Vision, Video and Graphics (2005)
E. Trucco, M. Chantler (Editors)

Bitvectors for Robust Hierarchical Template Matching

David Tweed

Institute for Perception, Action and Behaviour, Edinburgh University

Abstract

Many target detection problems involve objects where the primary variability in appearance is due to changes amongst characteristic configurations (as opposed to the systematic variability of object rotation or illumination changes). It is then important to utilise as much of the correlation between features as possible. Detecting pedestrians is such a problem and was tackled by Gavrila [Gav98] using a large set of exemplar *templates combined with hierarchical matching via Distance Transforms. We describe a variant using a* robust *distance function and explicit allowance for occlusions. Our innovation is using* bitwise logical operators *to test against multiple exemplars in parallel.*

Categories and Subject Descriptors: I.4.8 [Image Processing and computer vision]: Object recognition,

1. Introduction

We consider detecting objects of a given type in images where some key assumptions can be made. These are applicable to a wide range of problems, but aren't universally met. We will refer to the objects of the kind we are searching for as *targets*, and we assume that the targets have a continuous but constrained range of configurations. (For example, a tennis player may be in a forehand position, in a backhand position, or transitioning between the two; they aren't, say, performing handstands even though this is physically possible). We assume some simple process – such as an edgel or interest point detector [SonHlaBoy99] – can reduce the image to a relatively sparse set of feature detection locations. These give a corresponding set of *feature configurations* $\{X\}$ distributed according to an *a priori* unknown distribution $p(X) \doteq p(X|\mathsf{target})$. We also assume targets have a characteristic size in the image. A range of object appearances for a typical problem, sign-language hand-spelling decoding, is shown in Fig 1. This satisfies our assumptions as the hands would be a standard distance from the camera and the alphabet hand shapes provide the constrained set of object appearances.

1.1. Previous work

The standard target detection approach is to boil down the training data into some form of parameterised mathematical model of the appearance of the targets and declare the target detected when parameters can be found which cause the model to reproduce the image features. The constrained range of the target appearances ought to translate into *explicitly described* constraints on the model, but it is often difficult to carry over to the model, and indeed even to understand, the various subtle constraints on object appearance. For example, consider a kinematic-chain model for detecting tennis players: the natural 3-D joint angle constraints are difficult to extract purely from the 2-D images, and where exactly does the boundary between the allowed forehand and disallowed handstand configurations lie? A different approach is to develop a very minimal model for matching feature sets to the data in the training set, since the training set *implicitly obeys* the appearance constraints. This approach was taken by Gavrila for recognising pedestrians for an in-car warning system [Gav98, Gav99]. This is based upon making a separate *exemplar* from edge detection results for each target in the training set. A detection occurs when the chamfer distance between the detected outline and an exem-

plar image is within a threshold. This elegantly avoids needing a mathematical model for pedestrian appearance, and using a clever hierarchical search it is computationally feasible to use thousands of exemplars for an accurate pedestrian appearance model. This image-exemplar approach has been developed further researchers (eg, [ToyBla01] adapts probabilistic modelling to the metric space due to the exemplars) and applied, eg, for face recognition [KruZho02] and video analysis [GatSun02].

There are two minor complications when using Gavrila's framework. Firstly, every detected edgel contributes to the chamfer distance used to match contours, whereas occlusion, missed detections and deviations from the training set argue for a robust-statistics type matching measure. In particular, the matching process often fits a distinctive region of the exemplar – such as angled legs – very well whilst a human would argue that other regions of the exemplar don't meaningfully match at all. The chamfer distance is affected by the precise details of these non-matching regions, whereas it would be desirable to have all correspondences outside a relatively small degree of closeness making the same contribution to the match score. Secondly, the hierarchical match procedure works by making a *narrow, deep* decision tree which enables a match to be made whilst inspecting a small proportion of the tree, and hence set of exemplars. However, this narrowness makes the matching process sensitive to wrong decisions when the training set is not relatively homogeneous; it would be desirable to perform hierarchical matching via a *wide, shallow* tree. This paper describes a variation on the exemplar algorithm attempting to address these two issues which may be useful for some problems.

