Get me that Screwdriver! Developing a Sensory-Action Vocabulary for Fetch-and-Carry Tasks

Jonathan H. Connell
IBM T.J. Watson Research Center
Box 704, Yorktown Heights, NY 10598
jhc@watson.ibm.com

Abstract

This report describes a partially implemented system for finding and manipulating small objects. The system is meant to be used for picking up hand tools from table tops using a 6 DOF force-sensing robot arm mounted on a mobile platform. The visual recognition system is designed to use a wrist-mounted camera to search for coarsely modelled objects. In this report we develop a vocabulary of useful visual operators for object localization and describe their use. We also develop a vocabulary of sensor-based motion primitives and describe a control scheme for combining them to achieve tasks. Finally, we discuss how the complete system might be trained instead of programmed, and examine what sort of information needs to be extracted to accomplish this.

Introduction

Constructing a mobile robot that can navigate around a building is a difficult project, but one on which a lot of research has been done. A much more exciting, and useful, goal is to build a robot that can both fetch and carry.

To implement the fetch part we need to understand what sort of manipulation is required for the task. It is not precise positioning or highly dextrous handling, nor does it require much trajectory planning. It is mostly just "snarfing" – somehow grabbing a viewed object so that it does not fall out of the gripper. Although it might be nice to have some of the other capabilities so the robot could ballistically grab its target, the state of the art is generally not up to this challenge. Arms always have some inaccuracy in them and their slop increases rapidly as the price drops. Vision systems can give a rough idea of the position and orientation of an object but have a hard time building metrically accurate representations of their surroundings. Furthermore, the most robust vision systems usually require a geometric model of the target object – something we would like to avoid since it is difficult to acquire.

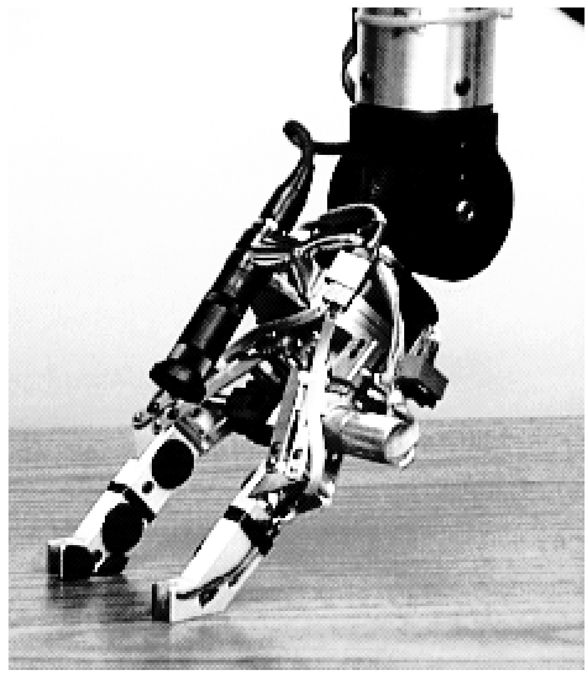

Figure 1: BRANTA consists of an 6-DOF arm with a color camera mounted on it. The parallel jaw gripper has proximity detectors in the fingers and the wrist has a force/torque sensor.

Instead we propose breaking up the act of snarfing into a strategic and tactical component. A vision system would provide a rough idea of where the object was and at what angle to grasp it. A variety of end-point sensors on the hand would then interactively determine the true position and orientation of the object and perform the specified grasping action. In this way one could build a working system which does not demand excessive performance from either the vision system or the arm.

The interesting aspect of this approach is to define what sort of primitive behaviors might be employed by the robot to accomplish its task. For instance, visual operators such as "find-big-red-things", "search-above-table-edge", and "reject-unless-next-to" prove to be useful. A fairly simple visual system can accomplish most of this without resort to computationally complex (and often fragile) techniques such as full image segmentation, edge to model matching, texture-based stereo, algebraic surface fitting, etc.

Similarly, the basic motion vocabulary might include terms like "center-fingers-over", "align-grip-axis", and "flatten-hand-wrt-surface". In essence this provides a richer vocabulary for describing the task than standard robot control languages which have position and trajectory commands, guarded moves, and (sometimes) generalized-damper control. Discovering a generally useful set of behaviors would also be a boon to telemanipulation where bandwidth constraints and operator fatigue are considerations. It could also provide a much more productive search space for robot learning algorithms.

The rest of this paper discusses the BRANTA system under development here. This system is built around a Zebra Zero 6-DOF anthropomorphic robot with a six axis force/torque sensor (see Figure 1). The links of this robot have been lengthened to provide high maneuverability over a region three feet wide by one foot deep at tabletop level. To the parallel jaw gripper we have added hook-shaped fingers with short range (2mm) IR proximity detectors in each fingertip and two break-beam sensors in the jaws. There is also a small color CCD camera with a wide-angle lens mounted on the wrist (hence the analogy to a goose). As a disclaimer, this system is not yet totally integrated so what is presented here is a mixture of partial results and speculation.

Target Detection

The primary task for our robot is to pick up small hand tools from table tops. To do this, the robot needs to be able to find and identify the object to be grasped. In our case, BRANTA relies on its wrist mounted color camera.

Fortunately, the location task is not as hard as general-purpose vision: it is greatly simplified by the context of the task. First, it is not necessary to interpret all parts of the scene. Many potential targets can be eliminated solely on the basis of *position and size*. Second, a real understanding of the shape of the object is not required. Since the robot is looking for a specific object, *signature-like features* usually provide sufficient constraints to disambiguate it. Finally, our particular task is further simplified by the *nature of hand tools* – the handles are often brightly colored and have a roughly cylindrical shape.

The basic vocabulary we use for this sort of target detection is given below. The visual operators can be broken down into sets of routines that work on a pixel-by-pixel basis, ones that analyze the properties of images regions, and ones which pay attention to the arrangement of groups of objects and select between them.

Pixels –

- Color pass (based on hue, saturation, intensity)
- Edge pass (based on direction, magnitude)
- Motion pass (based on difference magnitude)
- Depth pass (based on correlation at vergence)

Regions –

- Connected components
- Shape gate (based on area, width, length)
- Position gate (based on image x, image y)
- Distance transform (erosion, dilation, Voronoi)

Groups –

- Overall stats (count, mass, centroid, orientation)
- Pick best (wrt. size, shape, position)
- Overlap tag sets of regions (either retain or poison)

To further explain what these operators are and to see how they are used, let us examine a particular example. The task is to find a blue set of pliers on a table. The input scene is shown in the upper left of Figure 2 (originally in color). This was taken using a camera with a 90 degree field of view positioned one foot out from the table and two feet above it.

The first step is to limit the area of search. Since we know the pliers are going to be on a table, we start by looking for the edge of the table. This is accomplished by passing the pixels which have a high edge magnitude and an orientation close to zero degrees (e.g. from the Sobel operator). The resulting binary image is shown in the middle left hand panel of Figure 2. Notice that there is a lot of clutter along with the edges we really want.

We can then use some further mild constraints to clean up this image. We know, for instance, that table edges are fairly long and that we have parked the robot so that the edge is likely to be in the lower third of the image. To enforce these restrictions, we compute the connected components in the binary image and determine the corresponding ellipse parameters for each blob. Any blob which is too short or whose centroid is too high in the image is eliminated. This yields the binary image in the lower left of Figure 2. Since we know the robot was parked roughly perpendicular to the table we can use the y coordinate (183 from the top) of the overall centroid of this image as an estimate of the table boundary.

