Recognising Sketch Objects as Gestures via Autonomous Agents

Khaireel A. Mohamed
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Geb. 51, 79110 Freiburg
email: khaireel@informatik.uni-freiburg.de

April 30, 2003

Abstract

Without drawing too much attention to the advent of technology, interaction between users and synthetic hardware systems – through appropriate software interfaces – have provided opportunities to utilise an important sensing modality, through the use of a digital pen (stylus). Recognising and reacting to the input sketches and gestures from the stylus assist in expediting responses needed to improve the user-interface (UI) supports for more natural communications. This article portrays the techniques and methodologies used in ciphering information obtained from the digital ink compositions that gave way to the conception of more elaborate and complex programs and toolkits. Focus is given towards the classification of *sketch-objects* for identification purposes while maintaining its native attributes, as well as approaches for interpreting *gestural* inputs from the stylus – [Re-do this!].

Keywords: Digital pen gestures, recognition, interpretation, strokes

CR Classification: G.1.10 [Mathematics of Computing]: Applications, H.3.3 [Information Systems]: Information Search and Retrieval - Query formulation. I.7.5 [Computer Methodologies]: Document Capture - Graphics recognition and interpretation,

1 Introduction

Publicised works in computer sciences, mathematical analyses, and psychological experiments indicate that users' experiences with the interaction technology between the stylus and the digital paper must not be overly obstrusive and awkward. This interaction becomes a burden and an obstacle for a successful natural sketching environment, if it draws too much attention to the technology or imposes a high cognitive load on the users than the tasks at hand [3]. But more often than not, using only the stylus for sketching inputs to the GUI programs is not enough to satisfy demanding scenarios – such as to quickly change a pre-edited document, or calling up menus for issuing commands to the applications. Inevitably then, a unique way to achieve this anomalously, without pressing the corresponding key on a standard keyboard, is to stroke a *gesture*.

Inking, a process that enables "digital ink" from a stylus, allows users to insert a sketch or drawing, jot down a chart, take free hand notes, and annotate existing documents seemlessly in situations where fiddling with a keyboard or a mouse is challenging. Maintaining *ink* as ink allows the preservation of *ink*-objects in their "native data types". This ideology paves the way for the development of adaptive goals necessary when contriving and appreciating new objectives.

The experiments administered for this report are conducted with the intention of reviewing the mechanics behind, and the processes involved in gesture recognition. Our approach uses primary *agents* that perceive the sketching environment and act accordingly to decipher the *ink*-objects as valid (or invalid) gestures (*GestureObjects*). These agents are then observed and studied to determine their viabilities as background helpers for their object-interpretation capabilities, as well

as their efficiency in communicating data between themselves and the system. Their rationality and intelligence lies on them giving correct inferences, with an attached value of statistical confidence, when determining the possibility of an *ink*-object as a recognised GestureObject. The conception of our agents is discussed in further details in section 3.

Embedded into these agents are the conventional methodologies adapted from Rubine's algorithm [9], based on his techniques for extracting features from *ink*-objects to supplement the heuristics and calculations of his algorithm, as part of our recogniser engine. In addition to those techniques, we also incorporated the features presented on their work on *Quill* by the Group for User Interface Research (GUIR) from the University of California, at Berkeley (UC Berkeley) [8]. As our set of experiments forms the foundation for our future work on "protocolling GestureObjects", such that we need to include some elements of dynamism when recognising gestures, supplementary techniques to extract *invariant* features are also augmented. Section 4 gives the various mathematical illustrations on how these features are derived.

The next two following sections accord Rubine's algorithm itself, and the incorporation of the algorithm into the background agents. The sections also provide some insights on the structure and organisation of the classes and packages of our Java codes. Complete details of figures, tables, and numbers are reported in the Results section, followed by a section discussing these results to complement our claims that the implementation of our agents are not only successful and efficient, but also accurate in giving correct interpretations of *ink*-objects as intended recognised GestureObjects.

2 Representation of Gestures and the Gesture Continuum

The power of gestures lies in their ability to convey expressive and meaningful strokes as commands to the system without having to leave the vicinity of the digital screen. Cutler and Turk [3], Barrientos and Canny [1], and Wexelbrat [13] noted in their reports the following functional roles of human gestures:

- Semiotic – to communicate meaningful information.
- Ergotic – to maniputlate the environment.
- Epistemic – to discover the environment through tactile experience.

Cutler and Turk further expanded their definitions to include *communication tasks* when interpreting the gestural inputs from the stylus. These tasks specify the commands and parameters for:

- Navigating through 2D space;
- Specifying items of interest;
- Manipulating objects in the sketching environment;
- Changing object values; and
- Issuing task specific commands.

In his sociological field of study, Kendon [7] remarked an important part of 'kinetics' research showing just how gesture phrases are organized in relation to speech phrases. We can parallel his arguements and reasonings to relatively coincide between the pen gestures (as natural human gestures) and the instantiated sketch-objects (as dictionised speech contents). Kendon stated that there is a consistent patterning in how gesture phrases are patterned in relation to the phrases of speech – just as, in a continuous discourse, speakers group tone units into higher order groupings resembling a hierarchy, so gesture phrases may be similarly organized.

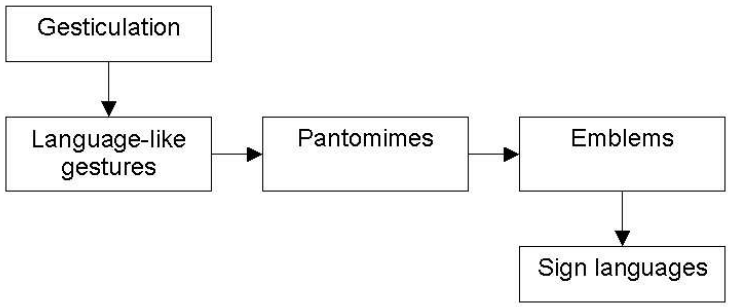

Figure 1: Kendon's description of the gesture continuum defining five different kind of gestures.

