CSE 364 Project - Colored Object Detection and Tracking
George Donnelly

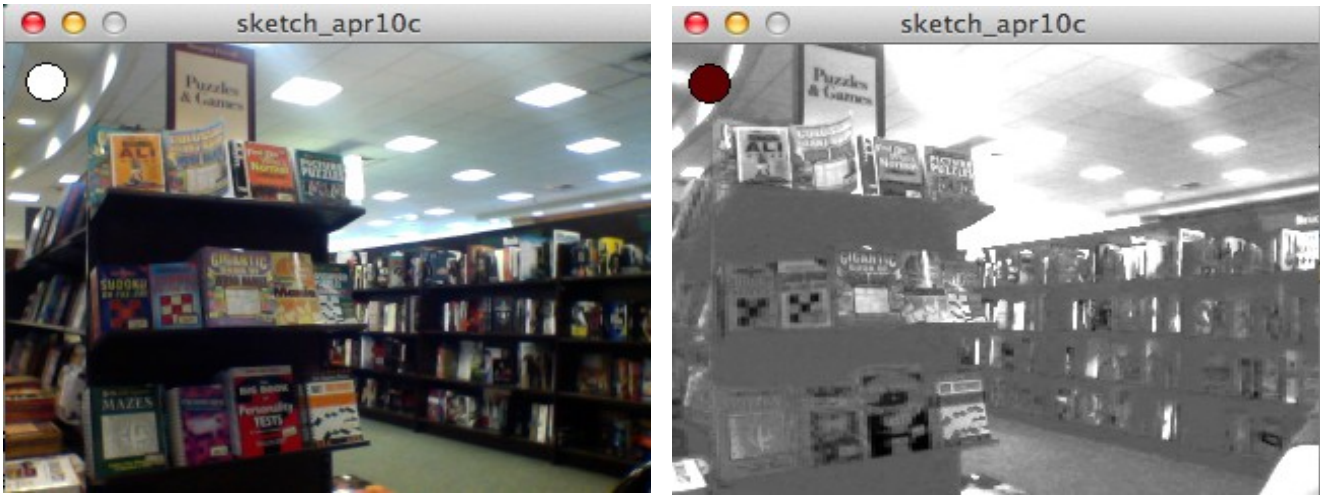
1. Introduction

Color recognition is a fascinating topic and can be applied to many fields that use computer vision such as robotics, camera security, machine learning and so much more. For an example in robotics it is important to have a great visual system, and this visual system will start with color recognition. A real world example of color and object recognition can be found in soccer playing robots that compete in the RoboCup. The robots in the RoboCup (http://www.youtube.com/watch?v=XLKKbz2mNyo) find the color ball on the field of play and play soccer. This task is accomplished by the robots fast color and object recognition that detects the color and shape of an object. In this project I will like to first understand the basics of color recognition for computer vision and if time allows apply color tracking using Processing.

2. Phase 1 – GOAL MET!

The goal of phase 1 of the project was to successfully get color recognition to work so that a human can notice how the program is working as well as an easy modulation to step into phase 2 of the project in color motion and tracking.

Figure 1.



In Figure 1. (left) the user will run the application and click on a color in the video screen using their webcam. Once the color is chosen the application will take the difference of all the pixels in the colored image and the color of the pixel chosen and repaint it onto the user webcam application (right) The difference will be in black (best closest value) and white (least closest value). The only flaw of this part of the application will be the webcam quality can the input resolution.

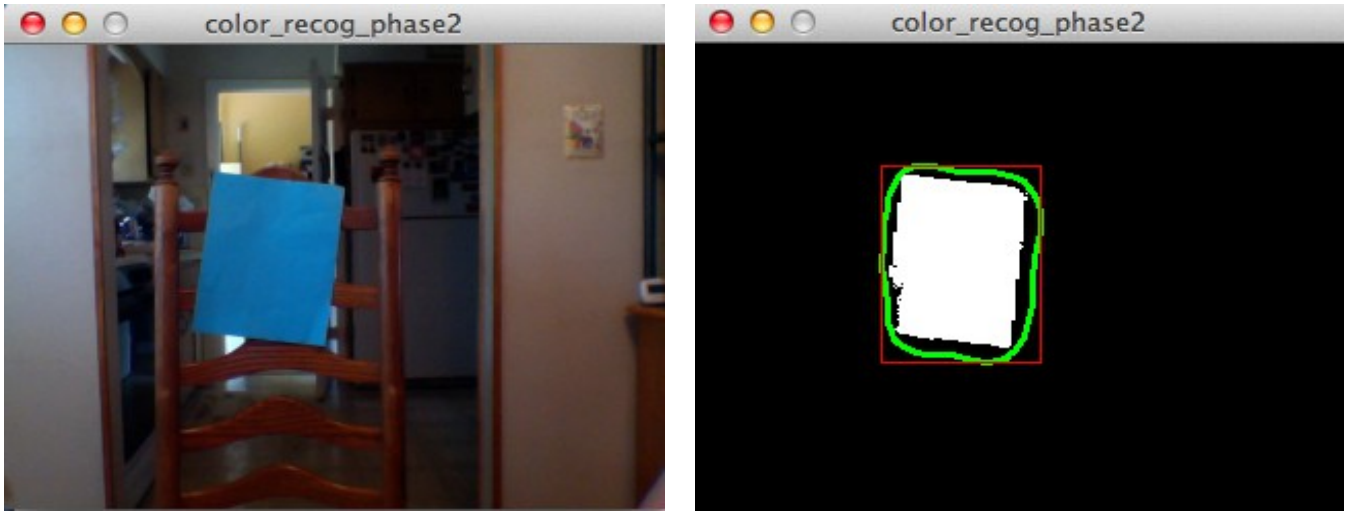
The input resolution can make the performance of the application either slow or fast so it is important to have the optimal resolution to perform pixel labeling. I neat feature about Processing is that it has the option to create a linear data-structure on a video capture so all calculations are done in O(n) time (http://processing.org/reference/pixels.html). Pixel labeling speed can have an excellent performance

