Interactive Systems labs.

Carnegie Mellon University
Pittsburgh, PA, USA

University of Karlsruhe
Germany

Pen-Based Gesture Recognition

Report

Edgar Seemann

Supervisors

Matthias Denecke
Alex Waibel

Acknowledgements

I would like to thank Alex Waibel for giving me the opportunity to do this project at Carnegie Mellon University and furthering my interest in gesture recognition.
I would also like to thank Matthias Denecke for his advice and guidance during this project. He was always there to render assistance whenever I needed it. Furthermore I would like to thank everybody who made my stay at CMU so enjoyable.

Abstract

Gesture Recognition is becoming more and more important in multi-modal user-interfaces. This work gives an overview of existing gesture recognition systems. It explains how these systems work and which techniques they use.

Based on experience with former systems, this work describes a new approach for recognizing pen-based gestures. This was done with a sample set of approximately 50 military symbols, drawn by 22 test subjects. The new system (GRec) is an on-line recognition system that uses a template-matching technique as its basic algorithm.

Results have shown GRec to be flexible with respect to the addition of new gestures and to possess high recognition accuracy. Furthermore it can be integrated into existing applications very easily.

1. Introduction 9
1.1. Motivation 9
1.2. Some Existing systems 10
1.2.1. Gregory B. Newby (Ph.D. Thesis) 10
1.2.2. D.H. Rubine (Ph.D. Thesis) 11
1.2.3. JOVE Project 11
1.2.4. ALIVE II 11
2. Recognizer technologies 12
2.1. Problems 12
2.2. Input devices 12
2.3. Preprocessing 13
2.3.1. Normalization 13
2.3.2. Feature Extraction 14
2.4. Recognition 15
2.4.1. HMM 15
2.4.2. Neural Networks 17
2.4.3. Template Matching 19
2.5. Conclusion 20
3. Pen-based gesture recognition (GRec) 21
3.1. Pen-based gesture recognition and Hand-writing 21
3.2. Motivation 21
3.3. GRec Overview 23
3.4. GRec Input 24
3.5. GRec Preprocessing 24
3.6. Recognition 27
3.6.1. Representation 27
3.6.2. Stroke-Matching 29
3.6.3. Template-Matching 30
3.7. Problems of the TemplateMatching approach 32
4. Implementation 34
4.1.1. GRec's own user interface 34
4.1.2. GRec as part of the CPoF system 36
4.1.3. Class hierarchy and interface definition 36
4.1.3.1. Application module 37
4.1.3.2. Stroke module 37
4.1.3.3. Displaying Module 37
4.1.3.4. Recognizing module 38
4.1.3.5. Template module 38
4.2. Conclusion 38

5. Evaluation 39
5.1. Data-collection 39
5.2. Performance 39
5.3. Conclusion 41
6. Summary 42
7. Future Perspectives 43
8. Appendix 44
8.1. Appendix A (Gesture Set) 44
8.2. Appendix B (Class Hierarchy) 49

Pen-based Gesture Recognition

1. Introduction

"A primary goal of gesture recognition research is to create a system which can identify specific human gestures and use them to convey information or for device control."[1]

This sounds quite good, but how is a gesture actually defined. The Random House Dictionary[2], e.g., defines a gesture as "the movement of the body, head, arms, hands or face that is expressive of an idea, opinion, emotion, etc."

This is a rather general definition. And, in fact, it is even not easy for humans to interpret gestures correctly, since they are often ambiguous.

Gestures are used for a wide range of activities. They are used, for example, to point at persons/objects or to indicate a certain direction. The main application, however, is the emphasis of speech and it is well known that body language is an important part in the communication between humans.

For applications in computer science this general idea of gestures is yet too complicated. It is therefore necessary to restrict to a certain set of gestures, which are non-ambiguous in their application domain.

1.1. Motivation

Even though the computer technology has evolved rapidly in the last few decades the way humans interact with it has basically remained the same. The user has to type specific commands via the keyboard or use the mouse.

This involves difficulties particularly for novice users. A new, more intuitive way would be desirable. Users should not only be able to operate computers without

---

[1] A Brief Overview of Gesture Recognition
(http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/COHEN/gesture_overview.html)

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

a wide knowledge of these systems, but there should also be an effective and fast way to provide a computer with data of what the human operator would like to do.

Besides speech and handwriting recognition, gesture recognition plays an important role in achieving this new level of ease of use.

Imagine for example a word-processing program, which can be operated by speech, handwriting and gestures. If you want to write something, you can just use a stylus and write naturally or use speech. If you did something wrong, you can just draw an X gesture to cancel the operation.

In recent years there has already been great effort in the field of speech and handwriting recognition and a number of commercial and non-commercial software products have been released that are quite successful in respect to recognition accuracy. Gesture recognition, however, has so far been relatively neglected. Nevertheless it promises great new ways of interacting with computers in a variety of application domains.

1.2. Some Existing systems

1.2.1. Gregory B. Newby (Ph.D. Thesis)[3]

This system was concerned with recognition of the American Sign Language (ASL). It used data gloves as input devices.

Newby discussed two different approaches.

In one approach he tried to use a fixed-parameter model. That means that each gesture is recognized by a specific number of parameters, e.g. how many fingers are used.
The problem with fixed-parameter approaches is that you can use only a limited number of gestures. The main advantage, however, is the conceptual and computational simplicity.

In the second he tried a somewhat more sophisticated approach using statistical similarity. He measured the similarity by the "sum of squares" method. Even though this is a simple statistical function it involves great computational complexity. But this approach can distinguish a large number of different gestures.

[3] Gesture Recognition using statistical similarity, Center On Disabilities Virtual Reality Conference 1993

1.2.2. D.H. Rubine (Ph.D. Thesis)[4]

In his PhD thesis Rubine deals with both single path gestures drawn with a mouse or stylus and multi-path gestures consisting of the simultaneous paths of multiple fingers (made with input devices such as a data glove).

The single-path gestures are used to represent word processing commands. His system requires the computation of 13 different features, e.g. distance between first and last point, total gesture length or maximum writing speed. These features are used to train the parameters of a linear classifier function which is used to discriminate between gestures.

The multi-path approach is somewhat more sophisticated. He uses algorithms to sort the different paths and cluster them according to their similarity. Finally a decision tree is used to classify the gestures.

1.2.3. JOVE Project[5]

The JOVE project was created to recognize infantry command signals. It was based on hand and arm positions of the standard army set gestures. There were both static and dynamic gestures.
The data was obtained from sensors on a persons wrists and shoulders.

The first method of recognition used was a template matching approach. Thereby regions of space are defined in which sensors would be positioned for certain gestures at a certain time.

Another method was trajectory matching. The idea is to view dynamic gestures as space curves. These space curves can be defined by their start point, curvature, torsion etc.

1.2.4. ALIVE II[6]

ALIVE identified full body gestures through basic image processing techniques. It was used to control virtual creatures in a virtual environment. The system distinguishes not only command types but also directions. For instance, the direction of a pointing arm is translated into a virtual creature's travel direction.
The system also uses background information of the virtual environment to interpret commands.

---

[4] The Automatic Recognition of Gestures, Dean Harris Rubine PhD Thesis
[5] http://www.hitl.washington.edu/scivw/JOVE/Articles/dsgbjsbb.txt
[6] A brief overview of Gesture Recognition
(http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/COHEN/gesture_overview.html)

2. Recognizer technologies

2.1. Problems

As we have seen, gesture recognition has a wide-range of applications. These applications still offer a large number of problems, which have to be solved.

First of all gestures normally don't have a well-defined semantics. Therefore it is necessary to define a certain meaning to a subset of gestures, which should be used for interacting with a computer system.

Defining gestures poses some difficulties. On the one hand you would like to define gestures, which are easily distinguishable by a classifying algorithm. On the other hand the gestures should be intuitive and easy to perform.

To achieve this compromise Palm Corp., for example, introduced a graffiti style handwriting in their handheld devices, which was easy recognizable for the software, but still relatively intuitive and simple to learn for a human operator.

Another big issue in the topic of gesture recognition is the question of representation. Often it is not obvious how to represent a gesture or a sequence of gestures in the computer. Is it reasonable to represent a gesture by a high-dimensional feature vector and which features should be used? Or is it cleverer just to store the trajectory of the gesture as a template?

Since recognition should work for every user, gesture recognition systems have to be very flexible. Either the used classifier can deal with exceptions and particular variations or the algorithm is designed to learn these exceptions from a training set. That is the reason why learning algorithms are very popular in all recognition problems.

2.2. Input devices

Input devices provide the interface between humans and the computer. In order to adapt to application specific necessities numerous input devices are used.

Vision-based systems provide the most general approach. But they also involve many difficulties. Arms, hands and the body have to be tracked and separated from the background. Then the trajectory of the different moving body parts has to be calculated from the recorded images.

Data-gloves are very useful devices for gesture recognition. They allow a wide-range of motions and still provide an easy way to transfer the information about the gestures to the computer system.

Pen-based gesture recognition systems are mainly based on touch screens or graphic tablets. They could offer an easy and comprehensive way of interacting with already existing applications. For example, the gestures could be used for editing in a word-processing application.

2.3. Preprocessing

Pre-processing of the data provided by the input devices is crucial for decent recognition results.

On the one hand the data is mostly noisy. This may be caused by the input device itself or simply because of the fact that human movement is not uniform especially at the beginning and at the end of the gesture.

On the other hand pre-processing prepares the data for the classifier that is used. This can be done by simple conversion of the data format or by applying sophisticated algorithms.

For the following it is assumed that gesture trajectories are already extracted from the data of the input devices. We therefore do not focus here on image processing techniques for vision-based solutions.

2.3.1. Normalization

Having the raw coordinates of the gesture trajectories, there are various techniques that can be applied to improve the classifying accuracy.

Particularly important for input devices with slow sampling rates is smoothing (sometimes called noise reduction). Smoothing takes the coordinates of the input data and adds points, which seem to belong to the trajectory. This can be accomplished, for example, with spline filtering[7].

Another technique, which is necessary in combination with some classifiers, is rescaling. In doing this, the dimensions of the trajectory are simply stretched by a certain factor. Most of the times this is used in combination with an orientation normalization, which tries to determine the actual orientation of the gesture and then rotates the coordinates by an appropriate angle.

---

[7] "Comutation of Smoothing and Interpolating Natural Splines via Local Bases", T.Lyche and L.L. Schumaker

Since many classifiers require a specific number of data points, and because of computation complexity issues, trajectories are often resampled. Which means that their coordinates are equally distributed along the trajectory (this is called a equal-arc-length resampling procedure). This can be achieved by interpolation. In most cases simple linear interpolation is applied, but in some areas different types of interpolation offer better results. These different types offer additional smoothing features.