Gestures that are put together to form phrases of bodily actions, which have characteristics that permit them to be 'recognized' as components of willing communicative action is represented in Figure 1. An elaboration of their meanings is given below:

- *Gesticulation* – spontaneous movements of the hands and arms that accompany speech.
- *Language-like gestures* – gesticulation that is integrated into a spoken utterance, replacing a particular spoken word or phrase.
- *Pantomimes* – gestures that depicts objects or actions, with or without an accompanying speech.
- *Emblems* – familiar gestures accepted as a standard.
- *Sign languages* – referring to the complete linguistic system, such as the American Sign Language.

Note that as the list progresses (moving from top-left to bottom-right of Figure 1), the association with speech declines, language properties increases, spontaneity decreases, and social regulation increases. This gesture continuum clearly states a useful background support for deriving systems that are able to receive continuous pen gestures as inputs, along with a string of referenced objects.

3 Autonomous Agents

Agents are best used in environments where there exist ample opportunitities for full expositions of knowledge representation and reasoning. Russell and Norvig [10] describe a knowledge-based agent at three levels as follows.

- *Epistemological level* or knowledge level, which is the most abstract level, outlines the agent by saying what it knows. For example, our autocratic agents might be said to know that there is a strong correlation between recognised gestures a and b, from the newly scribed *ink*-object c.
- *Logical level* is the level at which the knowledge is encoded into sentences. For example, the agent might be described as having the logical sentence $\mathbf{StrongCorrelation}(a, b, c)$.
- *Implementation level* is the level that runs on the agent's architechure, which is relative to the system that the agent has been designed to operate in.

In our implementation, we require our agents to immediately begin their interpretation, that a possible gesture has been scribbed, from the raw data obtained from the sketches of *ink*-objects (*SketchObjects*). Depending on the hardware environment [11], or the human factors associated

with this scribing [12], the SketchObjects may appear noisy and render it difficult for any algorithms to efficiently give good diagnoses. Hence the need for an intermediate level to filter out this noise and rationalise appropriately the sampled data in the SketchObjects.

Once we are confident that a SketchObject is rationalised and observable, our agents can then begin their diagnostics in extricating features that are unique only to a special class of recognised GestureObjects. Their analyses will refer to their epistemological and logical levels, which are instilled in their recogniser algorithms and knowledge angines.

3.1 Agents in the Architecture

Working ubiquitously in the background, we have structured that there be at least two specialised agents to receive the input data from any suitable interface, perceive the clarity of the information gathered, and to act accordingly by invoking necessary methods for identifying a SketchObject, compared to that of a recognised GestureObject from its library of databases. Obviously, one of the agents will be an active listener (*SketchAgent*), and the other a passive processor (*GestureAgent*) – that determines and associates a degree of probability to a SketchObject, compared to a recognised GestureObject. These information are then inferred for a DecisionAgent to accept or reject the hypothesis of the GestureAgent.

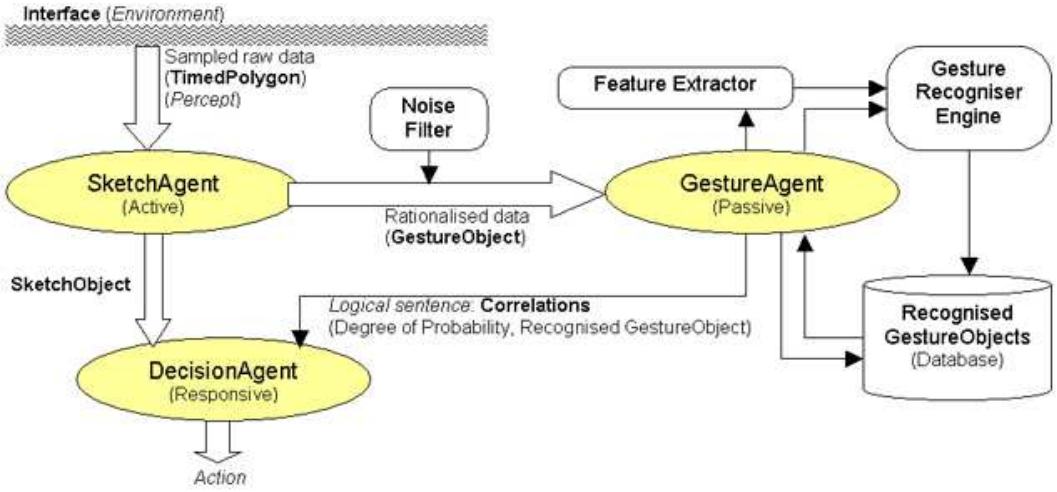

Figure 2: Deploying autonomous agents into the system's architecture for percepting, interpreting, and deciding whether a SketchObject is strong enough to be classified as a GestureObject.

The basic structure of how our three agents communicate within their embedded implementation in the system's architecture is illustrated in Figure 2. This structure is used for both the "learning phase", as well as the "application" phase when interacting with users. These two phases have different goals that complement each other; the former builds the database, while the latter retrieves from the database

3.2 Data Structures

The SketchAgent percepts its input from the interface environment through a series of ordered ink information, which are made up from the three-tupple primitive format (x, y, t). Here, x refers to the x-coordinate, y refers to the y-coordinate, and t is time at which the coordinates are sampled. The UC Berkeley group assembles this information collectively into an object class called TimedPolygon, which is an extension of Java's Polygon. These TimedPolygons correspond to our sampled raw data, as was previously explained.

