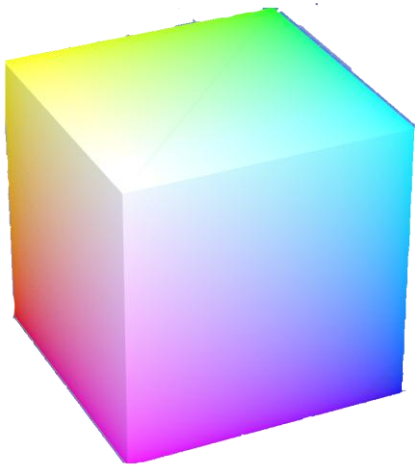
Vision

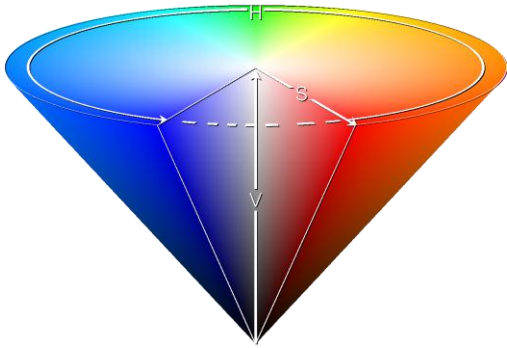
Geza Kovacs

Maslab 2011

Colorspaces

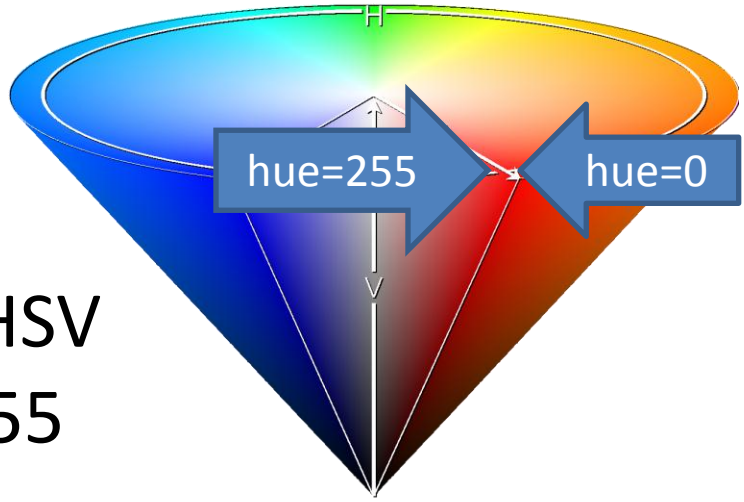


- RGB: red, green, and blue components
- HSV: hue, saturation, and value
- Your color-detection code will be more resilient to lighting conditions if you use HSV

