.

[88] G. Muller and R. Giulietti. High quadlity music notation: Interactive editing and input by piano keyboard. In *Proceedings of the 1987 International Computer Music Conference*, pages 333–340, San Francisco, 1987. Computer Music Association.

[89] Hiroshi Murase and Toru Wakahara. Online hand-sketched figure recognition. *Pattern Recognition*, 19(2):147–160, 1988.

[90] B. Myers and W. A. S. Buxton. Creating highly-interactive and graphical user interfaces by demonstration. *Computer Graphics*, 20(3):249–258, 1986.

[91] B. A. Myers. A new model for handling input. *ACM Transactions on Information Systems*, 1990.

[92] B. A. Myers, D. Giuse, R. B. Dannenberg, B. Vander Zanden, D. Kosbie, E. Pervin, Andrew Mickish, and Phillippe Marchal. Garnet: comprehensive support for graphical, highly-interactive user interfaces. *IEEE Computer*, 23(11):71–85, Nov 1990.

[93] B. A. Myers, B. Vander Zanden, and R. B. Dannenberg. Creating graphical interactive application objects by demonstration. In *UIST '89: Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 95–104. ACM, November 1989.

[94] Brad A. Myers. A taxonomy of user interfaces for window managers. *IEEE Computer Graphics and Applications*, 8(5):65–84, 1988.

[95] Brad A. Myers. Encapsulating interactive behaviors. In *Human Factors in Computing Systems*, pages 319–324, Austin, TX, April 1989. Proceedings SIGCHI'89.

[96] Brad A. Myers. User interface tools: Introduction and survey. *IEEE Software*, 6(1):15–23, January 1989.

[97] Brad A. Myers. Demonstration interfaces: A step beyond direct manipulation. Technical Report CMU-CS-90-162, Carnegie Mellon School of Computer Science, Pittsburgh, PA, August 1990.

[98] Brad A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, 1(1):97–123, March 1990.

[99] L. Nakatani. Personal communication, Bell Laboratories, Murray Hill, N.J. January 1987.

[100] T. Neuendorffer. *Adew Reference Manual*. Information Technology Center, Pittsburgh, PA, 1989.

[101] W. M. Newman and R. F. Sproull. *Principles of Interactive Computer Graphics*. McGraw-Hill, 1979.

[102] NeXT. *The NeXT System Reference Manual*. NeXT, Inc., 1989.

[103] L. Norton-Wayne. A coding approach to pattern recognition. In J. Kittler, K. S. Fu, and L. F. Pau, editors, *Pattern Recognition Theory and Applications*, NATO Advanced Study, pages 93–102. D. Reidel, 1981.

[104] K. K. Obermeier and J. J. Barron. Time to get fired up. *Byte*, 14(8), August 1989.

[105] A. J. Palay, W. J. Hansen, M. L. Kazar, M. Sherman, M. G. Wadlow, T. P. Neuendorffer, Z. Stern, M. Bader, and T. Peters. The Andrew toolkit: An overview. In *Proceedings of the USENIX Technical Conference*, pages 11–23, Dallas, February 1988.

[106] PC Computing. Ten other contendors in the featherweight division. *PC Computing*, 2(12):89–90, December 1989.

[107] J. A. Pickering. Touch-sensitive screens: the tecnologies and their application. *International Journal of Man-Machine Studies*, 25:249–269, 1986.

[108] R. Probst. Blueprints for building user interfaces: Open Look toolkits. Technical report, Sun Technology, August 1988.

[109] James R. Rhyne and Catherine G. Wolf. Gestural interfaces for information processing applications. Technical Report RC12179, IBM T.J. Watson Research Center, IBM Corporation, P.O. Box 218, Yorktown Heights, NY 10598, September 1986.

[110] J. Rosenberg, R. Hill, J. Miller, A. Schulert, and D. Shewmake. UIMSs: Threat or menace? In *CHI '88*, pages 197–212. ACM, 1988.

[111] D. Rubine and P. McAvinney. The Videoharp. In *1988 International Computer Music Proceedings*. Computer Music Association, September 1988.

[112] D. Rubine and P. McAvinney. Programmable finger-tracking instrument controllers. *Computer Music Journal*, 14(1):26–41, 1990.

[113] R. W. Scheifler and J. Gettys. The X window system. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

[114] K. J. Schmucker. MacApp: An application framework. *Byte*, 11(8):189–193, August 1986.

[115] Kurt J. Schmucker. *Object-Oriented Programming for the Macintosh*. Hayden Book Company, 1986.

[116] A. C. Shaw. Parsing of graph-representable pictures. *JACM*, 17(3):453, 1970.

[117] B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, pages 57–62, August 1983.

[118] John L. Sibert, William D. Hurley, and Teresa W. Bleser. An object-oriented user interface management system. In *SIGGRAPH '86*, pages 259–268. ACM, August 1986.

[119] Jack Sklanksy and Gustav N. Wassel. *Pattern Classifiers and Trainable Machines*. Springer-Verlag, New York, 1981.

[120] W. W. Stallings. Recognition of printed chinese characters by automatic pattern analysis. *Computer Graphics and Image Processing*, 1:47–65, 1972.

[121] Mark Stefik and Daniel G. Bobrow. Object-oriented programming: Themes and variations. *AI Magazine*, 6(4):40–62, Winter 1986.

