CHAPTER 5

Close encounters: recognizing nearby objects without contact

The followers of Om had lit their campfire in the crumbled halls of Gilash, just as the Prophet had said, and that counted, even though he'd only said it 5 minutes earlier, when they were looking for the firewood. (Pratchett, 1992b)

With the active segmentation behavior introduced in Chapter 3, the robot can familiarize itself with the appearance of nearby objects by poking them. This chapter is concerned with learning to locate, recognize, and segment those objects whenever they are present without further contact.

5.1 Approaches to object recognition

Physical objects vary greatly in shape and composition. This variety is reflected in their visual appearance. Unfortunately, it is not a straightforward matter to recover object properties from appearance, since there are many other factors at work – illumination, relative pose, distance, occlusions, and so on. The central challenge of object recognition is to be sensitive to the identity of objects while maintaining invariance in the face of other incidental properties. There are at least two broad approaches to recognition, geometry-based and appearance-based.

5.1.1 Geometry-based recognition

Image formation is a geometric process, so one way to approach recognition is to model invariant geometric relationships that hold for a particular class of object. These relationships can be between points, lines, surfaces or volumes. They may be known for many possible views of the object, or just one. When a new scene is presented, geometric relationships in it are measured and matched against the model. There are many details in what to measure and how to do the matching (there is a good review in Selinger (2001)). The main difficulty is the combinatorics of the search involved. There are a lot of free parameters to search over when we try to match an unsegmented scene to an object model – in particular, which elements in the scene correspond to the object, and what the transformation is between those elements and the object. For high-speed performance, geometric hashing is a useful technique (for a review see Wolfson and Rigoutsos (1997)). In this method, geometric invariants (or quasi-invariants) are computed from points in model (training) images, then stored in hash tables. Recognition then simply involves accessing and counting the contents of hash

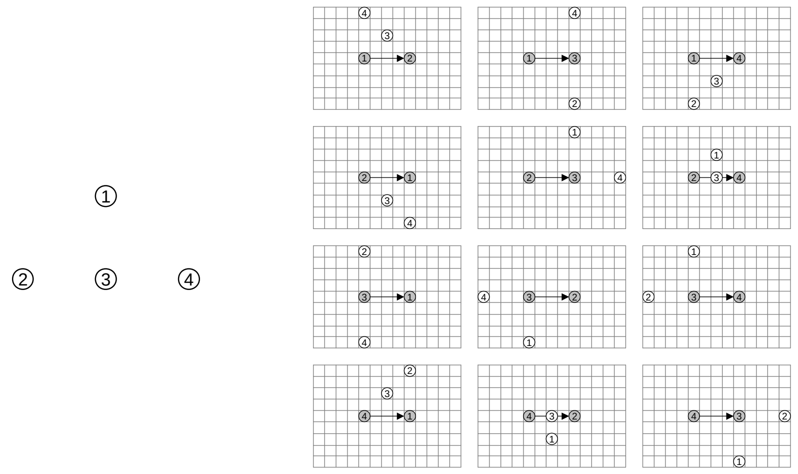


Figure 5-1: Geometric hashing for 2D-2D recognition. For the set of points shown on the left, each pair is considered in turn and used to normalize the rest for translation, orientation, and scale. The normalized locations of the points for each permutation are stored in a hash table, along with references to the model and pair of points that generated them.

buckets. One possibility for the geometric invariants is to take a set of points selected by an interest operator, and use each pair of points in turn to normalize the remainder by scale and rotation. The position of the normalized points can be stored in a 2D hash table, as shown in Figure 5-1. Invariants in 3D are more difficult to achieve, but various solutions have been proposed.

5.1.2 Appearance-based recognition

While the world is indeed geometric in nature, geometric features are not particularly easy to extract reliably from images. In appearance-based recognition, the focus is shifted from the intrinsic nature of an object to properties that can be measured in images of that object, including geometric properties but also surface properties such as color or texture. For example, Swain and Ballard (1991) proposed using the set of colors present in segmented views of an object as their representation. Regions of an image that contain the same color mix (as determined by histogram intersection) could contain the object. Histogram back-projection can be used to quickly filter out regions unlikely to contain the object, by assigning each pixel a weight based on the frequency of its color in the histogram. Some form of region-growing method then accumulates this evidence to find plausibly colored regions. This method is very fast, and for objects with distinctive colors it can be useful as a pre-filter to more computationally expensive processing. Color is also intrinsically somewhat robust to scale and rotation, but is sensitive to changes in the spectral profile of illumination.

Many appearance-based methods are window-based. A classifier is built that operates on a rectangular region of interest within an image. That window is moved across the entire image, at multiple scales, and sometimes multiple orientations. Responses of the classifier across locations and scales are combined using various heuristics. Variation in orientation (rotation in depth) is typically dealt with by training up multiple recognizers for various poses. These poses can be sampled quite sparsely, but still each pose requires iteration of the search procedure. There are ways to speed all this up, for example using a cascade of classifiers that reject clearly non-target windows early, devoting full analysis only to plausible targets. The actual classifier itself can be based on eigenspace methods or many other possibilities.