There are two similar approaches to pattern detection. The first [Bre92, Bre03] is based upon requiring the error metrics used to be robust, in the sense that beyond a certain threshold the badness of the match should not affect the error value. Breuel refers to this problem as *bounded error matching*, and the distance metric used in Sec. 2 is a very simple example of such a metric. Because such error metrics are non-smooth traditional gradient based fitting does not work reliably with them. However, with some reasoning about the detailed form of the distance function efficient algorithms for the fitting process are found using hierarchical subdivision and branch-and-bound [Bre92, Bre03]. Our work does not use such sophisticated techniques, but also focuses on the different problem of matching against many possible templates. The second approach is Olson & Huttenlocher's [OlsHut97] formulation of matching in terms of the Hausdorff distance between oriented edge maps of object models and observed edges, where again a threshold is placed upon the maximum error of solutions of interest. When attempting to match against a group of objects a hierarchical decomposition is build where common structures of oriented edge pixels are shared, so that they only have to be evaluated against once.

1.2. Interaction between algorithms and hardware

The above motivates the general exemplar approach from an image analysis viewpoint. However, the algorithm in this paper is also an example of the interaction between mathematically motivated algorithm design and hardware motivated design. Unfortunately, in computer vision development is often guided only by 'high-level' considerations. High-level analysis does generally yield the greatest improvements, eg, a mediocre implementation of the frequency domain correlation algorithm will be better than even the most highly-tuned code for direct image domain correlation [SonHlaBoy99]. However, it's also true that whilst modern CPUs have very low instruction cycle times their performance characteristics are starting to differ significantly (and in surprising ways) from the classical model that guides developers.

As an example, consider a module for colour segmenting images within a robotic system. In many applications various classes of object can easily be arranged to have characteristic colours. Colour segmentation is a well-studied problem in computer vision, with new algorithms for performing good segmentations on ever more challenging scenes being developed each year; however real-time performance constraints often force robotics researchers to develop specialised segmentation algorithms rather than use 'off-the-shelf' vision algorithms. In [BruBalVel02] – the paper that inspired this work – an image must be segmented into regions belonging to colours defined by cuboidal subsets of YUV space. It is noted that complex, unpredictable control flow (eg if–then–else) can perform poorly on modern architectures; because of this testing for cuboid membership in the obvious way is far too slow. However, they construct an equivalent technique using bitwise logical operators which *on modern architectures* performs in real time. (Bitwise operators are also used widely in the genetic programming community, e.g., [CagAdoMor02].) Their development is *not* purely low-level optimisation but involves *both* high- and low-level considerations.

We suggest that the development of *principled* variants of high-level vision algorithms with better real-world performance is a small but worthwhile niche to explore.

We next describe an algorithm for performing the

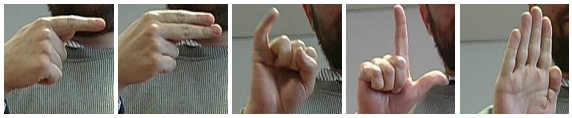


Figure 1: *Some examples of hand shapes for letters in American Sign Language*

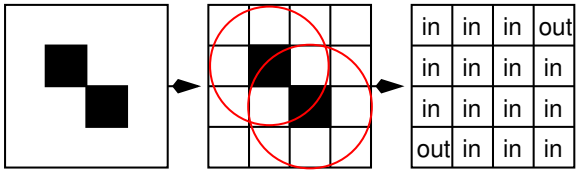


Figure 2: *distance transform: (a) feature pixels; (b) within τ of features; (c)* inlier/outlier

matching problem described above using a framework designed to make maximal use of hardware resources, and then present some preliminary experimental results.