[122] Jess Stein, editor. *The Random House Dictionary of the English Language*. Random House, Cambridge, Mass., 1969.

[123] Martin L. A. Sternberg. *American Sign Language: A Comprehensive Dictionary*. Harper and Row, New York, 1981.

[124] M. D. Stone. Touch-screens for intuitive input. *PC Magazine*, pages 183–192, August 1987.

[125] C. Y. Suen, M. Berthod, and S. Mori. Automatic recognition of handprinted characters: The state of the art. *Proceedings of the IEEE*, 68(4):469–487, April 1980.

[126] Sun. *SunWindows Programmers' Guide*. Sun Miscrosystems, Inc., Mountain View, Ca., 1984.

[127] Sun. *NeWS Preliminary Technical Overview*. Sun Miscrosystems, Inc., Mountain View, Ca., 1986.

[128] Shinichi Tamura and Shingo Kawasaki. Recognition of sign language motion images. *Pattern Recognition*, 21(4):343–353, 1988.

[129] C. C. Tappert, C. Y. Suen, and Toru Wakaha. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, August 1990.

[130] E. R. Tello. Between man and machine. *Byte*, 13(9):288–293, September 1988.

[131] A. Tevanian. MACH: A basis for future UNIX development. Technical Report CMU-CS-87-139, Carnegie Mellon University Computer Science Dept., Pittsburgh, PA, 1987.

[132] D. S. Touretzky and D. A. Pomerleau. What's hidden in the hidden layers? *Byte*, 14(8), August 1989.

[133] V. M. Velichko and N. G. Zagoruyko. Automatic recognition of 200 words. *Int. J. Man-Machine Studies*, 2(2):223, 1970.

[134] A. Waibel and J. Hampshire. Building blocks for speech. *Byte*, 14(8):235–242, August 1989.

[135] A. I. Wasserman. Extending state transition diagrams for the specification of human-computer interaction. *IEEE Transactions on Software Engineering*, SE-11(8):699–713, August 1985.

[136] D. Weimer and S. K. Ganapathy. A synthetic visual environment with hand gesturing and voice input. In *CHI '89 Proceedings*, pages 235–240. ACM, 1989.

[137] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, August 1962.

[138] A. P. Witkin. Scale-space filtering. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1019–1022, 1983.

[139] C. G. Wolf. A comparative study of gestural and keyboard interfaces. *Proceedings of the Humans Factors Society*, 32nd Annual Meeting:273–277, 1988.

[140] C. G. Wolf and J. R. Rhyne. A taxonomic approach to understanding direct manipulation. *Proceedings of the Human Factors Society*, 31st Annual Meeting:576–580, 1987.

[141] Catherine G. Wolf. Can people use gesture commands? Technical Report RC11867, IBM Research, April 1986.

[142] Xerox Corporation. JUNO. In *SIGGRAPH Video Review Issue 19: CHI '85 Compilation*. ACM, 1985.

Electronic Acknowledgement Receipt

EFS ID:	7082296
Application Number:	11677958
International Application Number:	
Confirmation Number:	1844
Title of Invention:	Ellipse Fitting for Multi-Touch Surfaces
First Named Inventor/Applicant Name:	Wayne Westerman
Customer Number:	69753
Filer:	Gregory Scott Weaver/Lisa Bronk
Filer Authorized By:	Gregory Scott Weaver
Attorney Docket Number:	10684-25086.04
Receipt Date:	24-FEB-2010
Filing Date:	22-FEB-2007
Time Stamp:	18:55:07
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	NPL Documents	37_RUBINE_1991_Dissertation.pdf	1709273 dc46cd8e7f6afe4d307d1d2d51690846ed06564c	no	285

Warnings:	
Information:	

Total Files Size (in bytes):	1709273

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (07-06)
Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number	Filing Date	☐ To be Mailed
11/677,958	02/22/2007	

APPLICATION AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =		OR	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =			X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT	02/24/2010	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	* 35	Minus	** 63	= 0	X $ =		OR	X $52=	0
	Independent (37 CFR 1.16(h))	* 3	Minus	***3	= 0	X $ =		OR	X $220=	0
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	0

AMENDMENT		(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)		RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.

** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".

*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".

The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Legal Instrument Examiner:
/DIANE JOHNSON/