by skipping pixels. Therefore instead of analyzing each pixel one can analyze every 3rd pixel or have the pixel interval of 3. The pro's of this method is the reduction of noise in the video! However I did not use a pixel interval but use the resolution of 320X240 and do not see any performance issues.

3. Phase 2 – GOAL MET!

The goal of Phase 2 was to accomplish successful color object detection, tracking, and display it on the users screen. There is no object recognition however it can be implemented after successful color/blob tracking. The user screen must only track a group/blob of colors that the user inputed into the program. After the success of color blob tracking the applications for object recognition and additional tracking are endless.

Figure 2.



The first goal of phase two was to display the color on screen for the object as white and the rest black. This was done like phase 1 of the project but changing all pixels values at a certain threshold to be black or white.

Second, achieving fast performance on blob detection the image does not need to be in the best resolution and could have some noise. I took the current frame of edited video and performed blurring. Blurring an image allows faster blob detections because the algorithm is looking at the brightness value of a specific area.

Third, edge detection and corning can not gain additional information about space in order to detect objects; in order to perform this operation blob detection was needed. Processing has a third party library for Blob Detection (http://www.v3ga.net/processing/BlobDetection/). The blob detection calculations vary depending on the developer and there are difference formulas. In general blob detection in general is calculating the areas whose brightness is above or below a particular value. After brightness level has been calculated the blob detection algorithm uses edges and corning algorithms to create the blob and uses edge and corning values to visually display the blobs edges and blob frames.

CSE 364 Project - Colored Object Detection and Tracking
George Donnelly

Processing Code:

```
// george donnelly
// phase 2 - color recognition and tracking

import processing.video.*;
import blobDetection.*;

Capture capture;
color target_color;
boolean start_tracking = false;

// blob detections
BlobDetection theBlobDetection;
PImage img;
// image to save for blob detection
boolean newFrame=false;


// set things up before displaying
void setup() {
  size(320,240);
  capture = new Capture(this,width,height,30);

  // smaller image on camera for effeciency
  img = new PImage(80,60);
  theBlobDetection = new BlobDetection(img.width, img.height);
  theBlobDetection.setPosDiscrimination(true);
  theBlobDetection.setThreshold(0.2f);
}

// mouse release event


// get the capture event and read in the video feed
void captureEvent(Capture capture) {
  capture.read();
  newFrame = true;
}

// main method to draw UI on the screen
void draw() {

  if (newFrame) {
```