2. Detection using exemplars

To take advantage of these restricted range of characteristic appearances we need to produce an approximation $\hat{p}(X)$ to $p(X)$ by analysing a set of training examples $\mathcal{S}$. Standard probabilistic reasoning shows that $p(\mathsf{target}|\mathbf{z}) \propto \sum_{\mathbf{x}} p(\mathbf{z}|\mathbf{x})\hat{p}(\mathbf{x})$, where $\mathbf{z}$ is a vector of observed features and $\mathbf{x}$ is the feature detections corresponding to a particular configuration of the target. We're only interested in relating target presence to observations so we push all the 'model complexity' into the $\hat{p}(\mathbf{x})$ term and virtually take $p(\mathbf{z}|\mathbf{x}) = \delta_{\mathbf{x}}(\mathbf{z})$ (see next section). However, it is important to note that approximating $\hat{p}(X)$ uses *both* a set of parameters *and* a computational process taking the parameters and observed data to a probability. Most common techniques – e.g., neural networks, Gaussian mixtures or graphical models [Bis95] – move most of the 'modelling complexity' into the computational process to reduce the number of parameters. In contrast we choose to approximate $p(X)$ directly from the training set $\mathcal{S} = \{(w_i, \mathbf{s}_i)\}$, where w_i is the relative frequency (scaled to sum to 1) and $\mathbf{s}_i$ is a sparse set of features detected. This allows $\mathcal{S}$ to encode priors on configurations whilst keeping $|\mathcal{S}|$ proportional to the number of *unique* configurations. The basic idea is to compare the observed features to the features detected in each of the exemplars in $\mathcal{S}$ individually, using a robust distance function to deal with the *essentially tiny* variations caused by imprecise feature detectors, tiny pose changes, etc. In some approaches (EGA, [ToyBla01]), some of the degrees of variation are handled by introducing an additional geometric transformation: A set of features from a new image is compared against an exemplar by first finding a transformation which minimises an appropriate distance between the feature set and the transformed exemplar. Whilst this increases the expressiveness of the model, recall we are using the exemplar approach precisely because we don't understand the detailed constraints which apply to the feature sets generated by the target. As it is almost impossible to come up with any neatly parametrised transformations other than shifts, rotations and scalings which generate only physically realisable transformed exemplars, it is debatable whether increasing expressiveness via transformations is the best course. In this work, we take the opposite approach of just adding more exemplars to training set $\mathcal{S}$. Thus, we keep the simple computational process but increase the number of parameters (ie, exemplars). With only small variations to deal with we can use simple feature-feature distance function

$$d(\mathbf{a}, \mathbf{b}) = \textbf{if } |\mathbf{a} - \mathbf{b}| < \tau \textbf{ then } \mathsf{inlier} \textbf{ else } \mathsf{outlier} \quad (1)$$

i.e., an image feature $\mathbf{a}$ matches an exemplar feature at $\mathbf{b}$ if it lies within a distance τ. (This particularly stark distance function is partly motivated by Section 3.)

2.1. A probabilistic formulation

We now cast 'matching' in terms of a generative model. Given a noise threshold τ we can construct the set Σ of offsets of magnitude less than τ and define the configurations generated by exemplar $\mathbf{s}$ as

$$\mathbf{s} \otimes \Sigma = \{\mathbf{s} + \sigma | \sigma \in \Sigma^{|\mathbf{s}|}\} \quad (2)$$

Using this a set of features is generated from $\mathcal{S}$ by simply selecting an exemplar $\mathbf{s}_i$ (with chance proportional to w_i) and then choosing uniformly an element of $\mathbf{s}_i \otimes \Sigma$. This gives

$$\hat{p}(\mathsf{target}|\mathbf{z}) \propto \sum_{(w_i, \mathbf{s}_i) \in \mathcal{S}} w_i \chi_{\mathbf{s}_i \otimes \Sigma}(\mathbf{z}), \quad (3)$$

where characteristic function $\chi_A(v)$ is 1 iff $v \in A$ and $\mathbf{z}$ is the vector of feature detections.

The key to efficiently evaluating $\chi_{\mathbf{s}_i \otimes \Sigma}$ is to form an 'image' D_i for each exemplar $\mathbf{s}_i$. This has all pixels within τ of a point in $\mathbf{s}_i$ to inlier and all others to outlier (as shown in Fig 2). We then form variables describing the matches of the individual image features $v_j \doteq D_i[\mathbf{z}_j]$, essentially using D_i as a lookup table. Then $\mathbf{z} \in \mathbf{s}_i \otimes \Sigma$ exactly when all the v_js have the value inlier, and as D_i depends only on $\mathbf{s}_i \otimes \Sigma$ it can be precomputed. This approach is the Distance Transform [RosPfa68] for our unusual d.

In practice other objects can occlude targets, so that

in a new image containing the target some features due to the target will not appear whilst some new features due to the occluder will appear. Since we 'detect' by requiring all observed features match with the features in the exemplar (and not all the exemplar's features to match with observed features), features *blocked out* by the occluder are not a problem. However, extra features *introduced* by the occluder are unmatched in the exemplar and so the strict matching above won't work. Thus we need a refined version of $p(\mathbf{z}|\mathbf{x})$. We use a simple solution: if at least α of the observed feature points are present in a pattern $\mathbf{s}'$ from $\mathbf{s}_i \otimes \Sigma$ then $p(\mathbf{z}|\mathbf{s}')$ is 1, otherwise it is 0; this translates to 'robustifying' $\chi_{\mathbf{s}_i \otimes \Sigma}$ in Eq 3.

