Blocks World Vision for the AIBO Robot

Matthew Carson
Advisor: Professor David S. Touretzky
May 5, 2006

Abstract

In existing robotic systems, robust object recognition is still a largely unsolved problem. On the AIBO robot, color segmentation is robust, so most existing object recognition is based on monochromatic objects whose appearance does not change with orientation. This research seeks to expand upon existing object recognition tools by developing ways to detect objects with more complex features, multiple colors, and appearances that change based on orientation.

The research focuses on developing algorithms to identify brick-shaped objects in the image seen by the robot. The final approach relies on knowledge of the colors of the sides of the brick. Adjacent blobs of color are selected as candidates for bricks, and specific corner information is extracted by fitting lines to the regions near adjacent faces, and by fitting quadrilaterals to individual faces. Other approaches for finding the corners of an individual face, and a different approach to detecting a brick using just the relationships between lines in the image are explored. The applicability of these algorithms toward detecting pyramid objects is also explored.

Contents

Introduction

This research is being applied within the Tekkotsu development framework for the AIBO Robot. Tekkotsu is being developed at Carnegie Mellon University to allow simpler development of complex robotic behaviors by providing higher level primitives that encapsulate routine operations like low-level motion control and visual processing. The ability to detect a brick or pyramid object is intended to be one such high-level primitive.

The fundamental paradigm of Tekkotsu's vision system uses dual methods of coding what the robot sees. Operations can be applied to iconic (pixel-based) "sketches", like the image the robot gets initially from the camera, and operations can also yield symbolic "shapes", like lines and ellipses. Shapes can be extracted from sketches, and sketches can be rendered from shapes, so both representations can be used interchangeably as needed when developing behaviors. Objects that are recognized appear in three different "spaces". All initial recognition is done within the image the robot sees, called the "camera space". In this space, the location of objects is just their position within the image visible to the robot, in pixels. Using the kinematics of the robot, Tekkotsu can project objects from camera space into "local space", where their position is known relative to the robot, in units of millimeters. In the local space, the origin is fixed at the location of the robot. As the robot moves around, the position of objects is integrated from the ego-centric local space into the allo-centric "world space", where all objects (including the robot) are located relative to a fixed origin. This research only utilizes the projection from camera space to local space.

The problem of detecting a brick or pyramid object was chosen as a natural extension to existing object recognition capabilities on the AIBO. With its low-resolution images (208x160) and vision based on color segmentation, most existing object recognition for the AIBO is done on uniform mono-chromatic objects, like balls, or fixed features in the environment. Bricks and pyramids provide interesting challenges to recognition, including differing views of the object depending on orientation, and combining multiple colors into a single object.

Bricks and pyramids were also chosen because they provide more interesting potential behavior for higher-level robotic behaviors. Their orientation can be manipulated in addition to their position. In addition, an algorithm that can detect these objects can be easily extended to detect other polyhedral objects.

Detecting a Brick

The objective of the brick detection algorithm is to accurately find the locations of all seven visible corners of the brick. Once the positions of the corners are found in the camera space, they can be projected into the world space, and desired properties like the location, size, and orientation of the brick can be computed.

The final algorithm for detecting a brick begins with knowledge of the coloring of different faces of a brick. Candidates for bricks are found from sets of two or three adjacent faces of different colors. Once a set of viable faces is found (Figure 1b), two different calculations are performed to compute possible corner locations.

Figure 1: a) The image captured by the robot; b) Three candidate faces that are adjacent.

Computing Interior Edges

The first calculation computes the line between a pair of adjacent faces. It can be quickly and robustly computed by finding the region close to both faces (Figure 2a), and then using built-in Tekkotsu functionality to fit a single line to the region (Figure 2b). The region close to both faces is defined as just the set of points that are within 5 pixels of a point in each face region. The endpoints of the resulting line are candidates for the corners of the brick that correspond to the intersection of the two faces. A very good estimate of the front-most corner can be obtained by finding the intersection of the three lines found in this way (Figure 2c). However, this method can only produce estimates for visible corners where two faces meet, or four of the seven visible corners of a brick.

