IEEE Conference on Computer Vision and Pattern Recognition, Hilton Head, SC, June 2000

Object Recognition for an Intelligent Room

Richard Campbell
Department of Electrical Engineering
The Ohio State University
Columbus, OH 43210
campbelr@ee.eng.ohio-state.edu

John Krumm
Microsoft Research
Microsoft Corporation
Redmond, WA 98052
jckrumm@microsoft.com

Abstract

Intelligent rooms equipped with video cameras can exhibit compelling behaviors, many of which depend on object recognition. Unfortunately, object recognition algorithms are rarely written with a normal consumer in mind, leading to programs that would be impractical to use for a typical person. These impracticalities include speed of execution, elaborate training rituals, and setting adjustable parameters. We present an algorithm that can be trained with only a few images of the object, that requires only two parameters to be set, and that runs at 0.7 Hz on a normal PC with a normal color camera. The algorithm represents an object's features as small, quantized edge templates, and it represents the object's geometry with "Hough kernels". The Hough kernels implement a variant of the generalized Hough transform using simple, 2D image correlation. The algorithm also uses color information to eliminate parts of the image from consideration. We give our results in terms of ROC curves for recognizing a computer keyboard with partial occlusion and background clutter. Even with two hands occluding the keyboard, the detection rate is 0.885 with a false alarm rate of 0.03.

1. Object Recognition for an Intelligent Room

This paper introduces a new object recognition algorithm that is especially suited for finding everyday objects in an intelligent environment monitored by color video cameras. For a typical room in a home or office building, we could base the following applications on object recognition:

- Customize a device's behavior based on location. A keyboard near a computer monitor should direct its input to the applications on that monitor. A keyboard in the hands of a particular user should direct its input to that user's applications, and it should invoke that user's preferences (*e.g.* repeat rate on keys).
- Find lost objects in a room like a television remote control.
- Infer actions and intents from which objects are being used. A user picking up a book probably wants to read, and the lights and music should be adjusted appropriately.

Besides the usual requirements for being robust to background clutter and partial occlusion, these applications share a need for moderate speed on cheap hardware if they are ever to be included in a consumer product. The main elements of the algorithm we develop – color lookup, edge detection, vector quantization, and image correlation – are standard image processing tasks that can run quickly on a normal PC. For recognizing a single object like a computer keyboard, our program runs at 0.7 Hz on a 500 MHz PC.

We are also sensitive to the need for a simple training procedure if this algorithm is ever to become a consumer product. While many object recognition routines require several carefully controlled training views of the object (*e.g.* on a motorized turntable) or detailed, manual feature selection, our algorithm only requires the user to outline the object of interest in one image for every unique face of the object. The training images for different poses are generated synthetically from the actual images.

In addition, we cannot expect a normal user to set several parameters for the program. Our program requires the setting of only two parameters, a sensitivity level for eliminating certain pixels from consideration based on their color and a detection threshold.

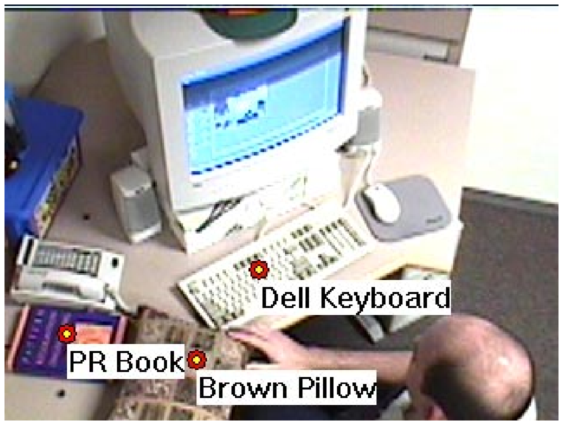

Figure 1: Our algorithm locates common objects that can be useful for interacting with intelligent rooms.