Our SketchAgent siphons these TimedPolygons into SketchObjects by finding the center of gravity (COG) of the *TimedPolygons*, and make the coordinate vectors relative to the screen

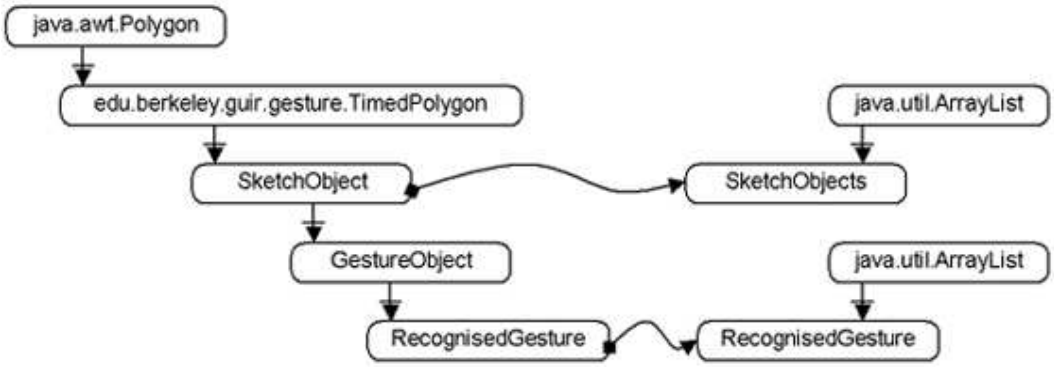

Figure 3: Object-oriented data structure for handling *ink-objects*.

origin at $(0, 0)$, while keeping a reference to its actual COG. Any future translations performed on the new SketchObjects can thus be done immediately without the extra hassle of additional projections to and from the center.

A noise filter is applied to the SketchObjects to remove points that are too close to one another that may affect the accuracy and correctness of calculations when applied. This results in a new instance of a GestureObject, in which features can be extracted assuringly.

*RecognisedGestures*on the other hand, are simply the extension of GestureObjects that contain appended information such as the name of the recognised gesture, the name of the library it is attached to, and the final weight values that are associated to a set of feature class (this is explained further in sections 4 and 5). Figure 3 gives an overview of the inception of the data structures that are being used by our agents.

4 Extracting Features from a GestureObject

In generative linguistics, *features* are defined as any of various abstract entities that combine to specify underlying phonological, morphological, semantic, and syntactic properties of linguistic forms and that act as the targets of linguistic rules and operations. Similarly, the features extricated from newly scribed GestureObjects relay useful information that can be used as parameters which uniquely identify a particular RecognisedGesture.

These features tell the angular information, velocity readings, and measurement of lengths.

4.1 Rubine's Standard Features

Rubine [9] used a set of 13 features, chosen such that each feature is incrementally computable in constant time per input point – this allows arbitrarily large gestures to be handled as efficiently as small ones. Also, each feature is made in such a way that it is meaningful enough to be used in gesture semantics as well as for recognition. Finally, there should be enough features to provide differentiation between all gestures that might reasonably be expected.

Figure 4 above illustrates the various algebraic terms and a geometrical view of some of the methods used to extract the standard features from a GestureObject. This is followed by the corresponding Equations (1) to (13).

$$f_1 = cos\alpha = \frac{x_2 - x_0}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}} \tag{1}$$

$$f_2 = sin\alpha = \frac{y_2 - y_0}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}} \tag{2}$$

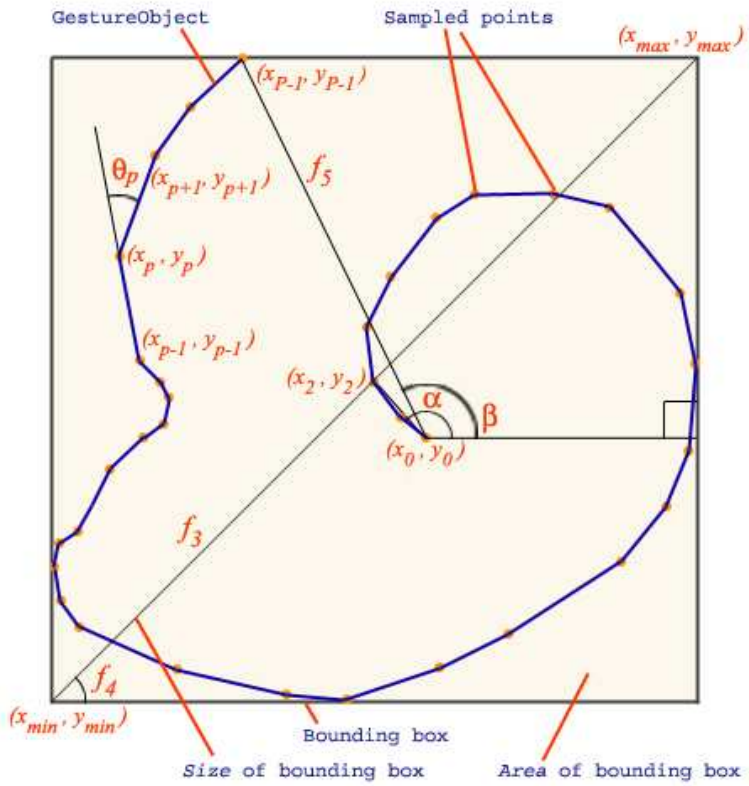

Figure 4: Features used by Rubine to identify strokes

$$f_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2} \tag{3}$$

$$f_4 = arctan\frac{y_{max} - y_{min}}{x_{max} - x_{min}} \tag{4}$$

$$f_5 = \sqrt{(x_{P-1} - x_0)^2 + (y_{P-1} - y_0)^2} \tag{5}$$

$$f_6 = cos\beta = \frac{x_{P-1} - x_0}{f_5} \tag{6}$$

$$f_7 = sin\beta = \frac{y_{P-1} - y_0}{f_5} \tag{7}$$

Let $\Delta x_p = x_{p+1} - x_p$ and $\Delta y_p = y_{p+1} - y_p$, and $\theta_p = arctan\frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1} \Delta y_p}{\Delta x_p \Delta x_{p-1} - \Delta y_p \Delta y_{p-1}}$, then

