IEEE Conference on Computer Vision and Pattern Recognition, San Juan, Puerto Rico, June 1997

Object Detection with Vector Quantized Binary Features

John Krumm
Intelligent Systems & Robotics Center
Sandia National Laboratories
Albuquerque, NM 87185
jckrumm@sandia.gov

Abstract

This paper presents a new algorithm for detecting objects in images, one of the fundamental tasks of computer vision. The algorithm extends the representational efficiency of eigenimage methods to binary features, which are less sensitive to illumination changes than gray-level values normally used with eigenimages. Binary features (square subtemplates) are automatically chosen on each training image. Using features rather than whole templates makes the algorithm more robust to background clutter and partial occlusions. Instead of representing the features with real-valued eigenvector principle components, we use binary vector quantization to avoid floating point computations. The object is detected in the image using a simple geometric hash table and Hough transform. On a test of 1000 images, the algorithm works on 99.3%. We present a theoretical analysis of the algorithm in terms of the receiver operating characteristic, which consists of the probabilities of detection and false alarm. We verify this analysis with the results of our 1000-image test, and we use the analysis as a principled way to select some of the algorithm's important operating parameters.

1. Overview and Context

Detecting objects in images and measuring their location is a fundamental task of computer vision, with applications in manufacturing, inspection, world modeling, and target recognition. Often the scene is inherently cluttered, the object may be partially occluded, and illumination may change. In this case, the algorithm must look at features internal to the objects' silhouette, and look at them in such a way that missing features and changing illumination are tolerated.

Researchers have responded to this need in many ways, including fairly recent, elegant object detection algorithms based on principle components of training images of the object[[6]][[10]]. In particular, Murase and Nayar[[6]] extract templates from training images of the object in different orientations, compute eigenvector principle components of these templates, and recover the object's orientation in new images by projecting them onto the principle components. They address the problem of illumination changes by taking training images under different lighting conditions. The whole-object template idea was improved by algorithms that look at only part[[7]] or parts[[4]][[8]] of the training templates. This helped to reduce or eliminate the effects of background clutter and partial occlusions.

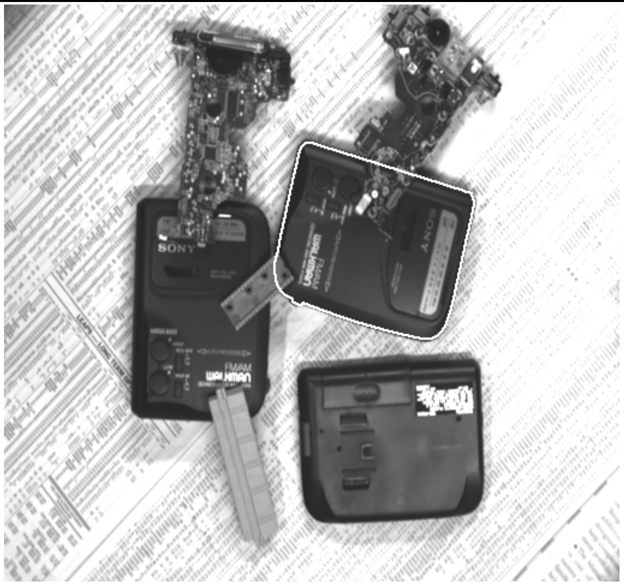
Figure 1: Result of detection algorithm in presence of background clutter and partial occlusions.

This paper presents and analyzes a new algorithm for object detection based on binary subtemplates (features) of the training images. Binary features are more robust to illumination changes than the gray-level features of previous methods. We replace the eigenvector principle components with binary vector quantization, a common method for image compression. This avoids any floating point processing after edge detection. An object is represented with a separate, distinct model for each pose in the training set, which avoids problems caused by self-occlusion and the movement of specular highlights. A model consists of vector quantized binary features and their relative spatial offsets with respect to each other. An example of our detection results in an image with background clutter and partial occlusions is shown in Figure 1. On a test of 1000 cluttered images containing the test object, the algorithm correctly detected the object in 99.3%. In order to determine the best values of the algorithm's important parameters, we derive and plot the receiver operating characteris-

This work was performed at Sandia National Laboratories and supported by the U.S. Department of Energy under contract DE-AC04-94AL85000.

tic. This shows the tradeoff between the probabilities of detection and false alarms.