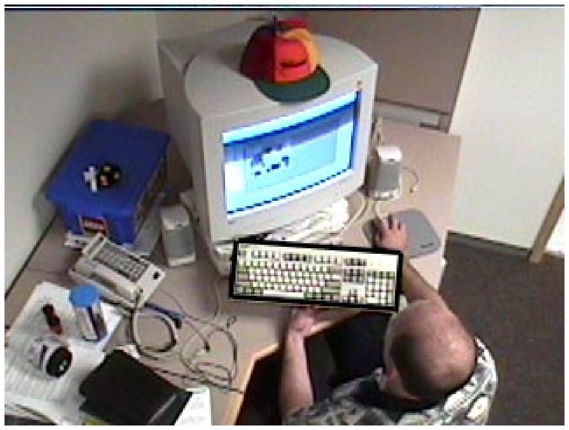

Figure 2: A user manually outlines the keyboard to make the single required training image.

Our problem started with an effort to recognize computer keyboards in images for the purpose of an intelligent room, as shown in Figure 1. We model the keyboard with a partial covering of 7x7 edge features, each of which has been matched to one of 64 prototype features. These prototype features are derived from training features using a new method of quantizing binary edge templates. Each prototype edge feature in an image gives evidence for thousands of possible locations for the center of the keyboard, which we encode with a "Hough kernel". The Hough kernels replace the "R-table" of the traditional generalized Hough transform [1], and are applied to the test image using image correlation. Using each of the 64 Hough kernels, we accumulate votes for the center of the keyboard. If any of the votes exceeds the detection threshold, we declare the keyboard found.

Using many small edge templates means the algorithm can tolerate partial occlusion, because we can set the detection threshold low enough that it does not require all the features to contribute a vote. Also, using edges instead of, say, gray scale or color templates, means that the algorithm is somewhat robust to changes in illumination, because edges are somewhat robust to the same.

This paper's main contribution is an object recognition algorithm that is simple enough for normal consumers to use on their own everyday objects. The algorithm's simplicity and efficiency are due to our new way of quantizing binary edge features and to our correlation-based implementation of the generalized Hough transform.

2. Training

We train our recognition algorithm by showing it images of the objects we want it to recognize. These training images are converted into object models, discussed in the next section. In modern object recognition, training is nearly always done on actual or simulated images of the object to avoid the labor-intensive process of constructing the models manually. Typically training images show the object in many different poses so the object model can cover all the anticipated appearances of the object. However, a normal consumer would have access to neither a renderable model of the object he or she wants to have recognized nor to a convenient means of presenting the object to the camera in many different positions and orientations. Instead, we require only that the user show a few images of the object to the training program. These images should ideally show the major faces *(i.e.* aspects) of the object that the camera will see. In our experiments with mostly flat objects with only one interesting face, a single training image was enough. After snapping an image showing the object, the user manually outlines the object to make a training image, as shown in Figure 2 for a computer keyboard.

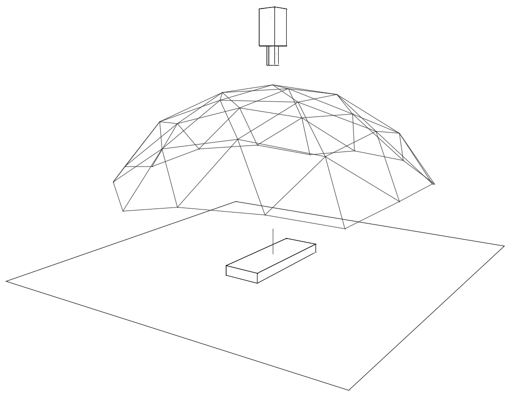

Figure 3: Synthetic training images are produced by rendering object as if its surface normal were pointed at the vertices of a tessellated, partial hemisphere and rotating around the object's surface normal.

