Report

An approach on how to combine object recognition methods

Author(s):
Wong, Gladys

Publication Date:
1990-06

Permanent Link:
https://doi.org/10.3929/ethz-a-000545883

Rights / License:
In Copyright - Non-Commercial Use Permitted
Gladys Wong

An Approach on How to Combine Object Recognition Methods

June 1990
An Approach on How to Combine Object Recognition Methods

Gladys Wong

Abstract

Object recognition methods are usually suited to specific classes of features. Nonetheless, real world objects seldom contain only one single class of features. The premise of this paper is that it would be possible to recognize more objects if several differing recognition methods were used independently on identical scene objects and their results were combined; furthermore, a combination would help to improve the quality of the recognition.

First, the original scene image (which may contain overlapping objects) is preprocessed several times; every method uses its preprocessed image as input in its recognition process. Every method delivers then a quality value for every object to reflect the result of the match. In a subsequent step, each object is weighted by taking into account the given scene, the object, and the class of features peculiar to the method. This weight gives the method’s suitability to characterize the particular object. The weights and quality values are then combined to produce a final list of objects sorted according to the probability that they occur in the scene.

We illustrate our system combining two methods for object recognition. We then present some results and discuss the future direction of our research.
1 Introduction

A scene image may contain any number of model objects, at any possible location. The model objects may be partially overlapped. The task of object recognition is to identify these model objects in the scene image.

In model-based object recognition, a description of each model object is made.\(^1\) Subsequently, the scene is described and a match is carried out between the descriptions of the model objects and the scene description. The result of the match is a list of model objects sorted according to the probability of the model object being found in the scene. In addition, the parameters needed to transform a model object onto the scene are usually delivered. These parameters are used to verify the matches.

Fourier descriptors have been used extensively for object recognition (Kuhl, 1982). Originally, Fourier descriptors were for global descriptions of objects found isolated in the image. Recently, they have also been employed to recognize partially occluded objects (Gorman and Mitchell, 1988) and even for describing line drawings and open curves (Vieveen and Prasad, 1989). Fourier descriptors characterize closed contours well if the number of harmonics is high enough. Fourier descriptors are best suited to recognize elliptic objects.

Geometric features for object recognition are popular in object recognition systems. For instance, Perkins (1978), Rummel and Beutel (1984), and Nakamura and Nagao (1988) use geometric primitives as a basis for their object description. These methods are effective in recognizing objects whose boundary can be approximated well with lines and arcs. There is indeed a large class of such objects, for instance, industrial parts. An application of such methods is in a robotics environment. Chin and Dyer (1986) give an excellent review of the many model-based recognition systems based on arcs and lines in robot vision. Yet, problems

\(^1\) This description may be saved for later use to avoid having to describe all model objects every time a new scene is given.
may arise if the object contains jagged structures. Segmenting squiggles with short straight segments is not always reliable. If the resolution of the images is not high enough, the segmentation of a rotated object will be considerably different from the segmentation of the original object. Thus a match will fail.

A careful analysis of the current object recognition methods will often show that though fast and reliable, they do have shortcomings. We could have come up with yet another type of feature for a new type of object description. Alternatively, we could have elaborated on the construction of more complex features or more intricate matching algorithms. Instead, we decided that combining methods would bring us the desired high precision in object recognition. The individual methods to be combined would not need to be that refined since the combination would have a complementary effect on the individual methods. Combining methods is indeed holistic—yet nontrivial.

2 Presentation

Let \( M_i \) denote method \( i \) for \( i = 1, \ldots, m \). The domains used in Figure 2.1 are:

- \( B = \{b_1, b_2, \ldots, b_k, \ldots, b_l\} \) is the set of all blobs found in the scene image. A blob \( b_k \) is a group of connected pixels; this group of pixels can be distinguished from the background. A blob may correspond to one or more objects and will often be dark on a light background.\(^1\)

- \( P_i = \{p_{i1}, p_{i2}, \ldots, p_{ik}, \ldots, p_{il}\} \) is the set of the preprocessed blobs which result from the preprocessing of the blobs \( \{b_1, b_2, \ldots, b_k, \ldots, b_l\} \) for \( M_i \).

\(^1\)Ballard and Brown (1982) use the word similarly when writing about "blob finding" in binary images. We do not mean a blob in the sense of a primal sketch feature (Marr, 1982).
Each object recognition method $M_i$ uses its domain $P_i$ to solve the problem. Every $M_i$ is especially suited to describe the preprocessed blobs found in its $P_i$. In its recognition process, $M_i$ uses its own precompiled descriptions of the model objects (presumably stored in a data base). Thus the following computation occurs $m$ times, once for each $M_i$:

$$P_i \rightarrow O \times U_i.$$  \hfill (2.2)

The lifting\(^2\) part of the problem happens in two stages. First, the elements of $O$ delivered by each $M_i$ are weighted. The weights $w_{i1}, w_{i2}, \ldots, w_{in}$ depend on $P_i, O$, and on the method $M_i$ itself.

Secondly, the answers delivered by the various $M_i$'s (the results of the computations in (2.2)) are combined with the help of the weights to produce the probabilities $V$ of $O$ being in the scene. The final “answer” is a sorted list of triples $<b_k, o_j, v_j>$ sorted according to $v_j$.

The reader may be wondering about several occurrences of an object in the scene. This is not a problem because there is a transformation associated with each recognized $o_j$, namely, the transformation from $o_j$ back to $b_k$. As a result, even if $o_j$ appears several times in the scene, (i.e. if there are several blobs corresponding to the one $o_j$) the triple $<b_k, o_j, v_j>$ will still be distinguishable due to the implied transformation. It is this implied transformation that will even identify various instances of $o_j$ that can occur in one same blob $b_k$.