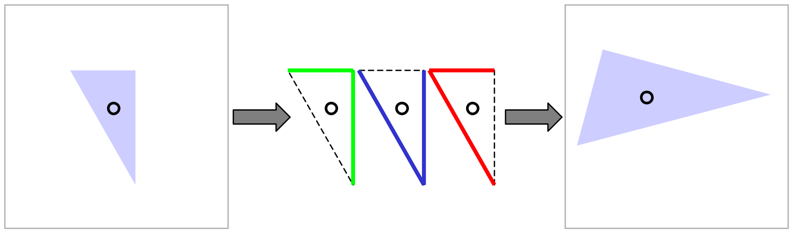


Figure 5-2: Rich features for hashing. Every pair of edges in the object to be recognized are stored, in terms of their relative position and orientation, along with samples of the colors between them. This representation is invariant to translation, scale, and in-plane rotation.

5.2 Hashing with rich features

The approach used here is like geometric hashing, but uses richer features that include non-geometric information. Geometric hashing works because pairs of points are much more informative than single points. An ordered pair of points defines a relative scale, translation, and orientation (in 2D). Further points may be needed for more general transformations, such as affine or projective, or to move to 3D (Wolfson and Rigoutsos, 1997). But, even staying with the 2D case, using just two points is somewhat problematic. First of all, they are not at all distinctive – any two points in an image could match any two points in the model. Hashing doesn't require distinctiveness, but it would be a useful pre-filter. Secondly, there is no redundancy; any noise in the points will be directly reflected in the transformation they imply.

One possibility would be to use triplets of points, or more. Then we have distinctiveness and some redundancy. But we also have an explosion in the number of possible combinations. Pairs of points are just about manageable, and even then it is better if they are drawn from a constrained subset of the image. For example, the work of Roy and Pentland (2002) uses histograms of the distance and angles between pairs of points on the boundary of an object for recognition.

In this work, pairs of edges (or more generally, any region with well-defined and coherent orientation) are used instead of pairs of points (Figure 5-2). Pairs of edges are more distinctive than pairs of points, since they have relative orientation and size. And if used carefully during matching, they contain redundant information about the transformation between image and model. A disadvantage is that edges are subject to occlusion, and edges/regions found automatically many be incomplete or broken into segments. But in all but the most trivial objects, there are many pairs of edges, so this approach is at least not doomed from the start.

The orientation filter developed earlier is applied to images, and a simple region growing algorithm divides the image into sets of contiguous pixels with coherent orientation. For real-time operation, adaptive thresholding on the minimum size of such regions is applied, so that the number of regions is bounded, independent of scene complexity. In "model" (training) views, every pair of regions belonging to the object is considered exhaustively, and entered into a hash table, indexed by relative angle, relative position, and the color at sample points between the regions (if inside the object boundary).

A useful extension of geometric hashing is coherency, where each match implies a particular transformation, and only "coherent" matches are aggregated (for example, see Lamiroy and Gros (1996)). This could be applied in the present instance. For speed, an abbreviated version is used here, where we filter by the centroid location each match implies for the object (this is information we would like to have anyway). There is no coherence checking by scale and orientation at the matching stage. This procedure means that we perform object localization simultaneously with matching.

Oriented regions are relatively sparse. Experiments showed that on a fast machine (800MHz) and at low resolution (128×128) it is possible to use triplets of regions as features at close to real-time. These can be very distinctive, and very redundant, and non-trivial objects have very very many possible triplets. But the frame-rate was just too slow (approximately 1 Hz) to be worth using here.

At the other extreme, another possibility would be just to use single edges. But they are not very distinctive, and sampling colors at an edge (generally a high-variance area) is problematic.

5.3 Details of matching

The basic feature used is pairs of oriented regions, containing the following information:

- ▷ Angle between the regions. The angle associated with each region is the mean response to the orientation filter in that region, and not the principle axis of the region itself, although these two measures will generally be highly correlated.
- ▷ Positioning of oriented regions relative to each other and to their projected intersection point (normalized for scale). The position associated with a region is the centroid of its silhouette.
- ▷ The color of sample points between the regions. Three points are sampled at quarter-length intervals along the line between the two region centroids, and their colors are compared. Color judgements are normalized for luminance and quantized to just a few bits.

We aim to have about the same number of regions as there are pixels in a row or column of an image, so the square is on the order of the number of pixels in the image, and hence on a scale that the computational architecture is designed to work with comfortably in real-time.

Now that each basic feature can make a prediction of where the object center should be, we can simply accumulate this and see if there is a convergence of evidence. This automatically gives high quality locations and scales the target may plausibly be at.

Segmentation of the object is then possible by seeing which regions contributed to locating the object. At this point, it is useful to find the consensus scale and eliminate matches at other scales.

Once we have proposed regions for the object, we can use any of the methods from the previous chapter to confirm the presence of the object.

