Auton Robot (2007) 22:425–435
DOI 10.1007/s10514-007-9024-0

Dual-coding representations for robot vision programming in Tekkotsu

David S. Touretzky · Neil S. Halelamien · Ethan J. Tira-Thompson · Jordan J. Wales · Kei Usui

Received: 8 March 2006 / Revised: 24 November 2006 / Accepted: 2 January 2007 / Published online: 30 January 2007

Abstract We describe complementary iconic and symbolic representations for parsing the visual world. The iconic pixmap representation is operated on by an extensible set of "visual routines" (Ullman, 1984; Forbus et al., 2001). A symbolic representation, in terms of lines, ellipses, blobs, etc., is extracted from the iconic encoding, manipulated algebraically, and re-rendered iconically. The two representations are therefore duals, and iconic operations can be freely intermixed with symbolic ones. The dual-coding approach offers robot programmers a versatile collection of primitives from which to construct application-specific vision software. We describe some sample applications implemented on the Sony AIBO.

Keywords Dual-coding theory · Visual routines · Robot vision · Educational robotics

D. S. Touretzky (✉)
Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
e-mail: dst@cs.cmu.edu

N. S. Halelamien
Computation and Neural Systems Program, California Institute of Technology
e-mail: neilh@caltech.edu

E. J. Tira-Thompson · K. Usui
Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
e-mail: ejt@andrew.cmu.edu

K. Usui
e-mail: kusui@andrew.cmu.edu

J. J. Wales
P. O. Box 821650, Vancouver, WA 98682
e-mail: wales@intuitiveimpulse.com

1 Introduction

We present an approach to robot vision programming in which objects are dually represented in iconic (pixel array) and symbolic (parameterized shape) forms. A correspondence is maintained between iconic and symbolic spaces so that the programmer is free to operate on objects in whichever form is most convenient. This facility forms part of Tekkotsu (Tira-Thompson, 2004; see also `Tekkotsu.org`), an application development framework for intelligent robots that is presently implemented on the Sony AIBO, with support for other platforms in progress.

1.1 Historical antecedents

Our approach draws on several ideas from cognitive science. The term *dual coding* comes from Paivio's "dual coding theory" of mental representations (Paivio, 1986), which posits parallel verbal and non-verbal (image-based) systems with extensive referential connections between them. An example of a verbal representation would be a syllogism ("All P's are Q's; all Q's are R's; therefore all P's are R's"); an example of a non-verbal representation would be a Venn diagram. On this view, cognitive problem solving invokes operations on items in one or the other of these two representational systems. Paivio hypothesized that more demanding tasks require translation between the two, which can be detected in human subjects through higher error rates and slower reaction times.

Another important source of inspiration is Ullman's notion of "visual routines" (Ullman, 1984). Ullman proposed a set of elemental operations for low to mid-level vision, some of which can be performed in parallel over the entire image. Examples of such operations include selecting pixels based on a sensory feature such as color, and bounded

spread of activation across pixels (coloring). The operations can be composed in many ways to produce specialized visual routines for different tasks. Somewhat higher level operations, not yet incorporated into our implementation, include boundary tracing and shift of attention. Roelfsema et al. (2000) conducted both psychophysical and neurophysiological investigations of Ullman's hypothesis using a curve tracing task. They found evidence for incremental spreading of activation along the curve, similar to what Ullman hypothesized, within the primary visual cortex of monkeys.

Our framework uses visual routines to operate on iconic (pixel-based/nonverbal) representations. But our argument in this paper is that incorporation of a complementary symbolic (semantic/verbal) representation is advantageous when constructing algorithms to parse the visual world.

Other uses of the term "visual routines" in AI and robotics, inspired by Ullman's work, have influenced our own. Rao and Ballard's (1995) active vision architecture made use of dual memory systems—one indexed by (x, y) values in the image space, the other indexed by object identities. This dual memory system is loosely analogous to our framework's dual-coding representations, but with less emphasis on interactions between the two representations. Forbus et al.'s (2001) work, concerned with qualitative spatial reasoning on maps, provided the basis for several of the iconic operators present in our framework. Agre and Chapman's project, PENGI, implemented Ullman's concept of visual routines as part of an automated system for playing a real-time two-dimensional video game (Agre and Chapman, 1987). Their work also introduced the notion of a "deictic representation:" a means for representing entities in the environment in terms of their relations to the perceiver, which may be a source of fruitful extensions to the work we describe here.

1.2 Map building and visualization

Reducing a camera image to a symbolic description, as occurs when we translate between representations, simplifies the problem of identifying points of correspondence across images as the camera moves. This permits construction of a world map from a series of overlapping images, freeing a robot from the limitations of a camera with a narrow field of view. The AIBO's single camera, located in its "nose," has a 56.9° horizontal field of view (Sony Corp., 2004), compared with around 200° for binocular humans and 300° for rats.

Our world map representation, like the representation of camera space, utilizes both iconic and symbolic components. It serves as a working memory for objects the robot has encountered but cannot currently see, and is also useful for path planning.

Another notable feature of our approach is an emphasis on transparency of computation. Operators that derive new iconic representations (called "sketches") or symbolic objects (called "shapes") from existing ones automatically record their parentage, so that computations performed within the dual-coding framework form a derivation tree with the initial camera image at the root and the most recently derived objects at the leaves. The structure of this tree, as well as the individual sketches and shapes that populate it, is made visible to the user via an interactive GUI tool that allows the robot's "mental imagery" to be monitored remotely over a wireless connection. This is not only valuable for debugging vision algorithms, it also provides a convenient way for the robot to communicate its parse of the world to its human companions.