2. Appearance Modeling with Binary Features

An object's pose has six degrees of freedom in general, and a general version of our algorithm would have a separate model for each discretized pose in this six-dimensional space. Such a high-dimensional model space implicitly accounts for the normally confounding effects of parallax, self-occlusion, and the movement of specular highlights. Our detection problem is less general in that we have a camera pointed down at objects resting on a plane which is perpendicular to the camera's optical axis. This eliminates all but one continuous degree of freedom. For now, we will describe the algorithm as having n_m models for a single object, with the models spread over the applicable degrees of freedom. In general, each model M_i, $i \in [1,2,3,\ldots,n_m]$, models one pose of the object to be detected.

A model consists of a set of binary feature vectors (square subtemplates) arranged with respect to each other in image coordinates. Each model comes from a training image of the object in a known pose. We get binary edge images using dilated zero crossings of Laplacian of Gaussian versions of the training images. The Gaussian filter has $\sigma = 2$, and we dilate with an n_d x n_d structuring element of 1's with $n_d = 3$. The idea to dilate the edges comes from [[3]], and we do so to help ensure some overlap between training features and possibly corrupted actual features. We designate the edge training images as $e_i(x,y)$, where $i \in [1,2,3,\ldots,n_m]$ indexes the model and $x \in [0,1,2,3,\ldots,n_x - 1]$ and $y \in [0,1,2,3,\ldots,n_y - 1]$ give the pixel coordinates on the n_x x n_y image. We eliminate the background by making binary masks using backlighting.

A model M_i represents features that are square patches of dilated edge points from $e_i(x,y)$. Using an idea from [[9]], we pick patches that are easy to localize in that they do not correlate well with their surroundings.

We introduce a windowing operator $\overline{\Omega}\{f(x,y);x_0,y_0,b\}$ that extracts a square region of size $(2b+1)$ x $(2b+1)$ pixels centered around (x_0,y_0) in $f(x,y)$ and scans the pixels in raster order into a column vector. The windowed neighborhood around each pixel in $e_i(x,y)$ is rated as a feature based on

$$r_i(x,y) = \min_{\substack{-d \le d_x \le d \\ -d \le d_y \le d}} \Big\{ D_H\Big[\overline{\Omega}\{e_i(x',y');x,y,b\}, \overline{\Omega}\{e_i(x',y');x+d_x,y+d_y,b\}\Big]\Big\}. \quad (1)$$

$D_H(\bar{r},\bar{s})$ is the Hamming distance between binary vectors $\bar{r}$ and $\bar{s}$. The Hamming distance simply counts the number of unequal elements in corresponding positions of its two arguments. In words, $r_i(x,y)$ is computed by taking a square of dimension $(2b+1)$ x $(2b+1)$ pixels centered on (x,y) in $e_i(x,y)$ and computing its Hamming distance with equal sized squares of pixels centered in the surrounding d x d neighborhood of $e_i(x,y)$. The minimum of these Hamming distances is the rating of the feature. The feature will rate highly if it is dissimilar to its surroundings. For our program, we chose $b = 7$ pixels to give binary features of size 15x15. We chose $d = 3$ pixels.

The best feature is taken as the $(2b+1)$ x $(2b+1)$ square surrounding the maximum value in $r_i(x,y)$. Subsequent features are chosen as squares surrounding the next highest value in $r_i(x,y)$ that does not overlap any previous features. Nominally, we chose $n_f = 40$ features based on the analysis in Section 5. The features chosen for one model of one of our objects of interest are shown in Figure 2.

The binary features are scanned into column vectors called $\bar{f}_{ij}$ with $i \in [1,2,3,\ldots,n_m]$ indexing the models and $j \in [1,2,3,\ldots,n_f]$ indexing the features within the models. The corresponding locations of the features in the training images are $\bar{x}'_{ij} = (x'_{ij}, y'_{ij})$. Since the absolute location of the object in the training image is unimportant, we take the feature locations as offsets with respect to the first feature in each model: $\bar{x}_{ij} = \bar{x}'_{ij} - \bar{x}'_{i1} = (x'_{ij} - x'_{i1}, y'_{ij} - y'_{i1})$.