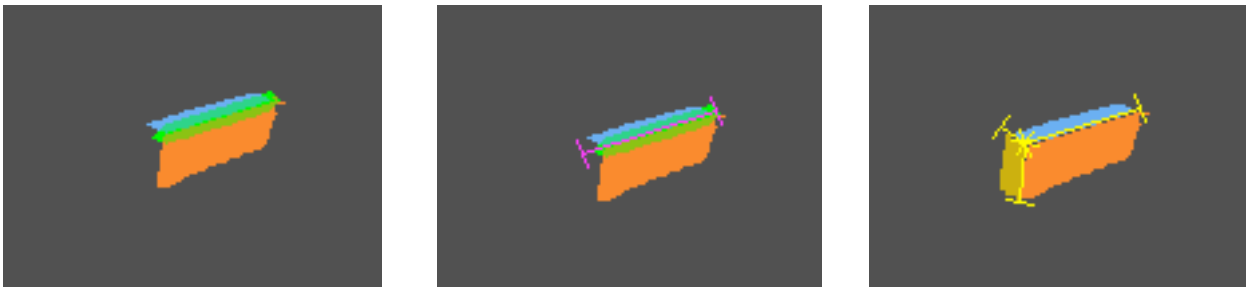

Figure 2: a) The region close to both faces; b) Fitting a line to that region; c) Lines for all pairs of faces.

Quadrilateral Fitting

The second calculation finds the corners of each rectangular blob of pixels that makes up a face of the brick by fitting a quadrilateral around the region. A candidate bounding quadrilateral is computed (Figure 3a), and then hill-climbing is applied to contract it around the face. Once the bounding quadrilateral reaches a local optimum, simulated annealing is performed to improve the fit.

The initial candidate region is found by taking the orientation of the longer of the adjacent edge lines found in the first calculation and extending that line and translating it away from the face and to the other side to make four points. Once initial candidates for a bounding quadrilateral are found, possible updated candidates are found by moving one corner a unit length toward or away from either adjacent corner. Only a single corner is updated at any time.

Quadrilaterals are scored by taking the area of the quadrilateral and subtracting the number of border pixels of the region that lie under one of its lines, times a scalar. Minimizing the score is desirable. Any quadrilateral that contains less than 80% of the points in the region is not considered. Until a local minima is first reached, the best candidate is chosen deterministically at every iteration.

Once the quadrilateral settles into a local minimum, moves are chosen probabilistically, with a weighting toward better scoring moves that slowly increases over time. The rate at which the weight increases is affected by the proportion of edge points that lie under the quadrilateral, so convergence happens faster in easier cases. Convergence is simply defined as when the weight reaches an arbitrary threshold, or in bad cases the process is capped at a number of iterations.

Figure 3: a) Candidate points found for a quadrilateral around this face; b) The converged quadrilateral.

Combining Candidates

Once all candidate corners are found by applying each calculation to each face or pair of faces, the final set of eight points that constitute a brick is computed by simply averaging all the candidates for each corner together. Corners with no candidate points are extrapolated geometrically from adjacent points.

In order to group the candidate corners into bins corresponding to the actual corner they represent, the faces and corners must both be sorted. Faces are initially sorted into “left”, “right”, and “top” classifications, based on their location relative to the other faces. Once the position of a face is known, one corner is identified based on being right-most, left-most, or bottom-most. The quadrilateral fitting computation is constrained to return corners in counter-clockwise order around the face, so after a single corner is sorted into its bin, the other corners can also be sorted.