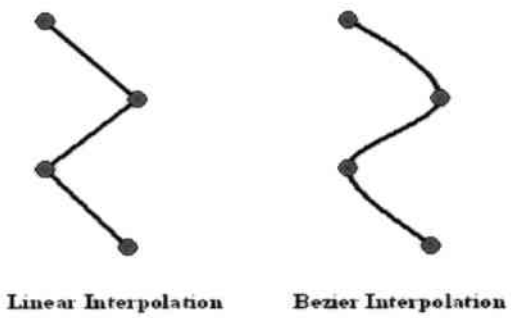


Figure 2.1 - Bezier Interpolation[8]

The reader should be aware that the goal of normalization is not to produce a perfect gesture, which isn't possible, but to process the gesture in a way, that makes the classifier achieve high recognition accuracies.

2.3.2. Feature Extraction

Feature extraction is one of the basic parts in recognition research. Determining which features to use and to compute them is strongly correlated with the recognition accuracy. Of course, the importance of the different features may differ from application to application.

However, some basic features are used in almost every gesture recognition system.

First of all, there are the features that are associated with the angles of the trajectory. Out of these angles you can calculate the curvature in each point and therefore determine whether a coordinate is an important feature point like a cusp (point of sharp directional change) or not. Other important feature points are turning points. These are the points where angle between the tangent and the axis turn from positive to negative or vice-versa.

Crucial feature points are also loops and crossings. These are rather writer-independent and invariant of the sampling rate.

---

[8] "On-line Erkennung kursiver Handschrift bei grossen Vokabularen", p75 PhD-Thesis Stefan Manke

Further geometric features are, for example, the maximum/minimum coordinate, initial direction etc.

Writing speed is a non-geometric feature, but it is a good indicator, since a user usually moves rather slowly in important feature points. That is why it can be used to confirm the above-detected geometric features.

2.4. Recognition

2.4.1. HMM[9]

Hidden Markov Models are widely used in speech recognition, but have also been applied in handwriting recognition systems. Their advantage is that they are very well able to process sequential data that is variable in time.

A Markov Models consists of a set of states:

$$\{S_1, S_2, ..., S_N\}$$

And transition probabilities between them:

$$a_{ij} = P(q_{t+1} = S_i \mid q_t = S_j)$$

Furthermore there are probabilities for the initial distribution:

$$\pi_i = P[q_1 = S_i]$$

As you can see from the definition, markov models assume that the probability of a state only depends on the preceding state. This is a strong assumption and the reader should be aware that this restricts the generality of markov models.

An example of a markov model is for example the weather as displayed in Figure 2.2 - Example of a Markov Model

[9] A tutorial on hidden-Markov models and selected applications in speech recognition, L.R. Rabinier 1989

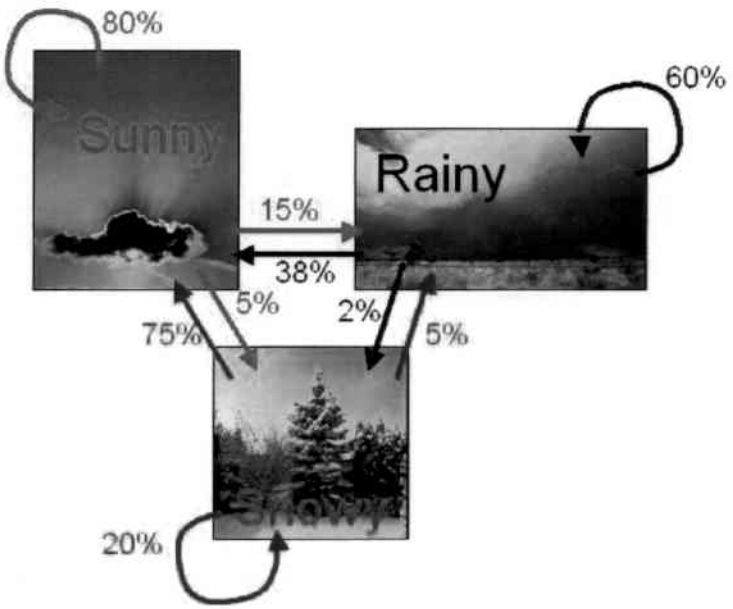


Figure 2.2 - Example of a Markov Model

Hidden Markov Models differ from normal markov models in the way that the states are not directly observable. Instead of that you have a number of observations that give you some information about a system:

$$\{O_1, O_2, ..., O_M\}$$

These observations give you an indication in what state the underlying markov model might be. The indication is represented by observation probabilities:

$$b_j(k) = P(v_t = O_k \mid q_t = S_j)$$

I.e. the probability of the fact that you observe an observation O_k while the system is in state S_j.

Let's keep up the link to the weather example. Imagine you're sitting in an office without a window. As we have seen the weather conditions represent the states, but what would the observations be? An observation could be the way people dress, for example. You know when somebody wears an umbrella the probability of rainy weather is quite high, but your not sure if it's really true.

How can apply the notion of hidden markov models to gesture recognition. In fact, the principal is quite simple. Known gestures represent the states of the markov model and the gestures actually executed represent the observations. There are plenty of ways how to associate the gestures with states and this association is the most difficult part in this approach.

Once you have decided which states you use and how the gestures represent them, the only thing you have to do is to assign a probability of how likely it is that the user wants to do gesture X, when he moves in a certain manner.

But how do I assign these probabilities?
Fortunately there's a learning algorithm (**EM**) that let's you train the parameters of the hidden markov model. Therefore you simply have to initialize the model and let the algorithm do the work for you.

The algorithm consists of two steps.
The calculation of the expectation (**E**-step):

- Compute $P(q_t = S_i \mid O, \lambda)$ for given $\lambda = (\pi, a, b)$

And the computation of a new λ out of these expectations (**M**aximization-step).

It can be proven, that the EM algorithm converges, but the reader should be aware of the fact, that the EM algorithm not necessarily converges to a global optimum, but only to a local one. The initial probabilities should therefore be set wisely.

2.4.2. Neural Networks

Neural Networks have been applied to almost every recognition problem. And, in fact, they are doing very well on many problem classes.

Neural Networks consists of a number of interconnected nodes traditionally referred to as neurons. A neuron has n input signals (x_1, ..., x_n) and one output signal (see Figure 2.3 - neuron) and actually computes a mathematical function f. Usually a sigmoid function for f is used, that means f is non-negative, monotone and asymptotically bounded (for $x \to \infty$).
The output y of the neuron is therefore:

$$y = f(\sum w_i x_i)$$

The parameters w_i define the neurons behavior.

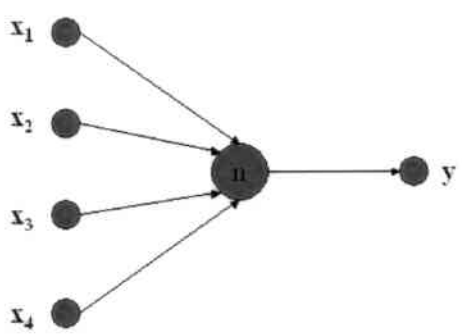


Figure 2.3 - neuron

A network consisting of only one layer of these neurons is called a perceptron (see Figure 2.4 – Perceptron). Perceptrons are able to compute a large number of mathematical functions, but unfortunately they are not able to compute any function, that is not linear separable, for example the XOR-function.

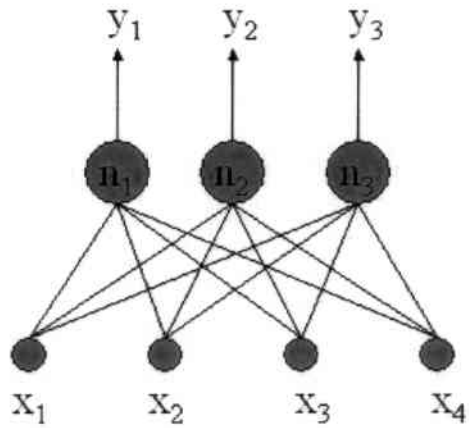


Figure 2.4 – Perceptron

More complex functions are representable by multi-layer perceptrons, which consist of an input layer, one or more hidden layers and a output layer (see Figure 2.5 - Multi-Layer Perceptron). The output of a neuron in layer j is:

$$y_j = f(\sum_i w_{ji} y_i)$$

where y_i is the output of the neurons of the preceding layer.

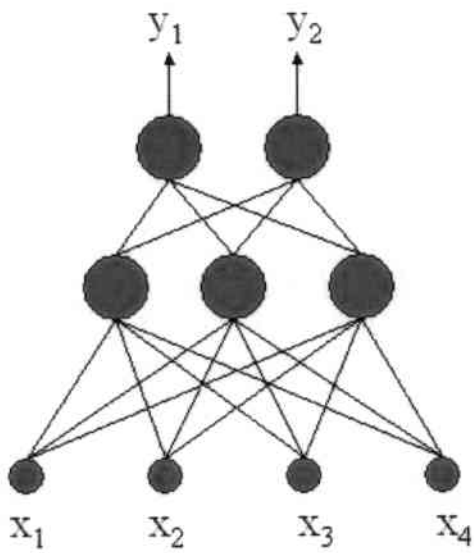


Figure 2.5 - Multi-Layer Perceptron

How can I use this technique for gesture recognition?

In a certain sense gesture recognition is also a mathematical function. You've got a specific input (for example the features of the gestures) and an output that determines which gesture has been recognized.

A multi-layer perceptron could therefore represent this function by having an output neuron for each gesture. The output neuron i would ideally be 1, if the gesture belongs to class i and 0 if the gesture does not belong to class i.

The actual problem now is, however, how to determine the weights w_{ij} of the edges, which connect the neurons and to determine which network architecture should be used (number of hidden-layers, number of neurons, connection rate etc.)

The question, which network architecture, is best, is still an unsolved problem, but there's an easy algorithm (backpropagation technique) that computes the optimal weights of the edges. It uses a gradient decent method to adjust the values gradually.

Consider the multi-layer perceptron produces the output $y_1, \ldots, y_n$. The actual output however should be $d_1, \ldots, d_n$. You can now assign an error function E that computes the difference between 2 outputs (an often used error function is the mean-squared-error method). The overall error is therefore err:

$$err = \sum E(d_i, y_i)$$

Since d_i and y_i are functions of the weights w_{ij}. The error function err is also a function of w_{ij}. To determine the new values for w_{ij}, we're computing the gradient of the error function and alter the each w_{ij} by a fraction of the gradient in the corresponding direction:

$$\Delta w_{ij} = -\delta \frac{d}{dw_{ij}} E(W)$$

with W weight matrix and $\delta \in (0,1)$.

Like the EM algorithm Gradient decent doesn't converge necessarily to a global minimum, but may get stuck in a local minimum.

2.4.3. Template Matching

The principle of the template matching approach is easy. Just store an example of each gesture in the system and compare the executed gestures with them.

Obviously this involves the question of representation, since the templates can be stored in a variety of manners. Depending on this representation the comparison algorithm should be chosen.