### 2.1 Preprocessing

We know that $M_i$ is best suited for recognizing only certain types of features in the objects, so we want to select out these primitives for $M_i$ even before $M_i$ begins to compute. The original set of scene blobs $B$ is preprocessed $m$ times to produce $m$ sets of preprocessed blobs $P_i$. The preprocessing is particular to each

\(^2\)We have borrowed some of the terminology, such as lifting, from the field of symbolic computing (Geddes, Labahn, and Czapor, 1990).
As mentioned above, the preprocessed blobs will be referred to as \( P_i = \{p_{i1}, p_{i2}, \ldots, p_{in}\} \). Note that having blob \( b_k \) map to an empty \( p_{ik} \) is allowed. This will happen when \( M_i \) cannot handle the type of features in \( b_k \). \( M_i \) will then treat only what it can process properly and will not receive those parts of the image that \( M_i \) would not have been able to use, or even worse, which would have caused \( M_i \) to deliver wrong results.

### 2.2 Object Recognition Methods

In this paper we are dealing with model-based recognition methods. Each \( M_i \) needs to be shown all possible object models at least once, so that \( M_i \) can use these as reference in the matching process.

Each method \( M_i \) matches the predefined model objects against the preprocessed blobs \( P_i \). The result of the match is a list of objects sorted according to their quality values: \( \{(o_1, u_{i1}), (o_2, u_{i2}), \ldots, (o_n, u_{in})\} \).

There are no restrictions on the kinds of methods to be used. Some methods will have a very elegant description whereas others focus on the matching algorithm. The more diverse the types of objects that the methods recognize, the fewer methods that are needed to identify all possible objects. Yet, we do not require that the participating methods be sophisticated since the list \( \{(o_1, u_{i1}), (o_2, u_{i2}), \ldots, (o_n, u_{in})\} \) delivered by a single \( M_i \) is later weighed and combined with the lists of the other methods to deliver one final list of sorted objects.

### 2.3 Calculation of Weights

The purpose of the weights is threefold:

1. To measure \( M_i \)'s ability to recognize an object.
2. To verify the recognition performed by \( M_i \) i.e., to verify the pair \( (o_j, u_{ij}) \); the value \( u_{ij} \) is increased for a good recognition by \( M_i \) and decreased \( u_{ij} \) for a poor one.
3. To normalize the values in \( U_i \) so that all the \( U_i \)'s have the same upper and lower bounds; this normalization is needed before the combination is done.

\( P_i \) is supposed to retain those parts of the original scene \( B \) that \( M_i \) can use appropriately in the recognition. The more similar that \( P_i \) is to \( B \), the higher the weights \( W_i \) because by saying that \( P_i \) is similar to \( B \), it is being connoted that \( M_i \) can process most of \( B \). It is tempting to compare \( P_i \) and \( B \) at a global level, namely, to assign one \( W_i \) to all the \( p_{ik} \)'s in \( P_i \); yet, doing this would imply that all there is only one type of feature in \( B \), which is usually not the case.

We have to calculate the weights locally, i.e., we have to calculate the weights exactly on those parts of the filtered image \( P_i \) that \( M_i \) used to deliver \( (o_j, u_{ij}) \). Unfortunately this is easier said than done. How can we measure how well a blob is being described when we do not even know what the object is? If we knew what \( o_j \) corresponded to the blob \( b_k \), then we would have solved the problem of object recognition altogether.

Before we describe the way we devised to compute weights, let us remember that in recognizing an \( o_j \), \( M_i \) also delivers at least one set of transformation parameters. These transformation parameters determine the rotation, translation, and scaling needed to "superimpose" \( o_j \) onto \( b_k \).

\[
\text{transformed}_{ij}(o_j) : \\
\text{return a new contour such that every point in } o_j \text{ is transformed using the parameters resulting from the match performed by } M_i \text{ between } o_j \text{ and } p_k.
\]

We use these parameters to define that local part of the scene image that needs to be considered when computing the weights. Furthermore, if the method delivers a wrong transformation for \( o_j \) (i.e., \( M_i \) does not recognize object \( o_j \)), then the local part defined by this erroneous, transformed \( o_j \) will not correspond to a \( p_{ik} \). The weight \( w_{ij} \) will then be set to 0.

\[\text{Here we say at least since there may be several instances (occurrences) of } o_j \text{ in the scene.}\]
Our scheme for computing weights is a function of $p_{ik}$, $o_j$, and of $M_i$. We have a procedure, called "corresponding" that returns the number of points in transformed$_{ijk}(o_j)$ that correspond (are close enough) to the points in $P_i$.

```
corresponding(transformed$_{ijk}(o_j)$, $P_i$):
    initialize(sum_of_corresponding_points);
    for (every objectPt in transformed$_{ijk}(o_j)$) do
        scenePt := point in $P_i$ closest to objectPt;
        if $|scenePt\cdot objectPt| \leq \delta$ then
            increment(sum_of_corresponding_points);
        end;
    end;
    return sum_of_corresponding_points;
end corresponding;
```

The threshold $\delta$ is set according to the tolerance wanted when measuring the distance from a point in transformed$_{ijk}(o_j)$ to a point in $P_i$.

In measuring the qualitative difference between a $p_{ik}$ and a $b_k$, we want to determine if important parts of $b_k$ were processed out. A first reaction might be to take the absolute difference between $p_{ik}$ and $b_k$ (e.g. the number of object points in $b_k$ minus the number of object points in $p_{ik}$). Sometimes this works well but not always. If $p_{ik}$ has retained the salient features of the object, then $w_{ij}$ should be high, even if a large part of $b_k$ was processed out.