Electronic Acknowledgement Receipt

EFS ID:	7194100
Application Number:	11677958
International Application Number:	
Confirmation Number:	1844
Title of Invention:	Ellipse Fitting for Multi-Touch Surfaces
First Named Inventor/Applicant Name:	Wayne Westerman
Customer Number:	69753
Filer:	Glen Masashi Kubota/Vivian Gutierrez
Filer Authorized By:	Glen Masashi Kubota
Attorney Docket Number:	10684-25086.04
Receipt Date:	11-MAR-2010
Filing Date:	22-FEB-2007
Time Stamp:	19:02:30
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$180
RAM confirmation Number	7702
Deposit Account	031952
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		106842508604SIDS-SB08.pdf	153685 31b22de4e1e654c86d0f8a0f2cc949f9e95020ba	yes	4
	Multipart Description/PDF files in .zip description				
	Document Description		**Start**	**End**	
	Transmittal Letter		1	3	
	Information Disclosure Statement (IDS) Filed (SB/08)		4	4	
Warnings:					
Information:					
2	Foreign Reference	7_JP2000163031_English.pdf	497635 1d7ab3a013e5e2c46076c40076bfcea2dbbd89df	no	63
Warnings:					
Information:					
3	NPL Documents	8_EP_SR_2_17_10.pdf	279352 08784e55f429404781f9037574c6f1997dea802e	no	6
Warnings:					
Information:					
4	NPL Documents	9_EP_SR_2_24_10.pdf	411224 fe1e4039187a1300b62205aa73c7913aa5718b02	no	9
Warnings:					
Information:					
5	NPL Documents	10_SUN_1992_Star7PDA.pdf	766047 c7f6c5d61614d2f4ac2b4b3530bc8c6dc861bdfc	no	7
Warnings:					
Information:					
6	NPL Documents	11_TOMITA_1997.pdf	285570 46b42d2e7277e28de4dfadf090b9d84b280e3e95	no	5
Warnings:					
Information:					
7	Fee Worksheet (PTO-875)	fee-info.pdf	30007 f2432187d722f49c4c4afc4aa33db58f3d67edb9	no	2
Warnings:					
Information:					

Total Files Size (in bytes):	2423520

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

ALTERNATIVE TO PTO/SB/08A/B
(Based on PTO 08-08 version)

Substitute for form 1449/PTO				Complete if Known	
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				Application Number	11/677,958
				Filing Date	February 22, 2007
				First Named Inventor	Wayne WESTERMAN
				Art Unit	2629
(Use as many sheets as necessary)				Examiner Name	Koosha Sharifi-Tafreshi
Sheet	1	of	1	Attorney Docket Number	106842508604 **Client Ref. No. P3950USC13**

U.S. PATENT DOCUMENTS					
Examiner Initials*	Cite No.[1]	Document Number Number-Kind Code[2] *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	1.	US-5,886,697-A	03-23-1999	Naughton et al.	
	2.	US-5,995,106-A	11-30-1999	Naughton et al.	
	3.	US-6,154,209-A	11-28-2000	Naughton et al.	
	4.	US-6,160,551-A	12-12-2000	Naughton et al.	
	5.	US-6,344,861-B1	02-05-2002	Naughton et al.	
	6.	US-7,240,289-B2	07-03-2007	Naughton et al.	

FOREIGN PATENT DOCUMENTS						
Examiner Initials*	Cite No.[1]	Foreign Patent Document Country Code[3]-Number[4]-Kind Code[5] *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages Or Relevant Figures Appear	T[6]
	7.	JP-2000-163031-A	06-16-2000	Seiko Epson Corp.		√

Examiner Signature		Date Considered	

*EXAMINER: Initial if information considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. [1] Applicant's unique citation designation number (optional). [2] See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3] Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [5] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST. 16 if possible. [6] Applicant is to place a check mark here if English language Translation is attached.

NON PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journ al, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T[2]
	8.	European Search Report mailed February 17, 2010, for EP Application No. 06016857.2, six pages.	
	9.	European Search Report mailed February 24, 2010, for EP Application No. 06016833.3, nine pages.	
	10.	SUN MICROSYSTEMS. (1992). "The Star7 PDA Prototype," located at <http://www.youtube.com/watch?v=Ahg8OBYixL0," last visited January 15, 2010, seven pages.	
	11.	TOMITA, A. Jr. et al. (November 9, 1997). "An Image Processing Method for Use in a GUI for the Visually Impaired," *Proceedings of the IECON '97: 23rd International Conference on Industrial Electronics, Control and Instrumentation*, New Orleans, LA, November 9 - 14, 1997, pp. 264-268.	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1]Applicant's unique citation designation number (optional). [2]Applicant is to place a check mark here if English language Translation is attached.

(19) Japanese Patent Office (JP)

(12) Publication of Unexamined Patent Application (A)

(11) Patent Application Disclosure

2000-163031

(43) Publication Date: June 16, 2000

(51) Int. Cl.[7]		Identification code	JPO File No.	FI		Subject Code (reference)
G 09 G	5/00	530		G 09 G	5/00	530A
		510				510G
G 06 F	3/00	656		G 06 F	3/00	656D
	15/02	310			15/02	310Z
	17/30			G 09 B	29/00	A

Request for Examination: Not yet Number of Claims: 7 FD Total pages: 30

Continued on last page

(21) Filing No.: H10-350728

(22) Filing Date: November 25, 1998

(71) Applicant 000002369
Seiko Epson Corporation
2-4-1 Nishishinjuku, Shinjuku-ku, Tokyo

(72) Inventor NOMURA, Yasuhiro
c/o Seiko Epson Corporation 3-5, 3-chome, Owa, Suwa-shi, Nagano ken

(72) Inventor CHIYO, Yoshika
c/o Seiko Epson Corporation 3-5, 3-chome, Owa, Suwa-shi, Nagano ken

(74) Agent 100090479
Patent Attorney INOUE, Makoto (and two others)

Continued on last page

(54) TITLE OF INVENTION

Mobile Information Device and Information Storage Media

(57) ABSTRACT

PROBLEM TO BE SOLVED
Providing an electronic book that achieves a favorable human interface with functions such as rotate, zoom-in, zoom-out, scroll, etc. of a map image that are easy to use and a mobile information device and information storage media used for these.