$$f_8 = \sum_{p=0}^{P-2} \sqrt{\Delta x_p^2 + \Delta y_p^2} \tag{8}$$

$$f_9 = \sum_{p=1}^{P-2} \theta_p \tag{9}$$

$$f_{10} = \sum_{p=1}^{P-2} |\theta_p| \tag{10}$$

$$f_{11} = \sum_{p=1}^{P-2} \theta_p^2 \tag{11}$$

Let $\Delta t_p = t_{p+1} - t_p$, then

$$f_{12} = \max_{p=0}^{P-2} \frac{\Delta x_p^2 + \Delta y_p^2}{\Delta t_p^2} \tag{12}$$

$$f_{13} = t_{P-1} - t_0 \tag{13}$$

$f1$ and $f2$ are the cosine and sine of the initial angle of the GestureObject, and highlights how rightwards and upwards the GestureObject is at the beginning, respectively. Rather than applying the actual angle, the sine and cosine of that angle is used instead to avoid the discontinuity as the angle phases through 360° and wraps back to 0°.

$f3$ and $f4$ deal with the information about the bounding box. The former describes the length and the latter describes the angle of its diagonal line. This diagonal length is also referred to as the size of the bounding box.

The distance between the starting and ending points of the GestureObject is given by $f5$. $f6$ and $f7$ convey the cosine and sine of the angle between the same starting and the ending points. The cosine of which tells us about the horizontal distance between the two points, whereby if the ending point lies on the left of the starting point, $f6$ will be negative. On the other hand, the sine of the angle tells us about the vertical distance, which will make $f7$ negative if the ending point lies below the starting point.

The actual total length of the GestureObject from its starting to ending points is given by $f8$, and the collective total angle traversed is summed up in $f9$. While the absolute value of the total angle traversed is defined in $f10$, the sum of the squared value of those angles is defined by $f11$. $f11$ here also tells us about the sharpness of the GestureObject, which is needed to distinguish between smooth and sharp gestures.

Finally, $f12$ calculates the sum of the maximum speeds squared, and $f13$ gives the duration it takes to scribe the GestureObject. Involving the time factor in the process for identifying gestures help to increase the likeliness of correctly interpreting intended gestures, and indicates that GestureObjects are dynamic objects and are not simply static pictures

4.2 Incorporated Advanced Features

The UC Berkeley group defined a set of advanced features and associated them as part of the "Human Perception of Gesture Similarity" [8]. Their application, *Quill*, tries to predict when people will perceive gestures as very similar, and this prediction is based on the geometric features of a GestureObject. These features are used in the context for similarity prediction, rather than for recognition.

4.2.1 Aspect

An *aspect* of a GestureObject describes the extent to which the bounding box differs from a square. A perfectly square bounding box has an aspect value of zero.

4.2.2 Curviness

This feature focuses on the strength of curve a GestureObject has, as opposed to being straight. The more curvy lines a GestureObject has the higher its value of *curviness*.

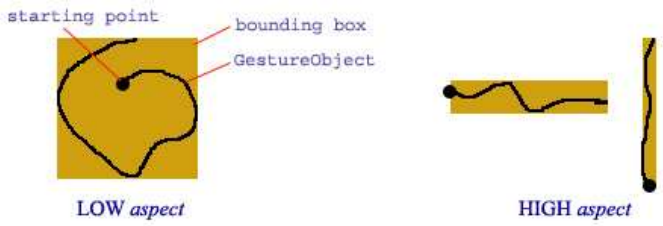

Figure 5: Degree of aspect of GestureObjects.

Figure 6: Degree of curviness of GestureObjects.

4.2.3 Roundaboutness

The Berkeley group defines this as the length of the GestureObject ($f8$) divided by its endpoint distance ($f5$). The closer the endpoints are, the higher the value of the *roundaboutness*. Although, care must be taken when evaluating endpoint distances that are too close to each other, or when the starting point equals that of the ending point – this will result in a value that is far too large, or Java throwing out a "divide by zero" error respectively.

4.2.4 Density

An insight as to how dense a GestureObject is with respect to its bounding box, is given by its *density*. This is obtained by dividing the length of the GestureObject ($f8$) by the size of its bounding box ($f4$).

4.2.5 Log of Area

As the name implies, the *log of area* of a GestureObject is defined by taking the natural logarithm of the area of the bounding box.

4.3 Invariant Features

It remains imperative that we are able to interprete intended gestures correctly, irrespective of their sizes, positions, and their disorientations (rather than orientations). As long as the elementary shapes of the GestureObjects scribed to the system are relatively similar to the ones contained in its *recognised-gestures* database, our agents should be able to identify these with strong degrees of confidence.

As such, all the features mentioned in sections 4.1 and 4.2 above should be rationalised before their values can be considered for any further calculations and comparisons. This is explained in further detail in section 4.4 below.

For now, we wish to determine additional attributes of the GestureObjects in the scene with the aim of using these to better distinguish their positions and orientations, that might otherwise be affected by noisy and undeterministic inputs. Invariant features involving moments of various orders, extracted from the GestureObjects, are incorporated as part of the gesture-recogniser input parameters. To a certain extent, the extraction of these moments from 1-D line-objects (which

Figure 7: Degree of roundaboutness of GestureObjects.