The histograms matched against are constructed by merging features from all the training views available. With these histograms, we are modeling both the noise in sensing and the variation in the appearance of the object. This index should be invariant to translation and in-place rotation; some robustness to the precise boundaries of the segmented region. Clearly, we could be unlucky, and small changes could push us over a histogram bin boundary. Could use smoothing, or multiple samples.

For multiple targets, can remove the intersections between histograms (or assign to the target for which a feature is more frequent). Some objects simply have more oriented regions than others, and so may have a greater response to random background. Hence the importance of an independent confirmation method.

5.4 Searching for a synthetic object in a synthetic scene

As a simple example of how this all works, consider the test case shown in Figure 5-3. The system is presented with a model view of the circle, and the test image. For simplicity, the model view in this case is a centered view of the object by itself, so no segmentation is required. The processing

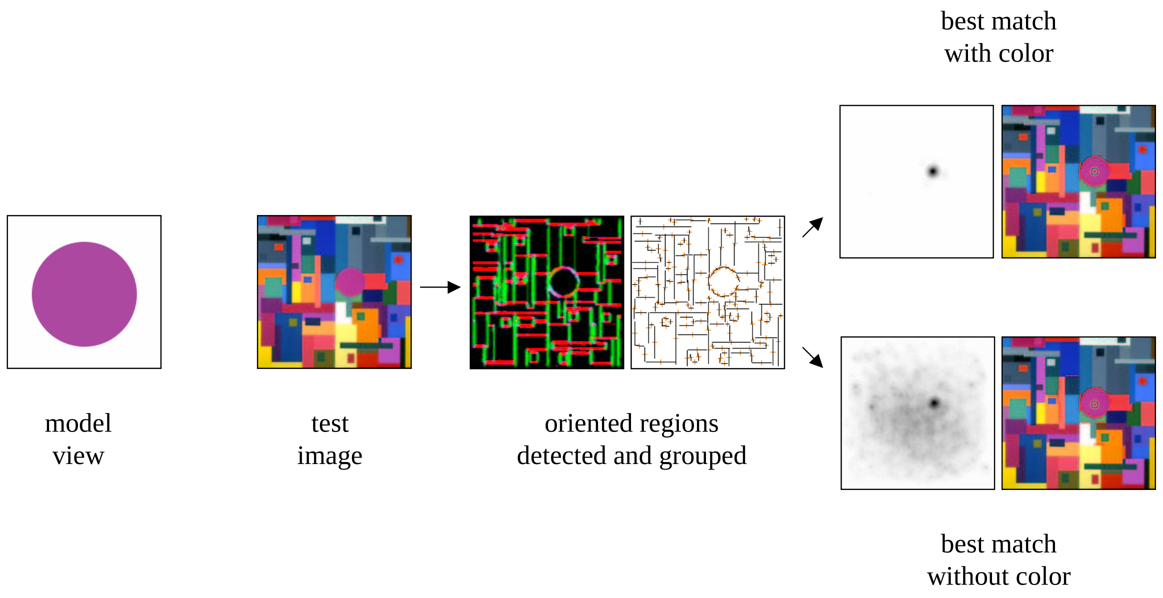


Figure 5-3: A simple example of object localization: finding the circle in a Mondrian.

on the model and test image is the same – first the orientation filter is applied, and then regions of coherent orientation are detected. For the circle, these regions will be small fragments around its perimeter. For the straight edges in the test image, these will be long. So finding the circle reduces to locating a region where there are edge fragments at diverse angles to each other, and with the distance between them generally large with respect to their own size. Even without using color, this is quite sufficient for a good localization in this case. The perimeter of the circle can be estimated by looking at the edges that contribute to the peak in match strength. The algorithm works equally well on an image of many circles with one square.

5.5 Searching for real objects in synthetic scenes

In Figure 5-4, we take a single instance of an object found through poking, and search for it in a synthetic image containing an abstract version of it along with various distractors. The algorithm picks out the best match, and lets us rank the distractors in order of salience. It is clear that a yellow square with anything in it is a good match, and having the internal purple square adds another boost. The closest distractor is a yellow square with a purple square inside it, rotated by 45°. Figure 5-5 shows another example. The object in question is a cube with a green face containing a red triangle. When presented with an image containing numerous variations on this theme, the most reasonable match (in the author's judgement) is selected.

5.6 Recognizing real objects in real images

Figure 5-6 shows examples of the cube being resegmented in real images. Testing on a set of 400 images of four objects (about 100 each) being poked by the robot, with half the images used for training, and half for testing, gives a recognition error rate of about 2%, with a median localization error of 4.2 pixels in a 128×128 image (as determined by comparing with the center of the segmented region given from automatic segmentation). By segmenting the image by grouping the