With the training images extracted by the user, we synthetically generate more views of the object at other orientations. This works best if the actual training image has the normal vector of the object's face roughly pointing at the camera. We simulate rotation around the object's normal vector by simply synthetically rotating in the image plane. We simulate out-of-plane rotation by synthetically pointing the object's face at 31 nodes of a tessellated hemisphere as shown in Figure 3. In these out-of-plane rotations we simulate simple scaled orthographic projection, meaning that we do not account for the camera's focal length. The 31 nodes of the hemisphere are spaced at approximately 20 degrees, and we have omitted nodes within 20 degrees of the equator of the hemisphere to avoid nearly edge-on views. We orthographically project the training image for each node of the hemisphere and then rotate this image in the image plane in 20-degree increments. This gives 31x18

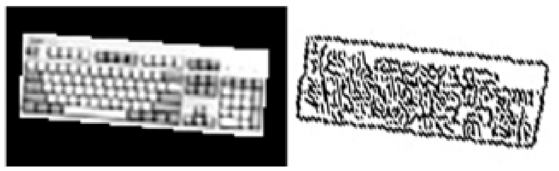

Figure 4: Synthetic training image and corresponding edges.

= 558 synthetic training images for each face of each object. One synthetic training image for the keyboard is shown in Figure 4.

We have intentionally neglected accounting for scale changes, as we found it was not necessary for the tests we did where the objects stayed generally close to the desktop near which we took the training images. It is easy to account for scale, however, by scaling the synthetic training images.

We detect edges on each synthetic training image with a Canny edge detector on the images' intensity component ($R+G+B$). We chose the Canny detector due to its generally good performance and speed [2]. Figure 4 shows the edges detected on one of the synthetic training images. Our raw features are 7x7 subwindows centered on each edge point in each training image. We designate these raw features as $f_i(x,y)$, with $f_i(x,y)\in[0,1]$, $x,y\in[0,1,2,\dots l_f-1]$, $l_f=7$, $i\in[1,2,3,\dots n_f]$, and n_f equal to the number of edge points in all the training images for a particular face of a particular object. Since each synthetic training image produces about 1000 edge points, and since we have 558 training images for the face of each object, we have an n_f of about ½ million. For each raw feature $f_i(x,y)$ we also record a 2D offset vector $\vec{c}_i$ going from the pixel coordinates of the feature's center on the training image to the pixel coordinates of the object's centroid. We round the elements of this vector to the nearest integer to save space. These vectors are used to accumulate evidence for the location of the centroid of the object in the image when a feature is detected, as explained in Section 4.

3. Binary Image Quantization

The ½ million raw features and their corresponding offset vectors cannot be used directly to locate the object in a new edge image. This is because the actual features are unlikely to ever be exactly reproduced due to image noise, in-between poses that were never trained on, and the effects of the object's surface relief that cannot be simulated from just one actual training view. In addition, looking for each of the ½ million features would make the program too slow.

We reduce the number of features and add noise tolerance by quantizing the raw features into 64 "prototype" features using a dendrogram. Each raw feature, from both training and test images, then maps to one of the 64 binary prototype features. The 64 prototype features for the keyboard object are shown in Figure 5. This is in contrast to the method of Ohba & Ikeuchi [7] who also use small feature templates to recognize objects. They use gray-level templates as opposed to our binary templates, and they use eigenvector principle components rather than binary quantization. Our method also contrasts with the traditional generalized Hough transform[1] which quantizes edge points based on their gradient, while we quantize based on the pattern of surrounding edge points. The idea of clustering the raw, binary features is most similar to the second author's work[6] using binary quantized features. However, there are significant difference in the generation of training images, the choice of raw features, clustering, and the implementation of the Hough transform.

We cluster features with a dendrogram[4], which is a means of clustering vectors by progressively merging the cluster centers that are nearest each other. In our case the vectors are the n_f raw, binary features. The dendrogram starts with each raw feature as its own cluster center and then merges the two raw features that are nearest each other. This merge produces a new cluster center that is representative of its member features. We continue this process of merging until the raw features have been reduced to a relatively small number of cluster centers. In particular, we continue merging until there are 128 cluster centers remaining and choose the $n_p=64$ out of these 128 that have the most raw features associated with them.