MEANS FOR SOLVING THE PROBLEM
An electronic book that includes a display area that can display map images. Based on the movement history of contacting the display area with a finger, execution instructions for at least one operation selected from rotate, zoom-in, zoom-out, or scroll and, at the same time, the quantity of operation can be input for a map image. Moving two fingers apart inputs a command to zoom in as well as the amount to zoom in on the map image. Furthermore, moving two fingers toward each other inputs a command to zoom out as well as the amount to zoom

out on the map image. In addition, a rotate command and amount of rotation can be input for the map image by rotating one finger with another finger as the axis.

Claims

Claim 1
A mobile information device with a display area that can display map images, comprising: finger movement detection means that detects the movement history of fingers contacting the display area displaying the map image, operation details determination means that judges at least one operation selected from rotate, zoom-in, zoom-out, or scroll of a map image to have been input based on said finger movement history, and map image generation means that generates a map image with at least one operation selected from rotate, zoom-in, zoom-out, or scroll performed for a map image displayed in the display area based on said judgment.

Claim 2
The mobile information device of claim 1, wherein said operation details determination means determines at least one operation amount selected from amount of rotation, amount of zoom-in, amount of zoom-out, or amount of scroll of a map image based on said finger movement history and said map image generation means generates a map image with at least one operation selected from rotate, zoom-in, zoom-out, and scroll performed for a map image displayed in the display area based on the quantity of operation determined.

Claim 3
The mobile information device of either claim 1 or 2, wherein if said finger movement detection means detects the moving apart of two fingers contacting the display area displaying the map image, said operation details determination means judges input of a zoom-in operation of the map image and said map image generation means generates a zoomed-in map image.

Claim 4
The mobile information device of any of claims 1 through 3, wherein if said finger movement detection means detects the movement of two fingers toward each other contacting the display area displaying the map image, said operation details determination means judges input of a zoom-out operation of the map image and said map image generation means generates a zoomed-out map image.

Claim 5
The mobile information device of any one of claims 1 through 4, wherein if said finger movement detection means detects one finger contacting the display area rotating with another finger contacting the display area displaying the map image as an axis, said operation details determination means judges input of a rotate

operation of the map image and said map image generation means generates a rotated map image.

Claim 6
The mobile information device of any one of claims 1 through 5, wherein the mobile information device is used as an electronic book.

Claim 7
Information storage media that can be used in at least one of a mobile information device or electronic book that can display a map image, comprising: information for detecting the movement history of a finger contacting a display area displaying a map image, information for judging input of at least one operation selected from rotate, zoom-in, zoom-out, or scroll based on said finger movement history, and information for generating a map image with at least one operation selected from rotate, zoom-in, zoom-out, or scroll performed for a map image displayed in the display area based on said judgment.

DETAILED DESCRIPTION OF THE INVENTION

0001
INDUSTRIAL FIELD OF THE INVENTION
The present invention is related to a mobile information device, an electronic book, and information storage media used for these.

0002
BACKGROUND TECHNOLOGY AND PROBLEMS THAT THIS INVENTION INTENDS TO RESOLVE
In recent years, information such as text, images, and audio have been recorded on electronic media such as CD-ROMs, FD, the like, and electronic books and mobile information devices that allow for a type of product similar to a book have been implemented.

0003
These types of electronic book and mobile information device can address many of the physical drawbacks for paper books. Furthermore, they can facilitate a non-linear reading style and the reader can enjoy a diversified reading experience from various perspectives.

0004
Furthermore, the speed and low cost of publishing using such devices enable easy publication by anyone, including the publication of specialized information where the number of readers is limited, thus achieving an ideal publication process.

0005

Here, when map information is stored in this type of electronic book or mobile information device, such a device may be carried while moving about or traveling, for convenient referral to the map image. In particular, unlike regular maps, electronic books and mobile information devices enable implementation of enhanced functions such as freely zooming in and zooming out as well as scrolling through parts that are needed. This is because electronic books and mobile information devices can implement dramatically enhanced functions for search and use of information, compared to regular paper books.

0006
However, even with the ability to achieve this enhanced function, if the operation is complex or difficult or is otherwise not user-friendly, it won't be used efficiently. In particular, the electronic book and mobile information devices are built small and are convenient for carrying around; therefore, an input method and an operation method suitable for these types of devices are desired.

0007
The present invention was invented in light of the problems described above and an objective thereof is to provide an electronic book and mobile information device that can display map images with a user-friendly interface with functions of rotate, zoom-in, zoom-out, and scroll and the like of the map image as well as information storage media used for these types of electronic books and mobile information devices.

0008
MEANS FOR SOLVING THE VARIOUS PROBLEMS
The present invention is a mobile information device with a display area that can display map images, comprising: finger movement detection means that detects the movement history of fingers contacting the display area displaying the map image, operation details determination means that judges at least one operation selected from rotate, zoom-in, zoom-out, or scroll of a map image to have been input based on said finger movement history, and map image generation means that generates a map image with at least one operation selected from rotate, zoom-in, zoom-out, or scroll performed for a map image displayed in the display area based on said judgment.

0009
Using the present invention, the user can input at least one operation selected from rotate, zoom-in, zoom-out, and scroll of a map image displayed in the display area through the movement history of his fingers.