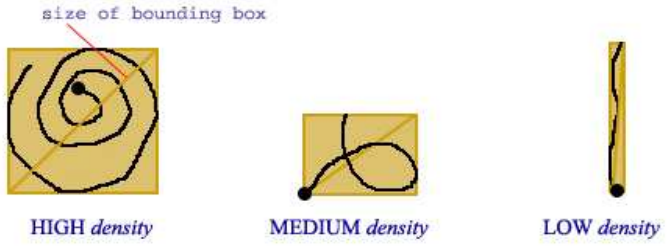

Figure 8: Degree of density of GestureObjects.

form the intrinsic architecture of our GestureObjects) are similar to that of extracting the moments from 2-D binary images

Figure 10 above contrasts the similarities in the way for excerpting moments between a 2-D binary image and a 1-D GestureObject. Equations used in our invariant features involve exploiting moments of the 0^{th}, 1^{st}, 2^{nd}, and 3^{rd} orders – the first three of which corresponds to the area, mean, and variance of a 2-D binary image. These are accorded to the length, median, and variance of a 1-D GestureObject

4.3.1 The 0^{th} Moment (2-D Area / 1-D Length)

An object in a 2-D binary image can be defined with the characteristic function given in Equation 14.

$$b(x,y) = \begin{cases} 1, & \text{for points on the object} \\ 0, & \text{for points on the background} \end{cases} \tag{14}$$

Assuming that the image is continuous (infinite resolution), the area is given by the 0^{th} moment of the object, as described in Equation 15.

$$A = \int\int b(x,y)dxdy = \sum_x \sum_y b(x,y) \tag{15}$$

Since the best way to estimate the pixel-area covered by the actual GestureObject would be to measure its length, we do it in the following manner. We know that the GestureObject is represented as a series of x and y coordinates with respect to the time they are sampled. Then, assuming that these sampled points form a smooth and continuos curve, we let $x = f(t)$ and $y = g(t)$, so that the 0th moment is the length of the GestureObject denoted by Equation 16.

$$L = \int_a^b \sqrt{f'(t)^2 + g'(t)^2}dt = \lim_{n\to\infty} \sum_{i=1}^{n} \sqrt{f'(t)^2 + g'(t)^2}.\Delta t \tag{16}$$

Figure 9: Log of area of GestureObjects.

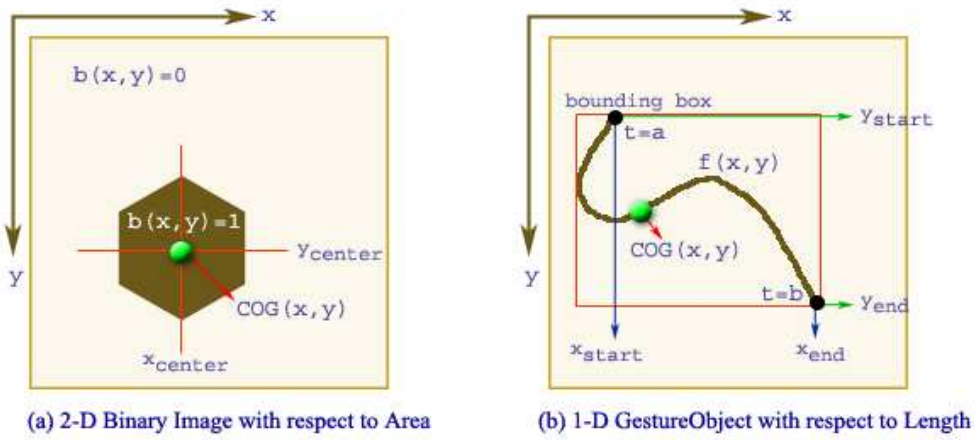

Figure 10: Retrieving moments in lengths of 1-D GestureObjects compared to retrieving moments in areas of 2-D binary images.

4.3.2 The 1^{st} Moments of Objects: Center of Gravity

The center of gravity (or center of mass), denoted here as $(\bar{x}, \bar{y})$ is given by the 1^{st} moments of the object defined in Equations 17 and 18 for an object in a 2-D binary image.

$$\bar{x} = \frac{\int\int xb(x,y)dxdy}{\int\int b(x,y)dxdy} = \frac{1}{A}\sum_x\sum_y xb(x,y)dxdy \tag{17}$$

$$\bar{y} = \frac{\int\int yb(x,y)dxdy}{\int\int b(x,y)dxdy} = \frac{1}{A}\sum_x\sum_y yb(x,y)dxdy \tag{18}$$

Similarly, we are able to obtain the center of gravity of a GestureObject with the same ideology involving the coordinates from the time samples. Defining $\underline{x} = \{x(t_0), x(t_1), ..., x(t_n)\}$ and $\underline{y} = \{y(t_0), y(t_1), ..., y(t_n)\}$, the 1^{st} moments of a GestureObject is defined in Equations 19 and 20. Using medians instead of means ensures the correct balance for a GestureObject sampled over time.

$$\bar{x} = \frac{\text{median }(\underline{x})}{\int_a^b \sqrt{f'(t)^2 + g'(t)^2}dt} = \frac{1}{L}\text{ median }(\underline{x}) \tag{19}$$

$$\bar{y} = \frac{\text{median }(\underline{y})}{\int_a^b \sqrt{f'(t)^2 + g'(t)^2}dt} = \frac{1}{L}\text{ median }(\underline{y}) \tag{20}$$

4.3.3 Moments of Higher Orders

Gonzales and Woods [6] classified objects with respect to their moments of various orders (Equation 22), and summed up moments of higher orders from the basic equation obtained from the moment of inertia as defined in Equation 21. These equations are based upon the areas (or lengths) and the centers of gravity of an object in an image, and strongly resemble that of statistical variance.

$$I = \int_S \sigma(x,y).\begin{bmatrix} x^2 & -xy \\ -xy & y^2 \end{bmatrix} dxdy \text{ where } \sigma(x,y) = \int\int (x-\bar{x})(y-\bar{y})b(x,y)dxdy \tag{21}$$

$$\mu_{pq} = \int\int (x-\bar{x})^p (y-\bar{y})^q b(x,y)dxdy \tag{22}$$

From here, Hu (1962) came up with a set of seven invariant moments, of which only the first four are being used in our implementation. The rest of which gave results that were too insignificant for our computations. We term these four invariant moments as phi equations and they are given in Equations 23 to 26. These four phi equations form the final four of our list of features that we use to extract from any input GestureObjects.