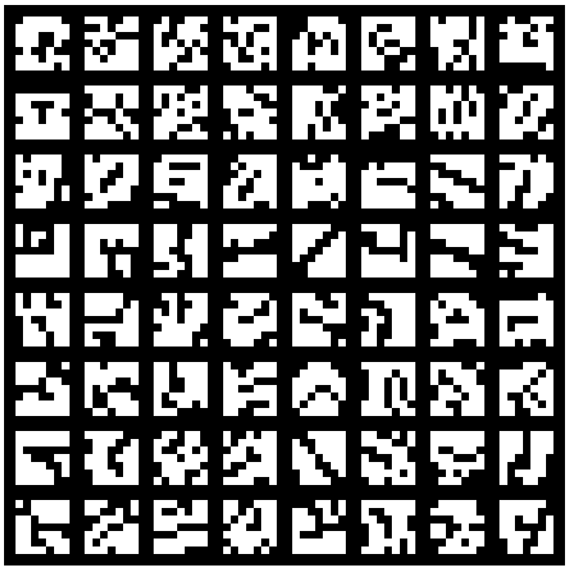

Figure 5: 64 prototype features from keyboard found with a dendrogram.

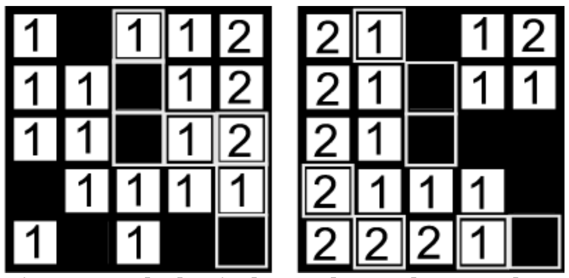

Figure 6: Black pixels on these edge templates represent edge pixels, and numbers represent grassfire transform values. Pixels surrounded by a gray box show locations of edge pixels in other template. Distance between templates is the sum of the numbers in the gray boxes, with zero implied for black boxes. Distance here is (1+1+2+1)+(1+2+2+1) = 11.

The two key elements of the dendrogram algorithm are a method to measure the similarity of features and a method to compute a representative feature of a group of features. For feature vectors whose elements are real valued, similarity is normally measured by distance (e.g. Euclidian, Mahalanobis), and the representative feature of a group is simply the group's centroid. For binary feature templates, we created our own methods for measuring similarity and for computing a representative prototype.

We measure similarity with a Hausdorff-like distance. The Hausdorff distance for use on binary images was introduced by Huttenlocher *et al.* in 1993[5]. For a feature $f(x,y)$ and prototype $p(x,y)$ (both 7x7 binary), we compute the grassfire transform of each one, $G_f(x,y)$ and $G_p(x,y)$, using 8-connected neighbors. Figure 6 shows the grassfire transform of a raw feature and a prototype feature. If edge points from the two features coincide at the same location, they contribute zero to the distance. For those edge points that do not coincide, we take the grassfire value in the other image as its contribution to the distance. In equation form, the distance is

$$D_{fp} = \sum_x \sum_y \{G_f(x,y)p(x,y) + G_p(x,y)f(x,y)\} \quad (1)$$

where an edge point in the features $f(x,y)$ and $p(x,y)$ is a one, and an edge point in the grassfire transforms $G_f(x,y)$ and $G_p(x,y)$ is a zero. Figure 6 is instructive in understanding this formula. In the dendrogram, we use this distance for deciding which pair of cluster centers to merge next. This grassfire-based method is better than a simple pixel-by-pixel comparison, because it accounts for the spatial closeness of edge pixels.