This software received a Technical Innovation Award at AAAI-05 for visualization for educational robotics. It is presently used in undergraduate Cognitive Robotics courses at Carnegie Mellon University and Spelman College. The course and its associated software will be implemented at several other colleges over the next two years.

2 The quasi-planar world assumption

Our initial experiments use a quasi-planar domain: the tabletop. It is "quasi" planar in the sense that objects are treated as having negligible height, but occlusions can still occur when objects overlap. Even this simple world is very rich, and provides a wealth of opportunities for experimental robotics. For example, Fig. 1(a) shows a view of a tic-tac-toe board through the AIBO's camera. Notice that several game pieces are partially occluding board lines. In order to play the game, the robot must parse camera images into board configurations. The first test of our framework is to show how this can be done in a straightforward way using the primitives we provide.

Tekkotsu's kinematics package calculates the current camera pose from joint angle information reported by the robot's sensors. Combined with the quasi-planar world assumption, this permits us to determine the projection of any camera pixel onto the ground plane on which the robot is standing, and thereby derive distance information from single camera frames.

3 Iconic representations: Sketch space

The iconic side of the dual-coding representation scheme is organized as a collection of sketch objects. Each sketch references a two-dimensional array of pixels of some type (bool, unsigned byte, etc.). All sketches within a "sketch space" have the same dimensions, so they are always in

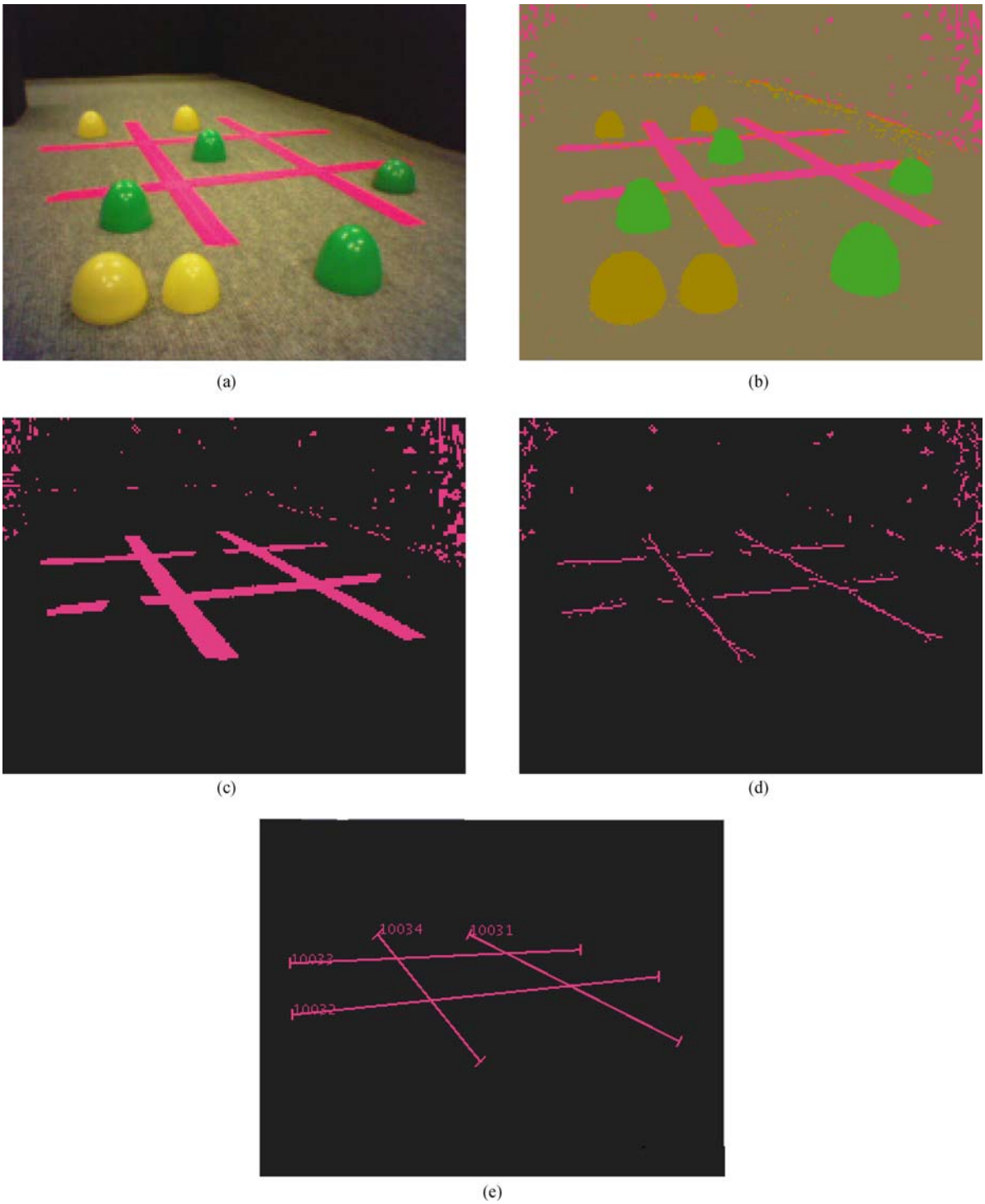


Fig. 1 Iconic operations underlying line extraction. (a) View of the tic-tac-toe board through the AIBO's camera; (b) color-segmented camera image; (c) extracting the pink pixels; (d) skeletonization; (e) extracted lines. The numbers next to each line are the shape ids; valid line endpoints are indicated by perpendicular tick marks

register. Sketches of unsigned bytes are used to store color segmented images; the bytes are indices into a color table. Sketches of bools are used to indicate which pixels have a certain property. Sketches of integers are used for tasks such as region labeling and distance maps.