0010
The movement history of fingers contacting the display area is a concept including a passage of time element and is distinguished from an operation of simply touching an input mark or the like with a finger that does not include a passage of time element.

0011
With the present invention, because various types of operation can be input using movement history, hardware such as buttons on the case or a keyboard does not need to be prepared, enabling provision of a more compact mobile information device. Furthermore, non-use of input marks and the like on the screen for operation promotes effective use of a small screen suitable for carrying around and makes a more visible screen display feasible.

0012
Furthermore, because simultaneous input of executable instructions and execution details with a single operation, such as [pushing] a button, etc., is difficult, multiple operations such as operations for commanding execution details and operations for performing execution instructions are required, and this is troublesome. However, with the present invention input is performed through movement of fingers where detection of the command range and movement range of the finger enables simultaneous indication of operation amount and operation range, providing a user-friendly interface.

0013
Furthermore, with the present invention, the aforementioned operation details determination means determines at least one operation amount selected from amount of rotation, amount of zoom-in, amount of zoom-out, and amount of scroll based on finger movement history and the aforementioned map image generation means generates a map image with at least one operation selected from rotate, zoom-in, zoom-out, or scroll performed for a map image displayed in the display area based on the operation amount determined.

0014
With the present invention, the user can simultaneously input not only execution instructions for at least one operation selected from rotate, zoom-in, zoom-out, and scroll of the map image displayed in the display area but also the quantity of operation by using the finger movement history.

0015
Execution instructions of the operation are instructions to execute one of the operations of rotate, zoom-in, zoom-out, or scroll or the like. Furthermore, quantity of operation means the amount of rotation for a rotate operation, amount of zoom-in for a zoom-in operation, amount of zoom-out for a zoom-out operation, amount of scroll for a scroll operation or the determined range to be displayed by zooming in, zooming out, rotating, or scrolling.

0016
With the present invention, the operation amount can be inputted for each type of operation based on the movement history of fingers; therefore, a user can input

the desired quantity of operation by adjusting finger motion according to his needs.

0017
Furthermore, because it is difficult to indicate an arbitrary quantity of operation and perform an execution instruction with a single operation using buttons or tags, multiple operations such as indicating the quantity of operation and commanding execution are required, making it complicated. However, with the present invention, input is performed through the movement of fingers where detection of command range and movement range of the fingers enables simultaneous indication of operation amount and operation range, providing a simple, user-friendly interface.

0018
Furthermore, with the present invention, if the aforementioned finger movement detection means detects an operation of two fingers contacting the display area displaying the map image moving apart, the aforementioned operation details determination means judges input of a zoom-in operation of the map image and the aforementioned map image generation means generates a zoomed-in map image.

0019
The action of moving apart of two fingers in contact with the display area displaying the map image is an action that would be thought of as zooming in on a map image and users can easily accept this action as a zoom-in action. In addition, one action can provide execution instructions indicating zoom-in details such as zoom-in range and amount of zoom-in based on position of action and amount of action.

0020
Therefore, with the present invention, a mobile information device that enables a user-friendly zoom-in operation can be provided to the user.

0021
Note that zooming-in of the map image can either be a constant rate of zooming for one action or zooming at a rate corresponding to the amount of distance the fingers are moved. When zooming in using a ratio corresponding to the distance the fingers are moved, the user can perform an execution instruction indicating the amount of zoom-in using one action, enabling provision of an interface that is user-friendly.

0022
Furthermore, it is preferable to include the location contacted by the fingers prior to zooming in, within the range of the map image displayed in the display area as the result of zooming in. This enables the user to perform an execution

instruction indicating the zoom-in range with one action, providing an interface that is user-friendly.

0023
Furthermore, the action of zooming in on a map image not only involves reducing the scale of the image but also can include changing from a map image with overall topography and location of cities as the main constituent of information before zooming in, to a detailed zoomed-in map of a city or a part of a city with information such as buildings and roads being the main constituent of the information.

0024
Furthermore, with the present invention, if the aforementioned finger movement detection means detects an operation of two fingers contacting the display area displaying the map image moving toward each other, the aforementioned operation details determination means judges input of a zoom-out operation of the map image and the aforementioned map image generation means generates a zoomed-out map image.

0025
The action of moving toward each other two fingers in contact with the display area displaying the map image is an action that would be thought of as zooming out from a map image and users can easily accept this action as a zoom-out action. In addition, one action can provide execution instructions indicating zoom-out details such as zoom-out range and amount of zoom-out based on the position of action and amount of action.

0026
Therefore, with the present invention, a mobile information device that enables a user-friendly zoom-out operation can be provided to the user.

0027
Note that zooming-out of the map image can either be a constant rate of zooming out for one action or zooming at a rate corresponding to the amount of distance the fingers moved. When zooming out using a ratio corresponding to the distance the fingers move, the user can perform an execution instruction indicating the amount of zoom-out using one action, enabling provision of an interface that is user-friendly.

0028
Furthermore, it is preferable to include the location contacted by the fingers prior to zooming out in the range of the map image displayed in the display area as the result of zooming out. This enables the user to perform an execution instruction indicating the zoom-out range with one action, providing an interface that is user-friendly.

0029
In addition, the action of zooming out from a map image not only involves increasing the scale of the image but also can include changing from a town-map–type map image where buildings and roads were the main constituent of information before zooming in [sic] to a map image where the main constituent of information is topography and location of cities.