A commonly used algorithm is DTW (Dynamic Time Warping), which uses dynamic programming to find the best way to match the gestures.

The algorithm tries to associate the characteristics of the gesture with a template. If there's no corresponding characteristic the algorithm omits it and adds a certain penalty.

For further information you can see an application of the algorithm in chapter 3.6.3.

2.5. Conclusion

As we have seen gesture recognition has a wide range of application and a general technique that is applicable for all gesture recognition problems is not feasible, since gesture recognition systems have to meet application specific necessities.

Learning algorithms are particularly useful, when it's even for humans not evident how to distinguish the gestures. The algorithm can then find its way of classifying them. This is getting more and more important the bigger the set of gestures gets, since normal algorithms might have to be adjusted to many exceptions.

The question however is which technique is appropriate for the application, which technique promises to have the best recognition accuracy and which technique is the easiest to implement.

3. Pen-based gesture recognition (GRec)

3.1. Pen-based gesture recognition and Hand-writing

Pen-based gesture recognition has many similarities with handwriting recognition.

Both recognition tasks try to classify a set of symbols. In handwriting recognition this is the alphabet, whereas it is the gesture set in gesture recognition.

You can distinguish between online and offline recognition. Offline recognition is based on a scanned image, whereas online recognition uses the pen-trajectory data, which is recorded during the writing/drawing process.

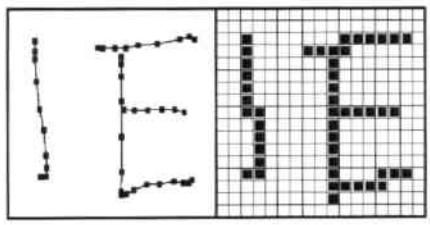


Figure 3.1 - online vs. offline recognition

In both cases you have similar pre-processing problems. The symbols have to be resampled, rescaled, rotated and the features have to be extracted.

Another issue is the so-called late strokes. These are strokes that are drawn much later than they actually should be. In handwriting recognition you have this problem, for example, in letters like t, where many writers draw the last stroke only at the end of a whole word. In gesture recognition this problem is even worse, because the gestures are often quite complicated and the strokes can be drawn in every possible order.

3.2. Motivation

The GRec gesture recognition system is part of the CPoF project (Command Post of the Future). The CPoF system is a program that allows military members to display and edit military units and tactical information on a map (see Figure 3.2 - CPoF Map).

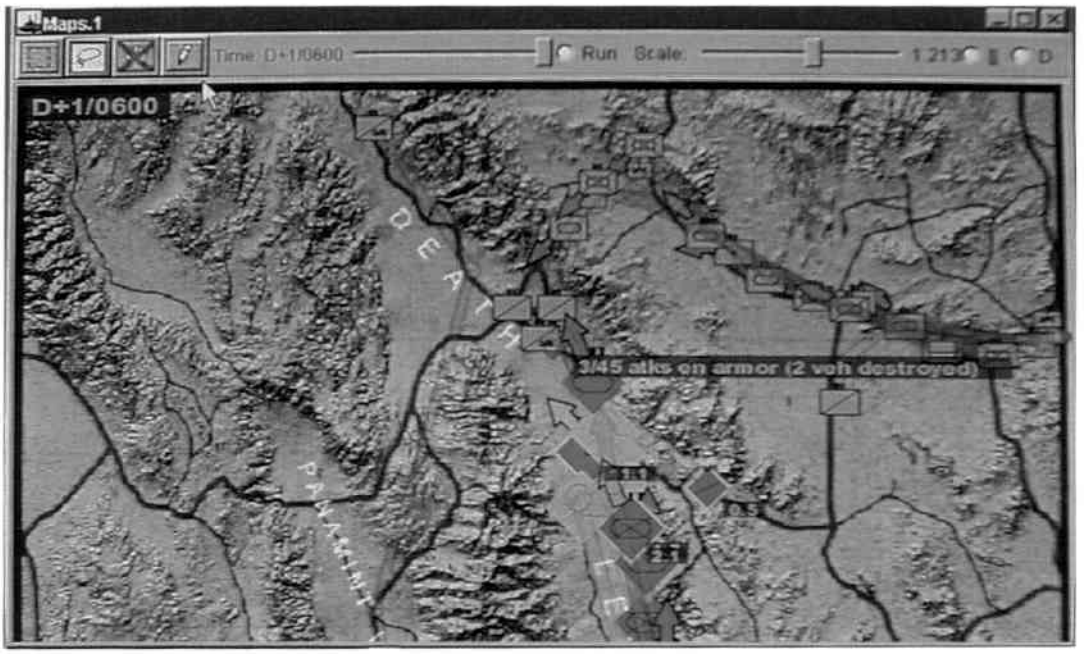


Figure 3.2 - CPoF Map

So far the system was mainly operated by speech commands. GRec's objective was now to integrate gesture recognition in the existing system. The recognizer should be able to classify a set of about 40 military symbols including some gestures for basic editing.

The operator should be able to draw the symbols directly into the map. After the recognition the corresponding image should be displayed. Furthermore it should be able to use the basic editing symbols for selecting and moving the units.

Most of the military symbols are modular. That means they consist of a basic symbol and some modifiers, which specify the exact properties of the unit (e.g. airborne, armored or motorized)

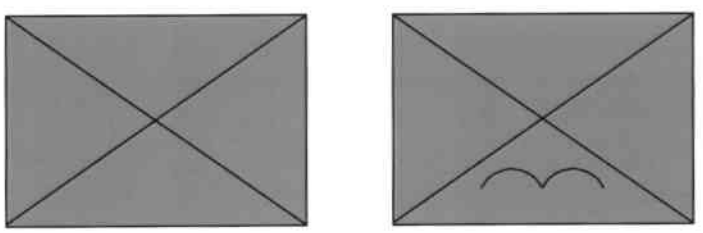


Figure 3.3 – Symbols

Therefore a gesture recognition system should be very flexible in the respect of adding and recognizing new symbols, since there are a variety of combinations the modifiers can be used with.

As the CPoF system is operated by a large number of people, the gesture recognition system should be writer-independent. This involves that it should be very flexible in respect of variations of how a gesture can be drawn.

3.3. GRec Overview

The GRec system tries to meet the requirements of an above mentioned gesture recognition system. It can deal with a large gesture set and new gestures can be added very easily.

GRec is based on an online template-matching algorithm and, as you will see, implemented a couple of techniques to prepare the drawn gesture for the classifying algorithm.

As in most recognition systems the GRec recognition process can be divided in 3 basic parts: Input, Preprocessing, Recognition.

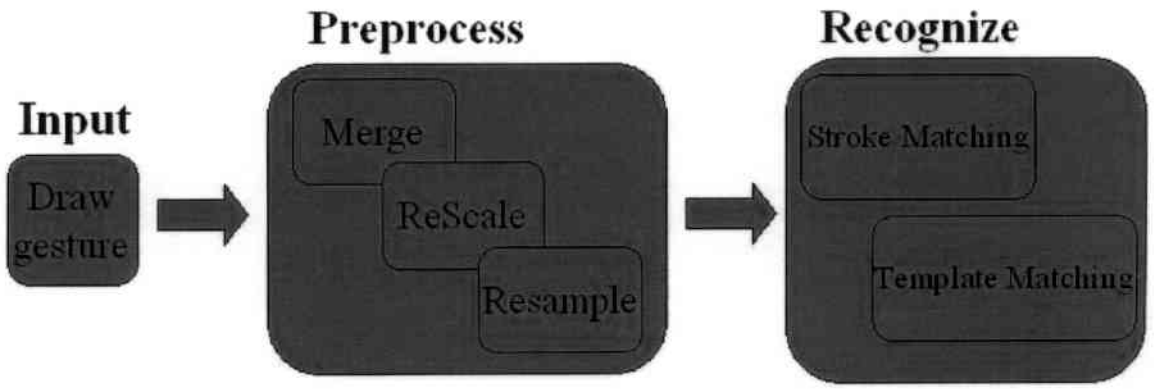


Figure 3.4 - Overview

Preprocessing consists of further steps. Firstly a merge routine tries to merge strokes, which belong together. Secondly a rescaling method resizes the strokes to a specific value. The third step is just a resampling of the data points.

The recognition process is split into two parts. The stroke matching function looks for corresponding strokes in the gesture and the template. To find the corresponding strokes it uses the template matching method for calculating an error value. This error value is accumulated for all corresponding stroke pairs and finally forms the result of the recognition process.

3.4. GRec Input

The input device for the system is a pen, which is used the same way as an ordinary computer mouse.

Therefore the system gets both "pen up/pen down" events and the pen coordinates at specific points in time. With each "pen down" event a new stroke is added. A "pen up event aborts the drawing of the current stroke.

Consequently you get several strokes that consist of a number of 2-dimensional points. These strokes are processed by the recognition system.

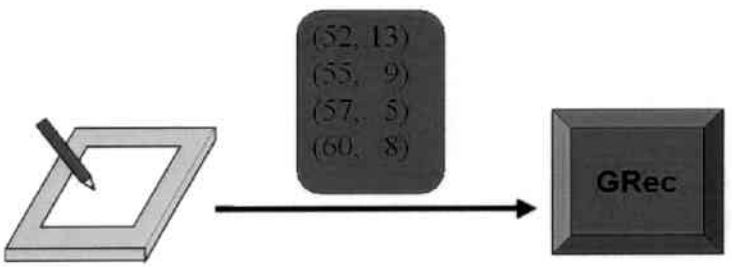


Figure 3.5 - Input of the system

3.5. GRec Preprocessing

Preprocessing is perhaps the most important part in gesture recognition. Even a really good classifying algorithm has a bad performance when the data is not correctly pre-processed.

In this particular case pre-processing involves a number of non-trivial problems.

Since the user should be allowed to draw a symbol the way he likes to draw, it is difficult to determine which stroke is to form which part of the gesture. E.g. a rectangle can be drawn with one stroke or with four strokes.

To make it easier for the recognizer a pre-processing function tries to merge strokes, which seem to form a certain component of the symbol. Since incorrect merging may result in a bad classification, the function computes various merging possibilities, which are transmitted to the recognizing algorithm.