3. Matching with a feature-set using boolean operations

As our robust distance measure takes only two values inlier and outlier we can represent them as the response to 'is the point an inlier?', i.e., inlier$\mapsto$1 and outlier$\mapsto$0. Thus detecting exemplar i if at least αN of the N image feature lookups $v_{1:N}$are inliers in exemplar i is equivalent to $count(v_{1:N})$ – the number of 1s in $v_{1:N}$ – being at least αN (using $v_{1:N}$ to stand for $v_1, \ldots, v_N$). We *could* compute *count* with standard addition instructions by simply incrementing a count each time a non-zero v_i is encountered, but the bitvectors used in the next section motivates using *only* boolean operators.

The key to efficient calculation of $count(v_{1:N})$ is relation

$$count(v_{1:N}) = count(v_{1:N/2}) + count(v_{N/2+1:N}) \quad (4)$$

i.e., the number of 1s in $v_{1:N}$ is the sum of the 1s in the two subsets; moreover *when* $count(v_{1:N/2})$ *and* $count(v_{N/2+1:N})$ *are* k*-bit numbers then* $count(v_{1:N})$ *has at most* $k{+}1$ *bits*. Our algorithm is to maintain an explicit binary representation of *count* and perform the additions at the level of program code; this gives us the freedom to compute additions using only the number of bits required. We can bound the operation count using the case $N{=}2^n$: Computer architecture shows that two k-bit binary numbers can be added in $5k$ logical operations [HenPat90] (an example is shown in Fig. 3a). Thus evaluation forms a *tree* (shown in Fig 3b) with the length increasing by at most 1 for each level, so that the number of operations is given by

$$\#ops(count(v_{1:N})) = \sum_{k=0}^{n} 5k{\cdot}2^{n-k} = 5(2^{n+1}{-}n{-}2) < 10N \quad (5)$$

in comparison to the N^2 operations without using the length constraint. It is plausible an application might work on high-resolution images where an object gives rise to 1–2 thousand feature detections (e.g., edgels when detecting human beings from their silhouettes). Although memory access costs will largely determine the algorithm's performance, the tree formulation requires 10–20 thousand operations per matching operation versus 1–4 *million* operations for the direct approach, enough to justify the extra 'complication'.

carry$\leftarrow$0
for i **in** $1, \ldots, k$**:**
  $t_1 \leftarrow y[i]$ **xor** *carry*
  $t_2 \leftarrow y[i]$ **and** *carry*
  $x[i] \leftarrow t_1$ **xor** $z[i]$
  $t_3 \leftarrow t_1$ **and** $z[i]$
  carry$\leftarrow t_2$ **or** t_3
$x[k{+}1] \leftarrow$ *carry*

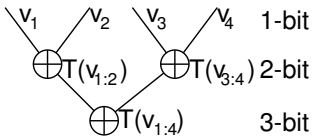


Figure 3: *(a)* $x \leftarrow y{+}z$ *using logic operations; (b) tree-wise computation of* $count(v_{1:4})$

3.1. Parallel evaluation with bitvectors

So far this merely reformulates the detection problem without computational benefit. However we now utilise the *bitwise logical operator* instructions of actual processors. For example the bitwise operator `&` takes two *bit vectors* and produces a new bit vector, each of whose positions is filled by **and**ing the bits in the corresponding position of the two inputs, performing the computation *in parallel*. A given processor has a small range of bit vector lengths: 32 is a common denominator but, e.g., the Pentium III/IV have 128-bit vectors via SIMD registers. We use these to check a set of features against 128 exemplars *in parallel*.

As bitwise operators process each bit independently all equations involving boolean operators have counterparts using the bitwise operators. (This correspondence was used in the segmentation algorithm of [BruBalVel02].) Imagine 'stacking' the images D_0 to D_{127} and taking a vertical 'core sample', as shown in Fig 4. This is equivalent to taking the $\mathbf{v}_i$s now as bitvectors whose jth entry is 1 exactly when exemplar j has inlier at position x_i. The 'binary operations only' algorithm shown in Fig 3 can then be used to compute *count* for each D_i in the stack in parallel. This gives us a *bitvector table* $\mathbf{D}^{(0:127)}$ where the bitvectors at position $\mathbf{D}^{(0:127)}[\xi]$, corresponding to the core sample from $D_0, \ldots, D_{127}$ at ξ can be precomputed as they depend only on the D_i images, can be precomputed.