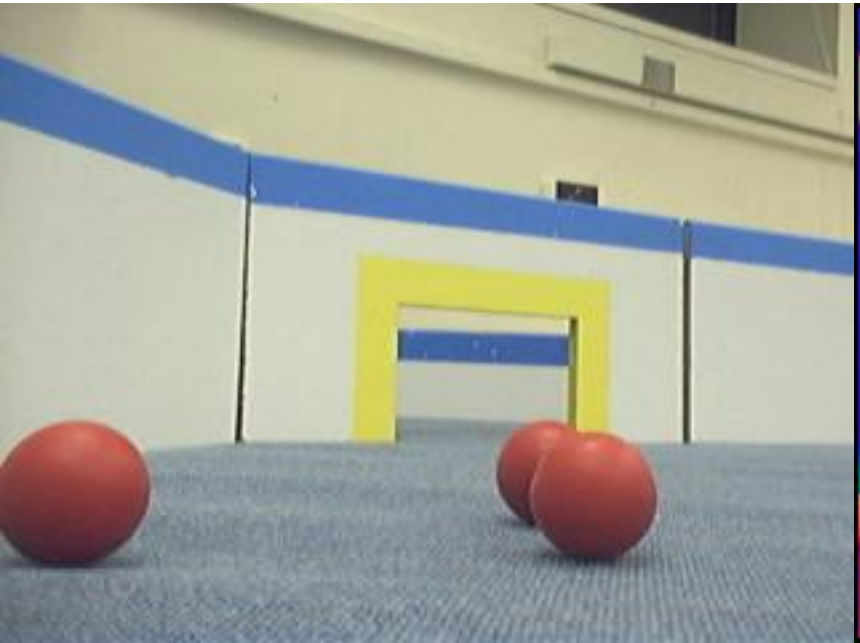


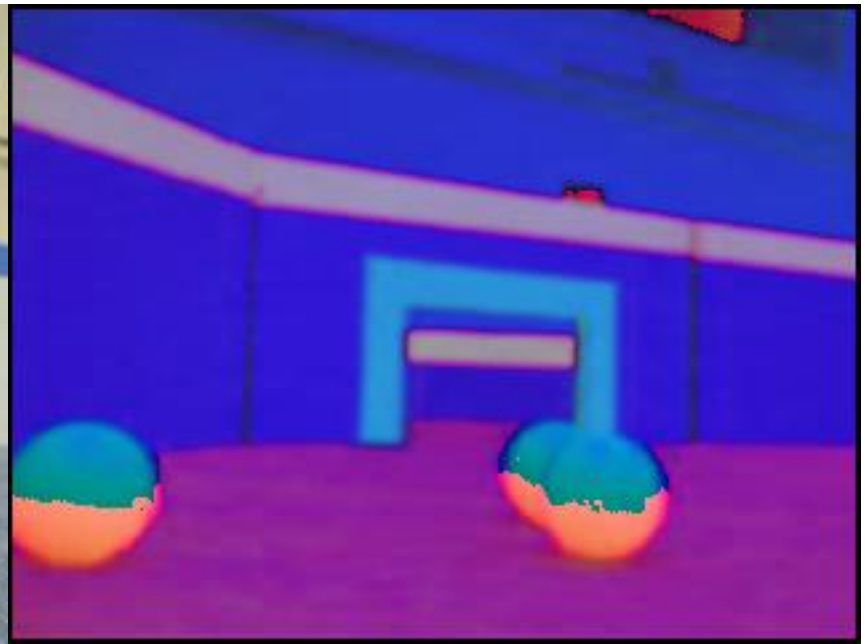
RGB: 212, 45, 45 HSV: 0, 201, 212	RGB: 102, 0,0 HSV: 0, 255, 102	RGB: 255, 105,105 HSV: 0, 105, 255

Colorspaces



- Note that because the hue in HSV wraps around, red is both h=255 and h=0
- See Tutorial for more info on HSV





Representation of Color in Bytes

BufferedImage img = …;