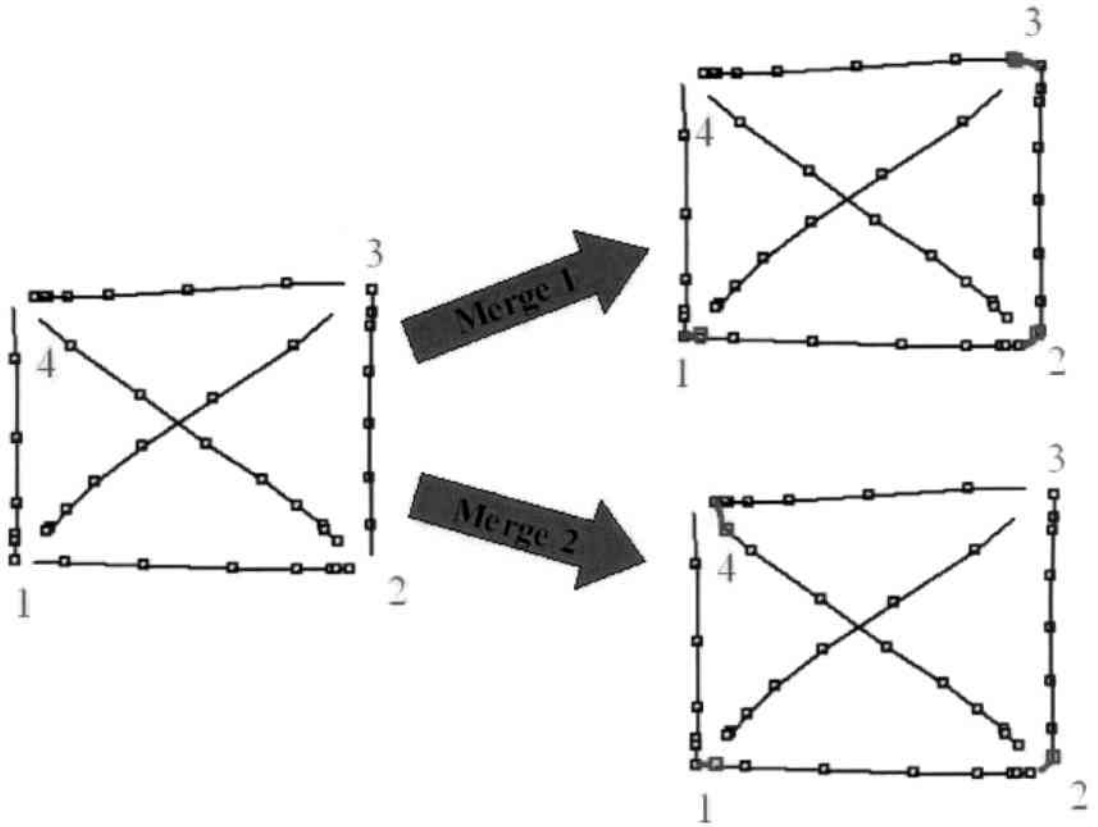


Figure 3.6 – Stroke merging possibilities

After merging the strokes some other pre-processing functions are applied to the data.

Firstly all the strokes will be rescaled to a certain size. On the one hand this is necessary because the drawn gesture will be matched with a template that has a specific size. On the other hand it is necessary to get an error that is comparable for all stroke sizes.

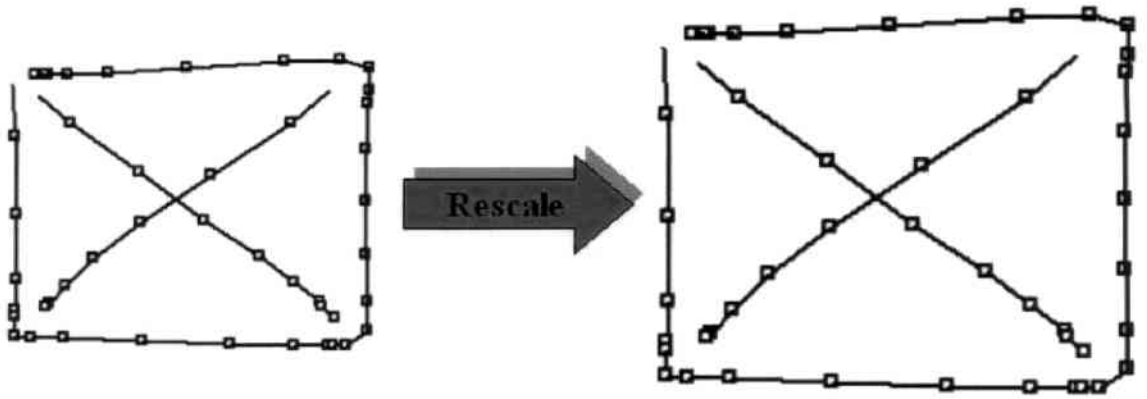


Figure 3.7 – Rescaling

Secondly, the strokes are resampled. That means that the points, which were sampled in equidistant time intervals during drawing, are distributed equally along the trajectory of the stroke. On the one hand resampling makes – as rescaling - the error more comparable, on the other hand it compensates the difference in drawing speed. After resampling the Euclidean distance d between any two consecutive points has the same value. This is particularly important when different input devices should be used.

The algorithm performing the resampling uses simple linear interpolation. That means that, if 2 points are two far away from each other additional points are added on the interconnecting line. In the case of point 2 is too close to point 1, point 2 is removed.

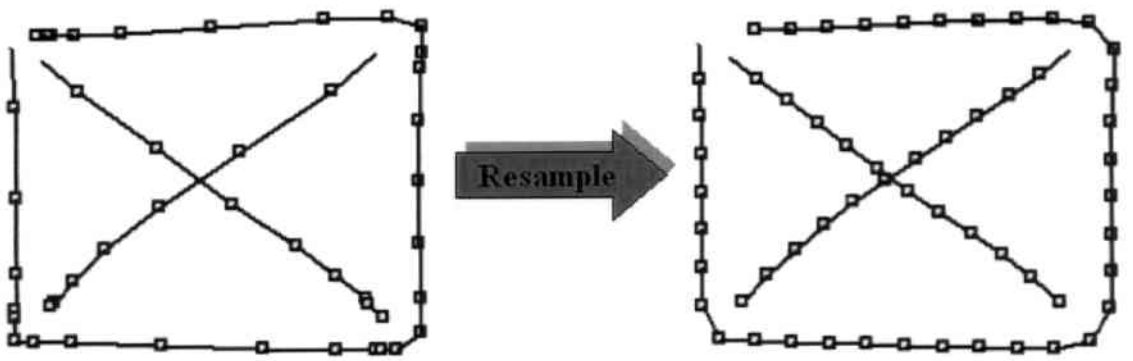


Figure 3.8 – Resampling

GRec has no smoothing feature, since the sampling rate of the input device is fairly high.

3.6. Recognition

After carefully considering all techniques that are applied in gesture recognition, I decided to use a template-matching approach as basic classifying algorithm. This was because template matching promises a good accuracy even if features like cusps are not easily extractable from the symbol and because it is very easy to implement.

3.6.1. Representation

The templates that are used to match a drawn symbol are represented as a compound of a number of basic components. E.g. an infantry symbol (see Figure 3.3 – Symbols) consists of a rectangle component and 2 lines. This is described in a file called `templates.txt`.

Example of a file entry in `templates.txt`:

```
infantry rectangle line(35) line(325) EOF
```

The values in the brackets are the angles by which the components are rotated and are relative to a horizontal line.

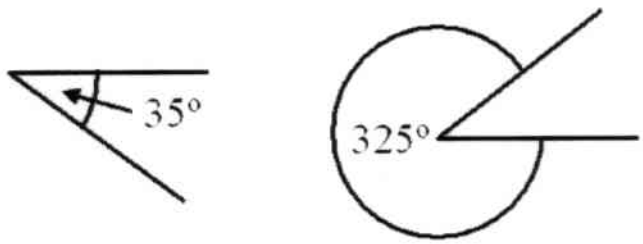


Figure 3.9 - Angles

The basic components itself are represented simply by the set of 2-dimensional points they consist of (at the moment they are painted with a special symbol editor and then stored in the GRec file format via serialization). To add a new component you have to add its filename to `components.txt`.

Since symbols can be drawn in various ways and a template-matching algorithm is sensitive for writing direction it is often necessary to draw a component in different manners, rotate the component or invert the writing direction.

Example 1: a triangle may be painted starting from an arbitrary corner. Therefore the component has to be rotated by 120 and 240 degrees in order to ensure correct recognition.

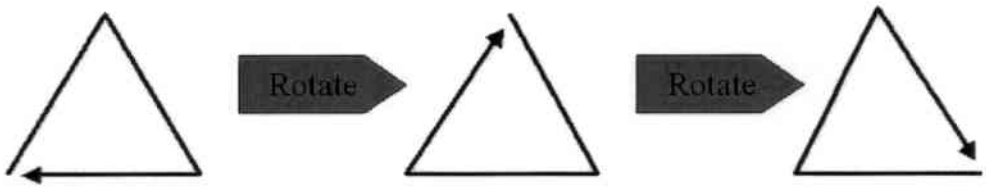


Figure 3.10 – rotating components

The rotation of a gesture requires a little bit of mathematics. Suppose the gesture consists of the points (x_1,y_1) … (x_n, y_n) or in matrix form:

$$G = \begin{pmatrix} x_1 \ldots\ldots\ldots x_n \\ y_1 \ldots\ldots\ldots y_n \end{pmatrix}$$

As we know a 2-dimensional rotation matrix looks like this:

$$R = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

The transformed gesture is thus:

$$G' = RG = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x_1 \ldots\ldots\ldots x_n \\ y_1 \ldots\ldots\ldots y_n \end{pmatrix}$$

Example 2: For a rectangle simple rotation is not enough, even though a rectangle is symmetric to both the x-axis and y-axis. Besides the rotation a second rectangle component is needed.

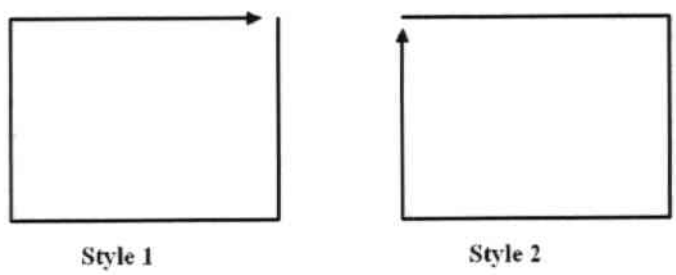


Figure 3.11 – different components

The different styles have to be added to the component file (using the GRec symbol editor). Each stroke in the file represents a different way to draw the

gesture. The rotation, however, is done dynamically during loading the templates from the filesystem.

Example of an entry in components.txt:

```
triangle 0:120 0:240
```

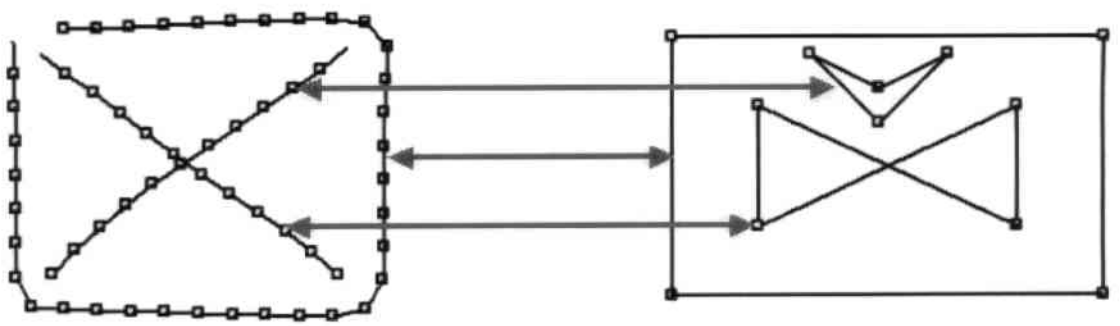
The values following the component name mean that stroke 0 should be added to the template rotated by 120 and 240 degrees (as you saw in Figure 3.10 – rotating components)

The loading of the templates and basic components is done during initialization of the GRec application. That is why GRec has to be restarted after modifying, adding or removing a template or component.