Now that we have determined generally where to look, the next step is to highlight possible candidate pliers. We do this by first passing pixels with a hue in the blue range. This yields the binary image in the upper right corner of Figure 2. In can be seen that the plier handles are picked out but that there is a lot of extraneous clutter, too. For instance, the floppy disk to

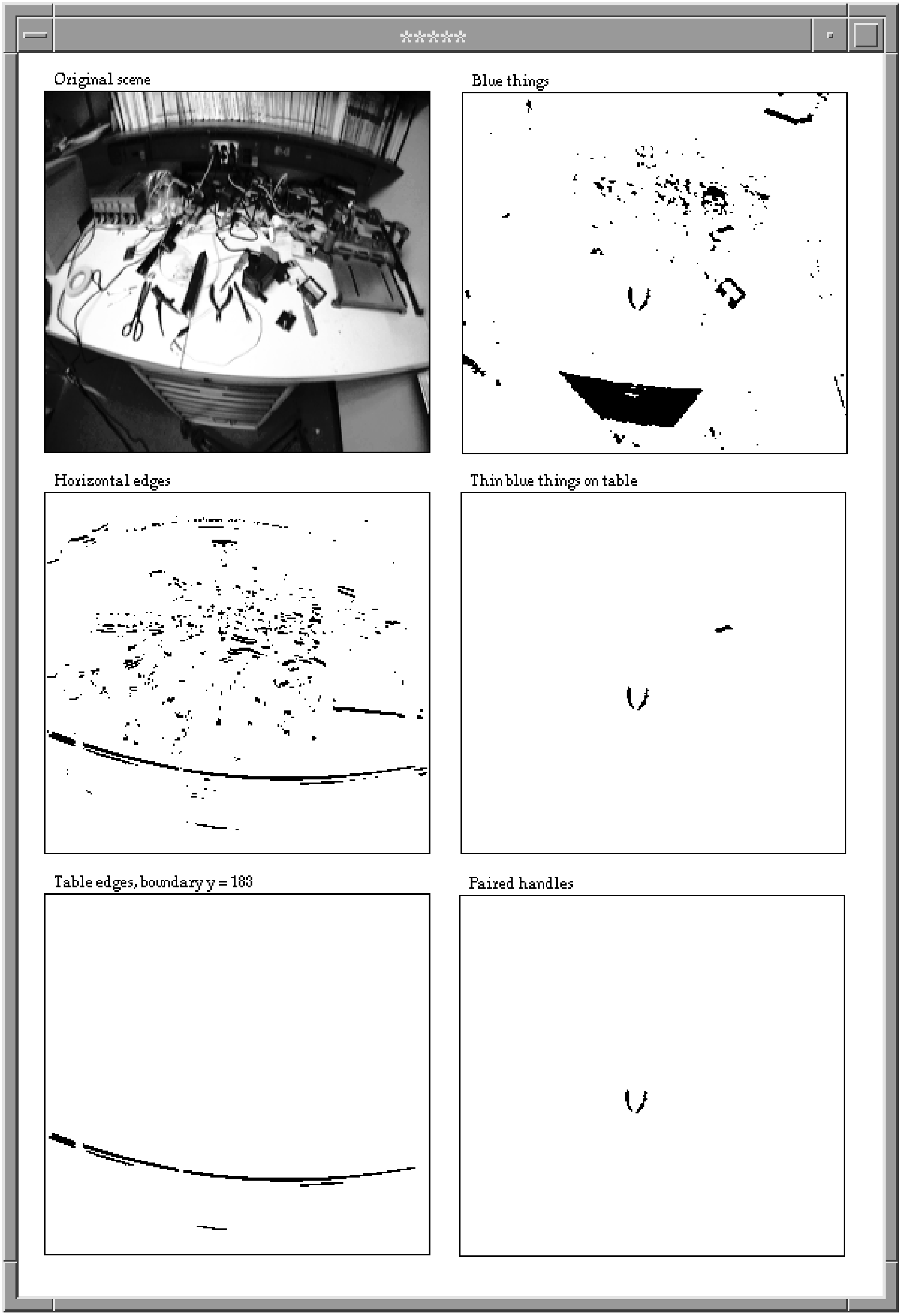

Figure 2: Steps in visually detecting a pair of blue pliers on the table.

the right of the pliers is blue, as are the drawers on the cabinet below the table, and the stripes on the databooks to the upper right. We now find the connected components in the binary ("blue") image. This allows us to apply the position constraint: we reject all blobs whose centroids are either below the table edge or too far above it.

The connected components also let us do further pruning. Since we know the configuration of the arm and the approximate location of the robot relative to the table, we can compute the position of the camera. For the setup used here, most of the reachable portion of the surface is between 2 and 3 feet from the camera. With this information we can set bounds on apparent pixel area of the pliers. We also know that in terms of rough shape the handles are somewhat elongated. Using these two constraints in conjunction with the position restriction yields the image in the middle right panel of Figure 2.

We now have three possible candidates for plier handles. However, we know that handles always travel in pairs and what the overall dimensions of the set should be. To weed out items that do not pass this test, we dilate each handle by the expected pair size and then erode by the same amount. This essentially wraps an envelope around suitable pairs. Once again we compute the connected components of the image and throw out all regions which are not the right size (i.e. too few or too many nearby blobs). Finally, we overlap this new image with the image of the candidate handles and "tag" all handle blobs that are touched by a pair blob. Untagged handle blobs are discarded. This yields the final result shown in the lower right of Figure 2.

In our example there is only one item (pair blob) that passes all the tests. However, in general there may be several. In such cases we could pick the best instance based on its distance (as judged by its height in the image), reachability (based on its offset from the center of the image), or some similar criterion. Alternatively, if we had some additional fine-grain matcher, like a gray scale template or Hough transform, we could go through the blobs serially to find the best. In terms of mechanism, this could be done by picking one blob randomly and using it to gate the corresponding portion of one of the intrinsic images (e.g. intensity or edge map). The parameters of the blob could then be used to normalize the patch with respect to position, scale, and orientation for the second tier matcher. If the blob failed this extra test (or we wanted to test another blob) the "overlap tag" operation could be used to combine the best blob image with the previous image of all candidates blobs. In this case the single blob would "poison" any blob it touched (such as itself in the candidate image).

The steps so far have served to select a single item in the scene which satisfies a set of constraints. From the resulting binary image we can find the camera-space coordinates and orientation of the object. Yet we might also want to know its depth. With this information would could obtain a ballistic target for the hand, determine whether the object really is within the workspace of the arm, or verify the object's true size (if there are false targets with fairly similar dimensions). A relatively easy way to do this is to displace the hand (and hence the camera) to generate a stereo pair. While stereo in general is hard, in this case the correspondence problem can usually be solved simply by applying the same visual operators to both images. Or, if desired, the actual parameters of the blob found in the first image could be used to tighten the search criteria for the corresponding blob in the second image. In either case, comparing the centroids of the target in both images yields an estimate of disparity (similar to vergence-based stereo). In our example, using a horizontal baseline of 3 inches and the same generic finding routine for both images, we arrive at a distance of 26.3 inches for the pliers.

A number of other things besides pliers can be found using the same visual operators. A number of these are shown in Figure 3. For instance, screwdrivers are similar to pliers in being elongated colored things (in this case yellow) but we don't need the pairs restriction. The resulting image (upper left in Figure 3) shows two possible candidates, both of which really are screwdrivers. A similar sequence of operators also serves to find the battery charger on the table (lower left) while ignoring the red pencil, red electric screwdriver, and various red items in the background.

Occasionally, blob parameters turn out to be too weak a shape description. Luckily, simple morphology can be used to resolve many of the ambiguities which arise. For instance, a mass of red wires might give rise to a equivalent ellipse similar to the one for the battery charger. But if we erode the wires slightly, their blob fragments into multiple pieces whereas the blob for the battery charger remains intact. This additional processing was not necessary for the either of the examples shown, however.