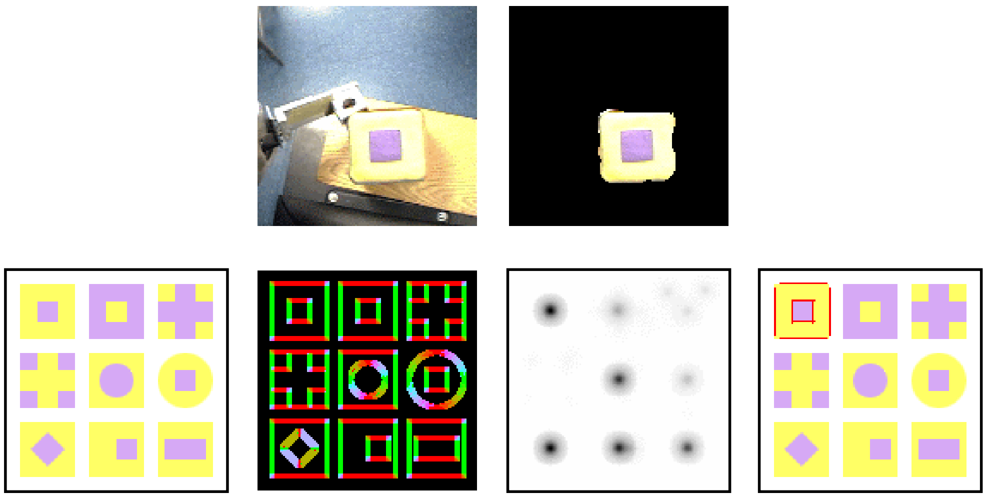


Figure 5-4: Looking for the best match to a cube found through poking in an image of a synthetic version of the cube along with a set of distractors. The superimposed lines on the rightmost image in the bottom row indicate the detected position of the object and the edges implicated. The image on its immediate left shows the strength of evidence for the object across the entire image, which lets us rank the distractors in order of attractiveness.

regions implicated in locating object, and filling in, a median of 83.5% of the object is recovered, and 14.5% of the background is mistakenly included (again, determined by comparison with the results of automatic segmentation).

5.7 Dealing with multiple objects simultaneously

There is nothing to stop us dealing with multiple matches in the same image, as shown in Figure 5-7. This is not in fact important for the robotic implementation, since it uses a foveated approach where objects are viewed sequentially.

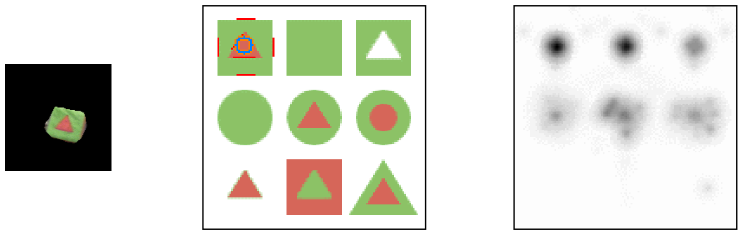


Figure 5-5: If the recognition system is trained on real images of a green cube (typical segmentation shown on left), and presented with a synthetic version of the cube along with a set of distractors (middle), we can evaluate what features are used for recognition. The superimposed circle and lines indicate the detected position of the object and the edges implicated. The image on the right shows the strength of evidence for the object across the entire image, which lets us rank the distractors in order of attractiveness. In this case, the most prominent feature used in recognition is the outer green square.

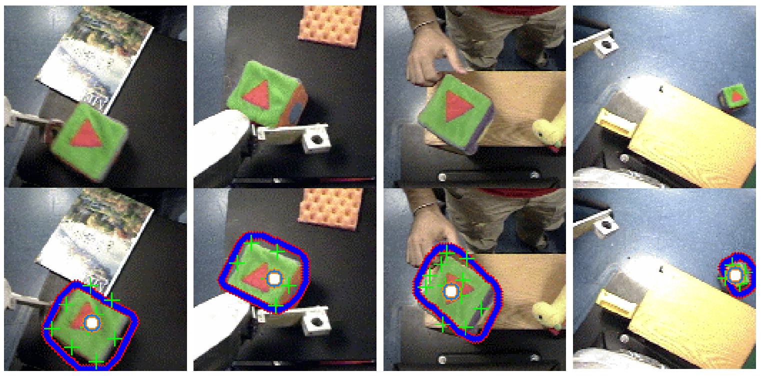
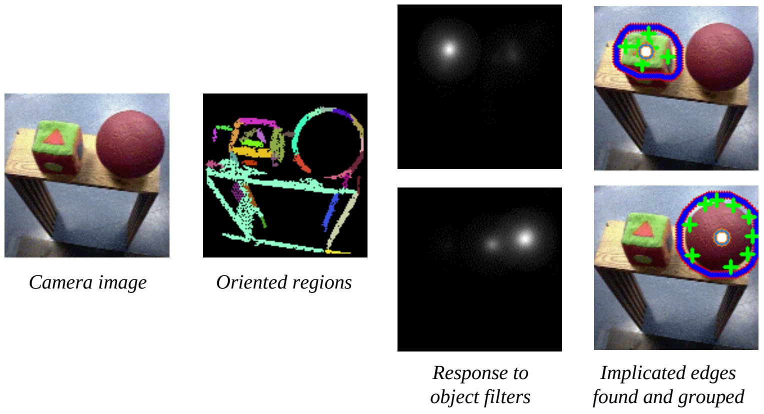


Figure 5-6: The cube being recognized, localized, and segmented in real images. The image in the first column is one the system was trained on. The image in the remain columns are test images. Note the scale and orientation invariance demonstrated in the final image.