The way that the qualitative difference between $P_i$ and $B$ is measured varies from method to method. What is central is that the correspondence between a transformed$_{ijk}(o_j)$ and a $p_{ik}$ be measured. Some factors that could be used to measure this correspondence are

- the number of object-points (e.g. the total length of the contours)
- the number of primitives
- the connectivity of the segments
- the curvature of the contour (e.g. the number of zero crossings)
- the saliency of an object

To calculate a weight $w_{ij}$, we take the object $o_j$ delivered by $M_i$ as a solution, transform this $o_j$, and check the correspondence between the transformed$_{ijk}(o_j)$ and $P_i$. We then adjust $w_{ij}$ to normalize $u_{ij}$.

If $o_j$ is not found completely in the scene $B$, the match between $o_j$ and $b_k$ cannot be a 100% match. $w_{ij}$ would reflect this occlusion. A weight can obtain its highest value only when

- the correspondence(transformed$_{ijk}(o_j)$, $P_i$) is equal to the correspondence($o_j$, $O$); this implies that $o_j$ was identified perfectly and that $o_j$ is found isolated and entirely in the scene
- and when correspondence(transformed$_{ijk}(o_j)$, $P_i$) is equal to correspondence(transformed$_{ijk}(o_j)$, $B$); this implies that $M_i$ is capable of dealing with all the features in $o_j$.

Since the range of the values of the members of $U_i$ for a method $M_i$ may not be the same as the range of the values of the members of $U_d$ for another method $M_d$, the $u_{ij}$'s need to be normalized. This normalization can be done by adjusting the weights to deal with this difference in ranges. Furthermore, there might be an a priori knowledge on a method's performance: it could happen that $M_i$ tends to deliver matches that are reliable or contrastingly, $M_d$ may tend to be sloppy. With the weights we can try to regulate the difference in the reliability of the methods.

---

4We use the word saliency in the sense of Turney, Mudge, and Volz (1985), namely, that saliency "measures the extent to which the boundary segment distinguishes the object to which it belongs from other objects which might be present".

5For instance, some methods are forced to deliver some match whereas other methods deliver only matches that have been thoroughly verified.
2.4 Combination

Once the weights for each object for each method have been computed, much of the work has been accomplished. The question raised now is "how should the objects o_j's with their values u_j's and corresponding weights w_j's be combined to deliver one final solution < b_k, o_j, v_j >?"

Bonissone (1987) reviews some of the current theories for combining. He partitions the approaches for combining into symbolic characterizations and numerical characterizations. He claims that for the most part non-numerical approaches are inadequate to represent and summarize measures of uncertainty. Yet, these symbolic approaches offer such possibilities as tracing from the sources how the final answer was obtained.

We determined numerical characterizations to be suitable for us as these seem to focus on the quality of the evidence. Henkind and Harrison (1988) review four uncertainty calculi. We consider these theories with respect to our application.

When recognizing objects, an \( M_i \) delivers pairs such \( (o_i, u_{ij}) \) implying that \( M_i \) recognized \( o_j \) with a certainty value \( u_{ij} \). In this section we use the abbreviation \( o_{ij} \) to mean \( M_i \) recognized object \( o_j \).

Bayesian Approach. Bayes's rule, based on the conditional probability, states that the posterior probability \( P(o_j|o_{1j}, o_{2j}, \ldots, o_{mj}) \) can be derived as a function of the conditional probabilities for the various \( o_{1j}, o_{2j}, \ldots, o_{mj} \), namely \( P(o_{1j}, o_{2j}, \ldots, o_{mj}|o_j) \), and the prior probability \( P(o_j) \). In this context, \( P(o_j|o_{1j}, o_{2j}, \ldots, o_{mj}) \) is the probability that the object \( o_j \) is found in the scene given that \( M_k \) recognized \( o_j \), \( M_2 \) recognized \( o_j \), \ldots and \( M_m \) recognize \( o_j \). The final value that we are looking for is \( P(o_j|o_{1j}, o_{2j}, \ldots, o_{mj}) \), namely \( v_j \). Using Bayes's rule,

\[
P(o_j|o_{1j}, o_{2j}, \ldots, o_{nj}) = \frac{P(o_{1j}, o_{2j}, \ldots, o_{mj}|o_j) \cdot P(o_j)}{\sum_{a=1}^{n} P(o_{1a}, o_{2a}, \ldots, o_{ma}|o_a) \cdot P(o_a)}.
\]

(2.3)

If we assume independence (cf. section 3.4), then

\[
P(o_{1j}, o_{2j}, \ldots, o_{mj}|o_j) = P(o_{1j}|o_j) \cdot P(o_{2j}|o_j) \cdot \ldots \cdot P(o_{mj}|o_j).
\]

Thus (2.3) becomes

\[
\frac{P(o_{1j}|o_j) \cdot P(o_{2j}|o_j) \cdot \ldots \cdot P(o_{mj}|o_j) \cdot P(o_j)}{\sum_{a=1}^{n} P(o_{1a}|o_a) \cdot P(o_{2a}|o_a) \cdot \ldots \cdot P(o_{ma}|o_a) \cdot P(o_a)}.
\]

(2.4)

\( P(o_{1j}|o_j) \) is the probability that given \( o_j, M_i \)'s answer which is \( o_{ij} \) be right. We let the weight \( w_{ij} \) be equal to \( P(o_{ij}|o_j) \) because the weight qualifies an \( o_{ij} \). Substituting \( w_{ij} \) into (2.4) yields

\[
P(o_{1j}, o_{2j}, \ldots, o_{mj}) = \frac{w_{1j} \cdot w_{2j} \cdot \ldots \cdot w_{mj} \cdot P(o_j)}{\sum_{a=1}^{n} w_{1a} \cdot w_{2a} \cdot \ldots \cdot w_{ma} \cdot P(o_a)}.
\]