To find other types of objects, more involved sequences may be required. For instance, to isolate the scissors (which are black) we first find long dark regions. Most of these are not parts of scissors. To determine which ones are valid we next look for small bright blobs corresponding to the holes in the handles. We then dilate these key areas slightly and tag any of the long dark regions that they overlap (upper right in Figure 3). This leaves a single candidate.

Sometimes it is also necessary to use edges to impose tighter constraints on the shape of objects. For instance, the electric screwdriver near the center of the original scene in Figure 2 is a medium-sized elongated red region. However, it may be useful to verify that it has straight sides. This would distinguish it from curved objects, like a red jar top seen in perspective, or ones in which the width varied, like totally red pliers. One way to do this is to find long vertical edges and also find medium elongated red regions. We then

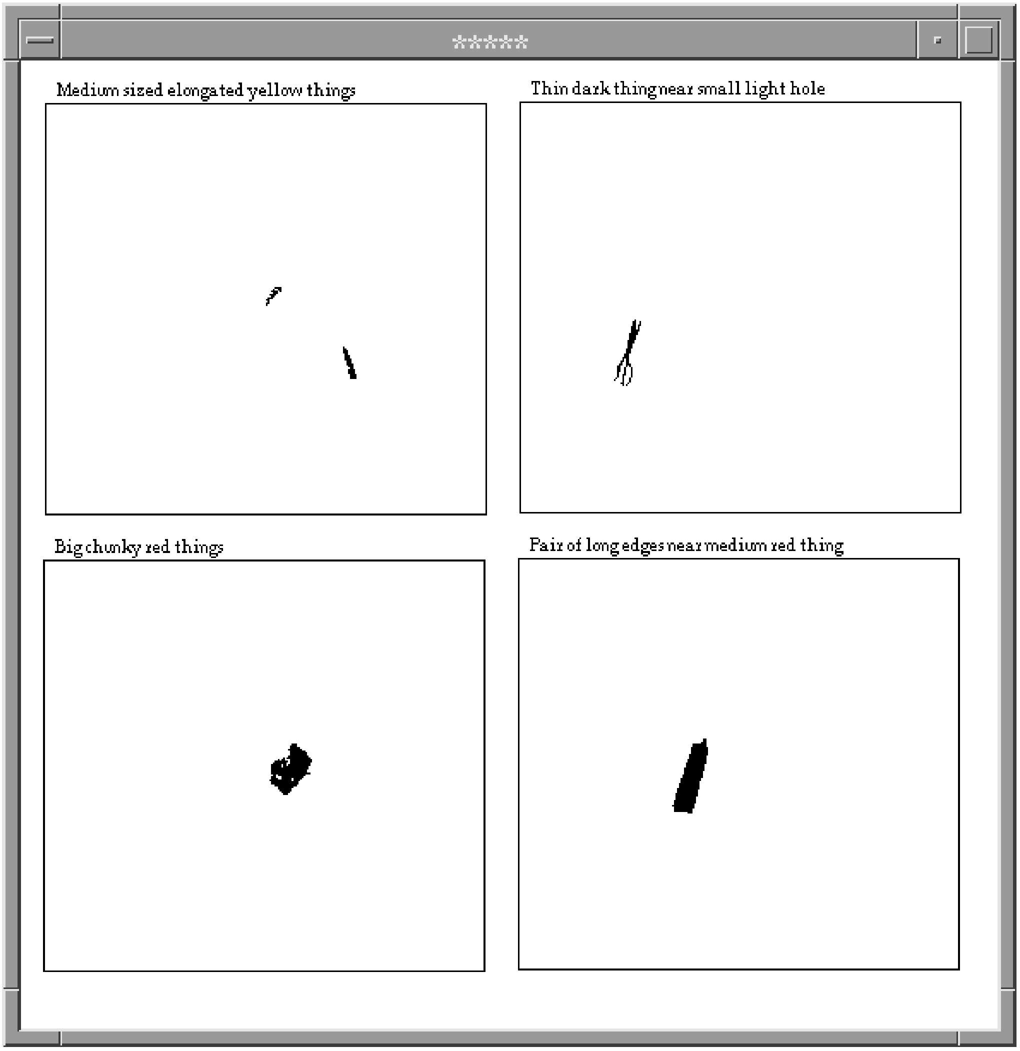

Figure 3: Other objects in the image can be found using the standard visual operators. These include the screwdrivers (top left), battery charger (lower left), scissors (top right), and electric screwdriver (lower right).

grow the red regions slightly and tag those edges that overlap these regions. Finally, we use the same morphological method as for the pliers (dilate, then erode, then size threshold) to find appropriate pairs of edges. The results of this are shown in the lower right panel of Figure 3.

As can be seen, our basic vocabulary of visual operators is very versatile. It allows us to use expectations about size, position, and orientation and to search for objects modelled in terms of color, rough shape, and the coarse configuration of subparts. The output of the target detector consists of the image coordinates and orientation of the desired object. The object's approximate distance and size can also be determined using with vergence-style stereo as we have shown.

Motion Generation

Now that the robot has visually identified the object of interest, it must figure out how to move its hand to the right place and actually grasp the object. Our proposed set of motion primitives is listed on the following page. We have segregated them into groups based on which of the robot's resources (actuators) they affect. Some of the operators are general policies, like driving the hand forward or thrashing. Others embody constraints on motion, such as rotating the hand if it is not sufficiently aligned with the axis of a viewed object.

BRANTA allows a number of these routines to run concurrently in order to perform a particular task. To achieve the necessary cooperation, each of the robot's degrees of freedom is controlled in terms of velocity. When a collection of routines is enabled, each behavior is given a specific priority. The coordinating controller then goes through the list of behaviors in order. If the primary behavior chooses not to specify a velocity for certain degrees of freedom, the next highest behavior gets a chance to set these variables. This continues until either all velocities have been set or there are no more behaviors. If some variable remains unset, it assumes a default velocity of zero.

Position –

- Drive direction relative/world
- Position seek relative/world
- Push relative/world
- Thrash
- Proximity attract/stop/repel
- Force halt/recoil
- Proximity center
- Contact center

Orientation –

- Rotate direction relative/world
- Orient seek relative/world
- Twist relative/world
- Center camera spot
- Align camera axis
- Wobble
- Proximity level/align
- Force level

Gripper –

- Width seek
- Beam close
- Proximity open
- Force open
- Contact stop/open

Many of the operators in our list are related to traditional position, velocity, and force commands. However, in general, none of our behaviors are full servo-loops. Operators like *Position seek* merely suggest motions toward their goal positions when the hand is too far away. They do not have exclusively control of the robot and they do not self-terminate. Although *Position seek* might be normally be driving the robot toward some goal position, it can be momentarily overridden by a behavior assigned a higher priority. This might happen, for instance, if *Force recoil* noticed that the hand bumped into something. Similarly, the robot might have gotten close enough to the target for *Position seek* to stop dictating velocity vectors. Yet instead of terminating, it remains quiescent just in case some external occurrence causes the robot to depart from this location.

Other operators directly implement useful stereotyped motions. For instance, there is a whole collection of behaviors based on the fingertip proximity sensors. These commands are useful for orienting the robot relative to some object or surface as opposed to some Cartesian position or coordinate frame. Many of these same functions can be performed using the force sensors as well. The proximity version, however, are especially useful when touching the object might disturb it in some undesirable way.

