

# OBJECT DETECTION: SUNGOD STATUE

Amin Rahimi

UCSD Electrical and Computer Engineering  
arahimi@ucsd.edu

## ABSTRACT

This paper discusses object detection in an image using previous knowledge of the object's color signature. Image processing techniques are used to simplify the image matrix so that simple pattern recognition algorithms may be utilized.

## INTRODUCTION

We are given the task of writing a program that will detect the presence of the Sun God in a given image. If the Sun God is found, the program must draw a bounding box around it. Our program of choice is Matlab 7.0.

There are numerous techniques that can be used in object detection. One method is the correlation filter of a model template matched over the entire image. This method, however, is impractical (1). We do not have previous knowledge of the Sun God's size with respect to the dimensions of the given image and, thus, the method would require that the template be resized and run across the image numerous times. This method would also fail to account for possible changes in image proportions and camera perspective due to changes in the position of the camera with respect to the statue. A method involving edge-detection and shape recognition would also pose problems. The contours of the statue's body are fairly abstract making it difficult to construct an algorithm for detection. These contours will also change shape with respect to changes in camera position thus adding to the complexity of such an algorithm.

We will therefore focus on identifying the statue by its color patterns. The advantage of using color for identification lies in the fact that color does not significantly change with camera position and the fact that there will be few objects in the background whose colors will interfere with detection of the Sun God's color scheme.

## COLOR SLICING

When a given color image is read into Matlab, it is processed in RGB space. This means that there are approximately 16 million colors available per pixel. If the program were to run each algorithm on this image alone, a large amount of processing power and programming time would be required. Thus, our first goal is to simplify the image's color scheme.

Because we are interested in only those colors that are present on the Sun God (red, orange, green, blue, and yellow), these colors will be extracted from the image while others are suppressed.

In RGB space, every color has three parameters (red, green, and blue intensities), each of which change with changes in lighting conditions. For example, to accurately identify the RGB range for the yellow sun on the Sun God's midriff would require knowledge of all the possible lighting conditions, including shadows, and their corresponding effects on the RGB values of the yellow sun. To simplify our problem, we look to HSI conversion. (2)

A pixel in HSI space has three components. The first component (and the one in which we are interested) is the hue value. Hue is a dimension of color correlated with the dominant wavelength of light (3). Thus, the hue of a color will not vary with increasing or decreasing lighting and with shadows cast upon an object.

To work in HSI space, a new matrix containing the HSI values of each pixel is created using the `rgb2hsv` command in Matlab. A `for` loop is then used to evaluate the hue of each pixel and to create a corresponding value in a third matrix for that pixel. For example, an orange pixel (hue value ranging from .04 to .14) in the original image will result in a gray pixel (with a value of .75) in the new image. This process is performed for each color stated above.

Once this process is complete, the image will be transformed from a matrix with a depth of 3 to a matrix with a depth of 1.

Figure 1 displays the results when this process is performed on the same object with two different lighting conditions.

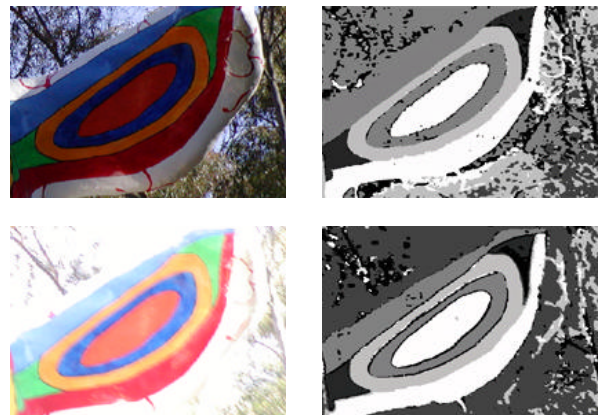


Figure 1. Color to 6-level grayscale conversion.

## PATTERN DETECTION: Finding the statue

Our first goal is to find the Sun God's left wing. We choose this wing because its design and pattern allow us to use a simple algorithm for detection. The inner portion of the wing consists of three concentric ellipses. The inner ellipse is red and is bordered by a blue ellipse which is bordered by an orange ellipse. Because of the nature of this shape, any line originating at a pixel in the inner red ellipse and extending radially outward must intersect the other two ellipses.

To find the wing, a new matrix, *findWing*, is created using **zeros**(m,n) where m and n are the dimensions of the original image. The program first searches the simplified grayscale image for "red" pixels (value '1') and then uses nested **while** and **if** statements to test for the color pattern. When a red pixel is found, the program looks directly to the right for an uninterrupted series of red pixels followed by an uninterrupted series of "blue" pixels (value '.5') followed by a single "orange" pixel (value '.75'). If a black pixel is found, the program disregards it and continues to the next pixel. Any pixel matching this criterion will be incremented by a value of .25 in the corresponding coordinates of the *findWing* matrix. This process is then carried out in the other N<sub>4</sub> directions.

What results is special grayscale version of the original image. This new version contains white pixels (value '1') where the program found the red-blue-orange pattern in all N<sub>4</sub> directions, contains light gray pixels (value '.75') where the pattern was only found in three of the N<sub>4</sub> directions, and so on. What this method allows us to do is to easily change the threshold for what constitutes a pixel as being "inside the wing". That is, we can require that the program only find this pattern in two directions by setting a threshold at .5 and having the program loop through the matrix and delete all values below this threshold. In the case of this program, we will require that this pattern be found in all 4 directions. A **for** loop is therefore run on *findWing* to change all non-'1' values to the value '0'. The coordinates of those pixels that have a value above the threshold in the *findWing* matrix are then recorded to *xycoord*, a 2by-n matrix (where n is the number of white pixels found in *findWing*). Figure 2 shows an image of the Sun God with its corresponding *findWing* image.