This is approach initially seems a little peculiar: conventionally we would use built-in addition on integers to compute *count* for each individual pattern

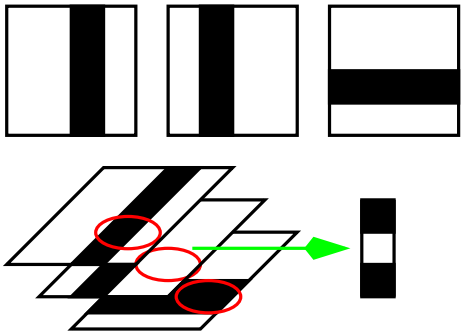


Figure 4: *a 'core sample' through the bitmap stack becomes a bitvector*

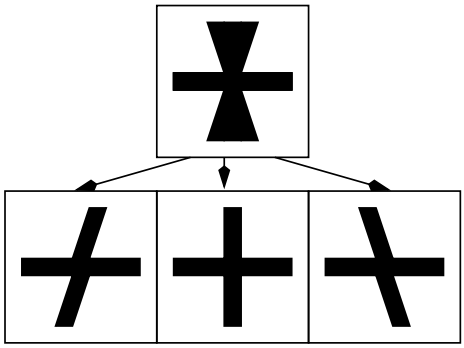


Figure 5: *combining bitmaps into a prototype*

sequentially, rather than matching patterns in parallel but then requiring addition to be performed at the program level. Better efficiency results as built-in addition always operates on *fixed-length* integers regardless of the actual contents. The aim is to allow using a robust distance function and 'wide' trees with efficiency similar to that of [Gav99]. The implementation was motivated by the platform used for development, but long bitvectors are widely available and the algorithm is extremely likely to remain competitive on future architectures.

4. A hierarchical search strategy

The previous section allows at most 128 exemplars and requires evaluation at each image point, even those 'obviously dissimilar' from all exemplars. We tackle both issues using the same idea as the hierarchical search in [Gav98].

4.1. Prototype exemplars

Suppose we have a set of exemplars $\mathcal{S}_i$=$\mathbf{s}_{0:N}$ and we want to perform some pre-screening to avoid performing a match against the the individual exemplars in the set if 'it's clear they aren't going to match'. One way to do this would be to composite *prototype exemplar* $\mathbf{p}^{(\mathcal{S}_i)}$, which is a synthetic but otherwise normal exemplar which we can try to match against. It needs to have two simple properties:

1. if observation $\mathbf{z}$ would match *any* element of $\mathcal{S}_i$ then $\mathbf{z}$ *must* match $\mathbf{p}^{(\mathcal{S}_i)}$.
2. *in most cases* observations $\mathbf{z}$s which don't match anything in $\mathcal{S}_i$ don't match $\mathbf{p}^{(\mathcal{S}_i)}$ either.

In essence these conditions say the prototype provides a *single* exemplar which rejects a large proportion of non-matches whilst never rejecting a true match: Condition 1 is required for correctness, whilst the degree to which 2 holds determines the efficiency of the matching process.

A simple way to produce such a prototype is simply to superimpose the exemplars making $\mathcal{S}_i$. This certainly achieves criterion (i). However to reject as many non-matches as possible it is desirable to align the exemplars so that common structures amongst the elements of $\mathcal{S}_i$ are overlaid. This is illustrated in Fig 5, where three cross-type structures are combined into a prototype where the common horizontal bar is overlaid: clearly any cross matching any of the 3 low-level crosses will match with the prototype. However, the fewer structures are overlaid the easier it is for a feature set to get a false match with $\mathbf{p}^{(\mathcal{S}_i)}$ by 'taking half the feature matches from one exemplar and half from another exemplar': in Fig 5 the prototype will match against an hourglass shape, even though this is not present in any of the underlying patterns.

To do this, we want to find a set of spatial displacements $\delta_{0:N}$ to apply to the elements $\mathbf{s}_{0:N}$ such that the overlaying of all the exemplars in exemplars $\mathcal{S}_i$ has as few distinct feature positions as possible. A similar problem occurs in the hierarchical exemplar matching of [Gav98] where the alignment minimising the match distance between the overlaid exemplars is used. This match is approximated effectively using simulated annealing. Unfortunately our binary distance function is very non-smooth so simulated annealing performs poorly; instead we are forced to use brute force search. Even with some highly sophisticated branch-and-bound techniques this search process takes a very long time. Luckily this is an *offline preprocessing* step which only needs to be performed once.