The other computational element needed for the dendrogram is a way to compute a single prototype feature that is representative of a set of raw features. For a group of features F, we compute the mean number of edge pixels m_F and the sum of the raw features' grassfire transforms, $G_F(x,y)$. In equation form, these are

$$m_F = \frac{1}{n_F} \sum_{i:f_i \in F} \sum_x \sum_y f_i(x,y) \quad (2)$$

$$G_F(x,y) = \sum_{i:f_i \in F} G_{f_i}(x,y) \quad (3)$$

where n_F is the number of raw features in group F.

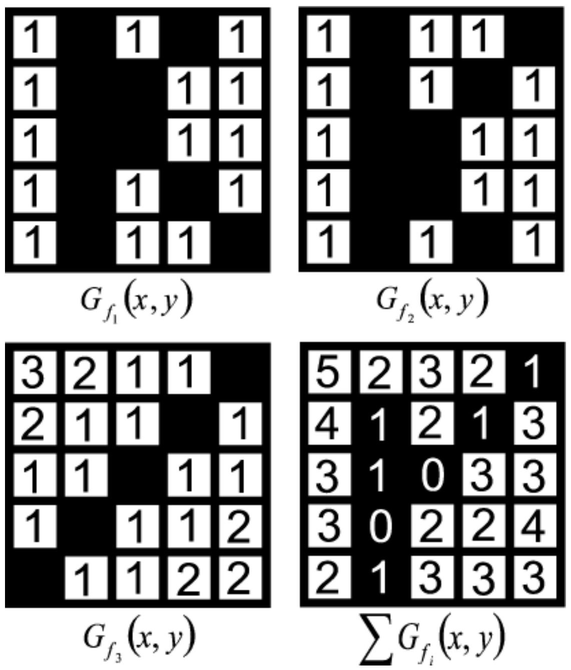

Figure 7: As an example, the grassfire transforms of three raw features are summed. The low parts of the summed grassfire are filled in to make a prototype feature as shown in the lower right. Here $m_F = 25/3$, and the prototype features has 7 edge points.

Summing up the grassfire transforms as in Equation (3) is shown in Figure 7. We fill in the low points of this sum with edge points until the number of edge points is as close as possible to m_F. The filled-in low points are a level set of $G_F(x,y)$. Intuitively, the low points represent those parts of the prototype feature that have the most nearby edge points in the raw features.

After running the dendrogram on the keyboard training images we get the $n_p = 64$ prototype features shown in Figure 5. To make training faster, we use only 10,000 randomly chosen raw features out of the approximately ½ million. The prototype features are used to convert each edge pixel in the training images into an integer $[1,2,3,\ldots,n_p]$ representing the prototype feature that this pixel's 7x7 surrounding region is most similar to, measured using the distance formula in

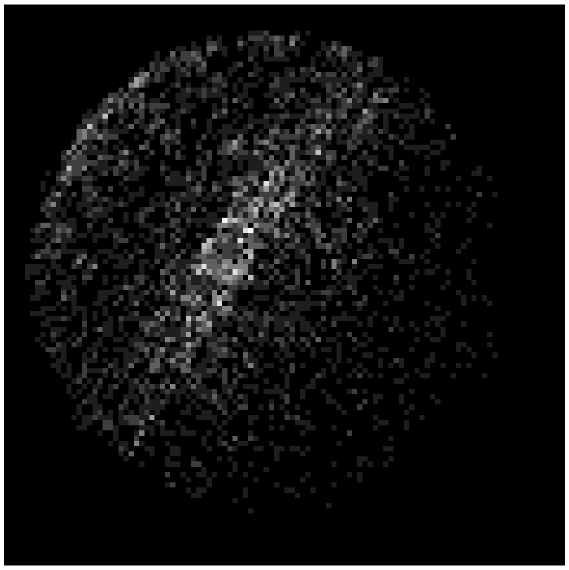

Figure 8: Hough kernel for one prototype feature. If this kernel's corresponding prototype feature occurs at the center of this image, then this kernel shows the magnitude and location of the votes that will be cast for the object's center in the vote image. These votes are cast using image correlation with slices of the indexed test image.