By picking an independent set of features from each training image, we are not forced to depend on tracking training features from pose to pose. This is difficult if features appear and disappear due to specular highlights or self-occlusions.

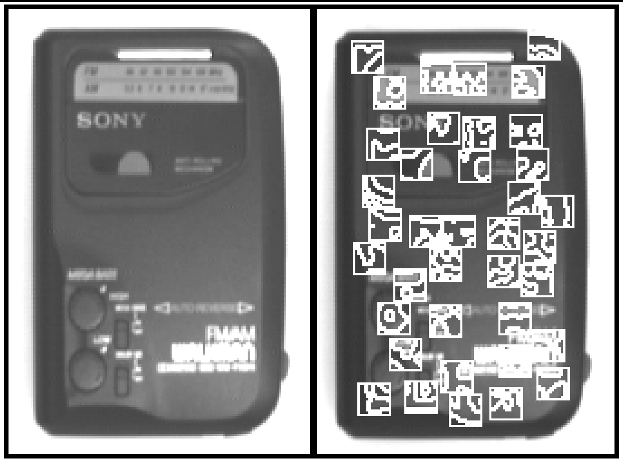

Figure 2: Binary features automatically selected for modeling object

As we mentioned above, our problem is one of detecting objects in stable poses resting on a plane with a camera looking down from above (*e.g.* conveyor belt). We model translation of the object parallel to the resting plane as proportional translation in the image, which assumes away effects of parallax and image distortion, *i.e.* the image of the object is shift-invariant. The degree of freedom that we model, then, is rotation around an axis perpendicular to the resting plane. Each model M_i models the object at angle θ_i.

We can compute the number of models n_m we need in this situation by considering the feature size. The training images of the object are separated by $\Delta\theta$ in angle. This angle should be small enough that a binary feature should not change appearance over the range $[\theta_i - \Delta\theta/2, \theta_i + \Delta\theta/2]$. Measured from the center of a $(2b+1)$ x $(2b+1)$ feature, the distance to the center of the farthest pixel is $\sqrt{2}b$. This pixel will move along an arc of length $\sqrt{2}b\Delta\theta$ between training images. We endeavor to make this arc length much less than one pixel to ensure that the feature will not change over the $\Delta\theta$ range in angle. If we set $\sqrt{2}b(\Delta\theta) = 0.2$ pixels, then $b = 7$ gives $\Delta\theta = 1.16^o$. We set $\Delta\theta = 1.0^o$, giving $n_m = 360$.

3. Encoding Binary Features

The recent work in subspace methods for object recognition [[3], [4], 6-[8], [10]], as well as the standard principle component analysis of pattern recognition, can be thought of as applications of data compression. A set of high-dimensional training vectors are projected into a lower-dimensional space to serve as efficient models of the items in the training set. When new data is to be classified, it is compressed in the same way as the training data and then compared to the compressed version of the training data. The advantage of this approach is not only efficiency, but that the compression process groups similar features, thereby tolerating the inevitable variations in features from image to image. We verify this assertion at the end of this section.

Looking to standard methods in image compression, we found no close analogue of eigenvectors for binary data like our training features $\bar{f}_{ij}$. Huttenlocher *et al.*[[3]] use binary whole-object templates compressed with real-valued eigenvectors to give real-valued compressed versions of the binary templates. One of the goals of our work was to avoid using any floating point processing on binary data. Thus, instead of eigenvectors, we chose to use binary vector quantization, which preserves the binary nature of the data in its compressed form.

Traditionally, the goal of vector quantization has been

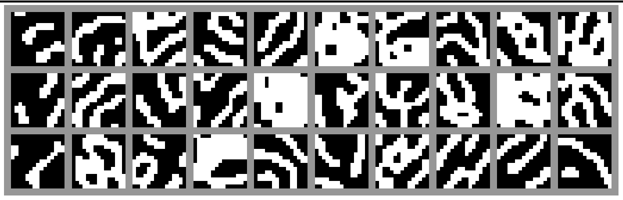
Figure 3: 30 code features from all training images of object in Figure 2.

image compression[[1]]. Each rectangular block in the image to be compressed is represented by the index of the most similar code block. In our case, we have a set of $n_m n_f$ training features $\bar{f}_{ij}$ that we want to represent with a much smaller set of n_c code features $\bar{F}_a$, $a \in [1,2,3,\ldots,n_c]$. Each code feature is the same size as the training features, *i.e.* $(2b+1)^2$.