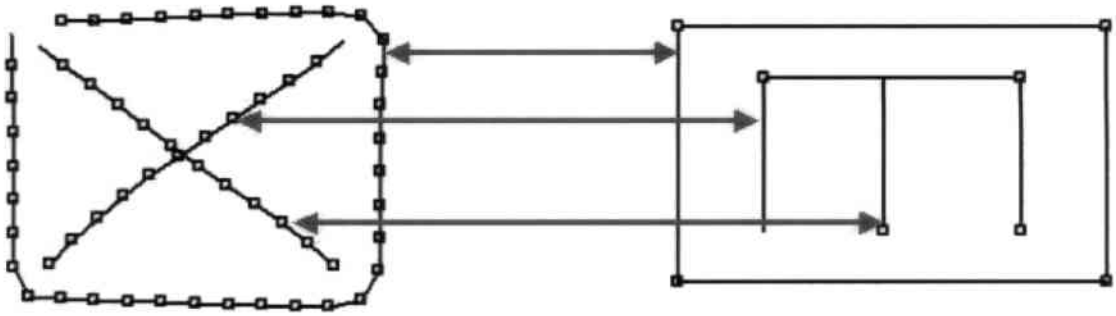
3.6.2. Stroke-Matching

The first step in the recognition process is stroke matching. The system tries to assign each stroke in the drawn symbol to a corresponding stroke in the template. The error value that is determined during the comparison of two strokes is calculated by a dynamic programming technique described in chapter 3.6.3.

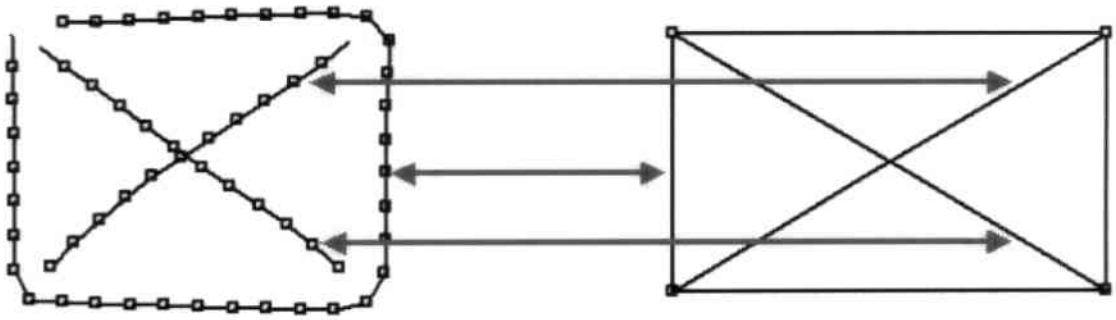
Example:



Total Distance: 13830



Total Distance: 8616



Total Distance: 1944

Figure 3.12 – assign strokes

The stroke-matching algorithm calculates the distance (error value) between each pair of strokes, chooses the pair with the minimum distance and removes them from the list of strokes.

So far the recognition process is aborted when the number of strokes in the template is not equal to the number in the drawn symbol.

For n strokes the computing complexity is $O(n^2)$ ($\sum_{i=1}^{n} i = \frac{n^2 + n}{2}$).

Since this has to be done for each of the m templates, the overall complexity for this step is $O(m*n^2)$.

3.6.3. Template-Matching[10]

[10] "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", Hiroaki Sakoe and Seibi Chiba

As you have seen above the actual template-matching algorithm is used to calculate the distance between 2 strokes.

It's a dynamic programming algorithm, which finds the best way to match the different points of the 2 strokes (referred to as stroke1 and stroke2).

This works as follows:

- Let $p1_i$ be the i-th point of stroke1 and $p2_j$ the j-th point of stroke2
- Move stroke2 on top of stroke 1. So that $p1_1$ and $p2_1$ are identical
- Label the i-th row of a matrix with $p1_i$ and j-th the column of the matrix with $p2_j$
- Let p be the current position in the matrix
- Set the position p to (0,0) (the upper-left corner)
- Assign 0 to the most upper-left element in the matrix (the 0 represents the distance of $p1_1$ and $p2_1$ which is 0, since we moved stroke2 before)
- For each iteration calculate the Euclidian distances between $p1_{k+1}$ and $p2_l$, $p1_{k+1}$ and $p2_{l+1}$, $p1_k$ and $p2_{l+1}$ (where (k,l) is the current position p in the matrix)
- Choose the shortest distance and move the position pointer to the new cell

You may notice that this algorithm terminates after a maximum of n+m iterations (where n is the number of points in stroke1 and m the number of points in stroke2), since during each iteration the algorithm increases either the row or the column index (or both).

The operation of increasing only the row/column index corresponds to leaving out a dispensable point in one of the strokes. The increase of both indexes, however, is similar to match the next 2 points together.

In the example below (see Figure 3.13 - template-matching example) the two strokes match pretty well. But stroke 2 has one point more than stroke 1. Therefore the algorithm finds the best point to leave out. As you can see in the matrix of Figure 3.14, this is point 4 of stroke 2.

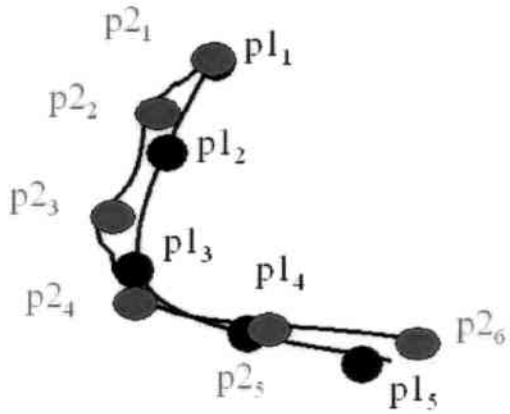


Figure 3.13 - template-matching example

S1/S2	$p2_1$	$p2_2$	$P2_3$	$p2_4$	$p2_5$	$p2_6$
$P1_1$		5				
$P1_2$	6		9			
$P1_3$		12			15	
$P1_4$			15	12		17
$P1_5$					15	

Figure 3.14 - matrix

3.7. Problems of the TemplateMatching approach

Even though template matching offers an easy and appropriate technique for gesture recognition, it has several drawbacks.

Firstly, the algorithm is sensitive to the writing direction, which makes it necessary to perform a variety of rotation and inversion operations. This may result in higher computational complexity compared to other techniques.

For the basic symbols used so far, it may have been easier to use off-line instead of on-line recognition. For further extensions, however, on-line recognition will be probably better, since it promises higher recognition rates.

Secondly, since template matching in its basic form does not support some kind of feature extraction, the technique is fragile to mixing up symbols that, for a human, seem pretty easy to distinguish.

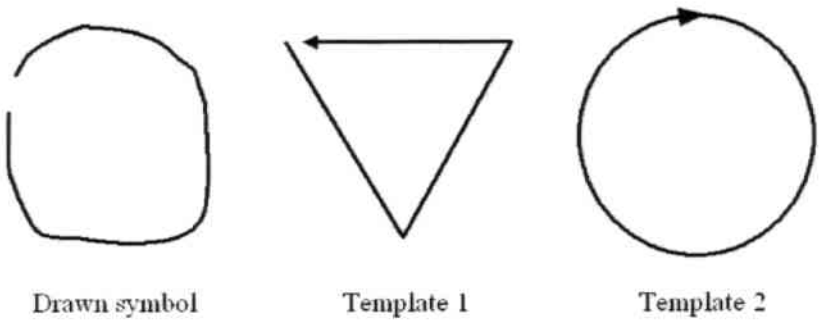


Figure 3.15 – mixing up gestures

As far as the development of the GRec system has evolved, no information about the orientation of the various components and modifiers are stored. This sometimes leads to misinterpretation of the symbol.

4. Implementation

GRec is developed using Microsoft Visual C++ 6.0 and uses numerous classes of the Microsoft Foundation Classes.

4.1.1. GRec's own user interface

The GRec system is an MDI (Multiple Document Interface) application. That means it is possible to load multiple documents into the main window.

A document represents a single open file. The document itself is not visible, but a specific view displays the document's data in a sub-window. This is accomplished by implementing two different classes at the code-level: a document class for storing the data and one or many view class/classes for displaying and editing a certain portion of the document's data.

In the GRec system itself there's only one view implemented. In the GRec data-collection tool, however, there is a further view that does not allow to edit the document's data.

The MFC class hierarchy contains classes - CDocument and CView - that make this structure easy to create, since they already implement the basic functionality.

The goal of the document class is to completely encapsulate the data for one open document. It holds the data for the document in a data structure in memory and knows how to load the data from the disk and save it to the disk. The view class uses and manipulates the data in the document based on user events. The view class is responsible for letting the user view the contents of the document.

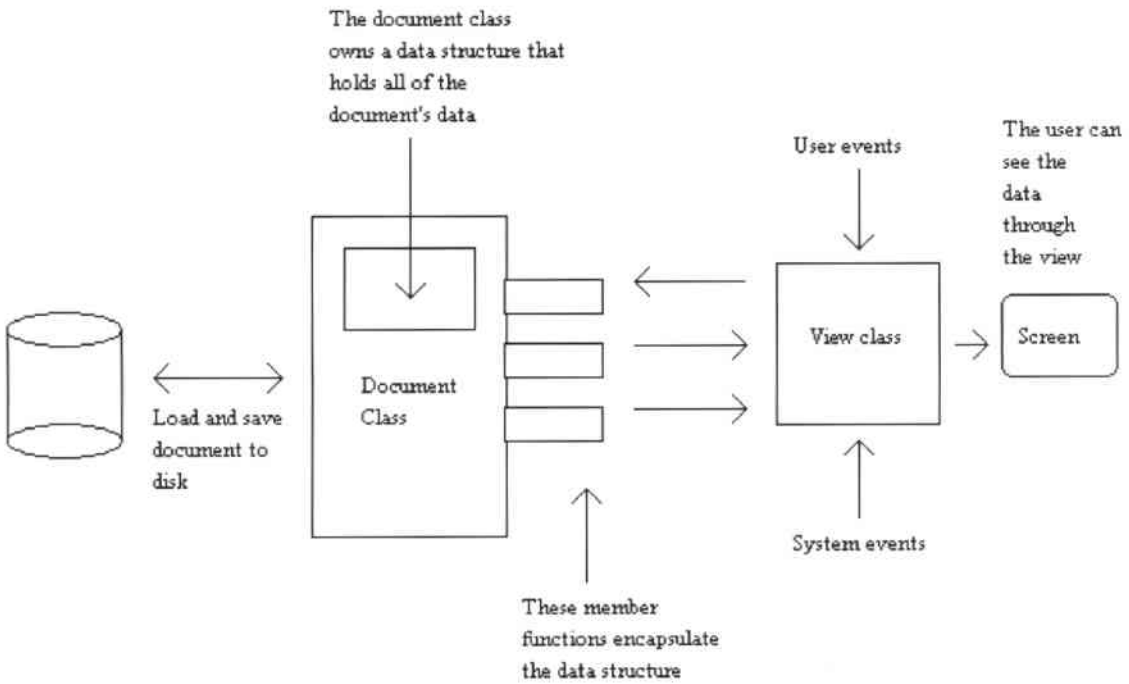


Figure 4.1 – document/view paradigm

The user-interface itself is pretty easy. You can create a new document via the toolbar or the menu and then draw a symbol into the opened sub-window.