Figure 4 shows how several of these behaviors operate. The proximity detectors return only two bits of information – whether anything is close to each of the two fingers. Based on these bits we set velocities for some of the robot's degrees of freedom. The top left illustration shows a situation in which the robot's fingers have come down on top of an object. Since the parallel jaws can not grip the object unless they encompass it, we open the gripper wider when both detectors see something. Another trick is shown in the center illustration. Here, the robot's gripper is already opened wide enough but is offset from the center of the object. In this case we can use the state of the two bits to determine which way to shift the hand. Now suppose we also enable *Drive direction relative -Z*, which causes the hand to slowly descend, and the *Proximity stop* behavior, which prevents the robot from lowering its hand if either detector sees something. These behaviors in conjunction with *Proximity open* and *Proximity Center* cause the robot to center itself on the target object, open the gripper the appropriate amount, and then move the fingers down around the object.

The proximity detectors can also be used with surfaces instead of objects. The upper right illustration in Figure 4 shows how the hand can align itself with the edge of something. Here the two bits of information are used to control the orientation of the hand. Finally, the lower illustration in Figure 4 depicts a situation in which the robot uses the proximity detectors to rock the hand so that the fingertips will squarely hit the surface. This *Proximity level* behavior might be combined with *Proximity align* as part of a routine that snagged a box and dragged it closer. The robot would gingerly move toward the front of the box until one of the proximity detectors was activated. It would then level itself and slowly start rising until at least one of the proximity detectors dropped out. At this point it would align itself with the lip of the box, execute a ballistic hop up and over the edge, and then pull backwards.

As should be apparent from the preceding example, sometimes it is necessary to change which behaviors are active during the course of a task. To allow this, we have adopted a finite state machine formalism. In a motion "program" there are a series of nodes connected by links. The nodes specify which behaviors should be enabled and assign them relative priorities. The links specify certain sensory events which shift the activation to a different. When this happens, all the behaviors in the old node are disabled in favor of those specified by the new node.

To see how this works, let us consider an example in which the robot picks up an object detected by its

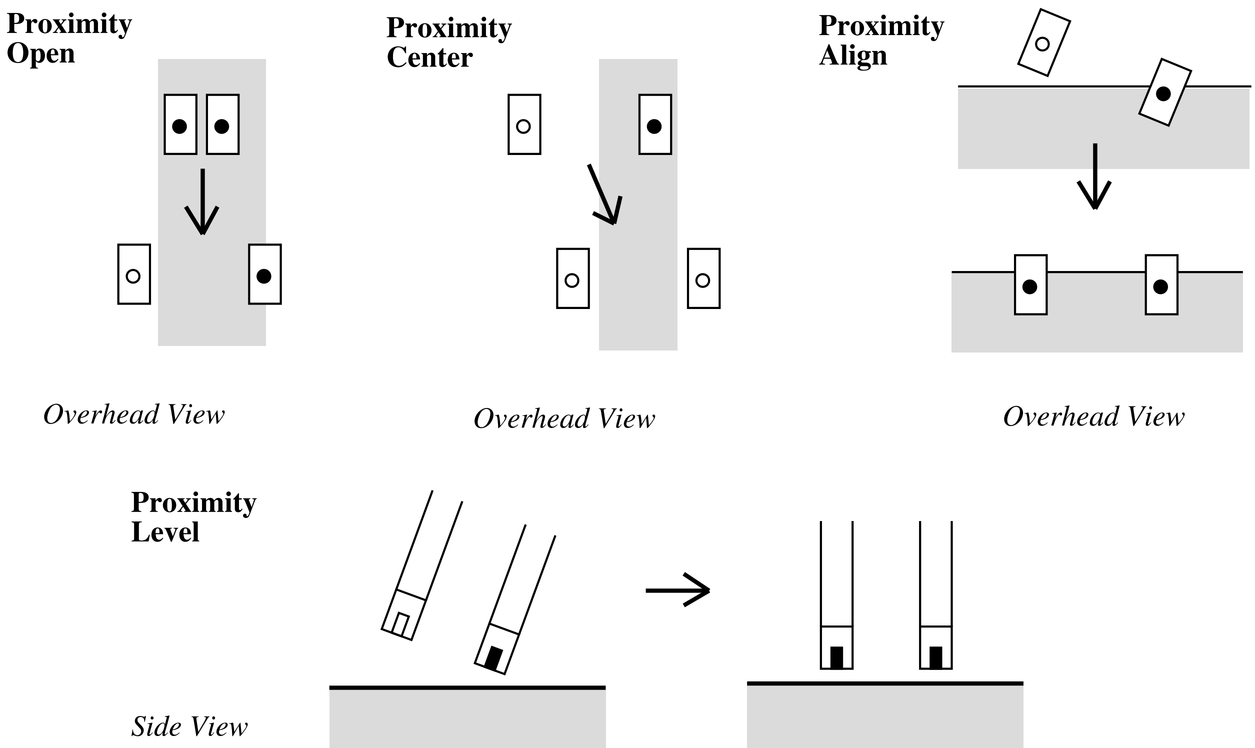

Figure 4: The two short range proximity sensors in the fingertips can be used for a number of useful behaviors. In these diagrams a black spot or bar in a fingertip indicates that the corresponding sensor has detected something.

camera. There are a number of ways to generate a trajectory for the hand to get it from the viewing point to the object. If we knew the real world coordinates of the target, we could directly move the hand to a point slightly above it and then descend. If we did not trust the distance calculated by the vision system, we could instead rely on the angular displacement of the target in the visual field. In this case, we would rotate the hand until the target was centered and then move the hand along the optical axis of the camera. Both this approach and the previous one assume that a free sight path to the object implies a free motion path to it as well.

Still another type of trajectory is embodied in the controller shown in Figure 5. Here, the arm moves far above the table until it is above the object, then descends to pluck it cleanly out of any nearby clutter. To accomplish this the system starts by activating the behaviors specified in the topmost box. The wrist ("O" resource) is constantly swiveled to keep the target object centered in the camera image while the hand is concurrently driven forward ("P" resource) at the current height. The hand moves straight toward the target since the "forward" direction is defined in terms of the hand coordinate system which in turn is re-oriented appropriately by the *Center camera target* behavior.

Eventually, the hand will be directly over the object. Since the camera is effectively slaved to the target, we can determine when we reach this position by looking at the global orientation of the hand. When the predicate "gripper vertical" is satisfied, control shifts to the next box in the diagram. At this point the hand stops translating and instead rotates to make the axis of the object appear vertical to the camera. Since the camera is rigidly attached to the hand, reorienting the target image also aligns the grip axis of the hand with the object. While this is happening, the fingers are brought together for the next stage of approach.

After satisfactory image alignment is achieved, control passes to the third node. This just activates a behavior which pitches the fingers up by 45 degrees so that the tips will hit the table on their flat sides. This is necessary because the camera is parallel to the long axis of the fingers, whereas the proximity detectors are at the bottom of the bent portion near the tips (see Figure 1). Notice that the condition for exiting this node is "gripper at 45". Since the only behavior that was active was the one that pitches the gripper, this whole node acts like a standard angular position command.