Camera image
Oriented regions
Response to object filters
Implicated edges found and grouped

Figure 5-7: It is possible to deal with multiple objects in the same image.

5.8 Online training

In geometric hashing, the procedure applied to an image at recognition time is essentially identical to the procedure applied at training time. We can make use of that fact to integrate training into a fully online system, allowing behavior such as that shown in Figure 5-10, where a previously unknown object can be segmented through active segmentation and then immediately localized and recognized in future interaction.

5.9 Extracting an object prototype

The model developed above is good for localization and recognition, but is difficult to visualize directly. It is useful to have a less powerful but more easily viewable model to get a quick sense of how well things are working. This section develops a simple procedure for doing so.

Since every view of an object has a segmentation mask, we can try to align these masks, and then average the views to get a prototype for the object. First the masks are centered. Then they are rotated to minimize the moment in the vertical direction – if the object's profile is asymmetric, this will normalize the object to two possible orientations. An iterative approach is them used to flip the masks such that the corresponding object views are as similar as possible. Finally, the resulting transformed views are averaged, as shown in Figure 5-11. The average is typically blurry, and there may be some aspect of pose that wasn't normalized (see for example the green cube in Figure 5-11). The final step is to find the segmentation that best matches the average. This gets us back to a particular view of the object, but one that is as representative as possible. This gives us a good feel for how good the segmentations are. The results here show, for example, that all four objects are well-segmented. The bottle seems to be systematically losing its cap, and the small strip below the label. This suggests that there may be problems when there is a strong internal edge slightly inside the boundary – the segmentation procedure may switch to using them to minimize perimeter length. This should be fixable, but the point is that it is much easier to see that kind of effect in this representation than in the earlier one, even though it would be much less useful for localization purposes.

5.10 Comparing segmentations

To build up a good object model, it is useful to combine information across multiple segmentations. But different segmentations may come from different objects. Hence the segmentations need to be compared and clustered. The images are coming from online experience, and so have timestamps associated with them. Images close in time are likely to be from the same object. However, it would be useful to be able to pool data across different training episodes. Hence other cues such as color, boundary shape, and oriented features are important.

A variety of measures were evaluated. Comparing segmented views of objects is a relatively easy problem, and so much work has been done on it that it seemed unnecessary to spend time on this. So a simple measure (color histogram comparison) was adopted. Clustering happens both online and offline. The online implementation is optimized for speed, the offline method is optimized for accuracy. As offline updates become available, they replace the online object models.

- ▷ **Clustering by color histogram:** A classic Swain and Ballard (1991) style implementation was used, giving a recognition accuracy of 98.6% on a set of 420 images of the four objects shown in Figure 5-11, measured using leave-one-out cross validation.
- ▷ **Clustering by orientation histogram:** If color information was replaced with a histogram of orientation angles detected within the object, then 88.6% accuracy was achieved (See Figure 5-8). Using orientation is complicated by the fact that histograms have an a degree of freedom as the object rotates, so when comparing two histograms they need to be aligned first. This is slow compared to using color.
- ▷ **Clustering by boundary shape** Using shape information based on the first four Hu moments of the segmented boundaries, an accuracy of 87.6% was achieved.
- ▷ **Clustering by behavior** In theory, objects could be clustered based on how the move when poked. Inaccuracies in motor control made this impractical. However, this topic is revisited in Chapter 7 where aggregate, statistical measures of object behavior are shown to indeed be measurable.

5.11 Stabilized perceptual interface

One problem with trying to continually refine models for recognizing objects is that changing models could confuse any modules downstream that refer to these models. To deal with this a *stabilized interface* approach is used. For each object (or word, see Chapter 9) that can be recognized, a 'feature line' is allocated. The contract between the recognizing module and the rest of that control

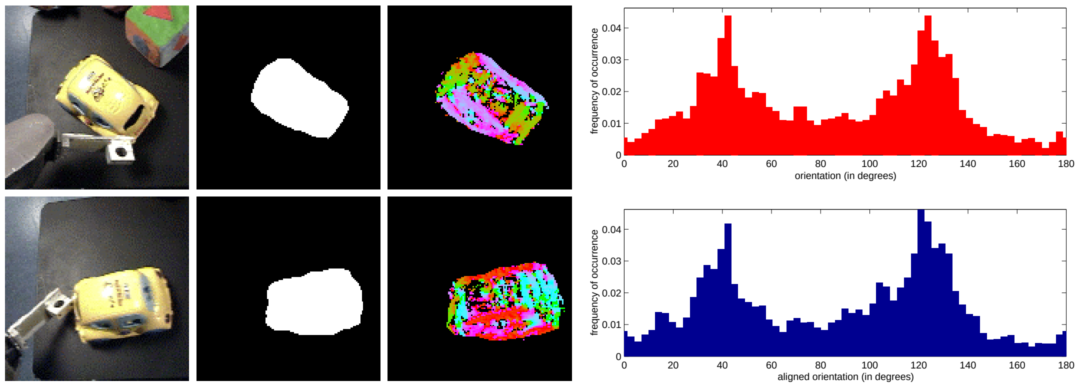


Figure 5-8: Comparing object segmentations using orientation histograms.