0030
With the present invention, if the aforementioned finger movement detection means detects one finger contacting the display area rotating with another finger contacting the display area displaying the map image as an axis, the aforementioned operation details determination means judges input of a rotate operation of the map image and the aforementioned map image generation means generates a rotated map image.

0031
Rotation of one finger contacting the display area with another finger contacting the display area displaying a map image as an axis is similar to an action when using a compass. Therefore, this will be thought of as rotating of a map image and the user can easily accept this action as a rotate operation. In addition, one action can provide execution instructions indicating rotation details such as rotation range and amount of rotation based on position of action and amount of action.

0032
Therefore, with the present invention, a mobile information device that enables a user-friendly rotate operation can be provided to the user.

0033
Furthermore, if a user wants to align the direction he is facing with the direction on the screen, he must rotate the map itself when using an actual map. However, with the mobile information device of the present invention, the same effect can be achieved by rotating the map image displayed. Therefore, the user can view a map image rotated in his desired direction through a simple operation, thus enabling the provision of a mobile information device that is more user-friendly than an actual map.

0034
Note that rotation of the map image can either be a constant rate of rotation for one action or rotation at a rate corresponding to the amount of distance the fingers moved. When rotating using a ratio corresponding to the distance the fingers moved, the user can perform an execution instruction indicating the amount of rotation with one action, enabling provision of an interface that is user-friendly.

0035

Furthermore, it is preferable to include the location contacted by the fingers prior to rotating in the range of the map image displayed in the display area as the result of rotating. This enables the user to perform an execution instruction indicating the rotation range with one action, providing an interface that is user-friendly.

0036
Furthermore, the present invention is the aforementioned mobile information device used as an electronic book.

0037
In other words, [this is] an electronic book with a display area that can display map images, comprising: finger movement detection means that detects the movement history of fingers contacting the display area displaying the map image, operation details determination means that judges at least one operation selected from rotate, zoom-in, zoom-out, or scroll of a map image to have been input based on the aforementioned finger movement history, and map image generation means that generates a map image with at least one operation selected from rotate, zoom-in, zoom-out, or scroll performed for a map image displayed in the display area based on the aforementioned operation details determination means.

0038
Furthermore, with the aforementioned electronic book, the aforementioned operation details determination means determines at least one operation amount selected from amount of rotation, amount of zoom-in, amount of zoom-out, and amount of scroll based on the aforementioned finger movement history and the aforementioned map image generation means generates a map image with at least one operation selected from rotate, zoom-in, zoom-out, or scroll performed for a map image displayed in the display area based on the operation amount determined.

0039
Furthermore, with the aforementioned electronic book, if the aforementioned finger movement detection means detects an operation of two fingers contacting the display area displaying the map image moving apart, the aforementioned operation details determination means judges input of a zoom-in operation of the map image and the aforementioned map image generation means generates a zoomed-in map image.

0040
Furthermore, with the aforementioned electronic book, if the aforementioned finger movement detection means detects an operation of two fingers contacting the display area displaying the map image moving toward each other, the aforementioned operation details determination means judges input of a zoom-

out operation of the map image and the aforementioned map image generation means generates a zoomed-out map image.

0041
Furthermore, with the aforementioned electronic book, if the aforementioned finger movement detection means detects one finger contacting the display area rotating with another finger contacting the display area displaying the map image as an axis, the aforementioned operation details determination means judges input of a rotate operation of the map image and the aforementioned map image generation means generates a rotated map image.

0042
Furthermore, the present invention is information storage media that can be used in at least one of a mobile information device or electronic book that can display a map image, comprising: information for detecting the movement history of a finger contacting a display area displaying a map image, information for judging input of at least one operation selected from rotate, zoom-in, zoom-out, or scroll of a map image based on said finger movement history, and information for generating a map image with at least one operation selected from rotate, zoom-in, zoom-out, or scroll performed for a map image displayed in the display area based on said judgment.

0043
Furthermore, it is preferable for the aforementioned information storage media to be configured to include information for determination of at least one quantity of operation selected from amount of rotation, amount of zoom-in, amount of zoom-out, or amount of scroll of a map image based on the aforementioned finger movement history and for generation of a map image with at least one operation selected from rotate, zoom-in, zoom-out, or scroll performed for the map image displayed in the display area based on the quantity of operation determined.

0044
Furthermore, it is preferable for the aforementioned information storage media to be configured to include information for judging input of a map image zoom-in operation in the case that the moving apart of two fingers in contact with the display area displaying the map image is detected and for generation of a zoomed-in map image.

0045
Furthermore, it is preferable for the aforementioned information storage media to be configured to include information for judging input of a map image zoom-out operation in the case that the moving toward each other of two fingers in contact with the display area displaying the map image is detected and for generation of a zoomed-out map image.

0046

Furthermore, it is preferable for the aforementioned information storage media to be configured to include information for judging input of a rotate operation of a map image in the case that an action of rotating one finger in contact with the display area with another finger contacting the display area displaying the map image as an axis is detected and for generation of a rotated map image.

0047
EMBODIMENTS
1. Characteristics of the present invention
A characteristic of the present invention is performing rotate, zoom-in, zoom-out, or scrolling of a map image through finger movement on a map image displayed on a mobile[1] information device or electronic book that can display a map image.