We use the standard method for computing code features, the Lloyd algorithm, described in [[1]]. This is the same as the k-means algorithm from pattern recognition. The algorithm iteratively recomputes new sets of code features based on the centroids of nearest-neighbor clusters of training features. To measure distance between features, we use the Hamming distance, described above. In our case, we found that 10 iterations were enough to produce a good set of code features.

To compute the centroid of a set of binary vectors based on the Hamming distance, we have corresponding elements of each vector vote for "0" or "1". For determining each element of the centroid, the majority rules, with ties broken randomly.

The result of the Lloyd algorithm is a mapping from any training feature $\bar{f}_{ij}$ to its corresponding code feature $\bar{F}_{c_{ij}}$ whose index is c_{ij}. 30 code features for all 360 training images of the object in Figure 2 are shown in Figure 3.

We can assess the quality of the mapping by computing the probability that a feature will be miscoded. This helps to select the number of codes, n_c, that we will choose, as described in Section 5. In order to approximate the probability of miscoding, we took 360 test images of the object in Figure 2 at angles halfway between the $n_m = 360$ training images. For each feature $\bar{f}_{ij}$, we extracted a test feature at the same relative location as the training feature in the two adjacent test images. We coded these test features and took the probability of miscoding as the fraction of these features that were *not* mapped to the same code as their respective training feature. We repeated the experiment for different numbers of code features, n_c, rerunning the Lloyd algorithm each time we changed n_c. The result is plotted in Figure 4. Of course, this experiment does not account for all anticipated variations of the features (*e.g.*

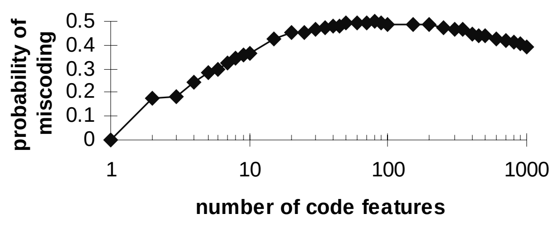

Figure 4: An experiment shows that the probability of miscoding a feature rises and then falls with the number of code features chosen.

illumination effects), but it does give an approximate idea of miscoding probabilities as well as the general behavior of the detection algorithm with variations in n_c.

As we expect, a small number of code features leads to a small probability of miscoding. This is because the nearest neighborhood of each code feature is large, so the chance of straying to another neighborhood is small. This supports our assertion at the beginning of this section that one advantage of subspace methods is that small appearance variations are tolerated by the many-to-one mapping. Based on this data, and the analysis in Section 5, we chose $n_c = 30$ codes, which gives a probability of miscoding of $p_b = 0.464$.

Figure 4 shows that the probability of miscoding peaks at about $n_c = 80$ and then starts to drop. We did not expect this behavior, and we are still speculating on the cause. It may be that for large numbers of code features, the code features represent training features that really belong together in some sense, while for smaller numbers of code features, there is a "forced marriage" between features that are really not very similar. An interesting extension of this work would be to explore the implications of large numbers of code features.

4. Detecting an Object

An object model M_i consists of a set of feature codes and locations for one pose of the object:

$$M_i = \left\{ (\bar{x}_{i1}, c_{i1}), (\bar{x}_{i2}, c_{i2}), \ldots, (\bar{x}_{in_f}, c_{in_f}) \right\}. \tag{2}$$

We search for these features in a preprocessed image in order to detect the object.