system is that the ‘semantics’ of that feature line will be preserved as much as possible – it will respond to the same situations in future as it did in the past – except for attempts to refine or purify the semantics (so that it is less affected by noise, for example, or responds to the same basic property in an extended range of situations). Offline updates are made initially without any regard to the contract so that off-the-shelf clustering algorithms can be used; then as a last step models are compared with previous models and aligned appropriately.

5.12 Completion and illusory contours

If there are not too many of them, the object recognition process will consider completing breaks in edges, if there are edges with the same orientation which, if extended, overlap with each other. The main purpose of this is to compensate for limitations of region grouping, which will sometimes erroneously split an oriented region into two fragments. As a byproduct, simple illusory contours can be dealt with, such as the Kanizsa triangle shown in Figure 5-9.

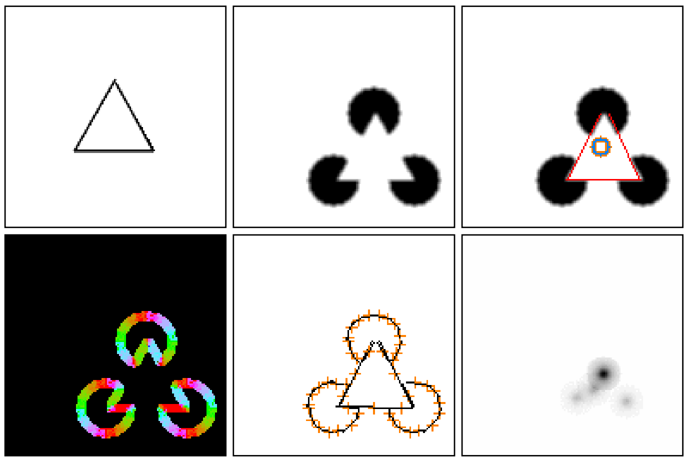


Figure 5-9: Breaks in edges are considered for completion during the object recognition process. Hence certain illusory contours such as the triangle in the right of this figure possesses, can be recognized.