Equation (1). We called these images the "indexed" training images, because the edge pixels index into the array of prototype features. We do the same conversion for each edge pixel in our test images in which we want to find the object, giving "indexed" test images. Not only does this compact representation of the raw edge features reduce memory requirements and computation time in the recognition phase, it also helps the algorithm ignore small variations among edge features. That is, if a few pixels in a raw feature turn from edge to non-edge or vice versa, that feature will still likely map to the same prototype feature.

4. Hough Kernels

A "Hough kernel" is an efficient form of bookkeeping and execution for the generalized Hough transform. In the traditional generalized Hough transform as first described by Ballard[1], individual edge points on an object's silhouette are quantized by their gradient direction. For each gradient direction, an R-table records all the observed offset vectors to a reference point on the object. The R-table is then used to increment an accumulator array based on the gradient direction of each edge point in the test image. While Ballard describes the technique only for boundary points, the extension to interior edge points, which we use also, is obvious.

In our algorithm each edge point is quantized to an index value j based on that point's surrounding pattern of edges. Each occurrence of an index value j in an indexed test image gives evidence for thousands of different possible centroid locations of the object. The Hough kernel for this index value represents this evidence in an efficient and usable way.

The Hough kernels are built based on the indexed training images. Each edge point i in the training images has a prototype index j associated with it. It also has an associated relative pixel offset vector $\bar{c}_i$ that points to the object's centroid in the training image. Thus, each index j has a collection of different offset vectors associated with it. When we see a particular index j in a test image, it gives evidence for a number of different center offsets, and the Hough transform casts a vote for each one using a Hough kernel.

We encode the voting for a particular index j with a Hough kernel, $H_j(x,y)$. These kernels are used to build up a vote image, initially all zeros, for the object's centroid in the test image. When we encounter index j at a particular location in an indexed test image, we lay down a copy of $H_j(x,y)$ on the vote image, with the origin of $H_j(x,y)$ centered at that index's location in the vote image. Then we sum this translated $H_j(x,y)$ into the vote image. An image of a typical Hough kernel for the keyboard is show in Figure 8.

In equations, $H_j(x,y)$ is given by

$$H_j(x,y) = \sum_{\substack{i:\, f_i(x,y)\text{'s} \\ \text{prototype is} \\ p_j(x,y)}} \delta(\bar{\mathrm{x}} - \bar{\mathrm{c}}_\mathrm{i}) \qquad (6)$$

where

$$\delta(\bar{x}) = \begin{cases} 1 & \text{if } \bar{x} = (0,0) \\ 0 & \text{otherwise} \end{cases}$$

and $i \in [1,2,3,\ldots n_f]$, with n_f equal to the number of edge points in all the training images for that object. There is one Hough kernel for each prototype feature, giving 64 Hough kernels in our case.

The vote image is built up from correlations between each Hough kernel and equal-value slices of the indexed test image. If the indexed test image is $I(x,y)$, its equal-value slices are

$$I_j(x,y) = \begin{cases} 1 & \text{if } I(x,y) = j \\ 0 & \text{otherwise} \end{cases} \qquad (7)$$

and

$$I(x,y) = \sum_{j=1}^{n_p} j I_j(x,y) \qquad (8)$$

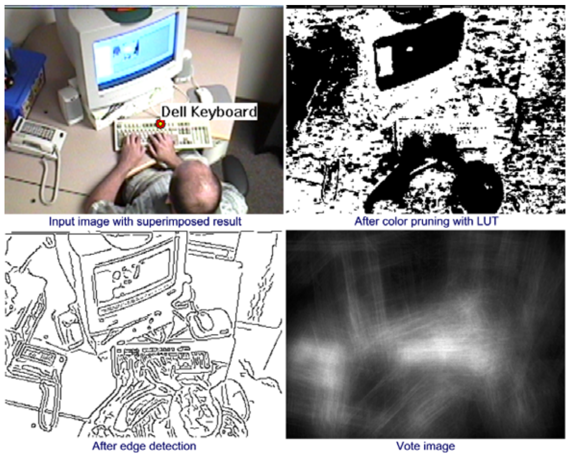
Dell Keyboard
Input image with superimposed result
After color pruning with LUT
After edge detection
Vote image

Figure 9: Result and intermediate images with occluded keyboard. Color pruning eliminates about 48% of the pixels. The vote image's peak is on the keyboard. Another peak (below the detection threshold) occurred on the telephone keypad.