img.getRGB(x, y) returns a 32-bit (4-byte) integer

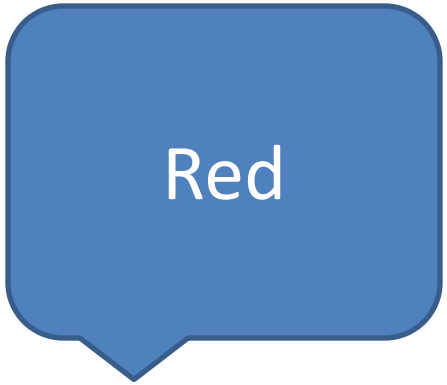
0x00FF0000

Representation of Color in Bytes

Alpha channel: basically transparency, not of interest

0x**00**FF0000

Representation of Color in Bytes



0x00**FF**0000

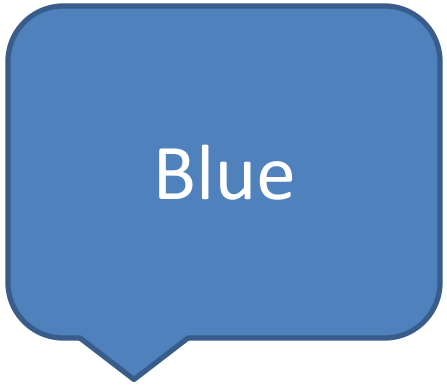
Red: 0xFF=255

Representation of Color in Bytes

0x00FF**00**00

Red: 0xFF=255 Green: 0x00=0

Representation of Color in Bytes



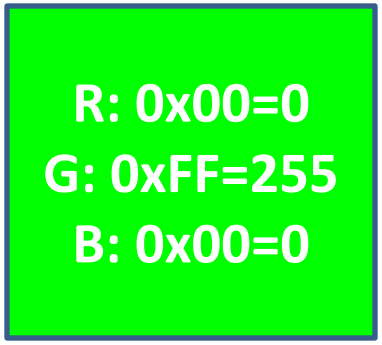
0x00FF00**00**

Red: 0xFF=255 Green: 0x00=0 Blue: 0x00=0

Representation of Color in Bytes