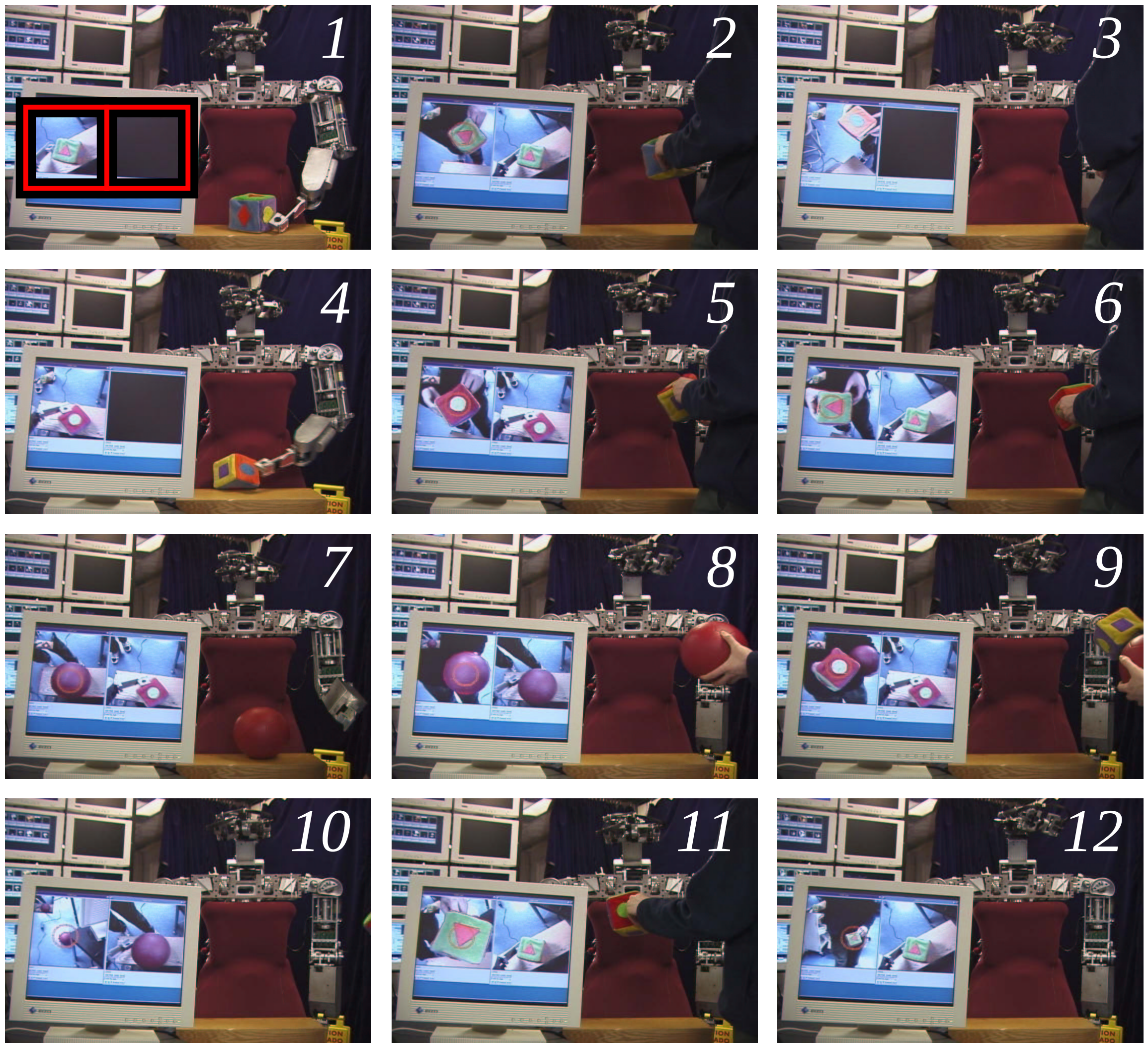


Figure 5-10: This figure shows stills from a three-minute interaction with Cog. The area of the first frame highlighted with a square shows the state of the robot – the left box gives the view from the robot's camera, the right shows an image it associates with the current view. Initially the robot is not familiar with any objects, so the right box is empty. It is presented with the cube, and pokes it (first frame). Then, if shown the cube again, it recognizes it (this recognition is evidenced by showing an image recorded from when the object was poked). If the cube is turned to another side, the robot no longer recognizes it (third frame). If that side of the cube is presented to the robot to poke (fourth frame), if can then recognize it (fifth frame) and differentiate if from the green side (sixth frame). If it confuses another object with what it has already seen, such as the ball in the seventh frame, this is easily to fix by poking (eighth, ninth frames). The final three frames show the invariance of the recognition system to scale.

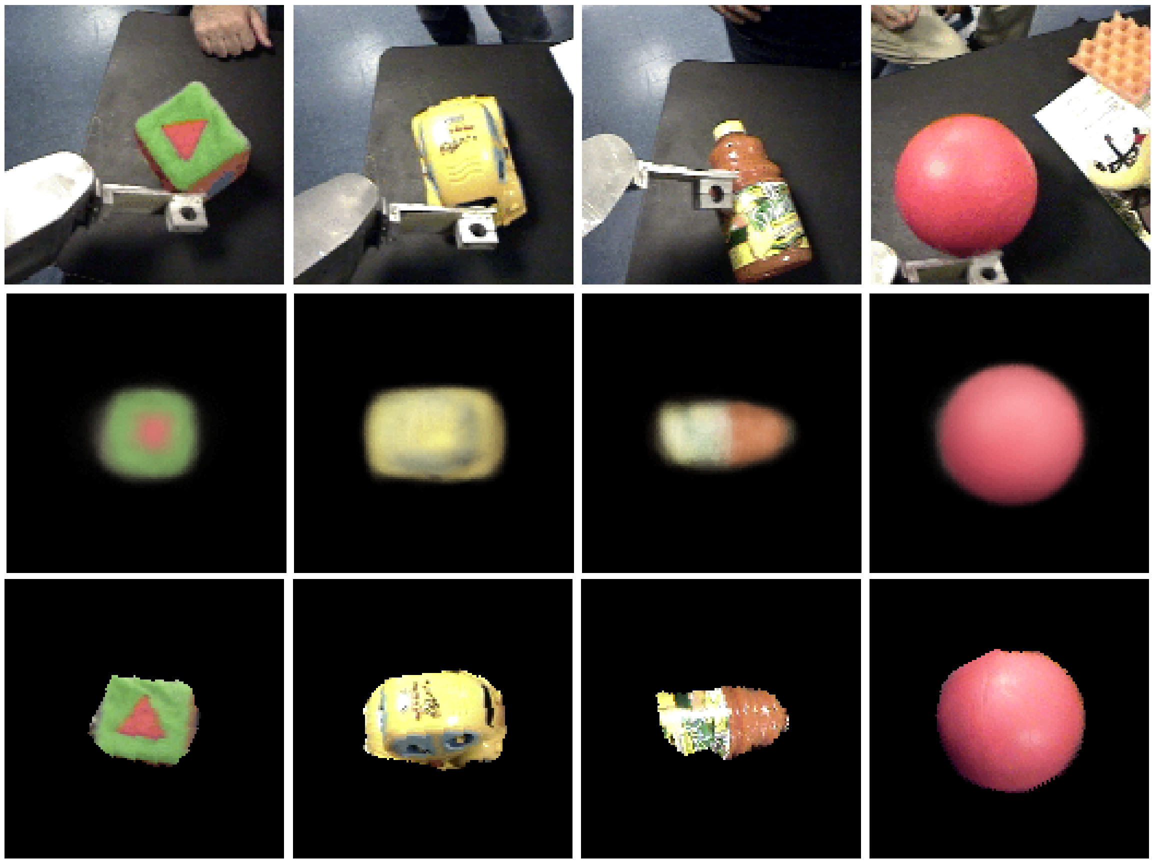


Figure 5-11: The top row shows the four objects used in this experiment, seen from the robot's perspective. The middle row shows prototypes derived for those objects using a naïve alignment procedure. None of the prototypes contain any part of the robot's manipulator, or the environment. These prototypes are used to find the best available segmentations of the objects (bottom row).