3.1 Iconic operations

We provide the usual arithmetic and comparison operators for element-wise operations on sketches. In addition, we provide a variety of standard image processing functions, such

as morphological thinning and thickening, edge detection, skeletonization, connected-components labeling, Manhattan distance, and region filling (Glassner, 1990). We have also implemented a bounded-distance operator that takes in *destination* and *boundary* sketches and applies the wavefront algorithm to calculate the boundary-respecting path distance between each non-boundary pixel and the nearest destination pixel. There also exist operators for calculating horizontal and vertical symmetry transforms over a sketch, and computing the convex hull. Most of these operators were described in Halelamien (2004).

Similar operators are found in other computer vision packages, such as OpenCV, the Intel open source computer vision library (Landre, 2004). Our current operator set is less comprehensive than that of OpenCV, and not optimized for efficiency. Our goal is not to compete with OpenCV; rather it is to show that an approach that combines iconic and symbolic representations is useful for robot vision. The iconic portion of our framework could in principle be built on top of OpenCV, although since the AIBO uses a MIPS chip, it would not benefit from Intel's architecture-specific optimizations (IPP, or Intel Performance Primitives) that speed up OpenCV on the Pentium.

3.2 Tic-tac-toe

The first stages of processing the tic-tac-toe board are shown in Fig. 1. We begin by importing a color segmented camera image as a sketch of unsigned bytes. The color segmentation is done using CMVision (Bruce et al., 2000). The result is Fig. 1(b). We then locate the pink pixels using a comparison operator, colormask, which performs an equality test between each pixel and a 1-byte color index value; the result is a sketch of bools (Fig. 1(c)). After shrink-and-grow region smoothing, we skeletonize the smoothed sketch (Fig. 1(d)) in preparation for extracting line shapes (Fig. 1(e)). Similar operations are used to extract the green and yellow pixels representing the game pieces.

Each sketch is assigned a unique integer id, and parent id's are maintained automatically by operators that derive new sketches from old ones. Thus, the parent of the skeleton sketch is the smoothed sketch, whose parent is the sketch indicating all the pink pixels, whose parent is the color-segmented camera image, which has no parent. For unary sketch operations like skeletonization, the parent relationship is obvious. For binary operations such as pixel-wise arithmetic or comparison, we have established the convention that the left argument is always the parent of the result. This makes parentage predictable and consistent. Parent information is used to organize sketches into a derivation tree, as discussed in Section 7, and also governs inheritance of certain properties, such as color for sketches of bools.

4 Symbolic representations: Shape space

While sketches are well-suited to manipulating regions in an image, for many operations it is more productive to describe image elements in symbolic form, in terms of geometric constructs such as points, lines or ellipses. (Circles lying in the ground plane are more properly treated as ellipses due to perspective effects.) We define a family of shape objects analogous to the family of sketches of various types. The most basic shape types are point, line, ellipse, and blob. Shapes exist within a shape space, which is directly associated with a sketch space. Parent information for shapes is maintained automatically, as for sketches.

Lines, rays, and line segments are all represented using the line shape. The difference is in the number of effective endpoints. A line segment has two endpoints, a ray has one, and a line has none. If fewer than two endpoints are known, a line can be created by specifying a point through which it passes, plus an orientation. When extracting lines from a camera image, an endpoint is noted but marked invalid if it occurs at the edge of the sketch, since in that case the true endpoint of the line is unknown.

Ellipse shapes have a center, an orientation for their major axis, and lengths for their major and minor axes. Blob shapes are represented by a bounding box, and by a run-length encoded list of pixels which form the region.

All shapes also have a color attribute, which is used when rendering them iconically or displaying them using the GUI tool. Color is an inheritable property of both sketches and shapes. Thus, when pink pixels are extracted from a color-segmented image by the colormask operator, the resulting sketch of bools is assigned color pink. When line shapes are then extracted from this sketch, the lines are also marked as pink (Fig. 1(e)).

4.1 Shape construction and updating

There are three fundamental kinds of shape operations: construction, updating, and relational calculations. The simplest way to construct new shapes is by directly specifying values for each of their components, e.g., constructing a line by giving the two endpoints. In practice it is more common to construct new shapes by specifying their relationships to existing ones. For example, given a line A, one might want to construct a bisector B that runs perpendicular to A and passes through its midpoint. We provide a variety of constructor functions to handle the common cases; users can add their own constructors to augment this set.

Each shape potentially has an iconic representation (as a sketch of bools) that is computed when needed and cached for reuse. Updating a shape by altering its parameter values invalidates this cached rendering. A common type of update for line segments is to temporarily inactivate their endpoints,

so that they become infinite lines and can serve as boundaries for region coloring operations.