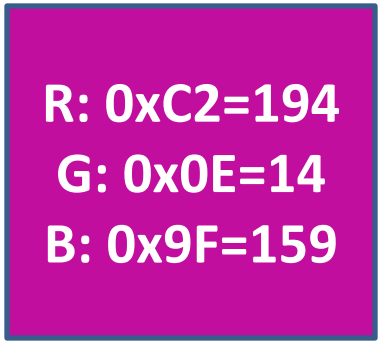
0x00FF0000

0x0000FF00



0x000000FF

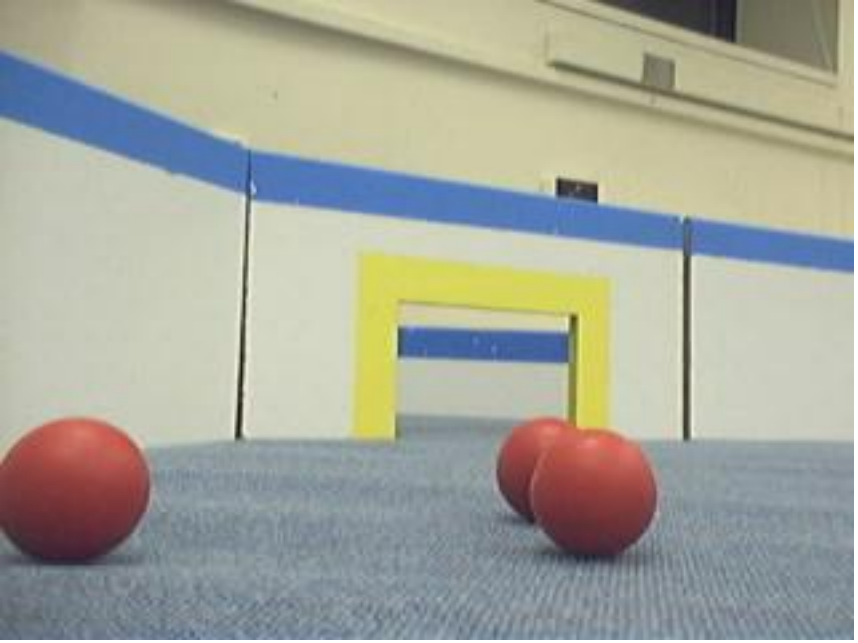
0x00C20E9F

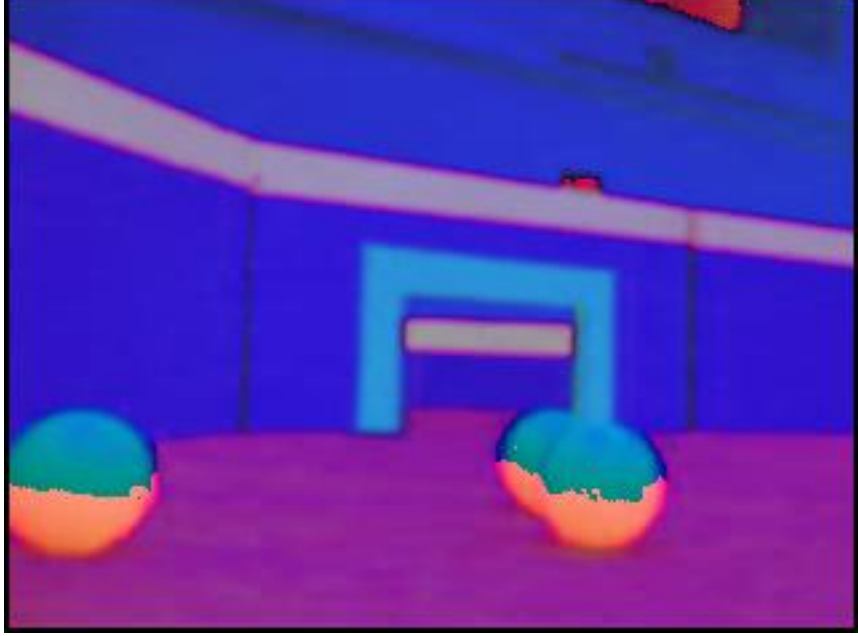


Extracting out R, G, B components

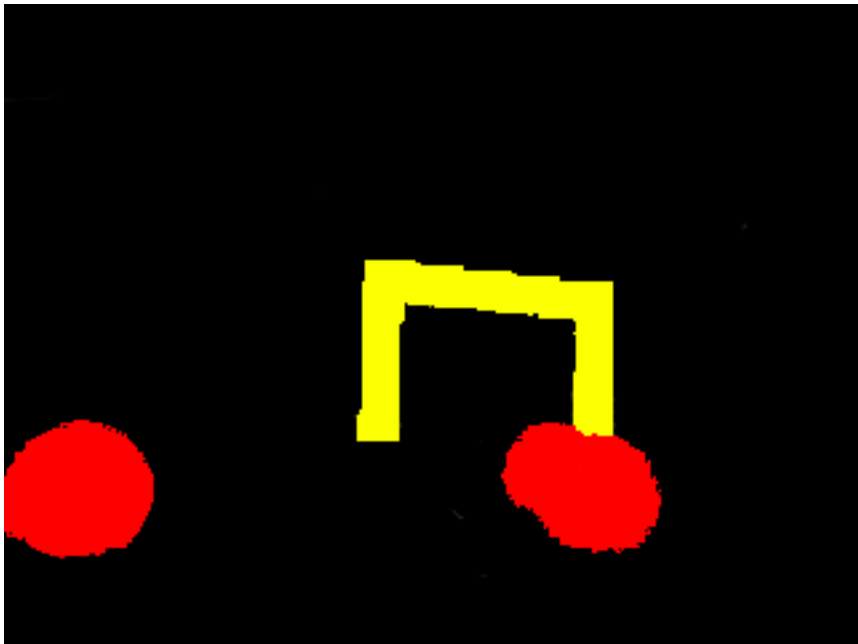
```
BufferedImage img = …;
int rgb = img.getRGB(x,y);
int r = (rgb & 0x00FF0000) >> 16;
int g = (rgb & 0x0000FF00) >> 8;
int b = (rgb & 0x000000FF);
rgb = b + (g << 8) + (r << 16)
```

- Also works if the image is in HSV format; just replace r with h, g with s, and b with v
- See Vision tutorial for more info