Besides the simple recognition button that classifies the symbol, you can choose to perform some single steps of the recognition process (merging, rescaling, resampling, rotating).
Furthermore it is possible to scan all recorded data in the recorded-directory. This is very useful to test the recognition accuracy of the system.

The application also offers the functionality for opening existing documents or storing documents to files.

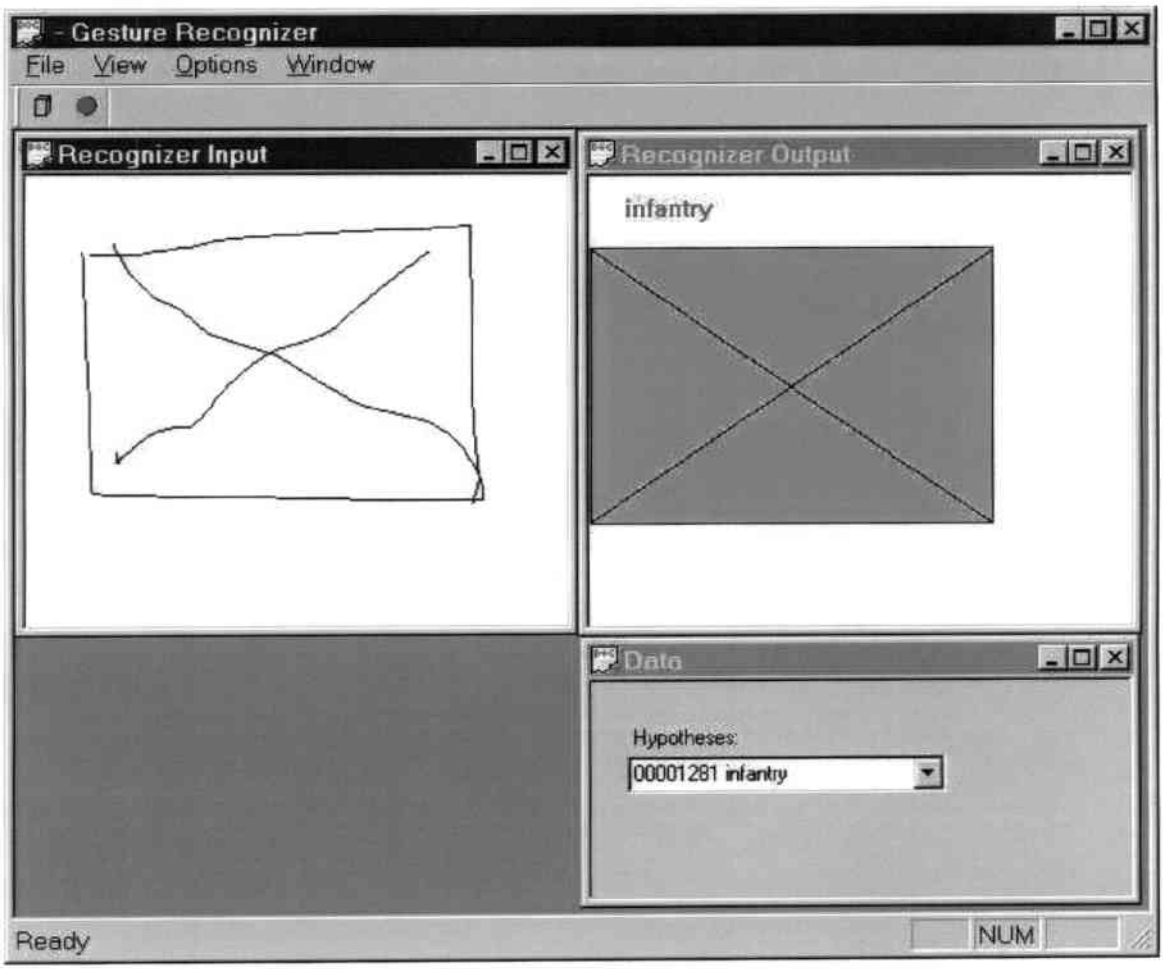


Figure 4.2 - Application Window

4.1.2. GRec as part of the CPoF system

The GRec system communicates with the CPoF system via a socket (CSocket class of the MFC). GRec is the server that waits on a specific port the data. The CPoF system operates as the client and records the drawn strokes. It converts them to a string of coordinates and then transmitted them over the socket to the GRec server. GRec converts the data into its format and tries to recognize the gesture. The result of the recognition is sent back to the CPoF system.

In order to separate the functionality - GRec has as server and as recognizer - the system is split up into two threads. A server thread, which takes care of the communication and a recognizer thread, which recognizes the gestures.

4.1.3. Class hierarchy and interface definition

GRec consists of 5 modules: application module, stroke module, display module, recognizing module and template module. These modules represent the basic function units of the system.

4.1.3.1. Application module

The application module takes care of the initialization, the user-interface and the event handling. Its code is mainly produced by the Appwizard of Microsoft Visual C++ and I will therefore not go into details.

4.1.3.2. Stroke module

The stroke module handles everything concerning storing and editing of strokes. Therefore it implements the basic pre-processing methods like rescaling, resampling, rotating etc. It consists of 3 classes.

The CStroke class represents an actual stroke and uses CList-template of the MFC to store the points that form the stroke. Since encapsulation is one of the main rules in object oriented programming this internal representation is only for private access. The class offers a variety of different public methods to get information about a stroke and edit it.

The CStrokes class is a container for all strokes that form a specific symbol. Similar to the CStroke class CStrokes holds a list of strokes using the CList-template. CStrokes implements all the methods, which can be applied to a set of strokes.

The CGestureMDIDoc class is basically used for storing/loading documents and the interaction with the application module. It implements the serializable-method for serialization.

4.1.3.3. Displaying Module

As the name suggests, this module takes care of displaying the data stored in the stroke module (more specifically in the document class).

The CPaint class is an abstract class, which implements the actual functionality of this module. On the one hand the class has methods for drawing the strokes on the screen. On the other hand it also provides also event-handling methods. These event-handling methods are used to add new points to a stroke or add a new stroke to the active document.
Since this class is abstract there is no possibility to create an instance of the class.

The CGestureMDIView class inherits from the CPaint class. Additional functionality is the interaction with the application module. This is e.g. necessary to retrieve the device context for drawing on the screen.

4.1.3.4. Recognizing module

The recognizing module is the core of the system. Principally it consists of only 2 classes.

The CStrokeMatcher class fetches the templates and tries to match the different strokes to their corresponding counterparts. For the assignment of the strokes it uses the CTemplateMatcher class.

The CTemplateMatcher class calculates the distance between two strokes. This is accomplished using the dynamic time warping algorithm described in chapter 3.6.3.

4.1.3.5. Template module

The template module is responsible for loading and storing the templates.

The CTemplateContainer class provides methods to load the components and template descriptions from the file system. The components are loaded via serialization. The templates itself however are described in a text file, which is loaded and parsed.

The CTemplate class represents an actual gesture template and stores the list of components of which it consists.

4.2. Conclusion

GRec's modules offer a good structuring of the system. They help to encapsulate the different parts and therefore make it easy to change the implementation of specific modules.

In future versions of the system, there will also be a separation between the user-interface and the actual recognition engine. This will enable it to integrate gesture recognition into other existing applications very easily

5. Evaluation

5.1. Data-collection

In order to get some data for testing and modifying the system, we had about 20 people, who volunteered to draw the symbols. For the recording we used a SmartBoard where the volunteers could draw with normal pen. However, some people had problems in pressing the pen strong enough on the board, which led in some cases to many dropouts.

The volunteers had never seen the symbols before. This resulted sometimes in confusion of how to draw the symbols. As a consequence some symbols where drawn in a really chaotic way.

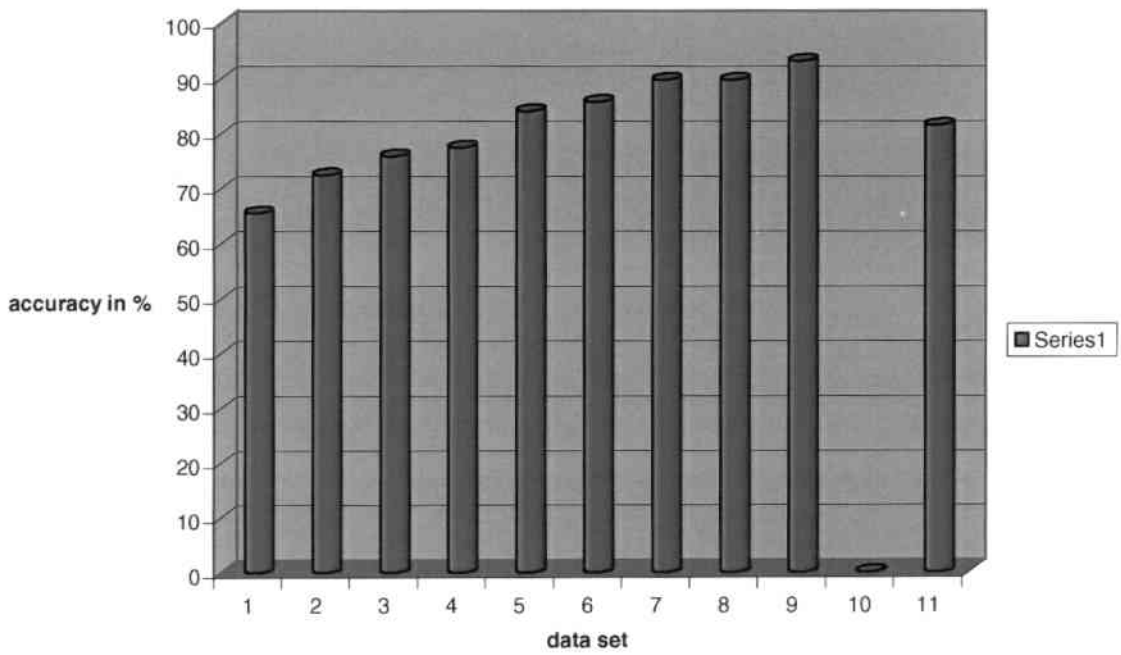
There are two data sets. One is based on a 29 gesture alphabet, the other on a 41 symbol alphabet. The first data set is a little bit older and since then templates have changed a little bit.