(2.5)

To resolve (2.5) we still need \( P(o_j) \). In theory, \( P(o_j) \) is the probability that a single object \( o_j \) will appear in the scene. Although we could have assigned this probability to be the reciprocal of the number of objects, we noticed that this value would have cancelled out in (2.5) because every object would be just as likely to appear in the scene. Thus, we let \( P(o_j) \) be equal to the product \( u_{1j} \cdot u_{2j} \cdot \ldots \cdot u_{mj} \). The reasoning behind is that the larger the individual \( u_{ij} \), the greater \( P(o_j) \) will be. Since \( P(o_j|o_{1j}, o_{2j}, \ldots, o_{mj}) \) is equal to \( v_j \), the final value associated with an object \( o_j \) is

\[
v_j = \frac{w_{1j} \cdot w_{2j} \cdot \ldots \cdot w_{mj} \cdot u_{1j} \cdot u_{2j} \cdot \ldots \cdot u_{mj}}{\sum_{a=1}^{n} w_{1a} \cdot w_{2a} \cdot \ldots \cdot w_{ma} \cdot u_{1a} \cdot u_{2a} \cdot \ldots \cdot u_{ma}}.
\]

(2.6)

\*Up to now, we have assumed that there are \( n \) objects; nevertheless, we need to consider that every \( o_j \) may appear more than once in a scene. Thus, \( n \) will refer to sum of all instances of all objects.
Dempster-Shafer Theory of Evidence. A feature of the Dempster-Shafer Theory is being able to associate a BPA (basic probability assignment) with any subset of the universal set. Applying this theory to our case implies that each subset of all objects \( O \) could have a BPA. Yet, we cannot assign a BPA to a subset of \( O \) since \( M_i \) assigns recognition values to single \( o_j \)'s, not to groups of \( o_j \)'s.

We could try to find subsets of the methods delivering matches about a distinctive object, but then again, a weight \( w_{ij} \) is local to \( B \). We need to conclude again that we are dealing with singletons and not with subsets with more than just one element.

In short, we cannot make full use of the Dempster-Shafer theory. If we were to apply the theory to the singletons, then we would be reducing the Dempster-Shafer theory to the Bayesian approach with independence, as pointed out by Henkind and Harrison (1988).

Fuzzy-Set Theory. The Fuzzy-set theory of Zadeh (1965) uses characteristic functions to determine the membership of objects to a set. Let \( f_i \) be the characteristic function that corresponds to the recognition of \( M_i \). \( f_i(b_k, o_j) = u_{ij} \) implies that \( M_i \) recognizes object \( o_j \) from blob \( b_k \) with a certainty of \( u_{ij} \). In other words, \( f_i = \text{probability}(o_j \in O) \).

The characteristic function \( g_i(b_k, o_j) = w_{ij} \) determines how well an object \( o_j \) can be described by an \( M_i \) given \( b_k \). There are \( m \) such characteristic functions \( g_i \), one for each method given. We say that \( g_i \) determines the probability that object \( o_j \) belongs to the set of objects that method \( M_i \) can describe properly given this particular scene.

In a first step, we need to combine the results of \( f_i \) with the knowledge delivered by \( g_i \). In other words, though method \( M_i \) recognizes object \( o_j \) with a recognition value of \( f_i(b_k, o_j) \), \( M_i \)'s opinion is only worth \( g_i(b_k, o_j) \). Thus \( f_i \) is weighted with \( g_i \):

\[
f_i(b_k, o_j) \cdot g_i(b_k, o_j)
\]

and this for every method \( M_i \) and blob \( b_k \).

In a second step, we combine the various products from (2.7) using the fundamental operators for fuzzy sets. The most fundamental operators for fuzzy sets are set intersection:

\[
f_r(b_k, o_j) \cap f_s(b_k, o_j) = \min(f_r(b_k, o_j), f_s(b_k, o_j)) \tag{2.8}
\]

and set union:

\[
f_r(b_k, o_j) \cup f_s(b_k, o_j) = \max(f_r(b_k, o_j), f_s(b_k, o_j)) \tag{2.9}
\]

The identities (2.8) and (2.9) are commutative and associative. Furthermore, there is a closure such that if \( f_r(b_k, o_j) \in V \) and \( f_r(b_k, o_j) \in V \) then \( (f_r(b_k, o_j) \cap f_s(b_k, o_j)) \in V \) and \( (f_r(b_k, o_j) \cup f_s(b_k, o_j)) \in V \).

Union (2.9) was chosen as the operator for combining the products of (2.7) because good answers are to be emphasized and bad ones are to be ignored:

\[
\bigcup_{i=1}^{m} [f_i(b_k, o_j) \cdot g_i(b_k, o_j)]
\]

Thus we determine the final value \( v_j \) (value to be used when sorting the final list of objects) of \( o_j \) to be

\[
v_j = \max(u_{ij} \cdot w_{ij}, u_{2j} \cdot w_{2j}, \ldots, u_{mj} \cdot w_{mj})
\]