Finally the large node in the lower left corner of Figure 5 takes command of the robot. There are four behaviors controlling the position of the hand and two controlling its orientation. The effect produced by the combination of *Proximity center*, *Proximity stop*, *Drive*

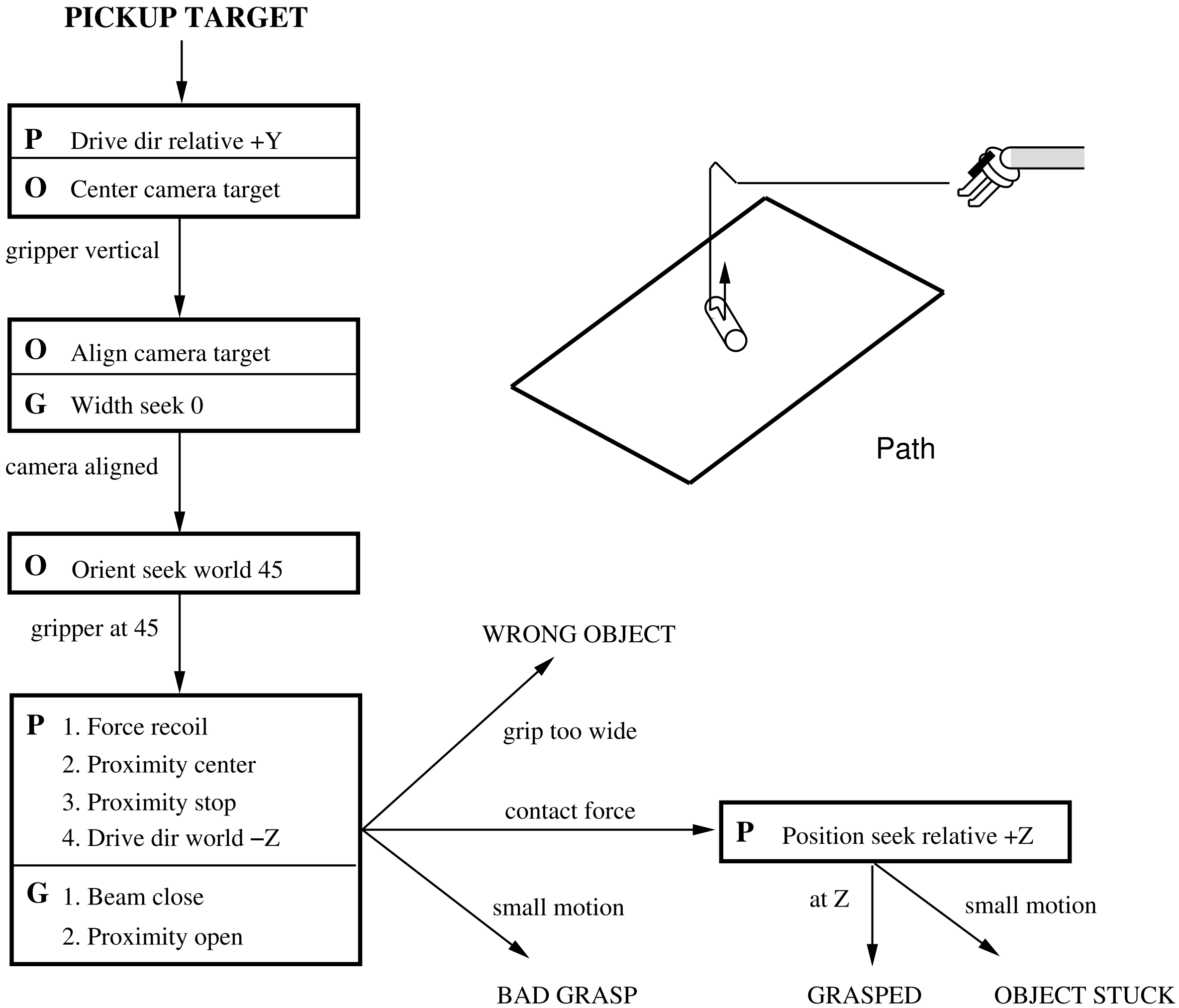

Figure 5: This finite state machine generates the hand trajectory shown to the right. Each node enables a prioritized set of behaviors that controls one of the actuator resources (P = position, O = orientation, G = gripper). The detection of a specific event switches the controller to the next behavior set.

dir world -Z, and *Proximity open* was described previously. One variation here is that *Force recoil* causes the hand to twitch upwards if the wrist force sensor detects a collision. The other is addition is that *Beam close* causes the gripper to squeeze the object when it comes between the fingers.

As can be seen by the number of arrows exiting this box, there are three typical reasons for deactivating this set of behaviors. If the fingers detect a resisting force, all is well and the robot has achieved a firm grasp on the object. If instead the hand is opened up all the way, it is likely that the robot missed the intended object or is trying to mistakenly grip the table. The third case is when there is no contact force, but the hand has stopped moving downward. This might happen if, for instance, the hand was over something that responded poorly to the proximity detectors (e.g. was black). In this case the robot might keep hitting its hand against the object and cause *Force recoil* to be repeatedly activated. Since the overall change in position of the hand remains small for an extended length of time, the "small motion" predicate would become true.

Assuming a reasonable grasp is obtained, command authority transfers to the last node. This just tries to raise the object a small distance above the table. If it succeeds in lifting the object, all is well. Otherwise it is likely that the robot grabbed the wrong thing or that some part of the hand has snagged on an nearby obstacle. Any time one of these error conditions is encountered it is up to the higher level controller (e.g. a human or a symbolic planning system) to analyze the situation and instantiate a new finite state controller.

Now that we have seen how one controller is built, we can write controllers for other purposes as well. For

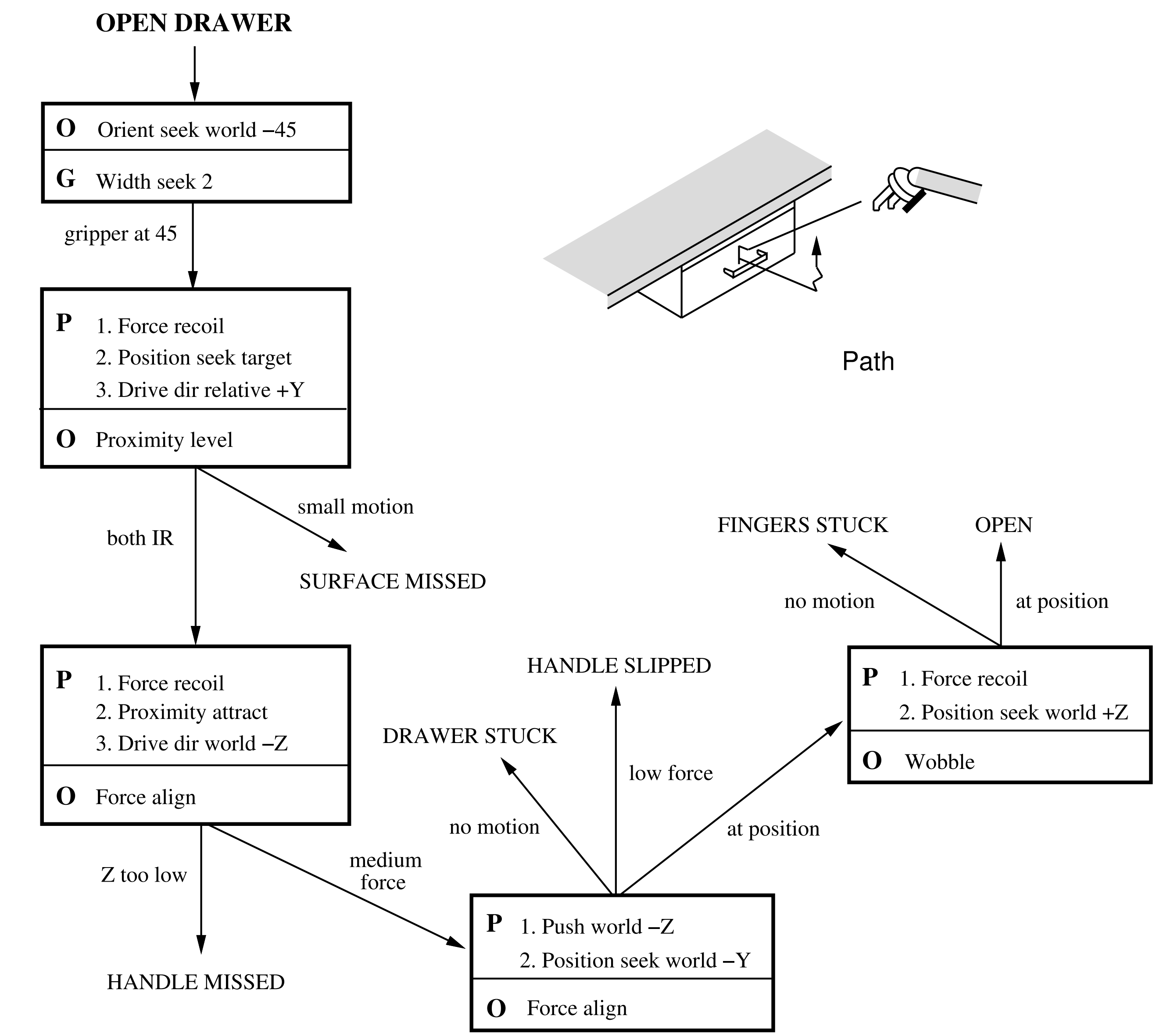

Figure 6: A finite state machine for opening a drawer with a hook grasp. Notice that there are a number of event transitions which signal various types of failures.