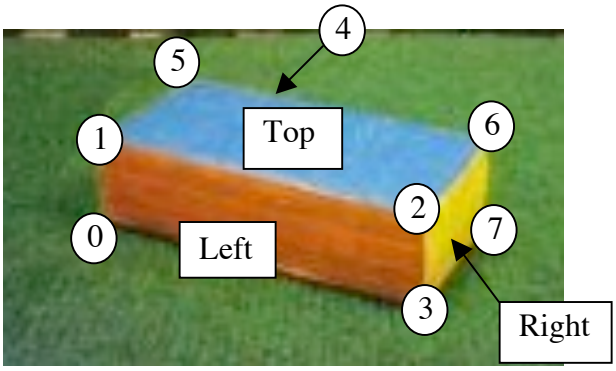

Figure 4: Face and Corner classification

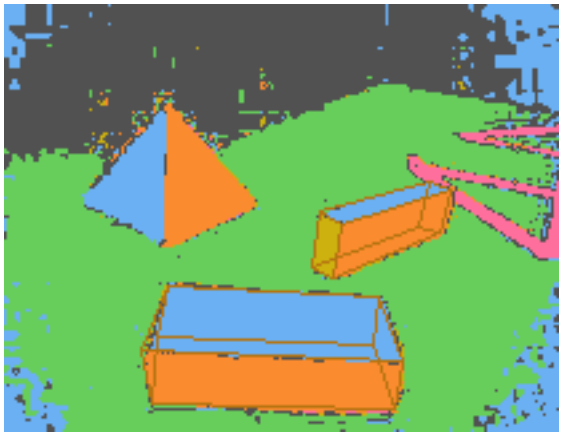

Figure 5: Extracted bricks.

Special Cases

Two Visible Faces

One special case, illustrated in Figure 5, constitutes a brick aligned parallel to the camera, so only two faces are visible. To handle this, the initial candidate selection accepts pairs of adjacent faces in addition to sets of three, and in these cases only a single inter-face line is found. Quadrilaterals are fit to the two visible faces normally, and the two visible faces are arbitrarily defined to be “top” and “left” so corner sorting remains simple. The two occluded corners are estimated geometrically from the adjacent visible corners.

Edge of the Image

If a brick is on the edge of the image, several corners will be uncertain, depending on how much of the brick is visible. If only one or two corners of the brick are outside of the image, it is still theoretically possible to recover the whole brick. The polygon fitting algorithm is not constrained to have all the endpoints lie within the image, so in the case where only a single corner is missing from a face, the quadrilateral fitter can potentially recover all corners correctly. However, in the more common edge cases, half of the brick or more is outside of the image, so it is not possible to accurately recover all corners of the brick. In these cases, the algorithms will recover corners that lie along the edge of the image. These corners will be retained to create a complete brick, but are marked as “invalid” so we know any computation using these parameters will be inaccurate.

Projecting a Brick

The extracted brick consists of eight corner points in the camera space. In order to find any useful information about the brick, these points need to be projected into the body-centric local space. Tekkotsu provides routines to project a point on the ground from camera space to local space. Using these routines on the bottom points of the brick provides desired information about the position, orientation, and two of three size dimensions of the brick (Figure 6b). The last desired dimension, the height, can be found by projecting the upper corners of the brick to world space (exploiting the assumption that they are on the ground), and then using the location of the camera and similar triangles to find the height of the top corner. In Figure 7 below, $\frac{dy}{dx} = \frac{cam_y}{cam_x}$. dy is the desired height, and dx, cam_x, and cam_y are all known.

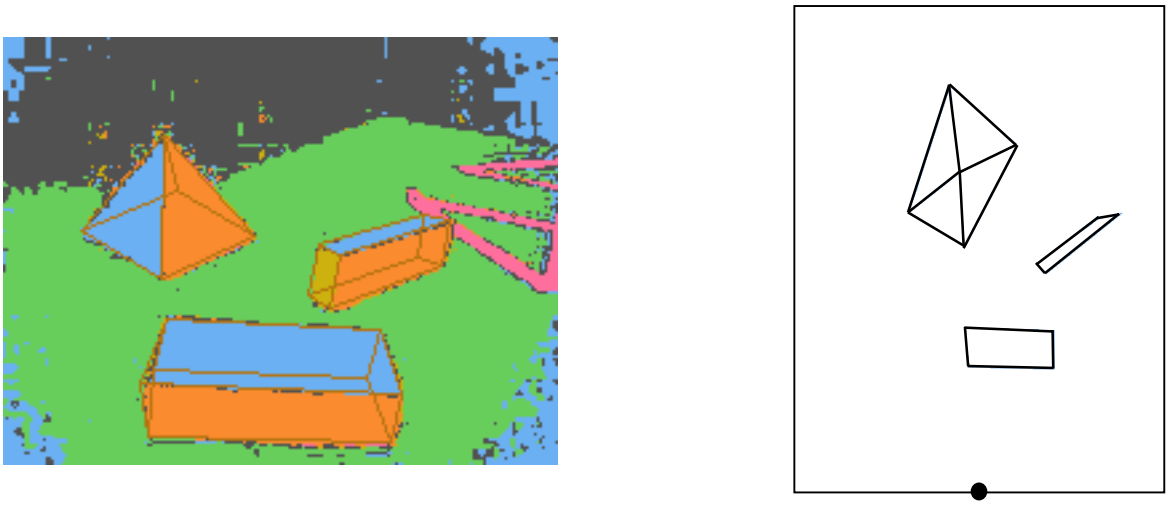

Figure 6: a) The original image seen by the robot; b) "Bird's eye view" of shapes projected to the local space.