4.2 Relational calculations

Relational calculations are the richest group of shape operations. Our goal is to provide a natural-sounding language for talking about shape relations, so that algorithms for parsing a scene can be described in intuitive terms. For example, the two endpoints that define a line segment must be distinguished somehow, but the choice of which should be the "first" endpoint and which the "second" is arbitrary. It is more natural to talk in terms of the "left" and "right" endpoints (for a line that can be seen as horizontal), or the "top" and "bottom" endpoints (for a line that can be seen as vertical). We provide accessor functions for lines that compute these relations and return the appropriate endpoint. Similarly, when looking at any two shapes, we provide functions to select the leftmost or rightmost of the two, or the topmost or bottommost, based on their centroids.

In some situations, such as when matching one line against another, absolute orientation isn't important, but we want to see whether corresponding endpoints match, e.g., whether two line segments are coextensive, or if one segment is a continuation of the other. We define the "first" endpoint of a line to be the leftmost endpoint if the line can be seen as horizontal (slope less than 60°); otherwise it is the topmost endpoint. Then we can talk about first and second endpoints without explicitly considering line orientation. But this raises a problem when comparing two lines near the horizontal/vertical cutoff: if only one of them passes the horizontal test, we could end up comparing the leftmost point of one with the topmost (which could be rightmost) of the other. The solution is for the endpoint selection function to allow one line to make the horizontal/vertical decision for both. Even if the second line is a little too steep to pass the horizontal test, it can't be that far off in orientation from the first line if the match is to succeed, so the notion of a leftmost point must still be well-defined.

The built-in line relationship tests include parallel, perpendicular, and colinear relationships (all with user-overridable comparison tolerances), an intersection test, and length comparison. The latter can be used with a sort primitive if we wish to consider lines in order of decreasing length.

5 Translation between representations

5.1 Iconic to symbolic: Shape extraction

Extraction operators translate from the iconic space to the symbolic space, i.e., they extract shapes from sketches. In the simplest case, these operators take as input a sketch of bools and produce zero or more shapes as output. Three such operators are currently provided. The line extraction operator uses moment statistics (Prokop and Reeves, 1992) to extract a line shape with the orientation and endpoints of the most prominent line segment in the image. Multiple lines can be extracted by applying a line-clearing operation (using the logical AND-NOT function) to the sketch and re-running the operator to extract another line. The user can specify the colors of "occluder" objects, and the line extractor will continue a line through any occluder regions. The user can also specify a maximum number of lines to be extracted, and a minimum length in pixels for each line. The result of the extraction is returned as a vector of line shapes.

The ellipse extraction operator returns a vector of ellipse shapes corresponding to all the ellipse-like regions in the sketch. It uses connected-components region labeling to find candidate ellipses, and computes the major axis length, minor axis length, and principal orientation using moment-based calculations (Prokop and Reeves, 1992). Regions smaller than a minimum size, or with extreme ratios of major to minor axis lengths, are discarded. Blob extraction is done by connected components labeling.

The complete set of extracted shapes from the tic-tac-toe board is shown in Fig. 2(a).

5.2 Symbolic to iconic: Rendering

All symbolic objects potentially have iconic renderings as sketches of bools, created on demand. Lines are rendered using variations on the Bresenham algorithm (Bresenham, 1965). Blobs are rendered by filling in their runs.

When a symbolic object is rendered, the iconic representation created in the sketch space links back to its symbolic parent in the associated shape space. The two representations form one entity, with the sketch dependent upon the shape, such that any change to the symbolic parameters will invalidate the sketch. If the user then attempts to access the iconic representation again, it will be re-rendered from the symbolic parameters.

On a broader view, the rendering of a shape onto a sketch can be seen as analogous to activity induced in primary visual cortex by higher level visual areas responsible for the perception of edges or boundaries in an image. These areas are responsible for such psychophysical phenomena as the illusory contours seen in the well-known Kanizsa triangle. Within our framework, the addition of gestalt perception operations for inferring boundaries in an image would, via the rendering operation, result in an explicit representation of these boundaries as activated pixels.