Confirmation Theory. The Confirmation Theory or Certainty Factor approach of MYCIN and EMYCIN has been popular for expert systems. Buchanan and Shortliffe (1984) give a detailed discussion of the advantages and disadvantages of this theory. Confirmation Theory has been criticized for being heuristic, yet the theory is computationally simple and apparently lends itself to a good estimation of parameters by the experts (Adams, 1976). The MYCIN calculus is based on a certainty factor \( CF \) that depends on a measure of belief \( MB \) and a measure of disbelief \( MD \). Later the EMYCIN calculus redefined the MYCIN \( CF \) to overcome some of the difficulties presented to be

\[
CF = \frac{MB - MD}{1 - \min(MB, MD)} \tag{2.10}
\]
MB[h, s] is the belief in the hypothesis h that results from the observation s. In our application, we only deal with measures of belief. M_i recognizes object o_j in the scene and assigns it a measure of belief u_{ij}. M_i does not express disbelief about other objects so that substituting MD[h, s] = 0 into (2.10) yields

\[ CF[o_j, o_{ij}] = MB[o_j, o_{ij}] = u_{ij}. \]

In the general theory, there is belief and disbelief, so that the CF is defined on the interval \([-1, 1]\). The corresponding combining function is divided then into cases depending on whether the CF's are negative or positive. When both \(CF[h, s_i]\) and \(CF[h, s_j]\) are greater than 0, the following rule applies:

\[ CF[h, s_i & s_j] = CF[h, s_i] + CF[h, s_j] \cdot (1 - CF[h, s_i]) \quad (2.11) \]

We know that in our case the CF will be positive or 0, so in using (2.11) we obtain

\[ CF[o_j, o_{ij} & o_{ij}] = CF[o_j, o_{ij}] + CF[o_j, o_{ij}] - CF[o_j, o_{ij}] \cdot CF[o_j, o_{ij}] \quad (2.12) \]

For three methods, the certainty value is

\[ CF[o_j, o_{ij} & o_{ij} & o_{ij}] = CF[o_j, o_{ij}] + CF[o_j, o_{ij}] + CF[o_j, o_{ij}] - CF[o_j, o_{ij}] \cdot CF[o_j, o_{ij}] - CF[o_j, o_{ij}] \cdot CF[o_j, o_{ij}] - CF[o_j, o_{ij}] \cdot CF[o_j, o_{ij}] \cdot CF[o_j, o_{ij}] \]

and so on.

Buchanan and Shortliffe (1984) evaluate the EMYCIN calculus and they mention on page 213 that "another limitation for some problems is the rapidity with which CF's converge on the asymptote 1". They suggest using "damping factors" which though not implemented, would solve the problem. We use weights to qualify the various CF's but keep the combination rule as in (2.12). Thus, the final recognition value \(v_j\) for object \(o_j\) given \(M_1\) and \(M_2\) is

\[ v_j = u_{1j} \cdot w_{1j} + u_{2j} \cdot w_{2j} - (u_{1j} \cdot w_{1j}) \cdot (u_{2j} \cdot w_{2j}). \]

### 3 Implementation

We describe Mingle as an implementation of the method presented in the previous chapter and report briefly on our results. At the present time Mingle combines two methods; however, an extension in the number of methods is both possible and planned for in the near feature. The methods that we adopted and implemented are taken from Eichenberger and Wong (1989) as they seem contrasting in nature and are for the recognition of objects even when these are overlapping; we shall refer to them as method G and method S.
3.1 Methods

Currently Mingle combines methods that are model-based and contour-based. A grey-scale image is taken with a camera. A binary image is produced by thresholding the grey-scale image. The contours of the objects are then extracted. There are many contour following algorithms (e.g., Duda and Hart, 1973) which involve tracing out the boundary between an object and its background.

3.1.1 Method G: Geometric Method G is typical of the many systems available based on geometric features (Chin and Dyer, 1986). The primitives arcs, lines, and circles, are used by method G to build the more complex characteristic features e.g., line-line-corners, line-arc-corners, arc-line-corners, and arc-arc corners. Each model feature is stored with references to similar features (of the same or of some other model objects). In the matching algorithm, a model feature that corresponds to a scene feature is used to create the hypothesis that \( o_j \) appears in the scene. This hypothesis is then verified. There is an intelligent search to determine the model features which should be considered when forming and testing the hypotheses. Though the primitives may be lines and arcs, the matching algorithms of such methods as Method G can vary drastically, e.g., Grimson’s technique (1989) on the recognition of curved objects.

3.1.2 Method S: Statistical and Frequency Analysis Method S is more statistical in nature and is based on frequency analysis using filters with Gabor functions. Given the contour of an object, the points along the contour that reflect a high change in curvatures are selected as “feature points”. At these feature points, feature vectors are computed by convolving the \( \psi - s \) curve of the contour with Gabor functions; by \( \psi - s \) curve here it is meant the angle \( \psi \) made between a fixed line and a tangent to the contour versus the arc length \( s \) of the contour traversed (Ballard and Brown, 1982). The features of the model objects are compared with those of the scene and a Hough Clustering is performed to find the best matches.

Other methods which are based on frequency analytical features would be Hong and Wolfson’s (1988) model-based matching method using footprints and Gorman and Mitchell’s (1988) Fourier descriptors using dynamic programming.

Method S has difficulty in describing objects with a constant curvature because the setting of the feature points depends on a changing curvature. Furthermore, if the \( \psi - s \) curve does not change, the feature vectors computed along the contour are no longer unique.

3.1.3 Method C: For Slowly Changing Curvature Method C needs to be a method that recognizes objects whose curvature change is gradual. A typical object of this class would be an ellipse. We are in the process of implementing Method C. An appropriate method of this type uses a description based on the medial axis transformation of an object as in Ogniewicz, Kübler, Klein, and Kienholz (1989). Another possibility could be Brady and Asada’s (1984) method that uses smoothed local symmetries.

3.2 Preprocessing

The preprocessing of a scene image occurs independently of the computations performed by \( M_i \).