0048
Fig. 1 is a functional block diagram of a mobile information device or electronic book with characteristic functions of the present invention.

0049
The finger movement detector 10 is for detecting the movement history of fingers on the display area displaying a map image for input of operations by the user such as zoom-in, zoom-out, rotate, scroll and the like. The finger movement detector 10 is a transparent touch panel or the like overlaid on the display area 60. Detection data obtained by the finger movement detector 10 are input to the processor 20.

0050
The processor 20 performs processing for generating a map image based on the aforementioned detection data and prescribed programming and the like. This processor 20 function is implemented using hardware such as a CPU (CISC type, RISC type), DSP, custom (gate array etc.) IC, memory and the like.

0051
The information storage medium 70 is for storing a program and data. The function of the information storage medium 70 is implementation using hardware such as a CD-ROM, cassette, IC card, MO, FD, DVD, hard disk, and memory and the like. This processor 20 performs various types of processing based on a program and data from this information storage medium 70.

0052
The processor 20 is made up of an operation details determination part 30, a map operation processor 40, and an image generation part 50.

0053

[1] Translator's note: Original Japanese literally: "morphological," which is a homonym for "mobile." Likely a typographical error.

The operations details determination part 30 judges the operation details input by the user based on the finger movement history detected by the finger movement detector 10. Specifically, the operations details determination part 30 judges finger movement history detected by the finger movement detector 10 of two fingers moving apart as input of a map image zoom-in operation. Furthermore, for example, the operations details determination part 30 judges finger movement history detected by the finger movement detector 10 of two fingers moving toward each other as input of a map image zoom-out operation. Furthermore, the operations details determination part 30 judges finger movement history detected by the finger movement detector 10 of one finger rotating with another finger as an axis as input of a map image rotate operation. Furthermore, for example, the operations details determination part 30 judges finger movement history detected by the finger movement detector 10 of action of moving one finger as input of a map scroll operation.

0054
The map operation processor 40 performs processing to generate a map image with the operation judged by the operation details determination part 30 implemented and includes a zoom-in processor 42, a zoom-out processor 44, a rotation processor 46, and a scroll processor 48.

0055
The zoom-in processor 42 performs necessary processing for generating a zoomed-in map image corresponding to the distance that two fingers are moved apart. The zoom-out processor 44 performs necessary processing for generating a zoomed-out map image corresponding to the distance that two fingers are moved toward each other. The rotation processor 46 performs necessary processing for generating a rotated map image corresponding to the angle of rotation of one finger.

0056
The scroll processor 48 performs the necessary processing for generating a scrolled map image corresponding to the movement of one finger.

0057
The image generation part 50 generates a map image output to the display area based on processing performed by the map operation processor 40. Generation of a map image can be a method of maintaining image data at maximum resolution and generating a scaled image by pixel-skipping these image data, or a method of holding all the data as vector data and performing the necessary calculation to generate a map image.

0058
2. A favorable embodiment of the present invention
The present invention is described in detail below using an electronic book as a favorable embodiment of the present invention.

0059
(1) External appearance drawing
Fig. 2 (A), (B), (C) shows an external appearance drawing of the electronic book of this embodiment. In Fig. 2 (A), 110 is an external appearance front view with the electronic book for this embodiment when it is closed and 120 is a side view. 130 shows the electronic book of this embodiment when it is open. For convenience of carrying, the electronic book of this embodiment is configured to be roughly the size of a paperback book when the electronic book is closed, as illustrated by 110.

0060
Fig. 2 (B) is a diagram showing a disk 140 on which software used in the electronic book is stored. As shown in Fig. 2 (C), the set-up of a disk 140 with various types of software stored on it in the electronic book body 150 enables the electronic book of this embodiment to provide information corresponding to details of various types of software.

0061
(2) Operation processing of a map image using finger movement
Next, an operation example of a map image using finger movement in the electronic book of this embodiment will be explained.

0062
Fig. 3 is a diagram showing a screen example displaying the case where travel information software is set. The center of the screen is primarily used as an information display area and the edges of the screen are primarily used for search tag and toolbar display areas. The details noted on one page of a two-page spread are displayed as image information in the information display area. Furthermore, search tags 210 to 230 and the like are displayed in the search tag area. Furthermore, images 240 and 242 that simulate the “thickness of a book” are displayed on the right and left edges of the display area.

0063
[1] Map zoom-in operation
A user touches the “Map” 220 search tag with a finger 250, and a full map 260 of “India” is displayed on the screen (see Fig. 4). In the case that the obtaining of information near “Bombay” is desired, as shown in Fig. 6 (B), the placement of a thumb and forefinger near “Bombay” on the screen with the thumb and forefinger close together and the action of moving the thumb and forefinger away from each other (map zoom-in gesture) are performed (see Fig. 5). When performed in this manner, a zoomed-in map image corresponding to the movement history of the thumb and forefinger is displayed. In other words, an increased level of separation of the thumb and forefinger leads to a smaller scale and a more detailed map.

0064
Here, the zoom-in of the map on the screen can be performed in real time by the action of moving the thumb and forefinger apart. Or, a zoomed-in map image can be displayed after the movement of fingers stops where the final amount of zoom-in is set based on the amount of zoom-in.