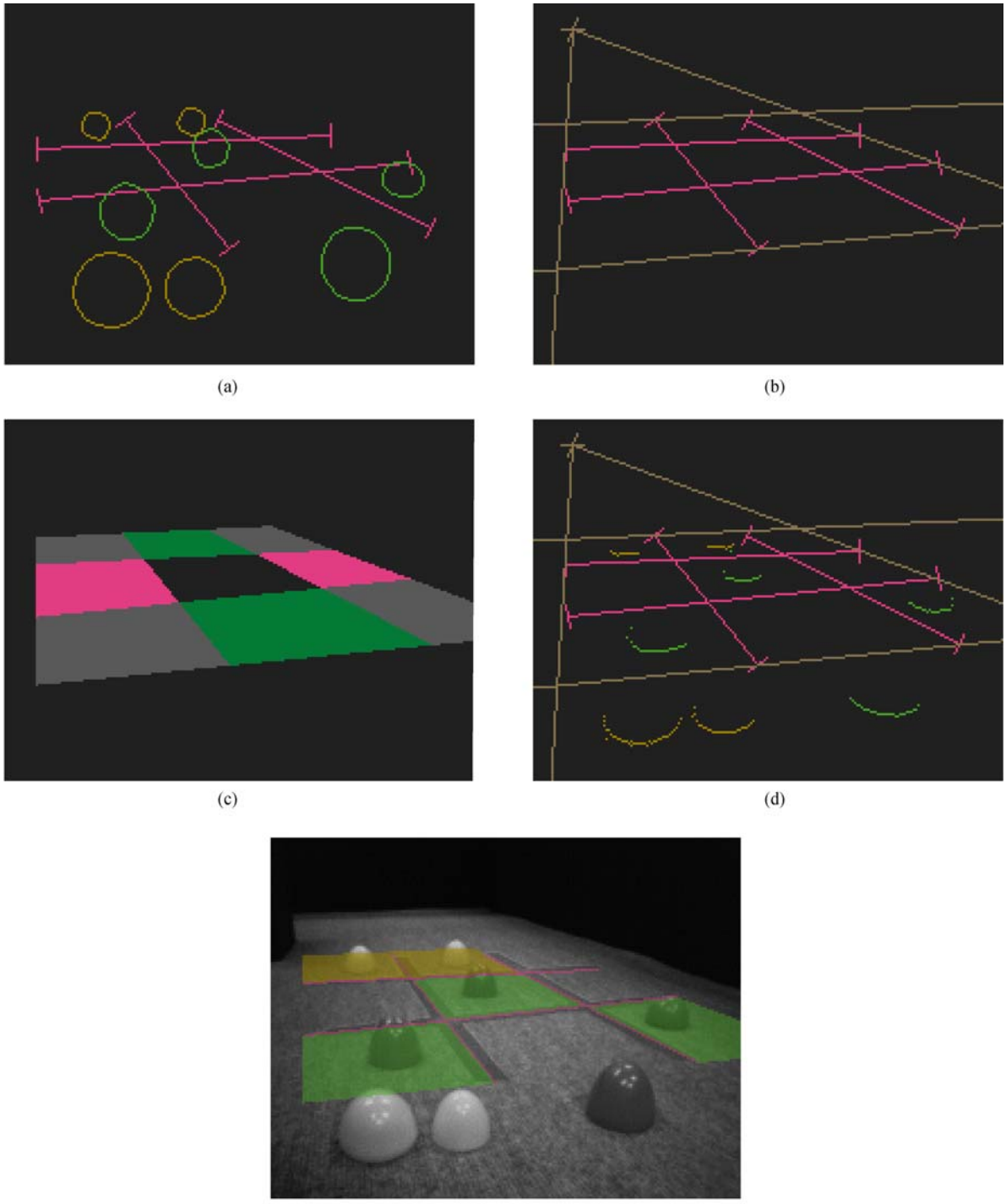


Fig. 2 Stages in parsing the tic-tac-toe board. (a) Extracted line and ellipse shapes; (b) board boundary lines; (c) board regions; (d) bottom edges of game pieces; (e) completed parse

6 Parsing the tic-tac-toe board

Because the robot is playing against a human, we must take pains to interpret the scene in the same way as the human opponent. The task of parsing real board images in a human-like way is complicated by two considerations. First, game pieces that have not yet been placed on the board may lie nearby. Thus it is necessary to find the outer boundaries of the board—which are not marked explicitly in the scene—in order to determine which game pieces are truly on the board

and which merely lie beside it. Second, perspective effects and noise introduced by the segmentation and line extraction processes lead to uncertainty in the line parameters. We must therefore seek heuristics to produce results in agreement with human assumptions about what the board must look like.

Having extracted a set of lines from the image, and two sets of ellipses, our heuristic algorithm for parsing the scene is as follows:

1. *Find the board lines:* From the set of extracted lines, find the longest pair of roughly horizontal lines. Then find the longest two lines that are not parallel to the first set. (We consider lines to be not parallel if their orientations differ by more than 20°.) Note that the latter two lines need not be vertical, nor mutually parallel, due to perspective effects.
2. *Bound the board:* Construct boundary lines from the most extreme points on the board lines, keeping in mind that perspective effects mean the two roughly vertical lines may actually be converging. If so, the vertical boundary lines should pass through this convergence point; otherwise they are made to parallel the slopes of the two vertical board lines. In either case, the vertical boundary lines pass through the most extreme leftmost or rightmost point of the two horizontal board lines. The result is Fig. 2(b). These tan-colored lines mark the edges of the board.
3. *Break the board into cells:* Locate the nine board regions delineated by the board lines and boundary lines. This can be done by coloring the half-planes defined by each line segment (a sketch operation), and then computing various combinations of intersections of colored regions. The result is Fig. 2(c).
4. *Determine occupancy:* decide which board positions are occupied, and by which color game pieces, by intersecting the board region sketches with the bottom edges of the renderings of the ellipses, as shown in Fig. 2(d). We use the bottom edges because the game pieces have non-negligible height, violating the quasi-planar world assumption. The bottom edges are the best indicator of where the ellipse contacts the ground plane.

The final parse is shown in Fig. 2(e). This example illustrates how a dual-coding representation fosters simple, natural descriptions of visual reasoning algorithms.

7 Visualizing the computation

To facilitate debugging of vision algorithms, we provide a graphical viewer for examining the contents of a sketch space and its associated shape space while the robot is running. Sketches and shapes are organized into a derivation tree, using automatically maintained parent information. The viewer displays this tree, as in Fig. 3, and the user can individually select and deselect objects to be displayed. If multiple sketches are selected, they are combined by addition. Selected shapes are then drawn on top of the sketch image.

