Cubelet Detection and Localization by Template Matching

Charles Chen
Electrical and Computer Engineering
University of Colorado at Boulder
Boulder, CO 80302

Charles Dietrich
Computer Science
University of Colorado at Boulder
Boulder, CO 80302

Chris Miller
Computer Science
University of Colorado at Boulder
Boulder, CO 80302

***Abstract*—We describe a method for detecting and localizing Modular Robotics Cubelets in a scene, for use as a subsystem in a system for autononmous assembly of Cubelets constructions using a robotic arm and grasper. We use a camera such mounted over the work surface. Our method returns two-dimensional position and angle of rotation as well as the type of Cubelet. We obtain position and rotation using template matching on images preprocessed with a novel line detection algorithm. We obtain the type of Cubelet by using color information. We also describe a method to obtain depth information from a camera with an RGB-D sensor such as the Xbox Kinect.**

I. Introduction

The purpose of this research is to produce a vision subsystem to the system necessary to accomplish the overall goal of 'robots building robots'. The 'robots building robots' project involves the assembly of robots made up of Modular Robotics Cubelets. Cubelets are a robotic toy consisting of a set of small cubes. Each cube, or Cubelet, is approximately 50mm on a side. Cubelets attach to one another by way of keyed magnetic faces. Each Cubelet contains a small computer. A Cubelet may be an actuator, a sensor, a thinking block or a battery block. Actuators have an actuation face and respond to incoming signals. Sensors have a sensor face and sense the environment and send out a signal. Thinking blocks change the signal received from neighbors in a predetermined way, e.g. a Inverse block inverts the signal received. The battery block provides power.

For the purposes of the overall goal of 'robots building robots', we are concerned with autonmously assembling the Cubelets into a predetermined structure that forms a robot. We assume that the Cubelets start scattered over the work surface, a flat, featureless surface. We intend to build the construction using the Clam arm, a small robotic arm intended for research, with a suitable grasper. We also intend to use ROS (Robot Operating System) to bring together the entire system.

We assume that a vision subsystem is a necessary subsystem of the overall system. The vision subsystem will be used to direct the arm and grasper. The vision subsystem reifies visual and aligned depth camera data into an ontology that represents the arrangement of Cubelets on the work surface. This ontology can then be consumed by other parts of the system.

Our engineering solution provides a partial technology for providing a useful ontology. The ontology should contain position and rotation for each Cubelet as well as the type of Cubelet. Our technology aims to provide two dimensional position and rotation information and partial information on the type of Cubelet at a reasonable frame rate.

We hypothesize that template matching on images preprocessed with a novel line detection algorithm can provide position and orientation data. We hypothesize that color matching on matched areas can provide partial information on the type of Cubelet.

A. Background

The problem of providing the desired ontology is largely one of object detection, a well-known problem in computer vision. Object detection methods include feature detection methods such as SIFT and SURF as well as template matching, segmentation and classification of segments, and flexible template matching. Feature based-methods are generally faster than template matching. We present evidence that SURF does not work well with Cubelets due to their particular appearance.

II. Matters and Materials

Our experimental setup consists of a Xbox Kinect camera mounted 65cm above the work surface. This distance is far enough for the depth sensor to work even if the blocks are stacked, but close enough to make out features on the faces of the Cubelets. The work surface is covered in white paper. The lighting is not controlled but is normal office lighting.

We generated two sets of 20 images each under the same setup and lighting conditions. Each image takes a picture of an arrangement of Cubelets. The Cubelet arrangements are determined by us. The first set was used as a training set to train the vision subsystem. The second set was used as a test set to generate the experimental results.

Our software is written in Python and uses ROS with OpenCV and openni_kinect.

A. General Methods

To manage the information related to the templates, we've created a single file with a variety of template images, scaled to approximately the same size. In addition to this, we have an xml file marking the (x,y) locations of each template cube

within the larger image, as well as the idea template radius for that cube. For the actual templates, we use the region around the circlular portion of the cube face.