The input image is processed the same way as the training images to give $e(x,y)$, an image of dilated Laplacian of Gaussian zero-crossings. We then code the $(2b+1) \text{ x } (2b+1)$ neighborhood around each pixel with the code features $\bar{F}_a$ computed from the Lloyd algorithm. The corresponding image of code indices is

$$c(x,y) = \underset{a \in [1,2,\ldots,n_c]}{\arg\min} \left\{ D_H \left[\bar{\Omega}\left\{ e(x',y'); x, y, b \right\}, \bar{F}_a \right] \right\} \tag{3}$$

To detect an object, we search the image for all the models M_i of the object. Since we assume that the object's appearance is space-invariant, we search over all translations of all models M_i. We keep track of the search with a three-dimensional Hough transform H_{ixy}, whose three indices correspond to the three search parameters:

$$\begin{aligned} & i \in [1,2,3,\ldots,n_m] \text{ indexes models} \\ & \text{x} \in [0,1,2,\ldots,n_x - 1] \text{ indexes image column} \\ & \text{y} \in [0,1,2,\ldots,n_y - 1] \text{ indexes image row} \end{aligned} \tag{4}$$

The bins of the Hough transform keep track of how many features were found for each model M_i at each location in the image.

The Hough transform is filled by consulting a simple geometric hash table upon encountering each code index in $c(x,y)$. Each code index lends evidence to several different model and translation possibilities. Upon encountering a particular code index, the algorithm indexes into the hash table to find models that contain that code. Each occurrence of an equivalent code in a model causes one bin of the Hough transform to be incremented. The incremented bins correspond to the model index i and the position of the first feature on that model translated by the position of the code index in $c(x,y)$. A standard geometric hash table must be indexed by pairs of points (or more)[[5]]. Since we have a distinct model for each pose of the object, we can index on single points.

Each bin in the filled Hough transform contains a count of the number of features found that support a given model at a given location. We declare a detection of model M_{i_0} at location (x_0, y_0) if $H_{i_0 x_0 y_0} \geq k_T$. The integer k_T is a detection threshold that specifies the number of features that must be found for the object be considered detected. Our nominal value is $k_T = 13$, based on our analysis in Section 5.

We tested the program on 1000 images of the unoccluded object in a cluttered background (images like Figure 1 but with no occlusion). Of these 1000, the object was correctly detected on 99.3% with at least $k_T = 13$ features and with no other detections exceeding the number of features correctly found. We also tested the algorithm on images with partial occlusion as in Figure 1. The algorithm works in such cases, but we have not performed a statistically significant test.

After offline training, the program takes 5.5 minutes to compute the pose of an object in a new image, running on a 50 MHz Sun Sparc 10. The bulk of the time is devoted to encoding the image (≈ 2.5 minutes) and filling the Hough transform (≈ 3.0 minutes).

5. Receiver Operating Characteristics

The “receiver operating characteristic” is a way of measuring the performance of a signal detection algorithm[[11]]. The receiver operating characteristic graphically shows the tradeoff between the probability of a detection and the probability of a false alarm. The tradeoff in our case is controlled by the detection threshold k_T. If k_T is low, then the algorithm is more likely to find the object, even if many of the features are not found. But, a low k_T will also increase the chance of a false alarm. Conversely, a high k_T will decrease the chance of a false alarm, but also decrease the chance of a valid detection. The receiver operating characteristic is useful for assessing and optimizing the performance of the algorithm.

Traditionally, the receiver operating characteristic considers the case of a single (possibly multidimensional) measurement. For our case, it is more useful to modify the receiver operating characteristic to show what we are most interested in for a computer vision application. We will base our probability calculations on the algorithm’s behavior on a search through the entire image rather than just a single measurement. The two probabilities that we calculate are:

$P(\text{detection}) =$ probability that algorithm will find correct model in correct location given that unoccluded object appears somewhere in image

$P(\text{fase alarm}) =$ probability that algorithm will find any model anywhere in image given that object does not appear anywhere in image

This analysis is similar in spirit to that of Grimson and Huttenlocher[[2]], who analyze the Hough transform for pose measurement. However, they analyze a situation where each feature fills a swath of bins in the Hough transform, while our formulation only fills several disconnected bins. They warn that the probability of false positives can be very high, but our theory and experiment show that it is easy to bring the probability of a false positive arbitrarily low with our algorithm.