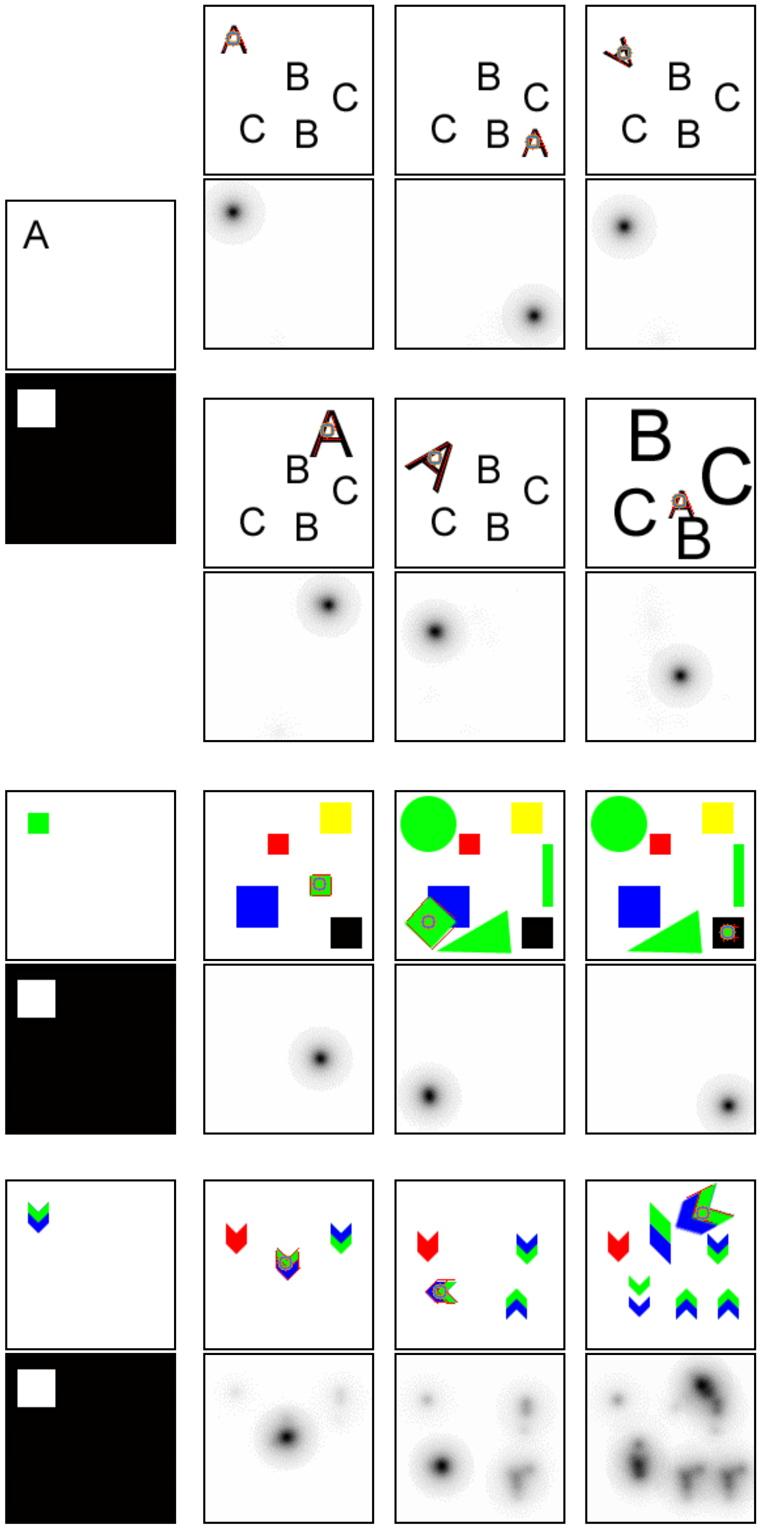


Figure 5-12: More localization examples. The left-most column shows the object prototype and mask for training. The top half of the figure shows the letter A being located at various positions, scales, and orientations (B and C work fine too, but letters that are physical supersets of each other with the same center, such as C and O, cannot be distinguished). The small circle indicates the best matching location, and the full map of responses is given underneath each image. The lower half of the figure shows the same algorithm working on examples of a different class of object, colored shapes: a simple square, and a chevron.