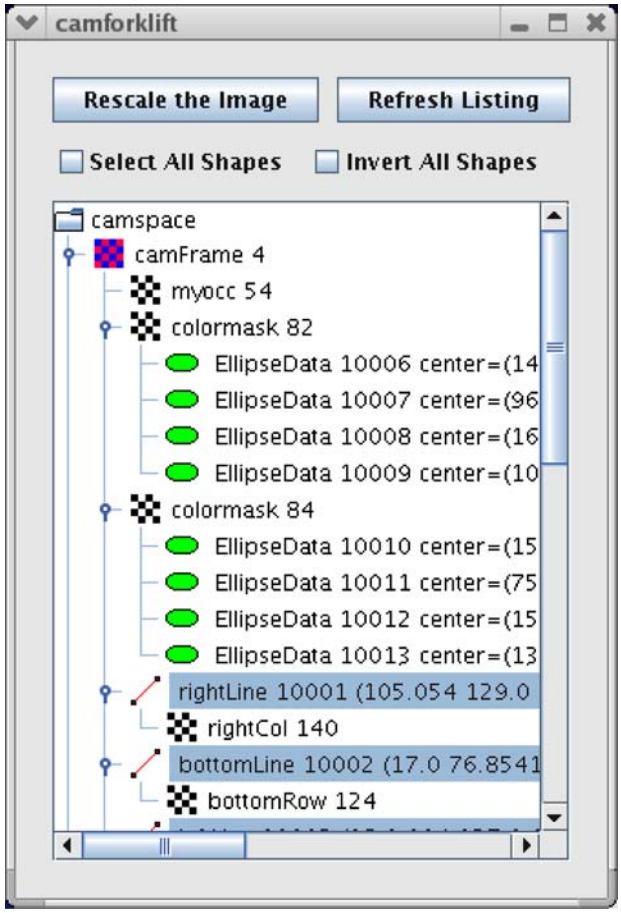


Fig. 3 GUI tool for examining the derivation tree. Clicking on one or more items causes them to be displayed in another window; this is how the images in Figs. 1(b)–(e) and 2 were produced

To begin examining the parse of an image, the viewer program downloads a "table of contents" from the robot: a listing of all available sketches and shapes. To conserve time and wireless bandwidth, the robot does not transmit all its sketches at once. Instead, individual sketches are downloaded to the viewer on demand when the user clicks on the corresponding menu item. But since each shape is described by just a few parameters, such as endpoints for a line, centroid and major/minor axes for an ellipse, or bounding box for a blob, full shape descriptions are included as part of the table of contents rather than being transmitted on demand.

Complex scene parsing algorithms can generate many intermediate results, producing a cluttered derivation tree. We allow the programmer to control which objects are visible to the viewer by setting an attribute of the sketch or shape. Unimportant entities can thereby be suppressed from the display.

8 Building a world map

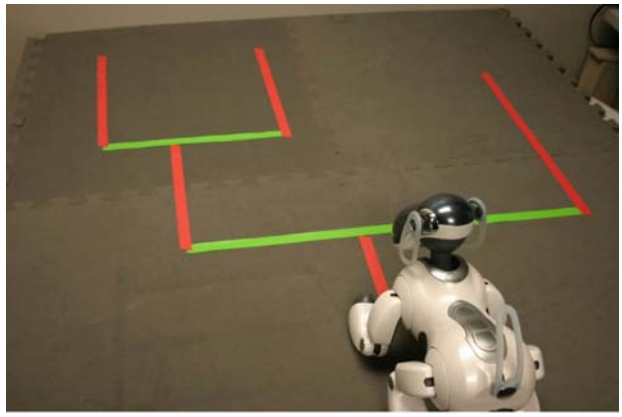
In the preceding example, the tic-tac-toe board fit within a single camera image, so all processing could be done in camera space. But in general this will not be the case. We have incorporated a map building facility that can piece together larger maps from multiple snapshots. Starting with an initial camera image, the first step in this process is to project the extracted shapes onto the groundplane. We thus go from a camera-centered reference frame to a body-centered one. This projection requires knowing the camera's pose relative to the ground, which is computed by Tekkotsu's kinematics engine. The orientation of the ground plane with respect to the body is calculated by using the AIBO's accelerometers (which indicate the gravity vector) and a heuristic that attempts to determine three points on the frame of the body most likely to be in contact with the ground.

Once shapes are translated into body coordinates, they are used to create the first draft of a local (egocentric) map. When the next camera image is processed, its shapes are also projected onto the groundplane and then heuristically matched against those already in the local map, taking noise and uncertainty into account. Heuristics are used to generate a sequence of gazepoints to maximize the information gained with each head movement. For example, if a line runs off the edge of the camera frame, a new gazepoint will be proposed to try to locate the true endpoint; if an object has only been detected in one image, a new gazepoint centered on the object will be proposed to confirm that the object is really there. In this way, lines that are only partially visible in one camera frame can be pieced together from several images, while spurious shapes can be rejected.

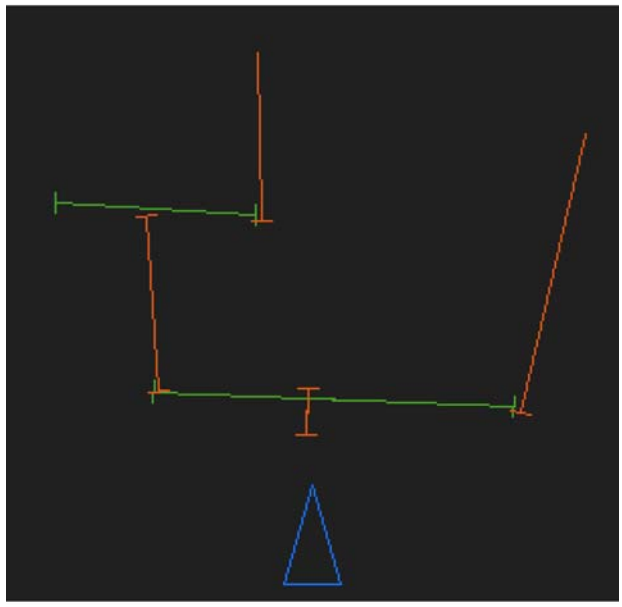
The map builder also has a "rapid scan" mode in which it moves the head along a continuous trajectory to rapidly scan a user-specified region, looking for color patches of potential interest. It then goes back and examines those patches systematically to extract shapes.