4.2. Exemplar match trees

To use prototypes for hierarchical matching, suppose we have a large set $\mathcal{S}$ of N exemplars $s_{0:N}$, where N

find the locations $\mathbf{z}_{1:N}$ of the features in W
initialise *work*←[(*root_node*, $\mathbf{0}$)] and *results*←{}
while *work* is non-empty**:**
  $(n, \boldsymbol{\Delta})$←*work.pop_front*()
  for ξ **in** test node offsets $R^{(n)}\times R^{(n)}$**:**
    $\boldsymbol{\Delta}'$←$\boldsymbol{\Delta}$+ξ
    for $\mathbf{z}_i$ **in** $\mathbf{z}_{1:N}$**:** set $\mathbf{v}_i$←$\mathbf{D}^{(n)}[\mathbf{z}_i+\boldsymbol{\Delta}']$
    match_vec←*map_over_bitvector*(*count*($\mathbf{v}_{1:N}$) > αN)
    if *match_vec*≠$\mathbf{0}$**:**
      if n is a leaf node **then** insert *match_vec* into *results*
      otherwise *work.add_end*((*best_child_node*(n, *match_vec*)), $\boldsymbol{\Delta}'$))

Figure 6: *algorithm for hierarchical matching within a window W*

is much larger than 128. We can divide this into 128 subsets $\mathcal{S}_0, \ldots, \mathcal{S}_{127}$, where each subset contains primarily *similar* exemplars. From these we can build both a bitvector table $\mathbf{D}^{(\mathcal{S}_i)}$ and a prototype $\mathbf{p}^{(\mathcal{S}_i)}$ for each subset. As prototypes are sets of feature responses just like standard exemplars, we can stack them just like we did for the individual exemplars in each subset.Thus we compute a bitvector table $\mathbf{D}^{(\mathcal{S}_0:\mathcal{S}_{127})}$ which summarises the information in the prototypes. Hierarchical matching then proceeds in the obvious way: we first test our observation $\mathbf{z}$ against the bitvector table for the prototypes $\mathbf{D}^{(\mathcal{S}_0:\mathcal{S}_{127})}$. Any prototypes which match we then test against the corresponding bitvector table $\mathbf{D}^{(\mathcal{S}_i)}$ for the exemplars that the prototype summarises. These final results correspond to the matched exemplars in the original set $\mathcal{S}$.

This can be seen as searching in a two level tree. The tree has a wide fan-out and because we only proceed down branches corresponding to a matched prototype, only a small fraction of the leaf nodes are examined when a match operation is performed. Whilst in principle the tree could have full 128-way fan-out, in practice it is more robust to have only 16–32 fan-out, with a several prototype exemplars leading to the same secondary set of exemplars. This two-level construction can be extended in the obvious way to multiple levels, where higher levels contain prototypes produced from sets of prototypes rather than individual exemplars. However, since the tree capacity grows exponentially with the number of levels, there is unlikely to be enough training data for more than a few levels.

4.3. Multiresolution hierarchical search

This description of hierarchical matching assumes that we have an exact position for the feature set we are testing against the exemplars. In practice we want to test all locations in the image against the exemplar set; clearly performing a full match centred on every pixel will be very inefficient. We can use a the same idea of generating new exemplars which always match if there's a true match in a local neighbourhood but which suffer from some false positives. Observe if we progressively increase the noise threshold τ in Eq 1 we obtain a sequence of distance images $\mathbf{D}^{(\tau_1)}, \ldots, \mathbf{D}^{(\tau_k)}$ obtained using thresholds $\tau_1{=}\tau, \ldots, \tau_k$. These will match against an observation sets which more misaligned as τ_i increases, at the cost of also matching with more observation sets which don't truly correspond to the underlying pattern. These two facts parallel the two conditions which we used in section 4.1 to define what the prototype should do. Thus, we can regard a distance image with an increased threshold as a "misalignment prototype" which can be used to prescreen matches on a coarse search before trying pixel resolution matching.

In detail: if we have an exemplar which would match at a position ξ then we'll get a match at position $\xi{+}\delta$ with $\mathbf{D}^{(k)}$ providing that $|\delta|{+}\tau{\leq}\tau_k$. So testing can then be performed first on a coarsely spaced grid using $\mathbf{D}^{(\tau_k)}$ giving a small number of 'test match here at finer resolution' locations. We lay down a finer grid around these locations and then test against the more discerning distance image $\mathbf{D}^{(\tau_{k-1})}$. This process is continued until we reach the original transform image $\mathbf{D}^{(1)}{=}\mathbf{D}$ at 1-pixel search granularity.