Each "1" in an $I_j(x,y)$ indicates the presence of index j at that point. If we lay down a copy of $H_j(x,y)$ at that point, it will show how that point votes for the location of the centroid of the object.

Mathematically, the vote image $V(x,y)$ is

$$V(x,y)=\sum_{j=1}^{n_p}\left[H_j(x,y)\otimes I_j(x,y)\right] \tag{9}$$

where "$\otimes$" indicates correlation. This is essentially the generalized Hough transform implemented as a sum of correlations. Since correlation and convolution are closely related, we could implement Equation (9) with an FFT for speed.

The Hough kernel is similar to Brown's "feature point spread function"[3] which he used to analyze peak shapes in the traditional Hough transform.

Even though each Hough kernel casts thousands of votes for the object's centroid, we expect the votes from all the features to accumulate into a tall peak at the object's actual centroid. The vote image $V(x,y)$ is thresholded to find likely locations of the centroid. An example vote image is shown in Figure 9. We examine the effect of different threshold values using R.O.C. curves in Section 6, which shows that it is easy to find a threshold that finds a keyboard in 95% of the test images with only 3% of the test images giving a false alarm.

5. Pruning with Color

The object's color distribution is useful for preventing votes from certain pixels in the test image. We build a 256x256x256 boolean lookup table (LUT) that determines whether or not any (R,G,B) in the test image can likely be part of the object. R, G, and B are all $\in[0,1,2,\dots,255]$, and there is one entry in the LUT for each possible (R,G,B). The LUT is built by examining all the pixels on the object's actual (non-synthetic) test image. We project each (R,G,B) pixel in this image to the unit sphere by dividing by $\sqrt{R^2+G^2+B^2}$. This normalization helps eliminate the effects of illumination intensity and shadows. We establish a small acceptance region around each projected color pixel. Each (R,G,B) in the LUT is also projected to the unit sphere. If it is not within the acceptance region of any projected pixels of the test image, that LUT location is marked as a color that should be eliminated from consideration as part of the object. For objects whose colors are quite different from the background, this technique increases the speed and efficiency of the algorithm significantly. An example image that has been passed through the LUT is shown in Figure 9. Here we are looking for the beige keyboard, whose normalized colors happen to match well with much of the background, including the gray desk. The size of the acceptance region is one of the two parameters that must be set by the user. This is done interactively, with the user watching the output of the LUT step to see if the object's pixels are being passed through.

6. Finding Objects

This section summarizes our approach and presents results. We have tested our algorithm on a few different objects as shown in Figure 1. Our original goal was to find keyboards in images for an intelligent room, so we concentrate on that object in this section.

As a potential consumer application, the program is simple to run from a user's perspective. He or she is asked to present the keyboard to the camera, oriented roughly perpendicular to the camera's line of sight. He or she snaps an image and outlines the keyboard in this image using the mouse. The program has five parameters that must be set: the size of the smoothing window for the Canny edge detector, the low and high edge strength thresholds for the Canny edge detector, the color match tolerance for the color pruning step, and the detection threshold applied to the vote image. The algorithm is somewhat insensitive to the Canny parameters, so the user would not be expected to change their default settings. As explained above, the color match threshold is set interactively by watching which pixels come through the LUT step while the keyboard is moved around in the scene. The detection threshold is also set interactively by watching the peak value in the vote image as the object is brought into and out of the scene. Overall the training procedure is quite easy from a user's perspective.