5.2. Performance

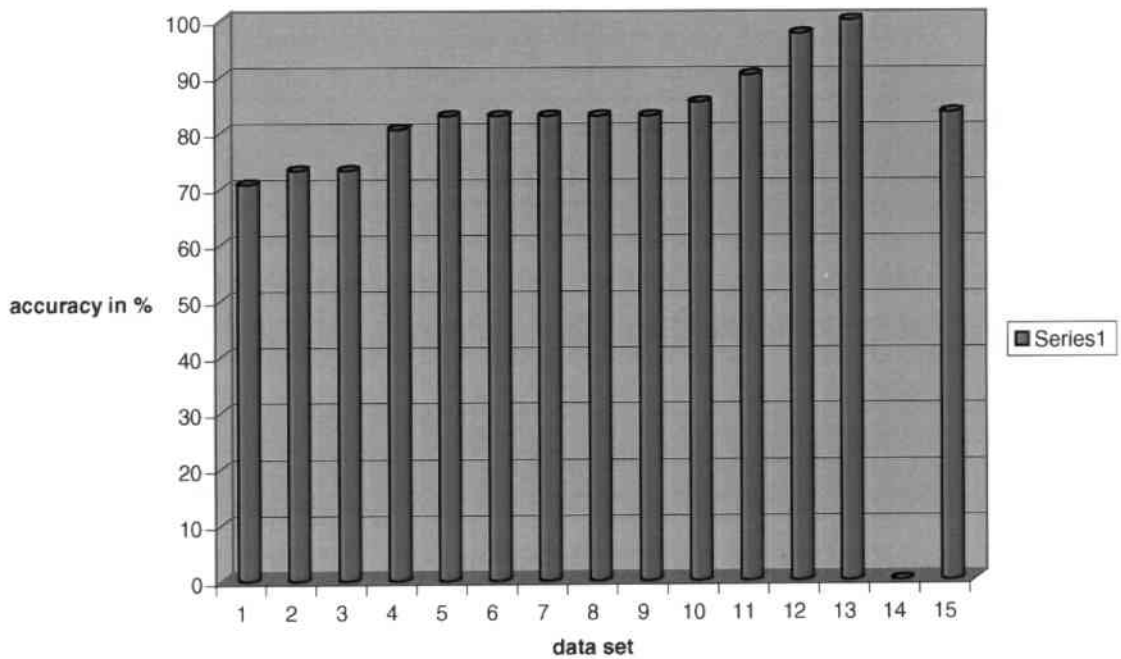
As you can see in the diagram below, the average recognition accuracy of the 29 gestures data set is 81% (column 15). The accuracies vary from 63% to 93%. An analysis of the wrongly recognized symbols has shown that most failures come either from drop outs of the smart board or ways of drawing gestures that are not covered by templates. Adding more templates should therefore increase the accuracy by some percent, but it may also lead to mixing up gestures more frequently.

The data set with 41 gestures had an accuracy of 83%, which was quite surprising for me, since I expected an increase of the error rate because of possible mix-ups. One of the reasons might be that I told the test persons to pay attention to SmartBoard dropouts.

Recognition Accuracy (29 symbols)



Recognition Accuracy (41 gestures)



5.3. Conclusion

For practical use the recognition performance of the system is quite acceptable. Particularly - as we have seen from the data-collection - if the gestures are drawn in a simple way. That means, for example, if the system does not get confused by lines that are drawn twice or just by a large number of strokes.

6. Summary

The whole GRec system works fairly well. It is able to recognize a large number of gestures and the recognition accuracy is quite stable even if new gestures are added.

We also have seen, that it is very simple to add new gestures. Most of the times it is enough just to add an entry into a text file and restart the program. You don't have to run a training algorithm, as it is common when learning algorithms are used.

Even though there are still some problems, the decision to use a template-matching approach was rather successful and there is still enough potential for further improvements.

7. Future Perspectives

As you have seen GRec works pretty well so far. But there's still a lot of work that can be done.

The first thing that should be implemented is a feature extraction. Some features, like cusp detection etc, are already implemented (CFeature class), but they are not taken into account by the recognizer so far. Feature extraction could boost the recognition accuracy considerably, since it would add a lot of new information, which could be used in addition to the template matching result. In this context it might be useful to use the convex hull for determining specific geometric features.

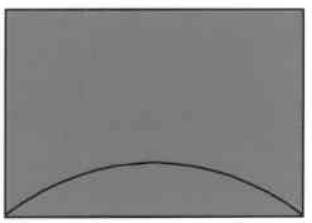
Another important issue would be to include some kind of orientation feature in the templates. That means a feature that indicates in which position a stroke resides in comparison to others. This would be especially useful to distinguish small modifiers, which are often drawn frowsy and therefore could be identified by their orientation.

To try different classifiers like neuronal networks or HMMs for the recognition would also be very interesting. Even though I do not expect much improvement from that, since most of the recognition errors are caused by the preprocessing routines. A way to tweak the existing template matching approach would be to try different metrics for the penalties. I already tried using squared-euclidean distance instead of Euclidean distance, but did not get better results. In addition to altering the recognition technique there is a lot of optimization in both performance and recognition issues that could be done.

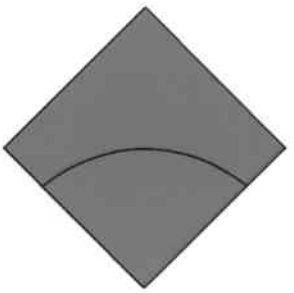
In my opinion the biggest accuracy improvement could be achieved by working on the stroke-merging algorithm. As already mentioned, most of the errors are caused by a wrongly merged strokes. Maybe a combination with an *offline* recognition technique would help to significantly boost the recognition performance.

8. Appendix

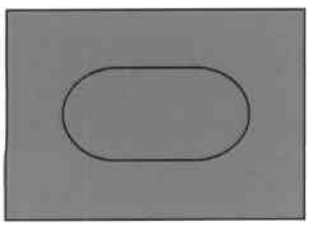
8.1. Appendix A (Gesture Set)



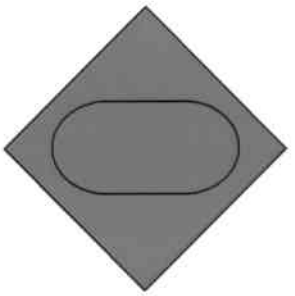
Air-defense



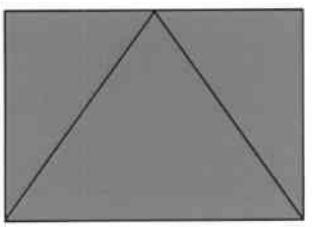
Air-defense (enemy)



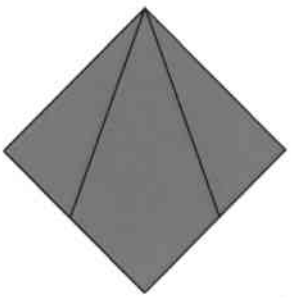
Armor



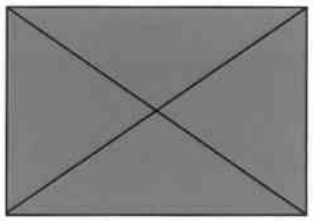
Armor (enemy)



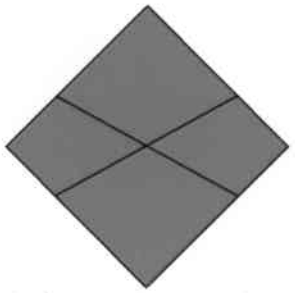
Anti-armor



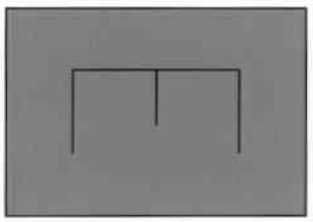
Anti-armor (enemy)



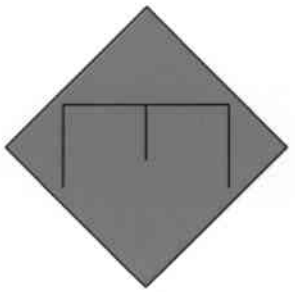
Infantry



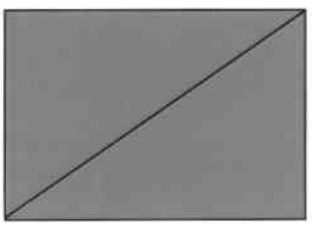
Infantry (enemy)



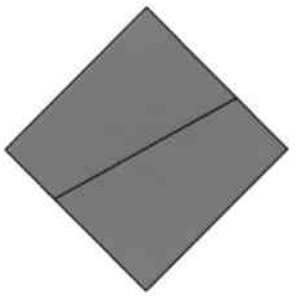
Engineer



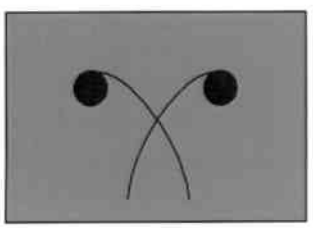
Engineer (enemy)



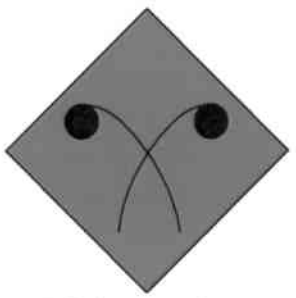
Recon



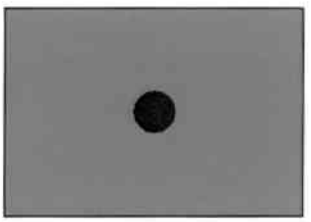
Recon (enemy)



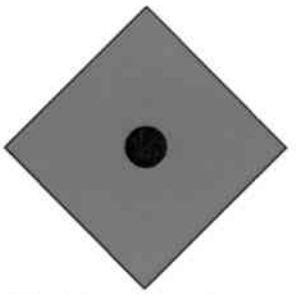
NBC



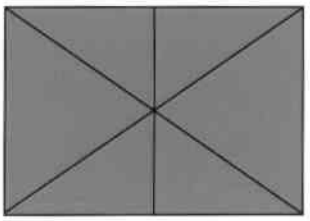
NBC (enemy)