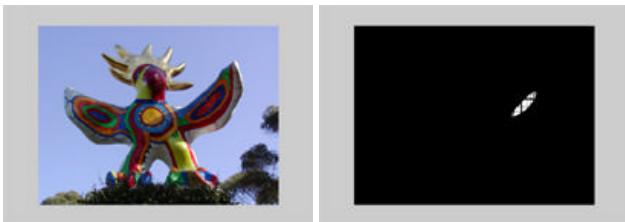


Figure 2. Sun God image with corresponding *findWing* image

We now have a matrix containing the coordinates of many of the pixels thought to be inside the wing. We make the assumption that a large amount of the pixels found using the criteria above are, in fact, in the wing and are not coincidental noise. Thus, the median of the coordinates in *xycoord* is assumed to be the coordinates of a pixel that lies in the red area of the Sun God's left wing.

To find the coordinates of the true middle of the wing, **bwlabel** is utilized. *redWing* is a binary matrix containing the value '1' in the coordinates corresponding to the red pixels of the original image. When **bwlabel** is run on *redWing*, each "red" region is labeled with an integer ranging from 1 to n (where n is the total number of regions). These labels are placed in a new matrix, *L*.

We already know the coordinates of a point inside the red area (from the median of *xycoord*). Therefore, *L*(x,y), where x and y are the coordinates determined above, will give the region number of this part of the wing. From there, *find*(), *min*(), and *max*() are used to determine the extremities of the wing. With these values, the size of the wing and the midpoint can easily be determined.

To verify that the object found is, in fact, the Sun God's wing, another region on the statue needs to be recognized. Using the same algorithm as before and using previous knowledge of the statue's dimensions, a **for** loop is run to search a specific area (proportional to the dimensions of the wing) to the left of the wing. The program searches for blue pixels with a red, blue, green, blue pattern extending to the left. The median of these blue pixels is then presumed to be a point just left of the sun on the statue's midsection. If this point is found, then the Sun God is in the picture.

## DISPLAYING RESULTS

Once the statue has been found, the program must draw a box enclosing it. This is done using simple algebraic functions on the dimensions that have already been found. The boundaries of the box are all determined by scaling the size of the wing and adding the value to the coordinates of the middle of the wing.

If the statue is not found, the program will display a pathetic sad face. The mouth is a semi-circle with radius *height*/4 (where *height* is the height of the image) and is centered at three-quarters of the height of the image. To draw the semi-circle, a **for** loop is used with parameter theta going from 0 to 1570 (this is (pi/2)\*1000). The x and y coordinates of the smile are then determined using polar coordinates. The eyes are simple vertical lines placed in the top half of the image and separated by a distance r.

Figure 3 shows the outputs for different images.



**Figure 3. Program outputs for different images.**

## RESULTS

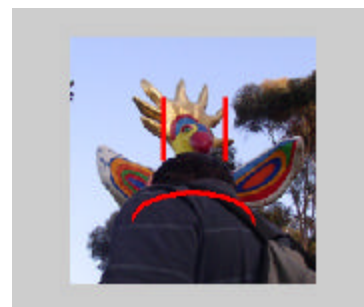
Of 16 images of the Sun God, the program was able to accurately detect the Sun God in 14 of them. Out of 30 images without the Sun God, the program found no false positives.

There are several strengths in using this method. The first, which has already been mentioned, is the program's insensitivity to changes in light. We have seen from figure 2 that the Sun God has been detected despite a large amount of glare from the sun. Another strength is the program's ability to process images of almost any size. There is thorough boundary checking throughout the program to assure that it gives no errors.

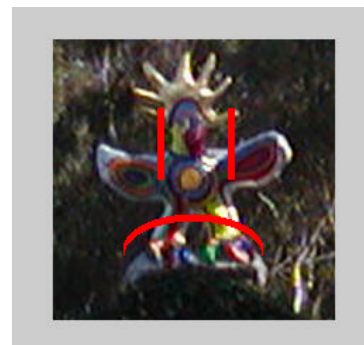
It is also very difficult for an image to create false-positive results. Because of the high demands of the search criteria, the odds of another object being detected as the Sun God are very low.

Unfortunately, these high demands also have negative effects. An obstruction, if in the proper place in the image, can easily keep the program from detecting the statue. Figure 4a shows such a case. Although it is clear to us that the Sun God is in the image, the program will not detect it because the area to the left of the wing is hidden. A simple, yet tedious way to get around this is to simply add more search points. For example, the program could also search for the statue's face and require that only two of the three search points be found to identify the object as the Sun God.

This program also requires that the image be above a certain quality. The negative result in figure 4b is the caused by the poor resolution among the colors of the statue. That is, the colors blend and are thus not distinguishable once the image is converted to a 6 level grayscale image. The solution would require more search points, as before, or the use of sharpening filters before the search processes are applied.



**Figure 4a**



**Figure 4b**

## CONCLUDING REMARKS

We have taken a complex color image, reduced it to its most elemental colors, and performed operations on this simplified image to detect color patterns known to exist on the Sun God. The dimensions of the entire statue were then estimated from the dimensions of smaller features found on the Sun God.

## REFERENCES

1. Buhmann, Joachim, et al. Image recognition: Visual Grouping, recognition, and learning, <http://www.cs.berkeley.edu/~malik/papers/pnas-gafos.pdf>
2. Gonzalez, Rafael and Richard Woods, Digital Image Processing, 2002
3. "Hue" The American Heritage® Dictionary of the English Language, Fourth Edition