5.1 Probability of False Alarm

We will begin by computing the probability of a false alarm, since this is easier, and the same techniques will be used in the next section. Our algorithm reports a detection if it finds at least one model M_i somewhere in the image supported by at least k_T features. For lack of a reason to believe otherwise, we will assume that the feature codes in an image with no object are uniformly distributed. Given this, the probability of observing a given feature index at a given location in $c(x,y)$ is n_c^{-1}, where n_c is the number of codes we choose to use. When the algorithm considers the existence of a given model at a given location, it looks for n_f code indices in a certain geometrical arrangement in $c(x,y)$. The probability of finding any subset $m \subset M_i$ of exactly l features at l different locations and not finding specific features at the remaining $n_f - l$ locations in $c(x,y)$ is given by a binomial distribution:

$$b\left(l; n_f, n_c^{-1}\right) = \binom{n_f}{l} n_c^{-l}\left(1 - n_c^{-1}\right)^{n_f - l} \qquad (5)$$

The probability of falsely reporting a detection of a given model *at a single location* is the sum of the probabilities of finding between k_T and n_f features:

$$p_f = \sum_{l=k_T}^{n_f} b\left(l; n_f, n_c^{-1}\right). \qquad (6)$$

It is clear from this equation that a lower value of the detection threshold k_T increases the chance of a false detection.

The probability of a false alarm, as we have defined it, is the probability that a model with at least k_T features will be found somewhere in the image. We calculate this probability as follows:

$$\begin{aligned} &P(\text{false alarm}) \\ &= P(\text{finding } \geq k_T \text{ features somewhere in image}) \\ &= 1 - P(\text{not finding } \geq k_T \text{ features anywhere in image}) \quad (7) \\ &= 1 - b\left(0; n_m n_x n_y, p_f\right) \\ &= 1 - \left(1 - p_f\right)^{n_m n_x n_y} \end{aligned}$$

where we calculate the probability of not finding $\geq k_T$ features anywhere in image as a binomial probability with zero successes out of $n_m n_x n_y$ tries with a probability of success on any given trial of p_f. As we expect, the probability of a false alarm rises with the size of the image ($n_x n_y$) and the number of models searched for (n_m).

5.2 Probability of Detection

A successful detection occurs when the algorithm finds at least k_T features on the object at the correct pose and no greater number of features from a single model anywhere else. Since we assume that the problem is translation invariant, the object’s position in the image makes no difference, and we start with the following probability of a successful detection:

$$P(\text{detection}) = \sum_{i=1}^{n_m} P\left(\text{finding model } M_i \middle| M_i\right) P\left(M_i\right). \qquad (8)$$

By the event “M_i” we mean that the object is at the orientation represented by M_i. We will assume that instances of the model in the images are uniformly distributed, so that $P(M_i) = n_m^{-1}$.

We further specify the probability of a detection as

$$n_m^{-1}\sum_{i=1}^{n_m} P\left(\begin{array}{l}\text{finding } m \subset M_i \text{ such that } S(m) \geq k_T \\ \text{at correct location } \textit{and} \text{ not finding} \\ \text{any } \tilde{m} \subset M_j \text{, j} \in [1,2,3,\ldots,\text{n}_\text{m}] \text{ such} \\ \text{that } S(\tilde{m}) \geq S(m) \text{ at any other location} \\ \text{in image} \mid M_i \end{array}\right). \qquad (9)$$

$S(m)$ is the number of elements in the set m. The event specified in this equation is that of finding at least k_T features out of the set of features that indicates the model and not finding a larger set of features from the models anywhere else. We split this term into mutually exclusive events based on the number of features found on the correct model, noting also that the two statements connected by "*and*" are independent events:

$$n_m^{-1}\sum_{i=1}^{n_m}\sum_{k=k_T}^{n_f}\Bigg\{P\left(\text{finding } m \subset M_i \text{ such that } S(m) = k \middle| M_i\right)$$
$$\left[1 - P\left(\begin{array}{l}\text{finding any } \tilde{m} \subset M_j, j \in [1,2,3,\ldots,n_m] \\ \text{such that } S(\tilde{m}) \geq k \text{ at any other} \\ \text{location in image}\end{array}\right)\right]\Bigg\} \qquad (10)$$