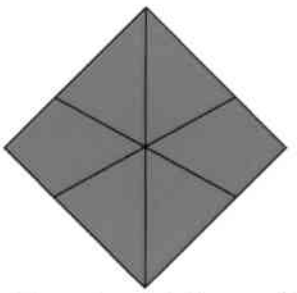
Field-artillery



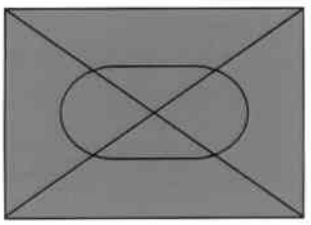
Field-artillery (enemy)



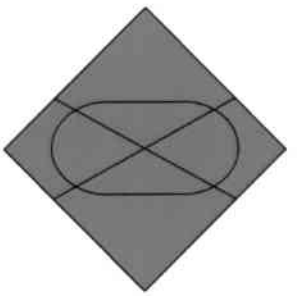
Motorized-infantry



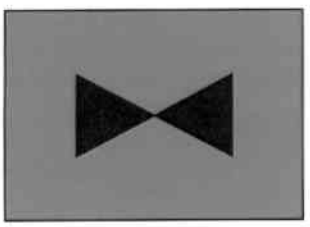
Motorized-infantry (enemy)



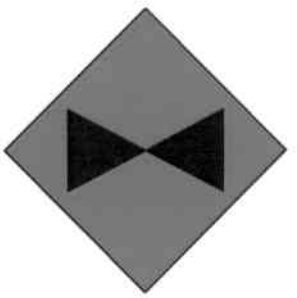
Mechanized-infantry



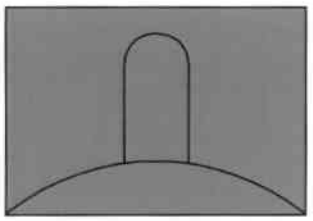
Mechanized-infantry (enemy)



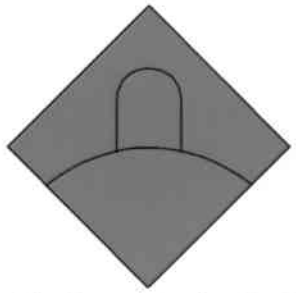
Fixed-wing aviation



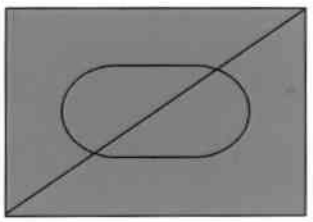
Fixed-wing aviation (enemy)



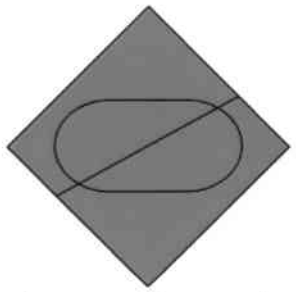
Surface-air-missile



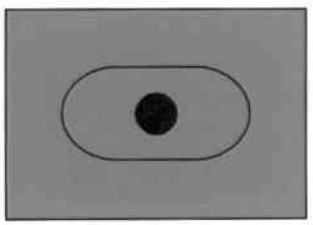
Surface-airmissile (enemy)



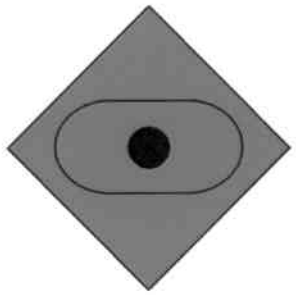
Armored-recon



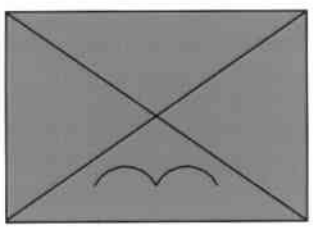
Aromored recon (enemy)



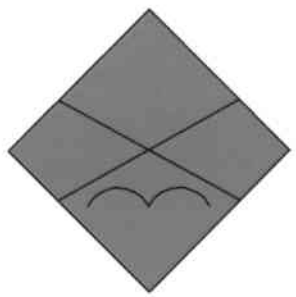
Self-propelled artillery



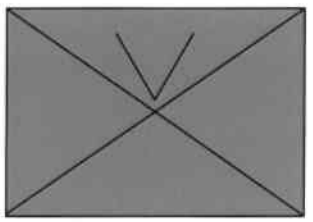
Self-propelled artillery (enemy)



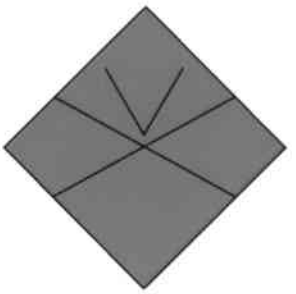
Airborne infantry



Airborne infantry (enemy)



Air-assault infantry



Air-assault infantry (enemy)

Fixed-wing UAV

Fixed-wing UAV (enemy)

8.2. Appendix B (Class Hierarchy)

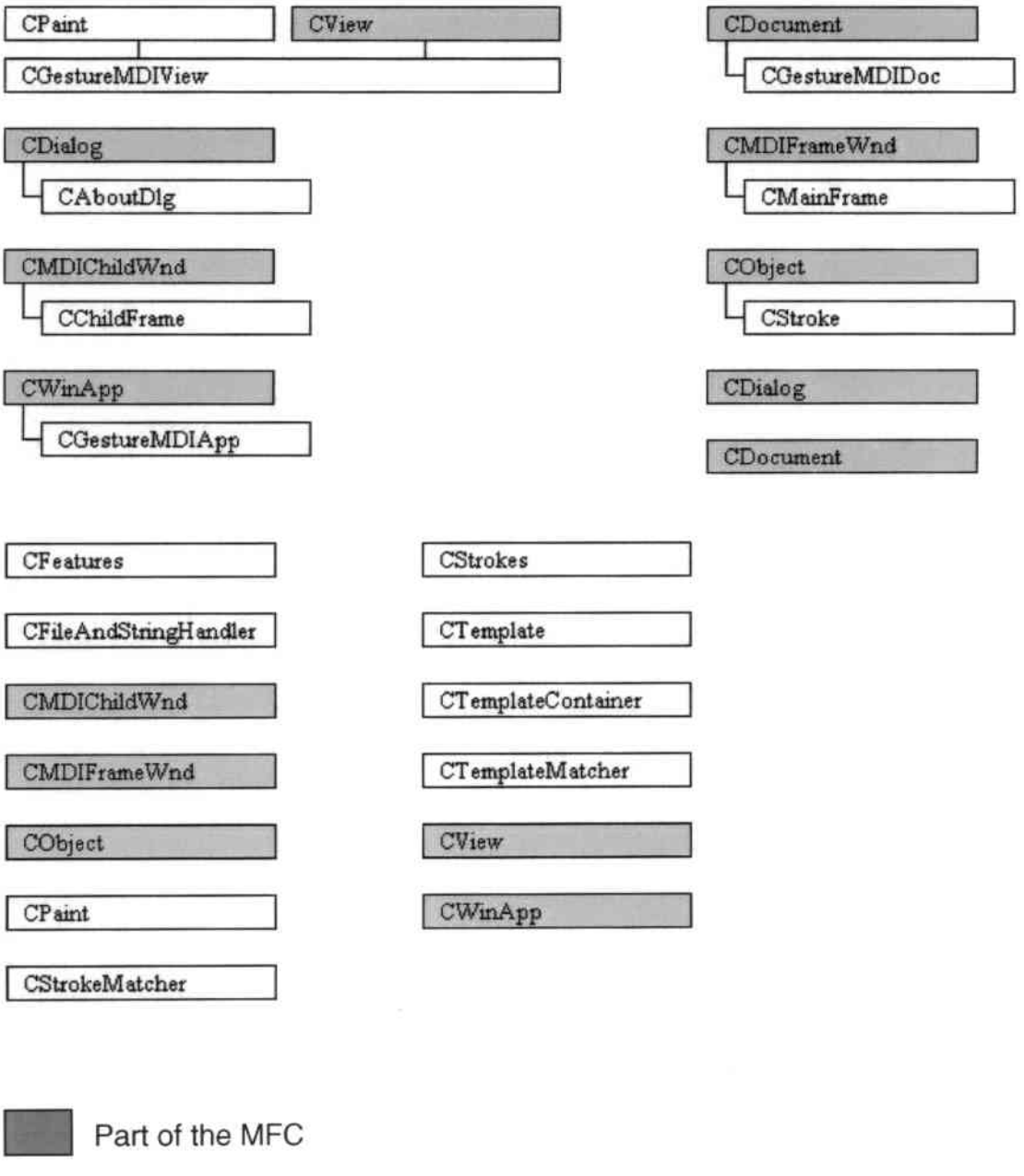


TABLE OF FIGURES

FIGURE 2.1 - BEZIER INTERPOLATION 14
FIGURE 2.2 - EXAMPLE OF A MARKOV MODEL 16
FIGURE 2.3 - NEURON 17
FIGURE 2.4 – PERCEPTRON 18
FIGURE 2.5 - MULTI-LAYER PERCEPTRON 18
FIGURE 3.1 - ONLINE VS. OFFLINE RECOGNITION 21
FIGURE 3.2 - CPOF MAP 22
FIGURE 3.3 – SYMBOLS 22
FIGURE 3.4 - OVERVIEW 23
FIGURE 3.5 - INPUT OF THE SYSTEM 24
FIGURE 3.6 – STROKE MERGING POSSIBILITIES 25
FIGURE 3.7 – RESCALING 26
FIGURE 3.8 – RESAMPLING 26
FIGURE 3.9 - ANGLES 27
FIGURE 3.10 – ROTATING COMPONENTS 28
FIGURE 3.11 – DIFFERENT COMPONENTS 28
FIGURE 3.12 – ASSIGN STROKES 30
FIGURE 3.13 - TEMPLATE-MATCHING EXAMPLE 32
FIGURE 3.14 - MATRIX 32
FIGURE 3.15 – MIXING UP GESTURES 33
FIGURE 4.1 – DOCUMENT/VIEW PARADIGM 35
FIGURE 4.2 - APPLICATION WINDOW 36

Bibliography

[1][6] "A Brief Overview of Gesture Recognition", http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/COHEN/gesture_overview.html

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

[3] "Gesture Recognition using statistical similarity", Center On Disabilities Virtual Reality Conference 1993

[4] "The Automatic Recognition of Gestures", Dean Harris Rubine PhD Thesis

[5] The Jove Project, http://www.hitl.washington.edu/scivw/JOVE/Articles/dsgbjsbb.txt

[7] "Comutation of Smoothing and Interpolating Natural Splines via Local Bases", T.Lyche and L.L. Schumaker

[8] "On-line Erkennung kursiver Handschrift bei grossen Vokabularen", PhD-Thesis Stefan Manke

[9] "A tutorial on hidden-Markov models and selected applications in speech recognition", L.R. Rabinier 1989

[10] "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", Hiroaki Sakoe and Seibi Chiba