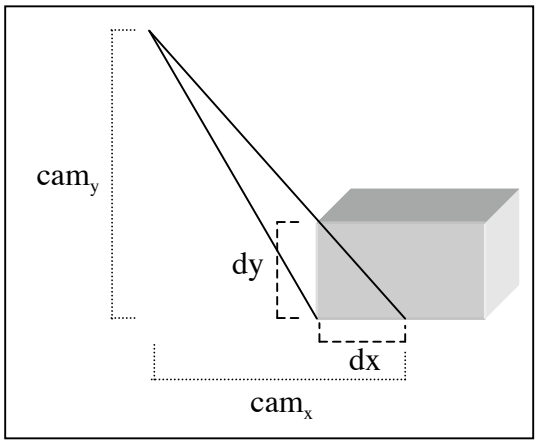

Figure 7 : Finding the height of one point above another

Detecting a Pyramid

Detecting a pyramid proceeds in the same manner as detecting a brick, with the simplifications of only having two visible faces instead of three, and the quadrilateral fitting computation fits a polygon with only three points to each face. Finding the candidates for polygon fitting is also simpler, as two points are just the end points of the boundary line, translated away from the face, and the third is the most distant point on the face from the boundary line, also translated away (Figure 8b).

To generate the final pyramid, the two faces are categorized as either left or right, and candidate corners are again sorted into bins and averaged, with the simplification that there are only 5 corners (4 visible).

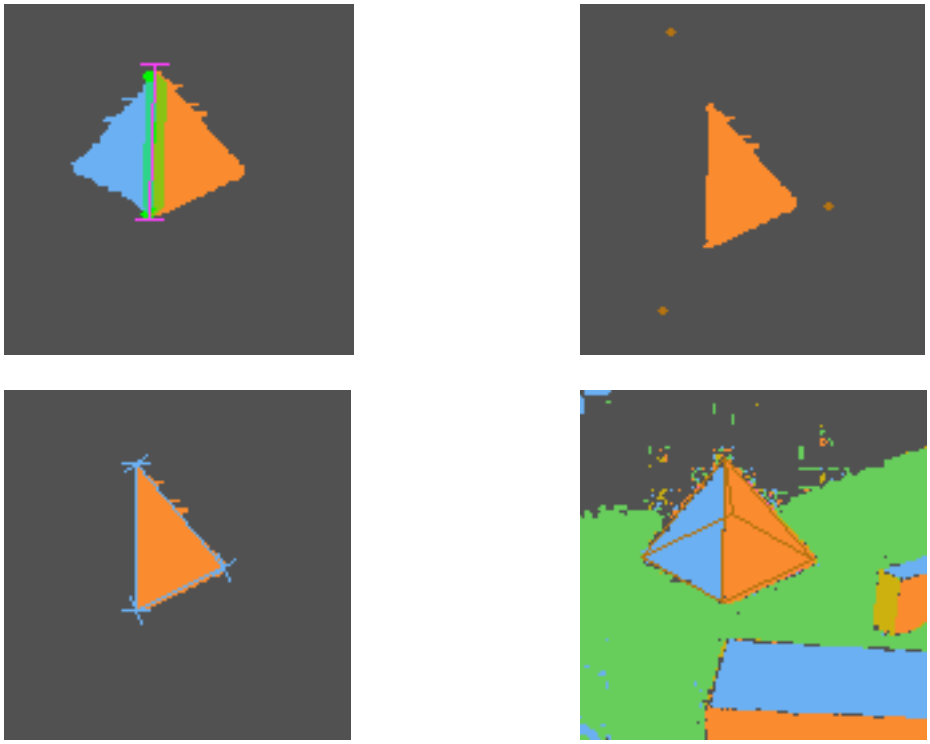

Figure 8: Stages of brick detection applied to a pyramid.
a) Finding the boundary region and fitting a line to it; b) Finding candidates for fitting a triangle;
c) The bounding triangle after convergence; d) The extracted pyramid.