In addition to the egocentric local map, the map builder also maintains an allocentric world map on which the robot's own position and orientation are represented. The world map is assembled from multiple local maps as the robot moves through its environment. A simple particle filter is used for localization. Since the local and world maps are both shape spaces, they are visible using the same GUI tool that monitors the camera's shape space.

Figure 4(a) shows another task to which we've applied our dual coding approach: walking a binary tree. The tree is laid out on a table top with colored masking tape. Only a small portion is visible at any one time through the AIBO's camera. Figure 4(b) shows a world map automatically constructed by the map builder from multiple camera images as the AIBO stood at the root of the tree. To produce such a map, the user simply indicates the colors of the different shape types and occluders expected in the scene, and certain other parameters such as the region around the body that is to be scanned, and the minimum acceptable length for any extracted lines. Head motion, shape extraction, and shape matching are performed by the map builder based on these parameters.



(a)



(b)

Fig. 4 (a) Binary tree, drawn with masking tape, which the AIBO is asked to traverse; (b) world map representation of part of the tree, constructed by the map builder. The blue triangle indicates the AIBO's position and orientation

A complication arises from the fact that different shape spaces use different coordinate systems. Camera space follows the usual image processing convention where the first coordinate is horizontal position, the second coordinate is vertical position, and the origin is at the upper left corner of the image. Local space, defined relative to the robot's body, follows the Denavit-Hartenberg convention used by Tekkotsu's kinematics package: the x coordinate increases in the forward direction from the robot's midline, and the y coordinate increases to the robot's left.

Shapes in camera space are projected into local space via a coordinate transformation based on the current camera pose. If the robot's head is pointing straight ahead, relationships between shapes in camera space should appear similar to those between corresponding shapes in local space, e.g., objects that appear near the top of the camera image (smaller y coordinates in camera space) should be judged farther away (larger x coordinates in local space). The GUI tool corrects for the difference in coordinate conventions, so shapes that are distant in local space appear near the top of the local space display.

To keep things intuitive for the programmer, relations like "left of" should mean the same thing in either display, so their implementation must be sensitive to the reference frame in use. Each shape space therefore has a reference frame attribute whose value can be *camera-centric*, *egocentric*, or *allocentric*. Relational primitives are defined separately for each reference frame type. "Left of" means a smaller x coordinate in a camera-centric space, and a larger y coordinate in an egocentric space. In world space, where the robot's position and orientation are explicitly represented, the meaning of "left of" is defined to be from the robot's perspective, so whether one object is viewed as "left of" another depends on the robot's current location on the map.

9 Conclusions

Our goal has been to show the ease with which a dual coding representation system can be employed to parse a scene and reason about spatial relationships. Geometric operations, e.g., finding parallel lines and determining relative line positions, are performed in shape space, where they are most easily described. Iconic operations, e.g., region labeling and manipulation, are performed in sketch space, where region-type spatial relationships are easily computed. The power of this approach comes in its use of meaningful higher-level functions in each space, enabling the programmer to express each processing step with clarity and ease.

Students in CMU's undergraduate Cognitive Robotics course had no trouble mastering these primitives and solving simple visual parsing problems, including parsing the tic-tac-toe board. They were provided with four sample images with varying viewing angles and board configurations to use as test data. Their solutions were then tested on an additional set of images to verify their robustness.

An example of a simpler problem the students were given is: find all the blue ellipses within a closed convex boundary formed from pink tape. This was initially solved by locating the exterior of the boundary with seedfill (which fills a bounded region starting from a seed location), using the top left pixel as the seed location. The result was then inverted to obtain the interior. But one should not assume that the upper left corner of an image cannot be part of the boundary. So a completely correct implementation must first search along the border of the sketch for empty pixels to use as proper seeds. (Multiple seeds makes the solution more robust.) Realizing this, we subsequently added fillInterior and fillExterior operators that automate this process.

The identification of useful primitives is continuing as we gain experience programming in the framework. As another example, in the first version of the tic-tac-toe board parser, to find the top row of board positions we manually deactivated the two endpoints of the top horizontal line segment, making it an infinite line, then rendered it, and then applied the seedfill operator to the rendering, with the seed pixel in the upper left corner of the sketch. This found all pixels lying above the line. We subsequently realized that determining what lies on each side of a line is a common operation, so we added topHalfPlane, bottomhalfPlane, leftHalfPlane, and rightHalfplane operators that take a line segment as input and return the desired result as a sketch of bools.

With the exception of one lab exercise, most of the students' programs in the first offering of the course operated on single camera images, because the map builder component was very new and not yet robust. It has subsequently been reimplemented and made easier to use. As we gain experience programming algorithms to operate on local and world map representations, our collection of useful sketch and shape primitives will likely expand further.