Let $\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$, where $\gamma = \frac{p+q+2}{2}$, then

$$\phi_1 = \eta_{20} + \eta_{02} \tag{23}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{24}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{25}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{26}$$

4.4 Rationalising Features

As we will see later in Rubine's algorithm for recognising GestureObjects (section 5), matrices are deployed in the calculations to solve linear equations. These matrices are the direct results from the extracted feature values defined in the sections above. Some of these matrices need to be inverted in order to find the answers we seek.

Our initial runs of retrieving features from a series of GestureObjects gave us values, v, that range from the very small ($v < 10^{300}$) to the very large ($v > 10^{+300}$). This often leads to our calculations being thrown out of bounds by Java, or given a "NaN" (not a number). Since all values calculated are important for the algorithm to determine the correct gesture, we took steps to ensure that these extracted feature values can be used in all calculations and will not result in silly computation errors.

Briefly running through the various features will tell us that gesture lengths ($f8$) are the main culprit in destabilising our calculations. Some GestureObjects may be short, while others very long. Related to this feature is also the duration of the gesture ($f13$), and its diagonal length ($f3$). Unfortunately, nothing can be done about the gesture length without losing precious information; except not to use it in the calculations directly – but rather, to use it indirectly. This gesture length serves as the basis for the rationalising of other affected features. For example, the rationalised diagonal length ($f3'$) is simply the diagonal length divided by the gesture length (i.e. $f3' = f3/f8$).

On the other hand, the duration obtained from the GestureObject ($f13$) is marked by milliseconds. This is rationalised by changing it to be marked in seconds (i.e. $f13' = f13/1000$).

5 The Recognition Algorithm

Each GestureObject is an array g of P time-stamped sample points: $g_p = (x_p, y_p, t_p)$, where $0 \leq p < P$.

Rubine's algorithm is a low-level recognition algorithm of two-dimensional, single-stroke gestures that are short and self-contained for both the classification and training algorithms [9]. There is a set of C gesture classes, numbered 0 through $C-1$. Each class is specified by example gestures. Given an input GestureObject g, the algorithm will determine the class to which g belongs – that is, the class whose members are most like g.

The main algorithm executes a two step statistical gesture recognition process; firstly, a vector of features, $\underline{f} = [f_1, ..., f_F]$, is extracted from the input GestureObject. And secondly, the feature vector is classified as one of the C possible gestures via a linear machine.

5.1 Identifying Gestures (Application Mode)

Provided with the feature vector f computed for an input GestureObject g, the classification algorithm associates each gesture class with a linear evaulation function over these features. Gesture class c has weights $w_{\hat{c}j}$ for $0 < i < F$, where F is the number of features. Per gesture class variables are written with hatted subscripts to indicate their class. The evaluations are calculated as shown in Equation 27.

$$v_{\hat{c}} = w_{\hat{c}0} + \sum_{i=1}^{F} w_{\hat{c}i} f_i \text{ , for } 0 \leq c < C \tag{27}$$

Classifying a gesture g is simply the c which maximises Equation 27.

5.2 Capturing Gestures (Training Mode)

The training part of the algorithm is a classic linear discriminator used for pattern recognition. The algorithm determines the weights $w_{\hat{c}i}$ from the example gestures without applying iterative techniques, which are avoided to get efficient training. Instead, it adopts a formula known to produce optimal classifiers under strict normality assumptions on the per-class distributions of feature vectors. Let $f_{\hat{c}ei}$ be the i^{th} feature of the e^{th} example of the gesture class c, $0 < e < E_{\hat{c}}$, where $E_{\hat{c}}$ is the number of training examples of class c. The sample estimate of the mean feature vector per class is the average of the features in the class (Equation 28).

$$\bar{f}_{\hat{c}i} = \frac{1}{E_{\hat{c}}} \sum_{e=0}^{E_{\hat{c}}-1} f_{\hat{c}ei} \tag{28}$$

The sample estimate of the covariance class c, $M_{\hat{c}}$, is computed as Equation 29. This is then averaged to yield M (Equation 30), an estimate of the common covariance matrix.

$$M_{\hat{c}} = \sum_{e=0}^{E_{\hat{c}}-1} (f_{\hat{c}ei} - \bar{f}_{\hat{c}i})(f_{\hat{c}ej} - \bar{f}_{\hat{c}j}) \tag{29}$$

$$M = \frac{\sum_{c=0}^{C-1} M_{\hat{c}}}{-C + \sum_{c=0}^{C-1} E_{\hat{c}}} \tag{30}$$

From here, we can determine the weights $w_{\hat{c}j}$ by inverting the common covariance matrix M, as explained in Equations 31 and 32 below.

$$w_{\hat{c}j} = \sum_{i=1}^{F} M^{-1} \bar{f}_{\hat{c}i} \text{ , for } 1 \leq j < F \tag{31}$$

$$w_{\hat{c}0} = -\frac{1}{2} \sum_{i=1}^{F} M^{-1} w_{\hat{c}i} \bar{f}_{\hat{c}i} \tag{32}$$

5.3 Probability Strengths

Linear classifiers always classify a GestureObject g as one of the C gesture classes. However, if there is a near tie for the maximum per-class evaluation function v_i, that gesture is then ambiguous. Still, we can estimate the strength of its relation to other recognised gestures within the library, and then deciding a threshold value for accepting (or rejecting) a particular GestureObject.