Other Algorithms

In the process of developing this algorithm, several other approaches were tested, with varying results. Two different approaches to finding the corners of a blob were implemented, both involving the maximum radial distance from the center of the blob (Figure 9). These approaches both showed promise, but were not robust in all cases.

Figure 9: Histogram of distances from the center as the circle is traversed.

The first approach starts from the assumption that the most distant point from the center of a region is guaranteed to be a corner. Given one corner point, a second corner should be the most distant point in the region opposite the centroid from the first corner. With these two points, a diagonal across the blob can be drawn (Figure 10a). With the blob bisected, the last two corners can be found by finding the most distant point from the diagonal line on each side.

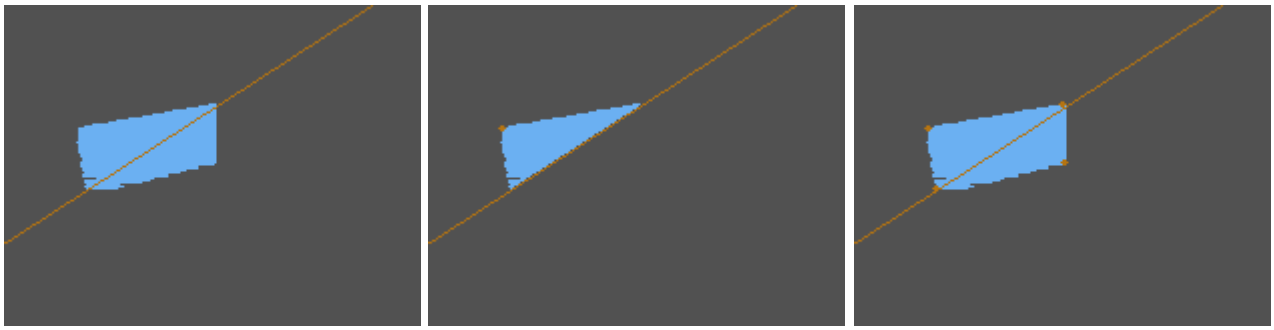

Figure 10: a) Diagonal across the blob; b) Finding a third point; c) All points computed

This approach fails in cases where the computed diagonal finds two adjacent corners rather than opposite corners, like in the case of the top of a brick facing directly at the camera, which tends to be the shape of a very long trapezoid.

The second approach using the radial distance finds the second derivative of that distance, and then searches for negative peaks (Figure 11). This works well for large shapes, and for shapes with more or less than four corners. However, it fails for small shapes because of the low resolution, and is very susceptible to finding extra points in the presence of noise.

Figure 11: a) Second derivative of the distance histogram; b) Extracted corners

Future Work

The focus of this work has been primarily on extracting accurate points for the visible objects. Less attention has been paid to discrimination between bricks and pyramids, and avoiding false positives. An arbitrary pair of different colored blobs could easily be extracted as both a pyramid and a brick, and faces from different nearby objects could be classified as a single shape. A simple sanity check on the final shape might be enough to handle most cases, however, being able to compute the number of sides a blob has without prior knowledge would be the best solution for discrimination.

Also. integrating different views of the same object is also an important, non-trivial objective. When applied in a robot behavior, brick extraction could be run many times on the same brick from different views as the robot moves around. The visible “left” and “right” faces won’t be consistent between views, so a viewpoint-independent scheme for describing brick parameters would make it easier to integrate multiple views. Careful handling of bricks that are occluded or at the edge of the image in one view is also important, so that the brick isn’t distorted when it is fully observed.

Finally, the quadrilateral fitting algorithm currently runs too slowly for real-time operation. During development, correctness, not speed was the objective, and several code optimizations were apparent to improve it but were not implemented. Also, the distance-based algorithms that were explored performed much faster than quadrilateral fitting. Using these in simple cases and only performing quadrilateral fitting when the others fail would produce a faster hybrid algorithm that also gives good results.