The framework can be extended in a variety of ways. We are currently working on perceiving three-dimensional entities such as spheres, bricks, and pyramids, using a combination of sketch and shape primitives. (The planar world assumption is only partially relaxed: we assume that these objects lie on the ground plane.) We have also extended the line representation to allow us to represent polygons in the ground plane, which the map builder can assemble from multiple images. Polygons are useful for representing the boundaries of the arena, or other closed spaces. With the addition of more abstract concepts such as object-centered frames of reference, gestalt perception operators such as boundary tracing and boundary completion, and mechanisms for shifting the focus of attention, we can see the beginnings of a "cognitive programming language" for robot vision.

Acknowledgments This work was supported by a National Science Foundation grant 0540521, an REU supplement to NSF award IIS-9978403 to support NSH, an NSF graduate fellowship to support JJW, and a grant from the Sony Corporation.

References

Agre, P. and Chapman, D. 1987. PENGI: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pp. 268–272.

Bresenham, J.E. 1965. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30.

Bruce, J., Balch, T., and Veloso, M. 2000. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 00)*, vol. 3, pp. 2061–2066.

Forbus, K.D., Mahoney, J.V., and Dill, K. 2001. How qualitative spatial reasoning can improve strategy game AIs. In *15th International Workshop on Qualitative Reasoning*.

Glassner, A.S. 1990. *Graphics Gems*. Academic Press.

Halelamien, N.S. 2004. Visual routines for spatial cognition on a mobile robot. Senior honors thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. Available at <http://www.Tekkotsu.org/media/thesis_neilh.pdf>.

Landre, J. 2004 Programming with Intel IPP (Integrated Performance Primitives) and Intel OpenCV (Open Computer Vision) Under GNU Linux. Accessed May 31 at: http://sourceforge.net/projects/opencvlibrary.

Paivio, A. 1986. *Mental Representations: A Dual-Coding Approach*. New York: Oxford University Press

Prokop, R.J. and Reeves, A.P. 1992. A survey of moment-based techniques for unoccluded object representation and recognition. In *CVGIP: Graphical Models and Image Processing*, vol. 54, pp. 438–460.

Rao, R.P.N. and Ballard, D.H. 1995. An active vision architecture based on iconic representations. In *Artificial Intelligence*, 78(1–2):461–505.

Roelfsema, P.R., Lamme, V.A.F., and Spekreijse, H. 2000. The implementation of visual routines. *Vision Research*, 40: 1385–1411.

Sony Corp. 2004. *OPEN-R SDK: Model Information for ERS-7*.

Tira-Thompson, E.J. 2004. Tekkotsu: A Rapid Development Framework for Robotics. Masters thesis, Robotics Institute, Carnegie Mellon University. Available at <http://www.cs.cmu.edu/~tekkotsu/media/thesis_ejt.pdf>.

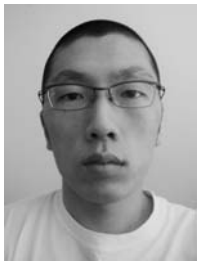
Ullman, S. 1984. Visual routines. *Cognition*, 18:97–159.

Neil S. Halelamien earned a B.S. in Computer Science and a B.S. in Cognitive Science at Carnegie Mellon University in 2004, and is currently pursuing his Ph.D. in the Computation & Neural Systems program at the California Institute of Technology. His research interests are in studying vision from both a computational and biological perspective. He is currently using transcranial magnetic stimulation to study visual representations and information processing in visual cortex.

Ethan J. Tira-Thompson is a graduate student in the Robotics Institute at Carnegie Mellon University. He earned a B.S. in Computer Science and a B.S. in Human-Computer Interaction in 2002, and an M.S. in Robotics in 2004, at Carnegie Mellon. He is interested in a wide variety of computer science topics, including machine learning, computer vision, software architecture, and interface design. Ethan's research has revolved around the creation of the Tekkotsu framework to enable the rapid development of robotics software and its use in education. He intends to specialize in mobile manipulation and motion planning for the completion of his degree.

David S. Touretzky is a Research Professor in the Computer Science Department and the Center for the Neural Basis of Cognition at Carnegie Mellon University. He earned his B.A. in Computer Science from Rutgers University in 1978, and his M.S. (1979) and Ph.D. (1984) in Computer Science from Carnegie Mellon. Dr. Touretzky's research interests are in computational neuroscience, particularly representations of space in the rodent hippocampus and related structures, and high level primitives for robot programming. He is presently developing an undergraduate curriculum in cognitive robotics based on the Tekkotsu software framework described in this article.

Jordan J. Wales is completing a Master of Studies in Theology at the University of Notre Dame. He earned a B.S. in Engineering (Swarthmore College, 2001), an M.Sc. in Cognitive Science (Edinburgh, UK, 2002), and a Postgraduate Diploma in Theology (Oxford, UK, 2003). After a year as a graduate research assistant in Computer Science at Carnegie Mellon, he entered the master's program in Theology at Notre Dame and is now applying to doctoral programs. His research focus in early and medieval Christianity is accompanied by an interest in medieval and modern philosophies of mind and their connections with modern cognitive science.



Kei Usui is a masters student in the Robotics Institute at Carnegie Mellon University. He earned his B.S. in Physics from Carnegie Mellon University in 2005. His research interests are reinforcement learning, legged locomotion, and cognitive science. He is presently working on algorithms for humanoid robots to maintain balance against unexpected external forces.