Given a gesture g with feature vector f classified as class i, that is $v_i > v_j$ for all $j \neq i$, then the estimate of the probability that g was classified correctly can be determined by Equation 33.

$$\tilde{P}(i|g) = \frac{1}{\sum_{j=0}^{C-1} e^{v_j - v_i}} \tag{33}$$

5.4 Feature Assignments

Taking into account of all the various features that can be extracted from a single GestureObject (as explained and illustrated in section 4), we used the assignments shown in Table 1 for our feature vector f as in Equation 27. This table is referred directly in our discussions for sections 7 and 8, with their corresponding values to $wc1, wc2, ..., wc18$.

f	**Assignment**	f	**Assignment**	f	**Assignment**
$f1$	Center of gravity	$f2$	Length per diag length	$f3$	Angle of diagonal line
$f4$	Cosine (initial angle)	$f5$	Sine (initial angle)	$f6$	Distance 1st to last pts
$f7$	Cosine (1st to last pts)	$f8$	Sine (1st to last pts)	$f9$	Total absolute angles
$f10$	Maximum speed	$f11$	Total duration	$f12$	Aspect
$f13$	Curviness	$f14$	Roundaboutness	$f15$	Density
$f16$	Log of area	$f17$	Phi vector 1	$f18$	Phi vector 2

Table 1: Assignment of feature index.

6 The Gesture Agent

Since Rubine's algorithm makes it necessary to have good example sets (obtained during Training sessions) in order to make reliable inferences when in application phases, our GestureAgent must be intelligent enough to set its goals accordingly when placed in the two different scenarios. Each goal invokes a different part of the algorithm; it either builds up a good set of example libraries, or refers to and retrieves relevant information from these libraries to perform the necessary calculations.

Figure 11 illustrates the internal works of our GestureAgent in its processing of information and communicating with its environment. The GestureAgent's goal is determined at start-up time by the system, and this same agent can be deployed in both *Training* and *Application* modes.

As we have discussed earlier, our implementation of Rubine's algorithm makes provisions in taking necessary precautions to rationalise feature values before performing any computations on them. But there are some unpredictable situations when even this precaution is overlooked, and is only discovered at the end of the entire evaluation – a rather costly rejection method. As such,

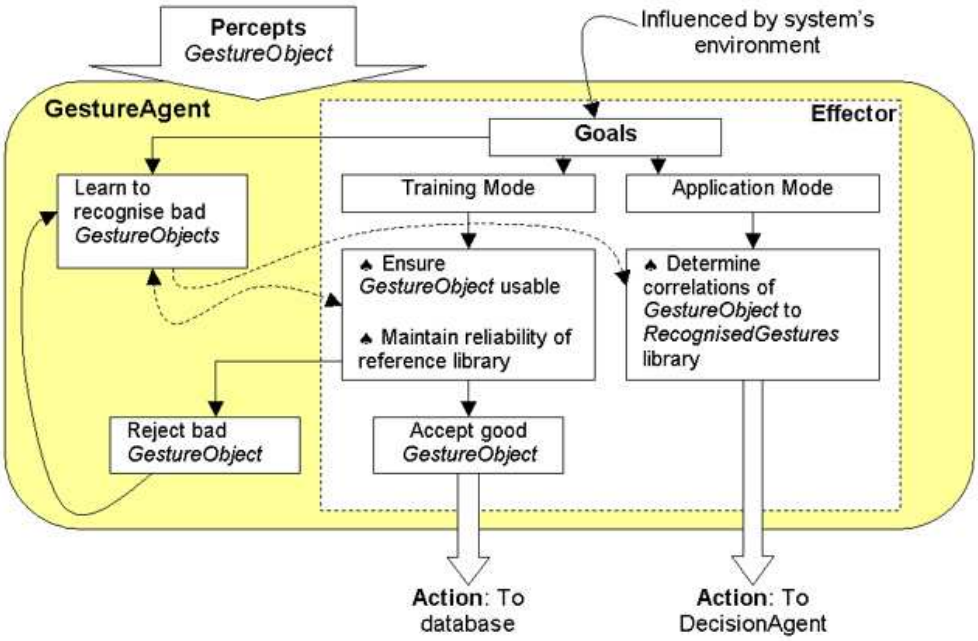

Figure 11: GestureAgent interacting with its environment.

our GestureAgent is programmed to learn the feature values that cause its calculations to break down, and we built mechanisms to intercept bad GestureObjects before processing them in both Training and Application modes.

7 Results

Our experiments are tailored to test the robustness of our agents in receiving and reacting to the information provided from the user interface. They also test the correctness of the implementation of Rubine's algorithm, within the GestureAgent, in making correct inferences and deciphering the uniqueness between the various classes of the recognised GestureObjects.

The section lists the complete details of the outcome of our experiments; in both the Application mode, as well as in the Training mode. The former lists the interpreted results with their associated evaluated-vector-values, V_c (from Equation 27), and their probability strengths, $\hat{P}(i|g)$ (from Equation 33). The latter illustrates graphs depicting the weights $w_{\hat{c}i}$ as they were trained.

Altogether, we structured nine sets of gestures for the experiment; the first five of which are straightforward and anomolous gestures. The last four, on the other hand, are made to evaluate the strength of the recognition algorithm as it is put to interprete gestures that are ambiguous in their orientations and directions of which they are scribed.

Also, the graphic details of our experimentations depict the proposed "ubiquitous" canvas-interface. Through it, users are able to view programs running in the background as they continue to use their stylus to write naturally on the semi-transparent canvas. With our application intended for use in "teaching" domains, this would serve as a useful property that does not block an open PDF document or a Powerpoint presentation.

Figures of various experiments will be included in subsection here.

8 Discussion of Results

Our experiments were conducted in a controlled environment with the main objectives structured to assess the stability of the autonomous agents running in the background, as well as to ensure that the intended gestures scribed by various different users are correctly interpreted with reference to the library of RecognisedGestures. Here, badly scribed gestures or gestures scribed that are not part of our library of RecognisedGestures are not taken into account for the report.

The efficiency of our agents, as a whole, are discussed by their time complexity analysis via the big '**O**' notation. As we did not stress on the implementation of the user interface, we have not included that discussion in this article.

8.1 Correctness and Completeness

The results we recorded and tabulated in the previous section all show that the strongest probability value $\hat{P}(i|g)$ for each input gesture scribed, correspond correctly to that of the user's intended gesture. This is also true for the evaluated-vector-values V_c.