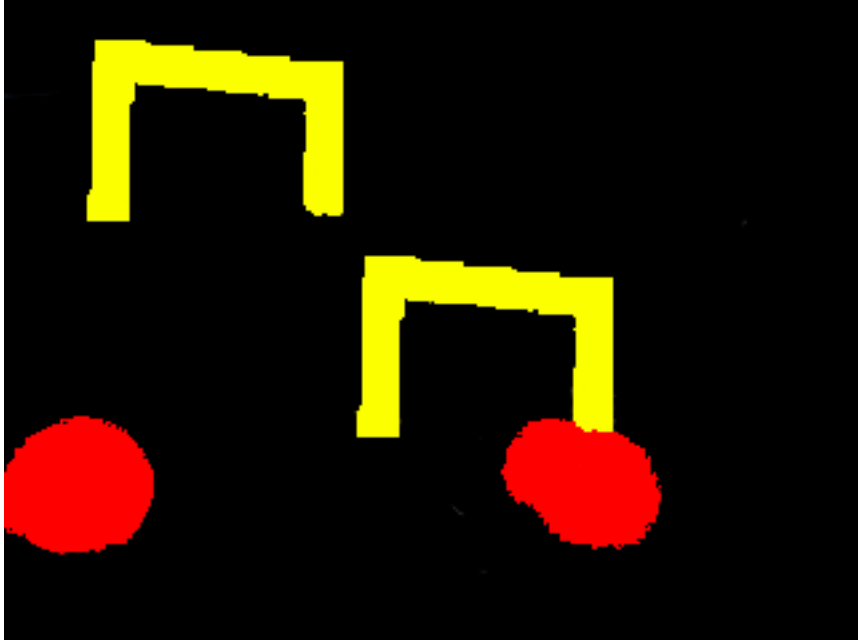


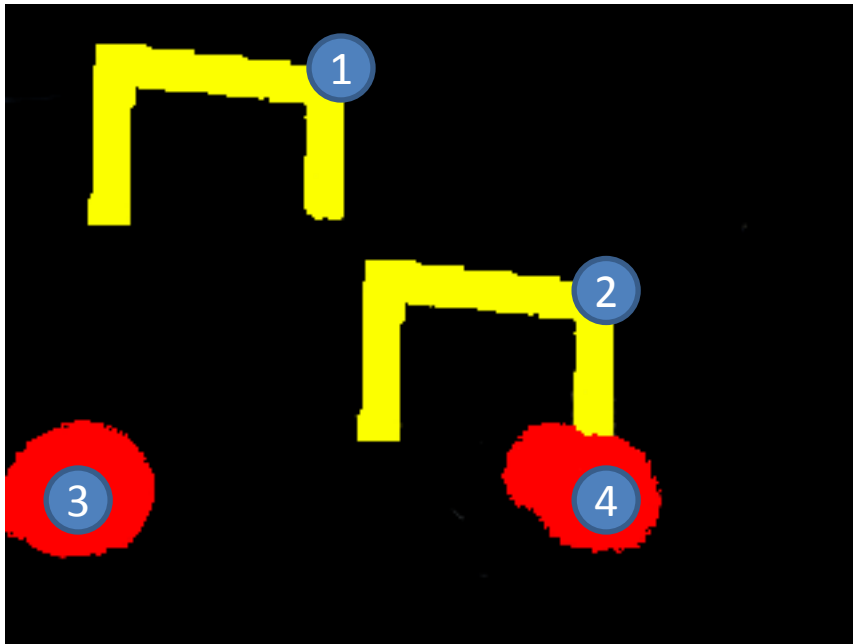
- By using color thresholds, (checking that hue is in a certain range), can classify pixels as being Red, Yellow, or other



Connected Component Labeling

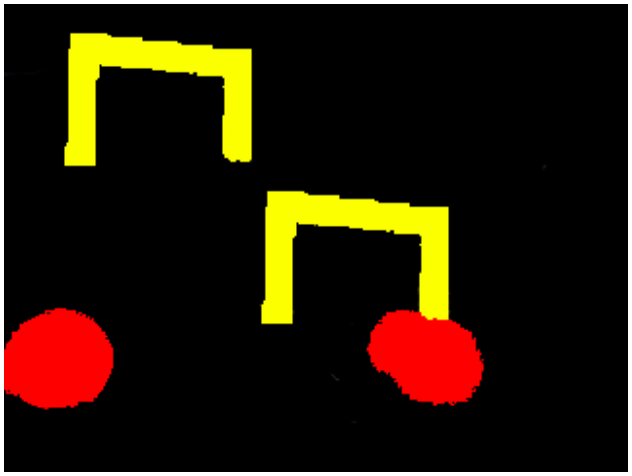
- As a first step in detecting balls and goals, we want to group connected yellow pixels, and connected red pixels into “blobs of interest”
  - That is, label each pixel with a number indicating the connected component it belongs to