The correct way to compute the first term in the product above is to consider all possible subsets $m \subset M_i$. Since there are so many possibilities, this is prohibitively expensive in computer time. Instead, we assume that each feature has the same probability of being miscoded, p_b, as computed at the end of Section 3. Then the probability of finding exactly k features and miscoding $n_m - k$ features for all possible sets m such that $S(m) = k$ is

$$b(k; n_f, 1 - p_b) = \binom{n_f}{k}(1 - p_b)^k (p_b)^{n_f - k}. \qquad (11)$$

For the second term in the product in Equation 12, we assume again that the feature codes in the background of the image are uniformly distributed. The probability of finding any specific combination ($\tilde{m} \subset M_j$) of k or greater features at a single location is

$$p_m = \sum_{l=k}^{n_f} b(l; n_f, n_c^{-1}). \qquad (12)$$

Omitting the correct pose, there are $n_m n_x n_y - 1$ opportunities to find k or greater features from any model. The second term in the product in Equation 12 becomes

$$1 - P(\text{finding } \geq k \text{ features somewhere in image})$$
$$= 1 - \sum_{i=1}^{n_m n_x n_y - 1} b(i; n_m n_x n_y - 1, p_m)$$
$$= 1 - [1 - b(0; n_m n_x n_y - 1, p_m)] \qquad (13)$$
$$= (1 - p_m)^{n_m n_x n_y - 1}.$$

Noting that we have assumed away any dependence on the particular model M_i, the probability of detection is

$$P(\text{detection}) = \sum_{k=k_T}^{n_f}\left\{b(k; n_f, 1 - p_b)\left[1 - \sum_{l=k}^{n_f} b(l; n_f, n_c^{-1})\right]^{n_m n_x n_y - 1}\right\}. \qquad (14)$$

5.3 Receiver Operating Characteristic Curves

Receiver operating characteristic curves are generally parameterized by a detection threshold. In our case, the detection threshold is k_T, which gives the number of features that must be found to consider the object present. A good receiver operating characteristic curve will look like the symbol Γ, with the vertical segment coincident with the vertical axis, and the corner being at point (0,1). Such a curve means that the false alarm rate stays low even for high probabilities of detection. Our goal is to adjust parameters in an attempt to reach an ideal curve shape and then take k_T to be the value at the upper left corner of the curve. Figure 6 shows receiver operating characteristic

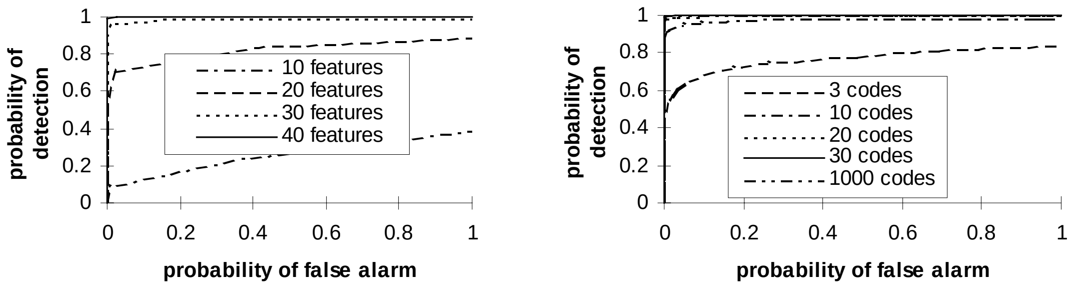

Figure 5: Theoretical receiver operating characteristic as a function of detection threshold. For varying numbers of features, the number of codes was kept constant at 30. For varying number of codes, the number of features was kept constant at 40. 40 features with 30 codes are adequate. The bends in the curves occur at a detection threshold of 13 features.

curves based on the equations above. The left plot of Figure 6 shows the effect of varying the number of features with the number of codes kept constant at $n_c = 30$. It shows that 40 features give a good curve. The sharp bend occurs at $k_T = 13$.

The right plot of Figure 6 shows the effect of varying the number of codes with the number of features kept constant at $n_f = 40$. It shows that 30 codes give a good curve, with the sharp bend in the curve occurring at $k_T = 13$. This is how we chose the number of features, number of codes, and detection threshold.