0065
In addition, as shown in Fig. 7, holding the thumb still and moving the forefinger away from the thumb can also be included in map zoom-in gestures.

0066
If a more detailed map image is desired after seeing the results of the zoomed-in map, repetition of the aforementioned zoom-in gesture enables obtaining a more detailed map image. Note, in this embodiment, if maps of cities are zoomed in to a certain extent, detailed maps of cities such as town maps are displayed.

0067
[2] Map scroll operation
As shown in Fig. 8, when a finger is placed on the screen and moved in the desired direction while pressing a finger on the screen (map scroll gesture), a map moved in the direction of movement of the finger and the same distance as the moved finger is displayed.

0068[3] Map zoom-out operation
When desiring a zoomed-out map for a case such as to see a map image with larger scale, or to display information for a wide range or the like, first place a thumb and forefinger apart on the map displayed on the screen as shown in Fig. 6 (A), then move the thumb and forefinger toward each other as shown in Fig. 9 (map zoom-out gesture). When performed in this manner, a zoomed-out map image corresponding to the movement history of the thumb and forefinger is displayed. In other words, an increased level of separation of the thumb and forefinger leads to a smaller scale and a more detailed map [sic].

0069
Here, the zooming-out of the map on the screen can be performed in real time by the action of moving the thumb and forefinger toward each other. Or, a zoomed-out map image can be displayed after the movement of fingers stops where the final amount of zoom-out is set based on the amount of zoom-out.

0070
In addition, unlike what is shown in Fig. 7, holding the thumb still and moving the forefinger toward the thumb can also be included in map zoom-out gestures.

0071
If a map image over a broader range is desired after seeing the results of the zoomed-out map, repetition of the aforementioned map zoom-out gesture

enables obtaining a map image for an even broader range. Note, in this embodiment, if maps of cities are zoomed out to a certain extent, the display will switch from detailed maps of cities such as town maps to a normal map display.

0072
[4] Map rotation operation
If change of orientation of a map without changing scale is desired, first place a thumb and forefinger apart on the map displayed on the screen as shown in Fig. 6 (A), then hold either the thumb or forefinger in one place and rotate the other finger with the other finger as an axis (map rotation gesture) (see Fig. 10). When performed in this manner, a map image with rotation corresponding to the movement history of the thumb and forefinger is displayed.

0073
As shown in Fig. 11, holding one of the fingers in place at the point O and rotating the other finger from point A to point B enables display of a map image in a rotated direction 370 equal to the rotated angle Θ from A to B.

0074
Here, the map can be rotated in real time on the screen with rotation of one of the fingers. Or, a zoomed-out [sic] map image can be displayed after the movement of fingers stops where the final amount of rotation is set based on the amount of rotation.

0075
Note that finger movement is not limited to movement of thumb and forefinger but can be performed using the thumb and middle finger or other combinations of digits.

0076
(3) Search processing
Next, a specific example of search processing based on finger action in an electronic book of this embodiment will be described.

0077[1] A normal search operation
For example, if a user desires hotel information in “Southern India,” a method such as that shown in Fig. 12 is available. Specifically, first select the “Southern India” tag to display the first page noted for “Southern India” information, then select the lodging tab as a method to display the first page providing information for “Southern India Lodging.” This method is useful when the user wants to obtain various information regarding “Southern India” by displaying a first page noting information for “Southern India” and by subsequently searching various information for “Southern India.”

0078
[2] Search operations using logical products

However, when the only information needed by the user is “Southern India Lodging,” displaying a first page noting information for “Southern India Lodging” directly in one operation is preferable. Here, as shown in Fig. 13 (A), if the user touches the “Southern India” tag with a finger on his left hand and touches the “Lodging” tag with a finger on his right hand at the same time, this embodiment is configured to display the first page noting information for “Southern India Lodging” directly with one operation.

0079
If information for “Southern India Lodging” is provided on several pages, as shown in Fig. 13 (B), information noted on the next and subsequent pages can be read by performing a page-turning input.

0080
(4) Page-turning, bookmark insertion processing, etc.
Next, a specific example of page-turning using the electronic book of this embodiment will be explained.

0081[1] Page-turning input operation
In the electronic book of this embodiment, as most operations are input through movement of fingers on the screen, there are very few buttons on the casing and the like for operation input. Another characteristic is that there are very few marks for operation input on the screen on which book details are displayed (see Fig. 3).

0082
In this embodiment, by installing travel information software in an electronic book, as shown in Fig. 3, the user can view the same details on the screen as those in a [paper] book in a similar manner as viewing a travel guidebook.

0083
When the user wants to see the next page, if the user traces a finger in the information display area on the screen as if turning a page of a paper book, as shown in Fig. 14, the display screen changes to the details on the next page with an image of turning as if there were a page.

0084
Here, the user is required to press a forefinger firmly on the screen and to move his finger in a horizontal direction as if rubbing. In this embodiment, in order to distinguish page-turning inputs from other inputs, if the finger contacts the screen within a prescribed surface area and with a prescribed pressure, and moves in a horizontal direction, this is judged to be a page-turning input.

0085
Note that the page-turning direction is reversed, depending on vertical text vs. horizontal text, similar to actual books. Specifically, in the case of a vertical-text book where pages proceed from right to left, when the movement shown in Fig.