These ideas are combined into the complete algorithm shown in Fig 6, with some details we clarify here. The double ended queue *work* is equivalent to breadth-first search and leads to elements on level l being evaluated consecutively, increasing the likelihood that the model bitvectors are in the processor cache. *map_over_bitvector*(*count*($\mathbf{v}_{1:N}$)>αN) has the ith bit set iff the integer represented by the ith bits of *count*($\mathbf{v}_{1:N}$) exceeds αN. *results* is a weighted set of matches and requires post-processing, eg, for simple detection performing non-maximum suppression

as there are less-supported matches around the true match.

5. Experiments

Exemplars require daunting volumes of marked up data, both for training and testing. We present some preliminary experiments using data sets which were only just big enough.

5.1. Written characters

Handwritten glyphs are a basic shape with variation due to pen slippage, muscle imprecision, neither of which is systematically geometric. A large number of examples of $\alpha\beta\gamma\delta\epsilon\phi\kappa\lambda\nu\omega\pi\psi\rho\sigma\tau\chi$ were written on paper and scanned to obtain a binary image where a glyph was 48×48 pixels. Edgels obtained using a binary-image edge detector were used as features. Each letter became a class and a 2-level 16-way tree was built using 128 examples of each letter. A prototype for each letter shown in Fig 7a (where whiter means more exemplars have a feature at that position). Fig 7b shows detection results on randomly drawn unseen examples: the first row is the number of correct detections whilst the second is the number of examples from that class.

5.2. Hand shapes

We obtained a 2500 frame video of a pair of hands running through the hand shapes corresponding to the letters a–z in ASL, with the region corresponding to the hands being around 140×140 pixels. Due to the *very small* differences between some shapes and the signer's poor competence we grouped the set of letter hand-shapes into just 8 classes with broadly similar characteristics. The video was divided into segments in the same category. Unfortunately a significant number of frames at points corresponding to 'in-transition' frames didn't fit any category well; in order to look at performance with 'rough-cut' data we didn't exclude any images judged to be transitioning. Edgels obtained using a standard edge detector were used as features. Random subsets of 256 (ie 2×128) frames from each class were used to build a 2-level, 16-way tree was built. Prototype exemplars for each branch shown in Fig 7c (so each prototype image combines data from one half of the images in the class). As in Fig 7b the left table in Fig 7d shows detection on a randomly drawn set of unseen frames from each class; there are no results for classes 0 & 12 as there were no unseen frames in the data set. The right table shows the confusion between true classes (rows) & detected classes (columns) as a fraction of the images tested; the deficit is from 'no detection'.

The results here are equivocal; we believe that due to issues with the datasets, and in particular the small sizes of exemplars in the images, these may not be representative.

6. Conclusions & future work

We described a variant upon the template matching of [Gav98] for detecting targets specified by a training set $\mathcal{S}$ by searching for matches between observed features and exemplars. We explicitly aim to make as few generalisations from the $\mathcal{S}$ as possible; to aid this a robust distance function is used combined with a proportion of features matched global criterion to ignore outliers and occlusions. Practical usage requires very many exemplars and we used bitvectors to compute multiple match scores simultaneously. Adding hierarchical search gives *wide* exemplar trees enabling robust, efficient computation. It is useful when the characteristic set of configurations is not easily encoded in a parameterised model. In future work we intend to produce larger annotated training sets so we can perform larger scale tests of the algorithm, both in as an isolated detection mechanism and when incorporated in the multiple object tracker of [TweCal02].

Acknowledgements

Andrew Calway and especially Mark Everingham for stimulating discussions on object recognition strategies.