instance, Figure 6 shows a finite state machine for opening a drawer. The semantics of this controller are the same as the previous one: there are a series of behavior sets which the robot swaps between on the basis of external sensory events. The component behaviors also largely overlap those for the grabbing system.

Drawer opening is accomplished using the trajectory shown in the upper right corner of Figure 6. The hand is first oriented so the proximity detectors in the fingertips are facing forward and the fingers are somewhat separated. The reorientation is done so that the robot will be able to sense the front face of the drawer and hook its fingers into the handle. The spreading is done to allow for better force sensing by the wrist in later stages. The next control node causes the hand to drive directly toward a visually sensed location corresponding to a portion of the drawer face above the handle. Just in case the visual estimate is off, we add two extra behaviors to compensate for any discrepancies. If the hand stops short of the actual surface, the *Drive dir relative* $+Y$ behavior causes it to continue moving. If the robot mistakenly tries to smash its hand through the drawer, the *Force recoil* behavior will make it back up.

If all goes well, the approach to the drawer face will result in a situation in which the two fingertip IR proximity sensors are activated. At this point, a third set of behaviors is activated to help the robot hook the handle. Basically, the hand is driven downward to engage the bent section of its fingers. Two other positioning behaviors try to keep the hand from scraping along the surface or getting too far away from it. At the same time, the *Force align* behavior looks for any torques produced by a single finger hitting the handle. If this happens, the behavior attempts to rotate the hand in a direction that will equalize the stress. Proper engagement is registered when the robot senses a moderate resisting force.

Next, the robot continues to balance the force on the two fingers while simultaneously pushing downward and pulling back. If the drawer moves properly, control passes to the final node in the finite state machine. In this stage, the robot attempts to release the handle so that it can use its hand to reach inside the drawer or pick up something to deposit there. The primary goal at this stage is to raise the hand a short distance. To help accomplish this the wrist is wobbled around and the hand is instructed to recoil from any collisions. Once a sufficient height is achieved the robot knows that the drawer is open and that the hand is free for other tasks.

This example of drawer opening and the previous example of grabbing objects are just two things that can be done. Many other control programs and actions are also possible. As we have shown, the basic motion vocabulary is very versatile and the combination of finite state sequencing and behavior-based controllers makes for robust systems.

Teaching

In the preceding sections we have shown how to locate and manipulate various items. Much of this hinges on the ability of the vision system to identify a target object in its natural setting. Although it does not need detailed geometric models of things, the vision system does require rough descriptions in terms of size, color, and configuration. These are not all that difficult to enter by hand, but sometimes it is inconvenient to measure the length of a new screwdriver or determine just what shade of yellow its handle is. It would be much more convenient if the system could somehow acquire these characteristic values on its own when prompted to.

Since we use the same camera for training and operation, one way to acquire models is to just isolate the training object from the background and then remember its color and shape parameters. While true size information might be useful and could be obtained using stereo, many times this is not necessary. If the system is trained with object at the same distance it would normally appear, the raw pixel dimensions can be saved instead. Thus, most of training depends on the system's ability to perform figure/ground separation.

There are several ways in which this separation can be accomplished. One of these is shown in Figure 7. Here, the teacher has set out the tools in an uncluttered area so that they are clearly visible. With this presentation the vision system can use a coarse texture algorithm to find them. This algorithm consists of finding strong edges, dilating and eroding them by an amount typical of the clear space around the objects, and then retaining only those blobs that are tool sized. The resulting blobs are measured and their shapes recorded. The blobs are also used to gate the corresponding portions of the original image so that statistics on hue, saturation, and intensity can be computed.

Another way to specify training objects is for the human to wiggle a finger near them. This might be used as a post-processing selection step for a routine like the texture-based separator which sometimes returns multiple objects. It could also be used in conjunction with other candidate object generators. An example of this sort is shown in Figure 8. Here, the system initially finds brightly colored regions that might be handles of tools. The system then watches as the human points out something. Two temporally separated images are differenced and thresholded to find regions of motion. These regions are dilated slightly and used to tag candidate blobs. Any blob not so tagged is discarded since it is unlikely that the human is pointing toward it. Although there are often motion differences along the arm, the differences due to the finger are typically closest to the center. This is because the human's hand usually enters from the side of the image. Thus, of the remaining objects, the system chooses the one closest to the center. Now that the training object has been located, its shape, color, etc. can be recorded as before.

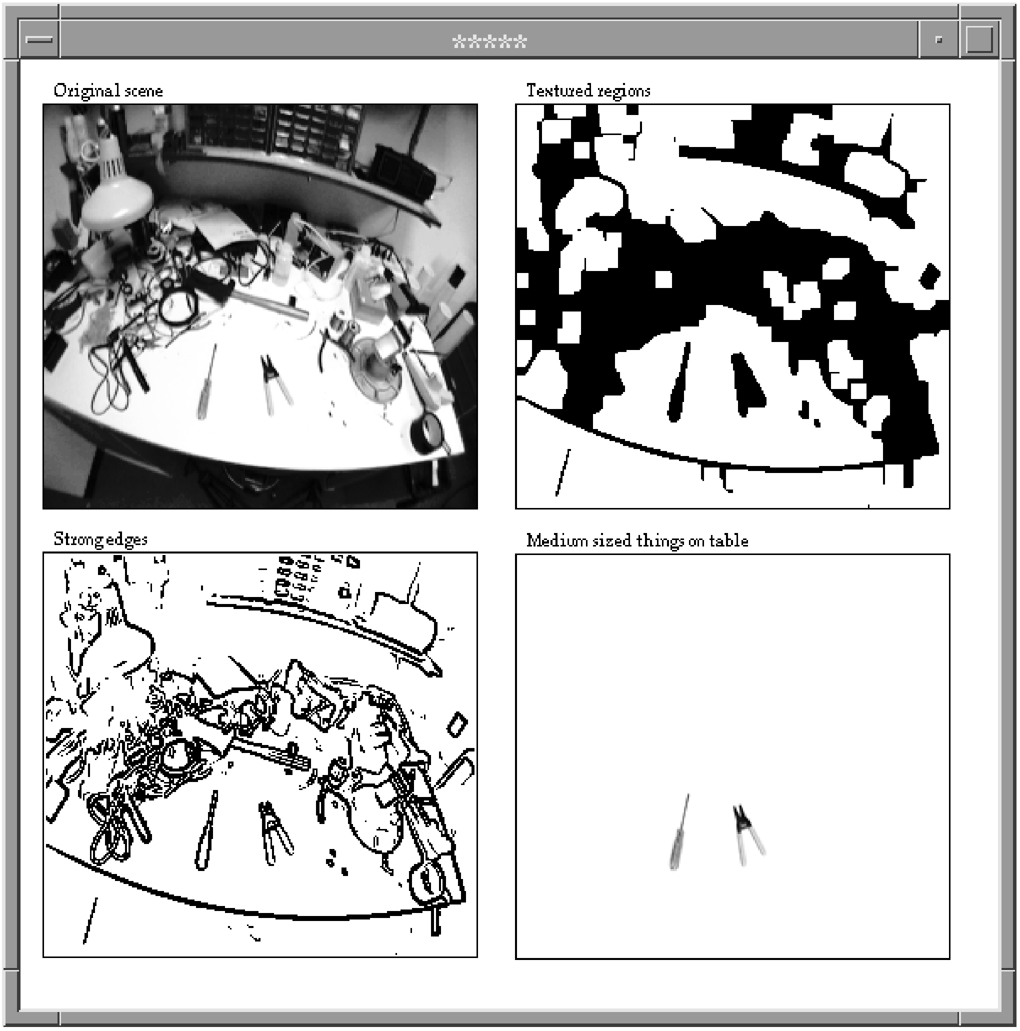

Figure 7: In certain cases the system can separate training objects from the background on the basis of texture.