CSE 364 Project - Colored Object Detection and Tracking
George Donnelly

```
      newFrame=false;
  image(capture, 0, 0);
  loadPixels();

  if(start_tracking == true) {

    // display to the use the color that was chosen
    fill(target_color);
    ellipse(10,10,15,15);
    println("Color search for #" + hex(target_color));


    // variable for the closest color match
    float best_color = 40;

    // XY for closest match
    int close_x = 0;
    int close_y = 0;

    // loop through the pixels and perform logic
     for (int i = 0; i < pixels.length; i++ ) {

       // get the color at loc
       color colorTmp = capture.pixels[i];

       // get the difference of the RBG of target color and current color
       // and divide by 3 because of 3 colors
       float diff =  abs(red(target_color) - red(colorTmp)) + abs(green(target_color) - green(colorTmp)) +
abs(blue(target_color) - blue(colorTmp))/3;

       if(diff <= best_color) {
         pixels[i] = color(#ffffff);
       } else {
         pixels[i] = color(#000000);
       }

     }

     updatePixels();
       PImage img2 = getFrame();
       img.copy(img2, 0, 0, img2.width, img2.height,
                                  0, 0, img.width, img.height);
       fastblur(img, 2);
       theBlobDetection.computeBlobs(img.pixels);
```

```
      drawBlobsAndEdges(true,true);



    }


  }
}


// on mouse release get the pixel color the user wants
void mouseReleased() {
  // get the target the user wants
  target_color = capture.pixels[mouseX + mouseY*capture.width];
  start_tracking = true;
}


// detect the edges and start drawing the blobs
void drawBlobsAndEdges(boolean drawBlobs, boolean drawEdges) {
        noFill();
        Blob b;
        EdgeVertex eA,eB;
        for (int n=0 ; n<theBlobDetection.getBlobNb() ; n++)
        {
                b=theBlobDetection.getBlob(n);
                if (b!=null)
                {
                        // Edges
                        if (drawEdges)
                        {
                                strokeWeight(3);
                                stroke(0,255,0);
                                for (int m=0;m<b.getEdgeNb();m++)
                                {
                                        eA = b.getEdgeVertexA(m);
                                        eB = b.getEdgeVertexB(m);
                                        if (eA !=null && eB !=null)
                                                line(
                                                        eA.x*width, eA.y*height,
                                                        eB.x*width, eB.y*height
                                                        );
                                }
                        }
```

CSE 364 Project - Colored Object Detection and Tracking
George Donnelly

```
                        // Blobs
                        if (drawBlobs)
                        {
                                strokeWeight(1);
                                stroke(255,0,0);
                                rect(
                                        b.xMin*width,b.yMin*height,
                                        b.w*width,b.h*height
                                        );
                        }

                }

    }
}

// ====================================================
// Super Fast Blur v1.1
// by Mario Klingemann
// <http://incubator.quasimondo.com>
// ====================================================
void fastblur(PImage img,int radius)
{
 if (radius<1){
   return;
  }
  int w=img.width;
  int h=img.height;
  int wm=w-1;
  int hm=h-1;
  int wh=w*h;
  int div=radius+radius+1;
  int r[]=new int[wh];
  int g[]=new int[wh];
  int b[]=new int[wh];
  int rsum,gsum,bsum,x,y,i,p,p1,p2,yp,yi,yw;
  int vmin[] = new int[max(w,h)];
  int vmax[] = new int[max(w,h)];
  int[] pix=img.pixels;
  int dv[]=new int[256*div];
  for (i=0;i<256*div;i++){
    dv[i]=(i/div);
  }
```

CSE 364 Project - Colored Object Detection and Tracking
George Donnelly

```
 yw=yi=0;

 for (y=0;y<h;y++){
  rsum=gsum=bsum=0;
  for(i=-radius;i<=radius;i++){
   p=pix[yi+min(wm,max(i,0))];
   rsum+=(p & 0xff0000)>>16;
   gsum+=(p & 0x00ff00)>>8;
   bsum+= p & 0x0000ff;
  }
  for (x=0;x<w;x++){

   r[yi]=dv[rsum];
   g[yi]=dv[gsum];
   b[yi]=dv[bsum];

   if(y==0){
    vmin[x]=min(x+radius+1,wm);
    vmax[x]=max(x-radius,0);
   }
   p1=pix[yw+vmin[x]];
   p2=pix[yw+vmax[x]];

   rsum+=((p1 & 0xff0000)-(p2 & 0xff0000))>>16;
   gsum+=((p1 & 0x00ff00)-(p2 & 0x00ff00))>>8;
   bsum+= (p1 & 0x0000ff)-(p2 & 0x0000ff);
   yi++;
  }
  yw+=w;
 }

 for (x=0;x<w;x++){
  rsum=gsum=bsum=0;
  yp=-radius*w;
  for(i=-radius;i<=radius;i++){
   yi=max(0,yp)+x;
   rsum+=r[yi];
   gsum+=g[yi];
   bsum+=b[yi];
   yp+=w;
  }
  yi=x;
  for (y=0;y<h;y++){
   pix[yi]=0xff000000 | (dv[rsum]<<16) | (dv[gsum]<<8) | dv[bsum];
   if(x==0){
```

CSE 364 Project - Colored Object Detection and Tracking
George Donnelly

```
        vmin[y]=min(y+radius+1,hm)*w;
        vmax[y]=max(y-radius,0)*w;
      }
      p1=x+vmin[y];
      p2=x+vmax[y];

      rsum+=r[p1]-r[p2];
      gsum+=g[p1]-g[p2];
      bsum+=b[p1]-b[p2];

      yi+=w;
    }
  }

}


// method retrieves the current webcam frame
PImage getFrame() {
  PImage img = new PImage(width, height);

  g.loadPixels();
  img.loadPixels();

  img.pixels = g.pixels;

  img.updatePixels();
  g.updatePixels();

  return img;
}
```