With the hand-segmented image of the keyboard, the training program renders 558 synthetic training images of the object at different angles of roll, pitch, and yaw. These training images are converted to edge images, and then 7x7 edge subwindows are extracted around each edge point in the training images. These raw features are clustered with a dendrogram to produce 64 prototype features. Finally a Hough kernel is created for each prototype feature. The whole training process takes about five minutes on a 500 MHz PC, with the user's involvement limited to the very beginning (outlining the object) and very end (setting the color pruning threshold and detection threshold).

Running the recognition program is also pleasant from a user's point of view. For a single object on a 320x240 image, it runs at about 0.7 Hz on the same 500 MHz PC. The ROC curves in Figure 10 show how the performance varies with the choice of the detection threshold on the vote image. When the keyboard is not occluded the detection rate is about 0.955 with a false alarm rate of 0.03 on a test of 200 images with the keyboard and 200 images without the keyboard. The test images with the keyboard showed the keyboard in various orientations on and above the desk. The ROC plot shows that the performance drops significantly when the color pruning is turned off. The ROC curves also show how much the performance drops when the keyboard is occluded by two hands typing, a typical image of which is shown in Figure 9. For the same false alarm rate of 0.03 as in the non-occluded case, the detection rate with color pruning drops by only 0.07 from 0.955 to 0.885. As in the non-occluded case, turning off the color pruning seriously degrades performance, although, curiously, not as much as in the occluded case as the non-occluded case.

The fact that the program performs as a pure recognizer with no tracking has advantages and disadvantages. The main advantage is that the keyboard can quickly appear and disappear in the frame anytime and anywhere, and it can move arbitrarily fast, because the program uses no assumptions about where the object should be. However, a crude form of tracking – averaging detection results over the past few frames – would help smooth over many of the detection errors at the expense of some delay in the reports.

7. Conclusion

In summary, we have presented a new algorithm for recognizing objects that is designed not only to work well but to be easy to use from a normal consumer's point of view. This sort of algorithm will be important in intelligent rooms where recognizing and locating objects can enable compelling behaviors. Part of the ease of use comes from the fact that our algorithm generates its own training images from just a few actual images of the object. In addition, there are only two parameters that must be set by the user. Based on this scarce training data, we introduced a new way of quantizing binary features and a simple, efficient implementation of the generalized Hough transform using Hough kernels. Combined with color pruning, the algorithm quickly and accurately finds objects using normal PC hardware.

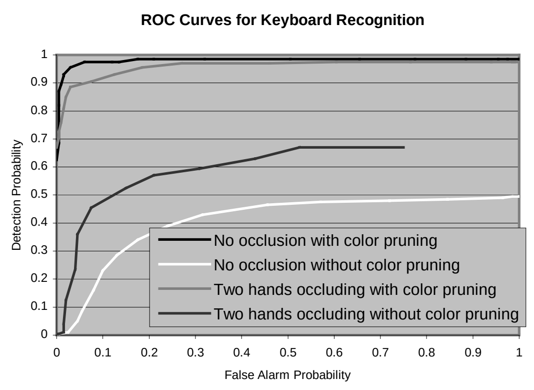

Figure 10: Receiver operating characteristic curves for keyboard detection. Occlusion degrades performance slightly, and color pruning helps greatly.

References

[1] D. H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111-122, 1981.

[2] K. Bowyer, C. Kranenburg, S Dougherty, "Edge Detector Evaluation Using Empirical ROC Curves," CVPR99, 1999, pp. 354-359.

[3] C. M. Brown, "Inherent Bias and Noise in the Hough Transform", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, no. 5, pp. 493-505, 1983.

[4] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[5] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing Image Using the Hausdorff Distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 850-863, 1993.

[6] J. Krumm, "Object Detection with Vector Quantized Binary Features", CVPR97, 1997, pp. 179-185.

[7] K. Ohba and K. Ikeuchi, "Recognition of the Multi Specularity Objects Using the Eigen-Window", ICPR96, 1996, pp. A81.23.