B. Edge Enhancement Filter

In order to better match the cubes, specifically the clear cubes, we found it necesary to apply a form of edge detection/enhancement to the images. Without doing this, the features of the clear cubes would not be very distinct, making matching of any kind very difficult. We found the Canny edge detector to be too imprecise; a slight adjustment to the upper and lower edge merge thresholds would change the shape of fine detail edges. Sobel suffered from a different problem in that the edges that it did detect were very weakly enhanced. In order to improve the edge enhancement, we wrote our own filter which is very similar to how Sobel operates. Shown below are examples of sobel and canny run on our test images, as well as the two versions of our edge enhancement filter.

[show images]

The first variant of the filter behaves as follows: for each pixel, the value of at pixel becomes the product of the difference between the pixels above and below and the difference between the pixel to the left and the right.

$$f(x,y) = |i(x+1,y) - i(x-1,y)| \times |i(x,y+1) - i(x,y-1)|$$

This filter enhances edge boundaries very strongly, but still suffers from a bit of noise. The second filter reduces the noise seen, but as a side effect it reduces some of the edge enhancement strength.

$$f(x,y) = \tfrac{1}{c}|i(x+1,y) - i(x,y)| \times |i(x-1,y) - i(x,y)| \times |i(x,y+1) - i(x,y)| \times |i(x,y-1) - i(x,y)|$$

[show example images of Canny, Sobel, and our Filters]

Before performing template matching, we apply these filters to the template image and the target image. Although this removes the color information, it does allow for the template matching process to detect all cubelets indiscrimitately.

C. Rotational Template Matching

In order to handle the possibility of missing a match due to a rotation, we take each template image, rotate it by 15 degrees, and then template match that rotated version to the target image. To accomplish the actual template matching, we are using the template matching function built into opencv, cv.MatchTemplate. The particular match value calculation equation is as follows:

[show the equation from the opencv python doc]

The output of this function is a grayscale image with pixel values equal to difference metric between the local window and the template. To find the matches in the image, we simply look for the minima within that image.

Once we have a list of prospective cube locations, we find the average color around the center of each match, and compare that to the color averages of each template image, selecting the smallest difference as the matching color.

D. Rotation Estimation

In order to determine the rotation of each cubes, we find try to utilize the relatively distinct edges of the cube. We once again start by applying our edge enhancement filter to the test image. We then blur, dilate, and erode the resulting image in order to merge the edges of each cube into a single connect component. This is to suppress the internal features of the cube, so that when we apply Canny edge detection to the image, we only get the outlines of the cubes. Once Canny has been applied, we use OpenCV's hough line detector find the line segments present. The detector returns lists of line segment endpoints, from which we calculate the rotation of each line segment, modulo $\frac{\pi}{2}$. For each detected cube, we located all the line segments within some radius of the cube center, then average the calculated angles for those line segments, which is assigned as the rotation angle for the cube.

E. Depth Information

III. Results

A. Match Improvement due to Rotational Matching

B. Match Improvement due to Edge Enhancement

C. Accuracy of Color Determination

D. Accuracy of Rotation Estimation

IV. Discussion

The color determination was not quite as accurate as we would have hoped. This mainly occured with the clear and black cubes, which in dim lighting have somewhat similar colors. A way to address this would be to instead compare color histograms of the a test image and a cube template. This would differentiate the black and the clear cubes well, as the black cubes would have a spike in the lower end of the histogram, and the clear cubes would have a more spread out distribution.

A. Further Work

The major limitation of our approach is that it requires a fixed overhead view of the scene. This constraint allowed us to use template matching to find cubelets easily within the scene, limits its usefulness on a robot that can look around in its environment. Our methods do not work very well if the cubes were to be seen from non-normal angle. This change in perspective would require us to have a more thorough method of pose estimation.

In order to address these concerns, we would need to find an object recognition method that is robust against scale, rotation, brightness differences, and affine transformations. As we have already found that SURF is insufficient, we would investigate matching methods using other feature detectors, such as MSER (Maximally-Stable Extremal Region), which is very robust

against affine transformations.

Given a set of matching points between a training image and the target image, we could find the transform relating those sets of points, and calculate the pose of those points in the camera space, provided that the camera parameters were known. Many different methods for doing this exist, including POSIT, which works on four non-coplanar points.

V. Conclusion

A. Subsection Heading Here

Subsection text here.

1) Subsubsection Heading Here: Subsubsection text here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.