3.2.1 For Method G The preprocessing for G consists of extracting lines and arcs from the images. We use a Hough Transformation (Duda and Hart, 1972) to extract the line segments and then a Hough Transformation to extract the arcs. The results from both transformations are superimposed to produce preprocessed blobs consisting exclusively of lines and arcs. We do not discuss the details of the Hough transformations here as a Hough Transformation is a technique that is commonly used; see, for instance, Ballard and Brown (1982) or Gerig (1987).

In Figure 3.2 the first column contains two original scenes. The
3.2.2 For Method S

In analyzing method S, we came to the conclusion that method S is appropriate for describing contours whose curvature is not constant. The preprocessing for S is supposed to leave winding, zigzag structures in the image. This preprocessing is like a low pass filter on the curvature.

The inspiration for the preprocessing for S came from the simple, standard procedure for segmenting a curve into a series of lines.

Duda and Hart (1973) call it the method of “iterative end-point fit”. An initial line is fit between two points. The distances between every point and this line are computed. When these distances are smaller than a predetermined threshold, then the process is finished. Otherwise, the point that is furthest away from the drawn line is chosen as a break point of the polygon and the original line is split in two.

In the preprocessing for S, we fix the length of the interval to be $n$ pixels. A line is drawn between the first and the $n$th pixel. The distances between the contour points and the line are then computed. The variance of these distances is estimated and if the variance exceeds a certain threshold, then this interval of the contour is kept; we say that this interval is squiggly.

The variance is used instead of the absolute distance between a contour point and the line to minimize the effects of “wild” points. The variance of the distances contains more information about the whole contour than a maximum distance point approach.

It can happen that the first and the $n$th points that determine the interval lie at an awkward place along the contour (for instance, at the corners). In such a case, the variance of the interval will be too low to exceed the threshold and the interval will be discarded unfairly. We seek to counterbalance this effect by doing a second pass on these intervals and choosing the points $\frac{n}{2}$ and $\frac{3n}{2}$ as the endpoints of the line. As before, we calculate the variance of the distances between the contour points and the line to check if this time the interval can be kept as squiggly or whether, the interval is indeed too flat or too round and should indeed be discarded.

We obtained satisfactory results using this preprocessing. For convenience we close the small gaps between the intervals of the contours in the preprocessing of S.

3.3 Calculation of Weights

3.3.1 For Method G

Method G delivers $o_j$ as the matching object for the preprocessed blob $p_{Gk}$. Using the function “corres-
ponding” of section 2.3, we calculate the weight for $o_j$ given $p_{Gk}$ as

$$w_{Gj} = \frac{\text{corresponding}(\text{transformed}_{Gj}(o_j), P_G)}{\text{length}(o_j)}$$

where “length” delivers the total number of points in $o_j$.

Let us consider the boundary cases. If there are no corresponding points between the transformed $\text{ajk}(o_j)$ and any $p_{Gk}$, then $w_{Gj} = 0$. On the other hand, if the number of points in the original contour $o_j$ is equal to the corresponding $\text{ajk}(o_j), P_c$ (and there is no overlapping) then

$$w_{Gj} = \frac{\text{number of corresponding points}}{\text{number of points in } o_j} = 1.$$

The more that method $G$ can approximate a contour with lines and arcs, the higher the weight $w_{Gj}$ should be. When calculating the weights for method $G$, we take the absolute difference between the number of pixels in the transformed $\text{ajk}(o_j)$ and the points of $P_i$.

3.3.2 For Method $S$ To calculate $w_{Sj}$, we transform the $o_j$ onto the preprocessed blob $p_{Sk}$, and we measure the longest uninterrupted contour segment. We scale the length of this segment and set it to be $w_{Sj}$. If $p_{Sk}$ has a long, uninterrupted contour segment, then there were several intervals that were not processed out because they were squiggly enough. These intervals were next to each other (to be able to compose the long contour segment) and this is significant. Thus

$$w_{Sj} = \text{adjust}(\max(\text{corresponding}(\text{transformed}_{Sj}(o_j), \text{segments of } P_i)))$$

where “adjust” is used so as to have an upper value for the $w_{Sj}$. Often the weights are in the range $[0, 1]$.

Suppose that $b_k$ is a large square. Here $p_{Sk}$ will contain only four isolated contour intervals, corresponding to the four corners of the square. When calculating the weight for this $p_{Sk}$, there will not be any long continuous contour segments and thus the weight $w_{Sj}$ for this square will be low – as it should be. In contrast, $w_{Gj}$ would assign a high weight to this large square because $p_{Gk}$ would be almost identical to $b_k$. Actually, what will probably happen is that method $S$ will not be able to match the square to any $p_{Sk}$. Thus $\text{corresponding}(\text{transformed}_{Sj}(o_j), P_S)$ and $w_{Sj}$ will also be 0.

3.4 Combination

Fuzzy-Set Theory. Fuzzy-set theory is easy to implement and is fast. Our first approach for combination of methods used Fuzzy-set logic. It worked up to the point to where it could have worked. We were encouraged by the good results but we realized that taking the maximum of two amounts is not additive. Is it right that $o_j$ should have a high value just because one single method with its weight delivered a high value? What about objects that have been suggested by more than one method but in none of the cases were assigned a high value? Remember that a single object $o_j$ could well contain several types of features or, alternatively, $o_j$ may have only one type of feature so that one single $M_i$ can describe the object well. Is it justifiable to take the best suggestion and ignore the rest?

Fuzzy-set theory is appropriate if it is known that the events are independent of each other. But it is questionable whether this is the case here. It is true that every $M_i$ works independently of each other; yet, each $M_i$ is trying to identify objects found in the same original scene.