Even when the input gestures are oriented at a variety of angles and scaled in different sizes, the algorithm is still able to recognise the anomolous gesture classes with strong confidence, and the ambiguous gesture classes with a comfortable margin that separates the next highest interpreted RecognisedGesture. The diversity in the styles possesed by seven different people when scribing our experimental gestures also resulted in the algorithm deciphering correct and *near*-correct (those interpretations that came in second in the list of probability strengths) judgements of the intended gestures.

As each information set is communicated between the SketchAgent and the GestureAgent, their epistemological and logical levels are purportedly observed to be correct and thus, complete. That is, it is ensured the recognition process is seamlessly integrated into the implemented system. These epistemological levels refer to the agents knowing that there exist a correlation of likeliness between the newly scribed GestureObject to its recognised library contents. These are followed by the logical sentences generated for each input GestureObject as the list of probability strengths, passed on to the DecisionAgent to accept or reject the hypotheses through its learned threshold value

8.2 Complexity Analysis

Generally analysing the time complexity of the whole agent-system implementation is done in three distinct steps; the feature extraction T_f, the training T_t, and the identifying T_i.

The features in a GestureObject are extracted sequentially, and some of these sequences depend on the number of points sampled, n, in the GestureObject. 18 features are extracted in our experiments, 12 of which involve straightforward calculations ($\mathbf{O}(1)$), and the rest are derived from their series of mathematical formulae ($\mathbf{O}(n)$). Hence, it takes $T_f(n) = 12O(1) + 6\mathbf{O}(n)$ amount of time to extract the features of an input GestureObject.

Again, the number of sampled points in any particular GestureObject play an importain role – this time, in determining the complexity of the training time. Equation 28 gives a complexity of $\mathbf{O}(nE)$, where E is the number of training examples for the class. We implemented this to build the common covariance matrix M simultaneously. Inverting M costs $\frac{1}{2} \times \mathbf{O}((nE)^4)$, as proven by Blum *et. al.* [2]. Solving the linear equations are easy after the inversion of M, and gives a complexity related to the number of features extracted, $18 \times \mathbf{O}(1)$. Thus, it takes $T_t(n) = \mathbf{O}(nE) + \frac{1}{2} \times \mathbf{O}((nE)4) + 18\mathbf{O}(1)$ amount of time to train one gesture example.

The identification process, at the moment, looks at all c RecognisedGestures in the system's library to calculate the evaluated-vector-values ($c \times \mathbf{O}(n)$) and the probability values ($c \times \mathbf{O}(n)$) for comparisons. This gives a time complexity of $T_i(n) = 2c \times \mathbf{O}(n)$.

9 Conclusion

Equally essential in a sketching environment, apart from the recognition and interpretation of gestures, is the management of multiple SketchObjects and GestureObjects. For in order to manipulate them capaciously, the system will need a basis for it to successfully match the diagrammatic similarity between the sketched-shape and its spatial relations [4, 5]. As the entire system stabilises after many uses, it would have amassed a collection of useful diagram references within its databases, that may contain helpful information such as its holding properties, transformation abilities, and its compatibility with other diagrams.

Since the contents in the database would already be sorted and tagged appropriately, the retrieval of these information for referencing may be indexed on its conceptual design features, on functions and purposes, on visual similarity, or shape – depending on the domain specificity of the system's implementation.

The three agents employed for this experiment will form the principal foundation of the intelligence of our system in its attempt to put together temporal and spatial information of the *ink*-objects for recognising more complex formations of sketches and gestures. The next step in our "protocolling" gestures (and sketches) will involve two additional agents to help in both the learning and recognising of special perspectives and a different paradigm to achieve our targeted goal of gesture programming

References

[1] Barrientos, F. A., and Canny, J. F. Cursive: Controlling expressive avatar gesture using pen gesture. In *Proceedings of the 4th International Conference on Collaborative Virtual Environments* (2002), ACM Press, pp. 113–119.

[2] Blum, L., Cucker, F., Schub, M., and Smale, S. *Complexity and Real Computation*, 2nd ed. Springer-Verlag, New York, Inc., 1998.

[3] Cutler, R., and Turk, M. View-based interpretation of real-time optical flow for gesture recognition. In *Proceedings of the Third IEEE International Conference on Automatic Face and Gesture Recognition* (1998), IEEE, pp. 416 –421.

[4] Do, E. Y.-L. What's in a diagram (that a computer should understand)? *Computer Aided Architectural Design Futures '95* (1995), 469–482.

[5] Do, E. Y.-L., and Gross, M. D. Drawing analogies - Supporting creative architectural design with visual references. In *3rd International Conference on Computational Model of Creative Design* (1995), M.-L. Maher and J. Gero, Eds., pp. 37–58.

[6] Gonzalez, R. C., and Woods, R. E. *Digital Image Processing*, 2nd ed. Addison-Wesley Pub Co, 2002.

[7] Kendon, A. An agenda for gesture studies [online], Available: http://www.univie.ac.at/wissenschaftstheorie/srb/srb/gesture.html [Accessed 19 January 2003]. *Semiotic Review of Books* (2001).

[8] Lam, Z. GUIR projects [online], Available: http://guir.cs.berkeley.edu/projects/ [Accessed 14 April 2003]. *Group for User Interface Research* (2003).

[9] Rubine, D. Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques* (1991), ACM Press, pp. 329–337.

[10] Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, USA, 1995.

[11] Smyth, K. The digital pen: Mapping and data aquisition. In *The Annual Conference and Exhibition: A GEOSPATIAL ODYSSEY, GITA Conference* (April 1998), GIS Development, pp. 261–268.

[12] Subrahmonia, J., and Zimmerman, T. Pen computing: Challenges and applications. *Proceedings of the 15th International Conference on Pattern Recognition 2*, 1 (September 2000), 60–66.

[13] Wexelblat, A. Gesture at the user interface: A CHI '95 workshop. *ACM SIGCHI Bulletin 28*, 2 (1996), 22–26.