So far we have discussed the acquisition of object models. The requirements for containers are somewhat different. As shown in Figure 9 there are several kinds of enclosures that the robot might conceivably be able to manipulate. Except for the case where the door has a knob-like handle (upper right picture) the robot needs to be able to locate an appropriate edge (or edges) in the scene. Also, all of the hook-grasp trajectories (arrows in diagram) require a specification of where the hand should initially touchdown. This, in conjunction with the orientation of the edge, specifies which direction of hook to use. In the case of boxes and doors (left panels) the robot then slides along the touchdown surface until it detects the boundary. At this point it executes a ballistic hop to get its fingers behind the surface. The case of drawers with handles has been discussed previously (see Figure 6).

Since furniture can not typically be moved and placed in an uncluttered surrounding, the finger twitch approach is more appropriate for specifying key edges. What is important for containers is usually the orientation and length of the edge. To help disambiguate this edge from similar ones later, the color or textural properties of the adjacent surface might also be recorded. Another important attribute is the edge's actual global position (e.g top drawer vs. middle drawer). This can be obtained using vergence-style stereo (with a vertical baseline in some instances) as described earlier. Finally, the touchdown point can also be indicated with a finger twitch. In this case no specific region or edge needs to be identified. All that matters is to which side of the edge the touchdown should occur.

Thus, not only does our robot require minimal models for operation, it looks like most of this modelling can be

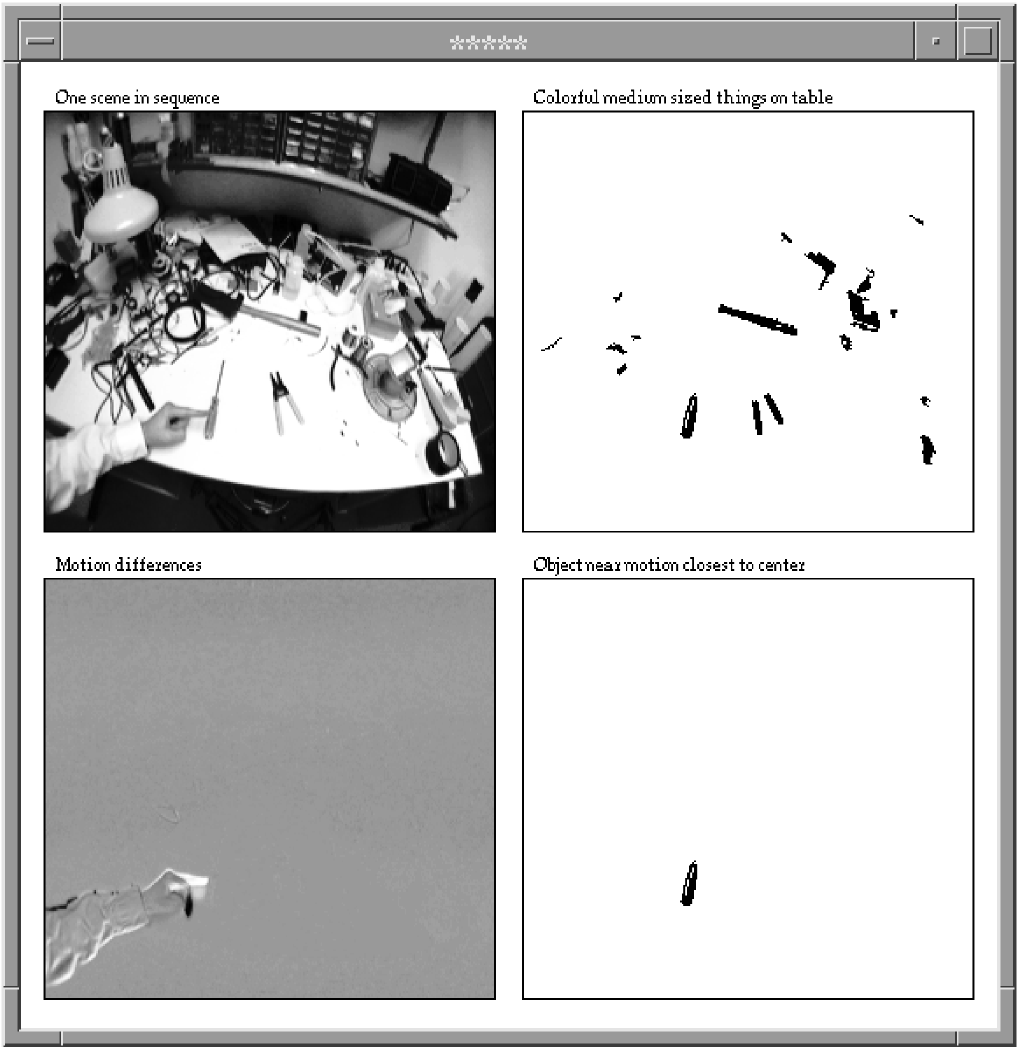

Figure 8: Motion can also be used to direct attention to particular objects or scene features.

done automatically. This would allow us to build a instructible fetch-and-carry robot which could be trained to respond appropriately to such high-level commands as: "Get me the big screwdriver from the top drawer in the lab".

Discussion

The BRANTA system is related to a number of other pieces of work that have appeared in the literature. In particular, our vision system takes a largely top-down knowledge-based approach. It knows what it is looking for and where the object is likely to be. It does not try to simultaneous solve the "what" and "where" questions (cf. [Ballard, 1991, pg. 73]) in a purely data-driven manner. However, it is still partially bottom-up to the extent that it relies on a number of general-purpose visual operators rather than detailed object models.

Several other research groups have used a similar mixed approach. For instance, the TEA system developed at Rochester [Rimey and Brown, 1992] incorporates a number of generic visual operators, such as a Hough transform circle finder and a color histogram matcher. These, however, are not adequate for directly locating the target object in a general setting. Yet with some knowledge of the expected structure of the scene, these weak operators can be sufficiently constrained to give useful results. The authors give an example where the location of a large round object (a plate) is used to limit the search space for a particular small round object (the adjacent cup).

