Objective: Write programs using object oriented approach; learn basic image processing methods.

1 Problem Definition

Object detection is one of the fundamental tasks in computer vision, the technology to let machines see as we do. Even though we can easily recognize many different objects, object recognition using computers is still far from mature. In this project, we study an easier problem in which objects have distinct colors or brightnesses from the background, e.g., white papers on a dark table. We would like to detect the sizes and locations of these objects. In this simple case, fast and effective algorithms can be implemented. In fact, we can detect or track objects in real time with a common computer.

![Figure 1: Detect objects in images.](image)

(a) Input image  (b) Thresholded image  (c) Objects detected  (d) Detected regions

Fig. 1 illustrates the object detection method. Since the objects in the input image shown in Fig 1(a) are much brighter than the background, a simple thresholding can separate them from the background as shown in Fig. 1(b). The thresholding procedure turns pixels brighter than a threshold into 1s and others into 0s. Foreground pixel clusters (the clusters of 1s) can be further detected; their sizes and centers are shown in Fig. 1(c). In fact, only the “large” clusters are labeled. Small clusters are thrown away because they are likely to be false detections. We use different colors to label the detected regions in Fig. 1(d).

In this assignment, you will write a class `Detector` to do object detection. The input is a color image and a threshold value, the output is the object labels in the image as shown in Fig. 1(c). Some methods, including the thresholding function and the connected region detector, are given to you. You need to use them to construct a class called `Detector`. In a client code segment, `Detector` class can be used as follows:

```java
Detector detect = new Detector("Objects.jpg", 0.5); // 0.5 is a threshold
detect.findObjects(); //detect objects
detect.show(); //show the results
```
Here we assume the input image is called “objects.jpg”. This image can be downloaded from the course website.

2 Object Detection

An image is a 2D array of pixels. For a grayscale image, each image pixel contains a single integer in the range of 0-255. A pixel of value 0 is the darkest and 255 the brightest. For color images, each image pixel contains three integers, which are the red, green and blue components. These integers are also in the range 0-255.

Use the following method to threshold the image and obtain a binary array:

```java
private int[][] color2bw(Picture pic, double threshold) {
    int w = pic.width();
    int h = pic.height();
    int[][] image = new int[h][w];
    for (int i = 0; i < w; i++)
        for (int j = 0; j < h; j++)
            {
                Color c = pic.get(i, j);
                double v = 0.299 * c.getRed() + 0.587 * c.getGreen() +
                           0.114 * c.getBlue();
                if (v > threshold)
                    image[j][i] = 1;
                else
                    image[j][i] = 0;
            }
    return image;
}
```

In the above method, we use the class `Picture` which can be downloaded from the course website.

In the thresholded array, ideally each object is a cluster of 1s. In a cluster, each 1 has at least one neighboring element that is also a 1. We assume that the neighbors of `image[i][j]` include `image[i-1][j]`, `image[i+1][j]`, `image[i][j-1]`, `image[i][j+1]`. Object detection hence becomes finding the connected 1s in the thresholded array. Interestingly, we can in fact compute the cluster mass and center as we label the connected clusters. To find the array elements connected with `image[i][j]` and compute the cluster mass and center, use the following recursive method:

```java
/* This method labels the connected 1s to image[r][c] with label id
 * 
 * Arguments:
 *   image is an array that contains 1s and 0s
 *   id should be a positive value other than 1
 */
private Cluster findCluster(int[][] image, int r, int c, int id) {
    int h = image.length;
    int w = image[0].length;
```
if ( \( r < 0 \) \| \| \( r > h \) \| \| \( c < 0 \) \| \| \( c > w \) )
{
    Cluster t = new Cluster(0, 0, 0);
    return t;
}

double v = image[r][c];
if (v != 1)
{
    Cluster t = new Cluster(0, 0, 0);
    return t;
}

image[r][c] = id;
Cluster c1 = findCluster(image, r-1, c, id);
Cluster c2 = findCluster(image, r+1, c, id);
Cluster c3 = findCluster(image, r, c+1, id);
Cluster c4 = findCluster(image, r, c-1, id);

double mass = 1 + c1.mass + c2.mass + c3.mass + c4.mass;
double row = (r + c1.rc * c1.mass + c2.rc * c2.mass + c3.rc * c3.mass + c4.rc * c4.mass)/mass;
double col = (c + c1.cc * c1.mass + c2.cc * c2.mass + c3.cc * c3.mass + c4.cc * c4.mass)/mass;
return (new Cluster(mass, row, col));
}

where Cluster is defined as

class Cluster {
    double mass, rc, cc;
    public Cluster(double m, double r, double c)
    {
        mass = m; rc = r; cc = c;
    }
}

Pay attention to the way that we use to return a bunch of values through a Cluster object. Read
the code and convince yourself that it indeed labels the cluster and computes its mass and center.
Due to the limitation of the Java stack, the size of a cluster cannot be too big (usually less than
40000 pixels). We can ignore this problem in this project.

Based on the above method, we can find all the clusters in the image: we scan each element in
image row by row. Once we find a 1 element, we find all the other 1s that are connected to this
element, and label them with an id that is greater than 1. By repeating this procedure, we can find
all the clusters including their mass values and centers. You can code the method findObjects() in Class Detector based on this procedure.
3 Display Results

To implement the `show()` method in the `Detector` class, we use `Draw` class which is a modified version of the one from the textbook. You should put `Draw.java` and `DrawListener.java` in the same directory with your class. These two files can be downloaded from the course website. To display an image using `Draw`:

```java
Draw draw = new Draw(w, h); // w and h are the width and height of the image
draw.setXscale(1, w);
draw.setYscale(1, h);
draw.picture(imageFileName);
```

Then, we draw points and circles that represent the center and mass of a cluster region:

```java
draw.setPenRadius(0.02);
draw.setPenColor(new Color(255, 0, 0));
draw.point(region.cc, h - region.rc);
draw.setPenColor(new Color(255, 255, 0));
double r = Math.sqrt(region.mass / Math.PI);
draw.circle(region.cc, h - region.rc, r);
```

where `region` has type `Cluster` as defined in the previous section. Notice that we use `h - region.r` as the y coordinate. You can use a loop to repeat this procedure for labeling all the clusters.

4 What to Submit

You should submit the Java programs. Pay attention to good Java programming style. Upload your Java files to webCT before the submission deadline. There will be 3 days grace period. But late submission would involve 10% point deduction for each day. Submissions later than 3 days are not accepted.