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*—In this paper we seek to describe a means to detect and localize structures created from Modular Robotics Cubelets using a combination of template matching and depth data from an RGB-D sensor, such as the Xbox Kinect.**

I. Introduction

The purpose of this research is to add another step in the framework necessary to accomplish the overall goal of 'robots building robots'. There are many ways to accomplish this, including the use of feature descriptors such as SURF to detect and match objects in an environment against a reference. However, these methods tend to be much less robust when dealing with very symmetrical and regular objects, as opposed to the highly textured and uniquely patterned objects that the algorithm is normally tested upon.

We hypothesize that template matching can provide a good means of cubelet detection, and with the addition of depth data, cubelet localization. The approach that we have taken involves starting with an example template, rotating it by increments, and applying template matching across a target image for each rotated template, then clustering points of high match likelihood. The centers of these clusters correspond to the centers of detected matches. We then use this information along with the captured depth data to determine the positions and pose of the cubelets.

If successful, this would provide a simple method for detecting and localizing the cubelets. The risk, however, is that the method will be very slow, as template matching methods usually are compared to feature based matching.

II. Matters and Materials

To accomplish our goals, we have leveraged the capabilities of the OpenCV computer vision library, specifically the python 2.1 bindings. In order to make the scope of this problem more manageable, we are also restricting the setup of the camera and depth sensor to being fixed in an overhead view above the cubelet work area.

A. 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. This is due to the difficulty in properly detecting the clear cubes, which had previously matched very closely to the background table. Using just the center circular portion of the cube as a template greatly increased the accuracy of the matchings for those cubes.

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.

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

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.

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.

III. Results

A. *Match Improvement due to Rotational Matching*

B. *Match Improvement due to Edge Enhancement*

C. *Accuracy of Rotation Estimation*

IV. Discussion

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

The conclusion goes here.

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.