Bayesian Approach. The following equation (copied from (2.6))

$$v_j = \frac{w_{1j} \cdot w_{2j} \cdot \ldots \cdot w_{mj} \cdot u_{1j} \cdot u_{2j} \cdot \ldots \cdot u_{mj}}{\sum_{a=1}^{n} w_{1a} \cdot w_{2a} \cdot \ldots \cdot w_{ma} \cdot u_{1a} \cdot u_{2a} \cdot \ldots \cdot u_{ma}}$$

(3.1)

gives the way to compute $v_j$ given the various $o_j$'s. Yet, there is a problem. If either $w_{ia} = 0$ or $u_{ia} = 0$, then the numerator will be 0. To resolve this problem, we introduce a minimum value which is assigned to all the $w_{ij}$ and $u_{ij}$'s which would be 0. It is important that the minimum value be smaller than the smallest $w_{ij}$ delivered by any method. We obtain small absolute values
for \( v_j \)'s but this is unimportant since the significant part is the ordering of the objects with respect to each other.

When comparing with Fuzzy-set logic, the Bayesian approach gives a higher preference to those \( o_j \)'s which were suggested more than once. Thus, even if \( o_j \) is assigned a high \( u_{ij} \) and its \( w_{ij} \) is also high, if there is an \( o_r \) which is suggested more than once, \( o_r \) will be preferred over \( o_j \).

Here we assume independence between the methods. The assumption that \( o_{rj} \) and \( o_{sj} \) are independent is restrictive. It is debatable whether this is a major difficulty or not. Lowe (1985) points out that a violation of this assumption will affect "all probability estimates by the same factor".

The results of a Bayesian combination are highly influenced by the poor matches rather than by the better ones. If an object \( o_j \) is not assigned a high value by all the methods, the overall \( v_j \) will be low. It is as though one were to take the minimum (or intersection as in Fuzzy-set logic). On the one hand this delivers safe results, on the other hand it impedes the individual method from excelling. It is only when all the methods deliver high \( u_{ij} \)'s that the resulting \( v_j \)'s are good.

**Confirmation Theory.** Confirmation Theory does not assume that every method delivers its pairs \((o_j, u_{ij})\) independently of each other. Here all values from the methods are added but their intersection is subtracted. The results seem to reflect our intuition.

### 3.5 Adding and Omitting Methods

We begin this section by describing what is needed to add a new method to the current implementation of *Mingle*. Since everything has been modularized, the addition of methods is simple. Figure 3.1 shows the system diagram. Based on this diagram, the new components that would have to be developed for a new method to be added are a branch \( i \) consisting of

- the preprocessing for \( M_i \)
- the method \( M_i \) itself

- the weight calculation for \( M_i \)

\( M_i \) must recognize 2-D objects that could be partially overlapping. \( M_i \) need qualify each model object \( o_j \) with a value \( u_{ij} \) that determines how likely it is for \( o_j \) to appear in the scene. Furthermore, \( M_i \) must provide a transformation that will superimpose \( o_j \) onto \( b_k \).

The method \( M_i \) needs to be analyzed seriously and honestly\(^1\): what classes of features in an object can \( M_i \) characterize? What kind of features does \( M_i \) have difficulty in processing: be it because this part of the object leads to erroneous recognition or because this part of the object does not add anything useful to \( M_i \).

Having admitted that \( M_i \) cannot possibly handle all types of objects, the preprocessing should preprocess images to leave only that which will be useful for \( M_i \). The weight calculation could be potentially difficult to write. In section 2.3, we present some factors that could be considered when calculating these weights. What is important is the difference between the preprocessed blob \( p_{ik} \) and the transformed \( o_j \). We highly recommend computing the weights locally using the transformation parameters for the match.

Once the preprocessing, \( M_i \), and weight calculation have been added, the combination part remains basically the same. The number of methods involved is not significant.

Generally speaking, a method is added to handle new, distinct classes of features. Adding this new method could possibly introduce wrong results. Let us consider the worst case, namely that the new method \( M_j \) does introduce a poor match \((o_j, u_{jf})\). Using the scheme of section 2.3 we verify the \( o_j \) delivered and set the weight \( w_{fj} \) to be low. Now, if the combination scheme being used is Fuzzy-set logic with set union, then the best value of any \( v_j = u_{jf} \cdot w_{fj} \) will be taken and the new and poor result \( u_{jf} \cdot w_{fj} \) will be ignored. In such a case, the addition is monotonic. On the other hand, the Bayesian approach would produce in general

---

\(^1\) Here we mean "honestly" in that the shortcomings of the method are admitted; it is not blindly said that the method can recognize all types of objects.
“safer” results emphasizing the lower valued pairs. In this case, the addition of a poor match does affect the final ordering of the objects.

What we cannot measure appropriately when adding a new method is the redundancy in computation due of the addition. Though ideal, it is difficult to require that the methods compute on classes of features that are mutually exclusive.

In direct contrast to adding a new method, it is also possible to omit existing methods. A user could know, perhaps, ahead of time what type of objects he would want to recognize. He would then determine that it would suffice to use one single method $M_i$ (with its corresponding preprocessing and weight calculation scheme). In such a case, it would trivial to not call the other methods. We have designed Mingle so that methods can be added or can be omitted without any difficulty.

3.6 Results

Our program Mingle is structured as in the diagram of Figure 3.1. We ran Mingle on a Sun Workstation running under Unix and obtained the following results.