In our test of 1000 images, we kept track of the number of features found for the top 10 detections for each image. Using this data, we plotted an empirical receiver operating characteristic curve, as show in Figure 7. We also plotted the theoretical receiver operating characteristic curve ($n_f = 40$ and $n_c = 30$). As shown, the false alarm rate is very well predicted by Equation 9. Equation 16 tends to overestimate the probability of detection slightly, which could be due to a higher probability of miscoding than what our experiment in Section 3 showed.

Nearly every computer vision algorithm comes with parameters that must be adjusted by the user - so-called "magic numbers". We list the magic numbers used by our program in Table 1. The ideal computer vision algorithm has no magic numbers, which means it does not have to be adjusted for different situations. The next best alternative is to provide a principled method to choose the parameters. As shown in Table 1, we were able to do this for about half the adjustable parameters, with the receiver operating characteristic accounting for three of the most important.

6. Summary

This paper presents a new algorithm for detecting objects in images. It uses models based on training images of the object, with each model representing one pose. Since each pose is modeled uniquely, this helps reduce the confounding effects of specular highlights, and eliminates the need to track features during training. Objects are modeled in terms of square binary edge patches that are automatically selected from the training images based on their dissimilarity with their surroundings. Internal features means the algorithm is robust in the face of background clutter. The features are compressed using binary vector quantization, which gives an efficient representation of the models. The detection algorithm fills a 3D Hough transform. We derive the probabilities of detection and false alarm (receiver operating characteristics) and use this analysis to determine some of the important operating parameters of the program.

References

[1] Gray, Robert M., "Vector Quantization", *IEEE ASSP Magazine*, April 1984, pp. 4-29.

[2] Grimson, W. Eric L. and Huttenlocher, Daniel P. "On the Sensitivity of the Hough Transform for Object Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3), March 1990, pp. 255-274.

[3] Huttenlocher, Daniel .P., Lilien, Ryan H., and Olson, Clark F. , "Object Recognition Using Subspace Methods", *Proceedings of the Fourth European Conference on Computer Vision*, 1996, pp. 536-45.

[4] Krumm, John C., "Eigenfeatures for Planar Pose Measurement of Partially Occluded Objects", *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, June 1996, pp. 55-60.

[5] Lamdan, Yehezkel and Wolfson, Haim J., "Geometric Hashing: A General and Efficient Model-Based Recognition Scheme", *Proceedings of the Second International Conference on Computer Vision*, December 1988, pp. 238-249.

[6] Murase, Hiroshi and Nayar, Shree K., "Visual Learning and Recognition of 3D Objects from Appearance", *International Journal of Computer Vision*, 14(1), 1995, pp. 5-24.

[7] Murase, Hiroshi and Nayar, Shree K., "Image Spotting of 3D Objects using Parametric Eigenspace Representation", *9th Scandinavian Conference on Image Analysis*, June 1995, 325-332.

[8] Ohba, Kohtaro and Ikeuchi, Katsushi, *Recognition of the Multi Specularity Objects using the Eigen-Window*, Carnegie Mellon University School of Computer Science Technical Report CMU-CS-96-105, February 1996.

[9] Shi, Jianbo and Tomasi, Carlo, "Good Features to Track", *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, June 1994, pp. 593-600.

[10] Turk, Matthew and Pentland, Alex, Eigenfaces for Recognition", *Journal of Cognitive Neuroscience*, 3(1), 1991, 71-86.

[11] Van Trees, Harry L., *Detection, Estimation, and Modulation Theory, Part I, Detection, Estimation, and Linear Modulation Theory*, John Wiley & Sons, New York, 1968.

parameter	value	how set
image size (n_x, n_y)	(512,480)	preset
smoothing σ	2 pixels	by eye
dilation n_d	3 pixels	experience
feature size b	7 pixels	experience
feature correlation distance d	3 pixels	experience
number of models n_m	360	subpixel feature change (Section 2)
probability of miscoding p_b	0.464 (for $n_c = 30$)	miscoding experiment (Figure 4)
number of features n_f	40	receiver operating characteristic
number of codes n_c	30	receiver operating characteristic
detection threshold k_T	13 features	receiver operating characteristic

Table 1: Settings of parameters (magic numbers)