References

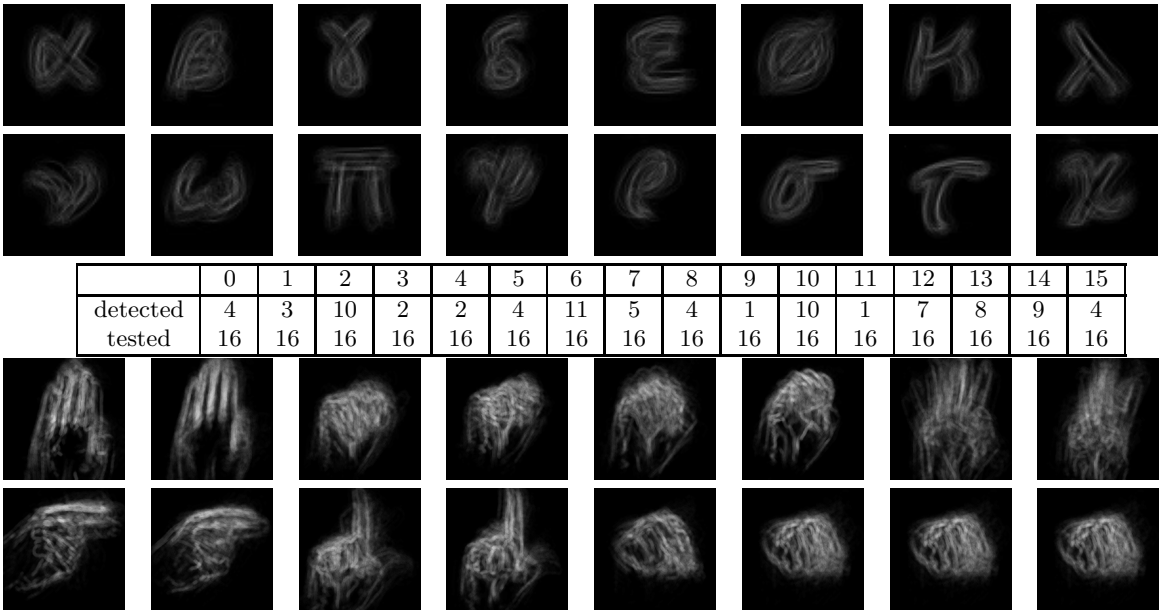
[Bis95] C M Bishop. Neural Networks for Pattern Recognition. *OUP,* 1995.

[Bre92] T M Breuel. Fast recognition using adaptive subdivisions of transformation space. *In Proceedings. 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.92CH3168-2),* 445–51, 1992.

[Bre03] T M Breuel. Implementation techniques for geometric branch-and-bound matching methods. *Computer Vision and Image Understanding,* 2003.

[BruBalVel02] J Bruce, T Balch, and M Veloso. Fast and inexpensive colour image segmentation for interactive robots. *In* IROS, 2000.

[CagAdoMor02] S Cagnoni G Adorni and M Mordonini. Efficient low-level vision program design using sub-machine-code genetic programming. *In* Conf. Assoc. Italiana per l'Intelligenza Artificiale, Oct 2002.



(a)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
detected	4	3	10	2	2	4	11	5	4	1	10	1	7	8	9	4
tested	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16

(b)

(c)

0	2	4	6	8	10	12	14
na	138	89	75	18	3	na	37
0	300	300	134	54	12	0	300

	0	**2**	**4**	**6**	**8**	**10**	**12**	**14**
2	0.42	0.43		0.00	0.02	0.0		
4	0.40	0.30	0.29	0.00	0.00	0.01		0.00
6	0.11	0.06	0.04	0.56	0.00	0.00		
8	0.04	0.43	0.0		0.34			
10	0.25	0.08	0.08			0.25		
14	0.19	0.06	0.04	0.01		0.03		0.12

(d)

Figure 7: *prototypes & detection statistics: (a-b) greek; (c-d) hands*

[GatSun02] D. Gatica-Perez and M.T. Sun. Linking objects in videos by importance sampling. *In* IEEE Conf on Multimedia & Expo, 2002.

[Gav98] D M Gavrila. Multi-feature hierarchical template matching using distance transforms. *In* ICPR, 439–444, 1998.

[Gav99] D M Gavrila and V Philomin. Real-time object detection for "smart" vehicles. *In* ICCV, 87–93, 1999.

[HenPat90] JL Hennessy and DA Patterson. *Computer Architecture: A Quantitative Approach. Morgan Kaufmann, 1990.*

[TweCal02] D Tweed and A Calway. Tracking Many Objects Using Subordinated CONDENSATION. *In* Proc. of BMVC, page 283–292, 2002.

[KruZho02] V Krueger and S Zhou. Exemplar-based face recognition from video. *In* ECCV, 28–31, 2002.

[OlsHut97] C F Olson and D P Huttenlocher. Automatic target recognition by matching oriented edge pixels. IEEE Trans Image Proc., 6(1):103–113, Jan 1997.

[RosPfa68] A Rosenfeld and J Pfaltz. Distance functions in digital pictures. Pattern Recognition, 1:33–61, 1968.

[SonHlaBoy99] M Sonka, V Hlavac, and R Boyle. *Image Processing, Analysis, and Machine Vision. PWS Publishing, 1999.*

[ToyBla01] K Toyama and A Blake. Probabilistic tracking in a metric space. *In* ICCV,**2**,50-59,2001.