Another related scheme is the VISIONS system developed at UMass Amherst [Riseman and Hanson, 1987]. Here the basic operators are full image segmen-

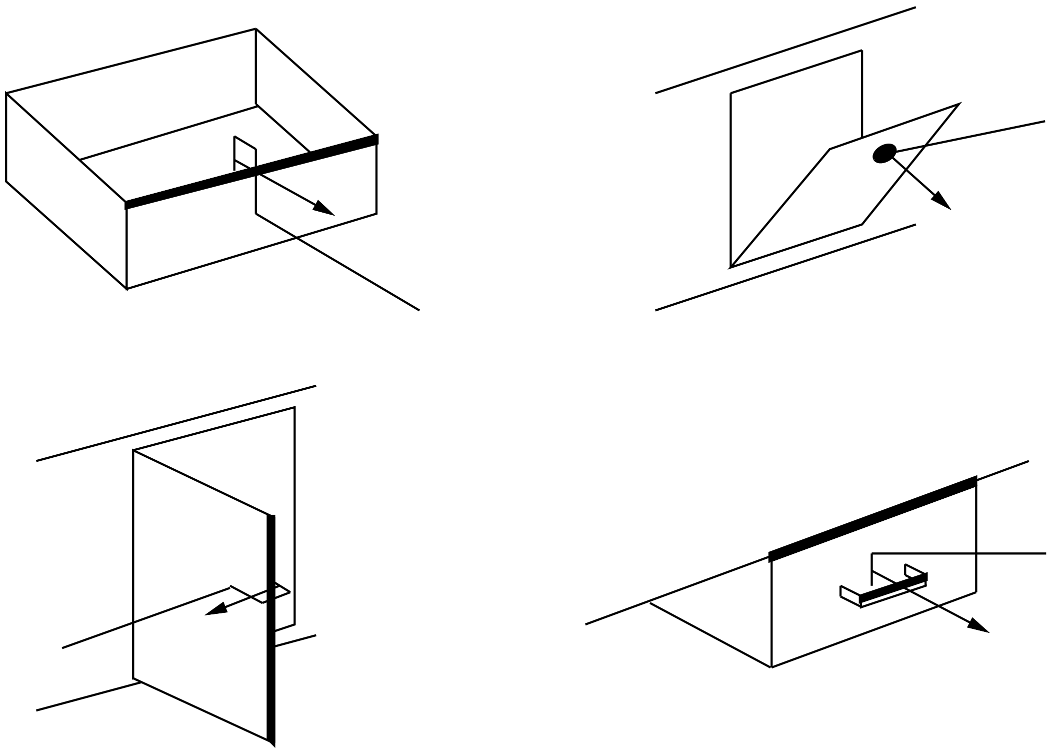

Figure 9: To manipulate a container the robot needs to find a particular edge for a hook grasp (left two cases), an initial contact area between a pair of edges (lower right), or a pinch-graspable handle (top right).

tation, color range gating, straight line finding, and texture energy determination. Once again, although these primitives are not powerful enough to uniquely identify objects in the scene, they become much more useful when combined with size and positional constraints. To find the roof of a house, initially large areas of a certain color and texture are sought. Next, one such region is selected and adjacent lines are extracted which run tangentially to it. Finally, these special lines are examined for parallelism and the collection is checked to see whether it forms a proper polygonal closed boundary for the roof. This sequence of steps is much easier than trying to interpret the myriad of original lines. It also yields much more accurate boundaries than could be obtained just by looking at colored regions.

Still another system of this flavor is Condor [Strat and Fischler, 1991]. This system combines a number of features extractors, such as color and texture, with geometric constraints, like LEANING and NO-SKY-BELOW, in order to interpret outdoor scenes. For example, to locate the sky in an image Condor first performs a full segmentation. It then finds regions which are either fairly blue, very bright, or weakly textured. Next, regions which extend below the horizon (as determined separately) are discarded. The final phase of processing uses serial attention to assemble a complete collection of regions which constitute the sky. The system starts by picking out the best region based on blueness, brightness, and smoothness as well as on the proximity of the region to the top of the image. It then checks whether the brightness of the next best remaining region is sufficiently similar to the current set of sky regions. If so, it is added to the set. This selection process is repeated until the full sky is found. In this way Condor follows the same basic steps as BRANTA: feature extraction, filtering based on region shape/size/position, and attentional selection.

In some sense Chapman's Sonja system [Chapman, 1991] and the earlier Pengi system [Agre and Chapman, 1987] also follow this general vision approach. Both systems first perform a bottom-up classification of scene regions and then use visual routines to identify specific items of interest. However, the operators used (like tracking and ray tracing) tend to be more elaborate than the ones in BRANTA. Also, because these programs were created to play video games, they have direct access to a perfect segmentation and classification of "pixels". Still, to determine the functional significance of a object, this bottom-up classification alone is insufficient – the object's geometric relationships also have to be considered. For instance, a region labelled as an "ice cube" could only be used as a "projectile" if its centroid was on the line connecting the penguin and the killer bee.

The second part of BRANTA, the vocabulary of motion primitives and the finite state controller, was based on previous experience with a simpler arm [Connell, 1989]. This robot used a behavior-based approach to bounce its way across a table in search of graspable objects. An improved version of the basic architecture was also used on a navigating robot [Connell, 1992]. Both systems operate by simultaneously activating a loosely coupled set of behaviors to achieve some task, an idea similar to Minsky's K-lines [Minsky, 1986, pg. 82]. The strong point of this approach is that routine anomalies in the environment are handled automatically by the low level tactical behaviors. The strategic level is only re-invoked when things when things go seriously awry or certain key events are detected. The type of variations handled autonomously by the arm included radial displacement of the target soda can as well as rises and dips in the table top terrain. The navigating robot was able to independently cope with small obstacles in its path and to stay in the center of hallways despite odometric drift.

The egg-flipping robot developed at Rochester [Pook and Ballard, 1992] also exhibits some of this local autonomy. For instance, in one step the robot rocks its spatula against the pan. This effectively compensates for small tilts produced by the initial grasping operation without requiring any re-planning of the overall sequence. Similarly, in the next step the spatula is slid along the bottom of the pan until it hits the far side of the pan. This serves to scoop the egg up irregardless of its exact position or mass distribution. However, unlike BRANTA, this robot does not use multiple concurrent processes to achieve its robustness.

In summary, we have described a robot manipulation system which uses information from vision, proximity, force/torque, and kinesthetic sensors to interact with containers and accomplish grasping tasks. We have also discussed a vision system based on both edges and regions which uses (at least to some extent) color, shape, structure, size, position, orientation, texture, and stereo to find target objects. While our robot system is not yet complete and has not been mounted on a vehicle, the preliminary results look encouraging and integration of all the parts seems feasible.

References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of action. In *Proceedings of AAAI-87*, pages 268–272, 1987.

[Ballard, 1991] Dana H. Ballard. Animate vision. *Artificial Intelligence*, 48:57–86, 1991.

[Chapman, 1991] David Chapman. *Vision, Instruction, and Action.* MIT Press, Cambridge MA, 1991.

[Connell, 1989] Jonathan H. Connell. A behavior-based arm controller. *IEEE Transactions on Robotics and Automation*, 5(6):784–791, 1989.

[Connell, 1992] Jonathan H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2719–2724, 1992.

[Minsky, 1986] Marvin Minsky. *The Society of Mind.* Simon and Schuster, New York, 1986.

[Pook and Ballard, 1992] Polly Pook and Dana H. Ballard. Contextually dependent control strategies for manipulation. Technical Report CS-416, University of Rochester, 1992.

[Rimey and Brown, 1992] Raymond Rimey and Christopher Brown. Task-oriented vision with multiple Bayes nets. In Andrew Blake and Alan Yuille, editors, *Active Vision.* MIT Press, Cambridge MA, 1992.

[Riseman and Hanson, 1987] Edward M. Riseman and Allen R. Hanson. A methodology for the development of general knowledge-based vision systems. In Michael Arbib and Allen Hanson, editors, *Vision, Brain, and Cooperative Computation.* MIT Press, Cambridge MA, 1987.

[Strat and Fischler, 1991] Thomas M. Strat and Martin A. Fischler. Context-based vision: Recognizing objects using information from both 2-D and 3-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1050–1065, 1991.