For our first set of examples we chose a scene image which could not have been recognized alone by neither method G nor method S. The given scene is drawn as a black contour and the dotted contour on top is the object recognized by Mingle. The combination scheme used here is Confirmation Theory. The first five objects recognized are instances of a 5 Rappen coin, $m_4$, and a 20 Rappen coin, $m_5$. These coins were recognized exclusively by method G. $m_3$ is then recognized whose end corner is identical to that of $m_2$. We purposely made the corners identical in an attempt to fool the methods. The keys $m_0$ and $m_1$ are correctly identified by method S even though they are very similar and heavily occluded. The last object which is correctly found is $m_3$. Note that 5 of the 6 corners of $m_3$ are covered and in spite of this the identification is correct. The reason that $m_3$ has a low value, say as opposed to $m_4$, is that $m_3$ is heavily occluded. We left some of the wrong matches at the end of the list—these have accordingly small values so that they may be ignored.

---

2Sun Workstation is a registered trademark of Sun Microsystems, Incorporated
3Unix is a registered trademark of AT&T
In Figures 3.3 and 3.4 we illustrate another scene. Here $o_1$ represents the object with the highest quality value, $o_2$ the object with the next highest quality value, and so on. The Bayesian Approach was used to combine in Figures 3.3 and Confirmation Theory was used in Figure 3.4. Yet, $o_1$ is the same in both figures; that is, the object that was best recognized is $o_1$. Since both method $G$ and method $S$ identified the object, since $o_1$ is not occluded, and since $o_1$ has features appropriate for method $G$ and method $S$, the resulting quality value for $o_1$ is high.

The next best recognized object in Figure 3.4 is $o_2$ with a quality value of 0.546 which is the result of combining with Confirmation Theory from $G$ a 0.508 ($= w_{G,o_2} \cdot u_{G,o_2}$) and from $S$ a 0.78 ($= w_{S,o_2} \cdot u_{S,o_2}$).

In contrast, in Figure 3.3 the next best recognized object is the result of combining from $G$ a 0.445 and from $S$ a 0.172. Here the value for $S$ is significantly higher than the 0.78 of the $o_2$ of Figure 3.4 though the value for $G$ here is lower than the 0.508 of the $o_2$ of Figure 3.4. The Bayesian Approach has a more "democratic" approach in that the final value reflects more the opinion of all the methods instead of giving preference to single high values. (In this respect Fuzzy-set theory is even more extreme than Confirmation Theory selecting the highest values delivered by any one method and ignoring the other values.)

Using the Bayesian Approach, in Figure 3.3, $o_7$ is recognized last because method $G$ cannot identify this object at all; the only information that $Mingle$ obtains comes from method $S$. Compare with Figure 3.4 where, using Confirmation Theory, $Mingle$ places the same object as fifth in the list. Though there is no information coming from method $G$, method $S$ identifies this object with a high recognition value and the weight for $S$ for this object is also high. Thus the object is named $o_5$ in Figure 3.4.

Though we have not illustrated the last object identified in Figure 3.3, $Mingle$ identifies $o_8$ as though it were $o_7$ but gives $o_8$ the lower quality value of 0.007. Some of the objects in our database were quite similar to each other to make the recognition task more difficult. The other objects follow; these are not right but their quality values are either 0.001 or 0.000. And similarly for Figure 3.4, the other objects identified were no longer correct but they had a corresponding lower quality value.
Figure 3.3: This scene depicts the order in which the objects were recognized according to their quality values. A Bayesian Approach was used to combine.

Figure 3.4: This scene depicts the order in which the objects were recognized according to their quality values. Combination Theory was used to combine.
4 Discussion

It is simple to run Mingle as a distributed system. Each branch \( i \) (consisting of the preprocessing for \( M_i \), of \( M_i \), and of the weight calculation for \( M_i \)) in Figure 3.1 can be computed independently of each other, e.g. in a separate processor. In this case the upper bound for the total recognition process depends on the slowest branch. Obviously in the serial case the overall complexity is the sum of the complexity of the individual methods.

We still need to investigate the matter of determining the minimum number \( m \) of methods needed to accomplish the task of recognizing objects perfectly (i.e., to perform the mapping (2.1)). This number \( m \) depends on the complexity of the individual methods involved. We hypothesize that it is better to have more methods but simpler ones, as opposed to having fewer methods that are quite refined. Furthermore, it is important that the sets of features that the methods can recognize be as mutually exclusive as possible: we want to avoid "overkill".

In this paper we write about a list of objects sorted according to their quality value and do not attempt to set thresholds determining when an object is or is not recognized. Yet, such thresholds will be needed if Mingle is to be used in say, a robotics environment; there the decision of recognition needs to be binary. We are now experimenting with comparing the results from the combination methods in a cross reference manner to determine where the thresholds should be set. These thresholds are not fixed; they are application dependent. We feel that if the first objects in the list delivered using Confirmation Theory are identical to those first objects in the list delivered using the Bayesian Approach, then these first objects are "recognized" (such is the case for the first objects in Figures 3.3 and 3.4). We still need to investigate this matter further but speculate that there is potential in using various combination theories.

5 Conclusion

A method for object recognition was presented. Instead of developing "yet another method for object recognition", we break down the overall task of recognition into several subtasks each with a simpler domain. These subtasks are solved by object recognition methods that are suited to recognize certain types of features in an object. Each of these methods delivers a quality value for each object to reflect the result of the matching process.

The piecing together of the solution to the original problem occurs in two separate phases. In a first phase, weights are computed to determine the suitability of a method to recognize the objects found in the scene. In a second phase, the quality values for the objects from the various methods are combined with the weights. The final answer is a sorted list of objects. As opposed to employing one single sophisticated method, with this approach we increase both the domain of discourse of model objects to be recognized and the accuracy of the object recognition process.
6 Bibliography