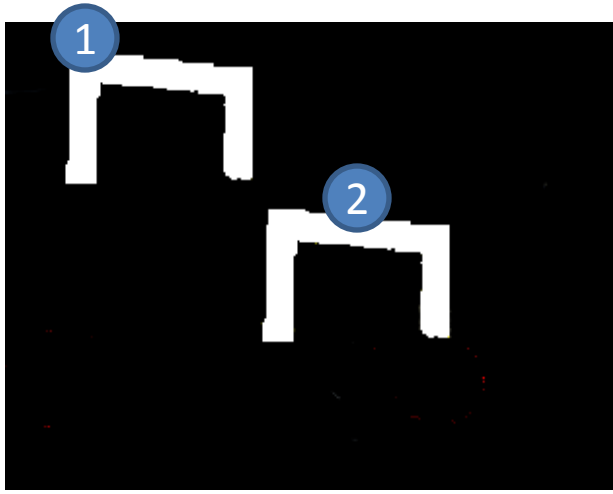


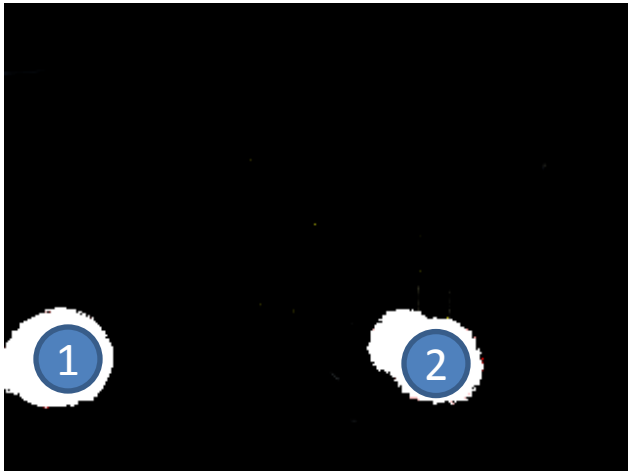


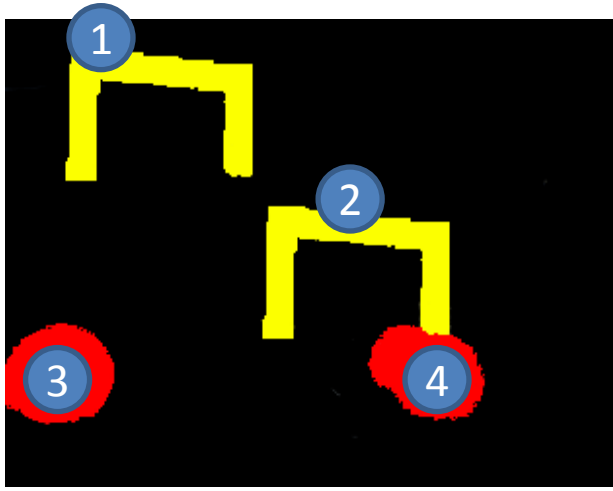
Connected Component Labeling

- Various efficient algorithms exist for finding connected components of white pixels in binary images
- We can use these algorithms if we consider colors one at a time

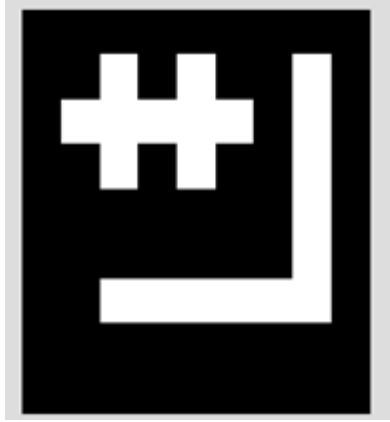






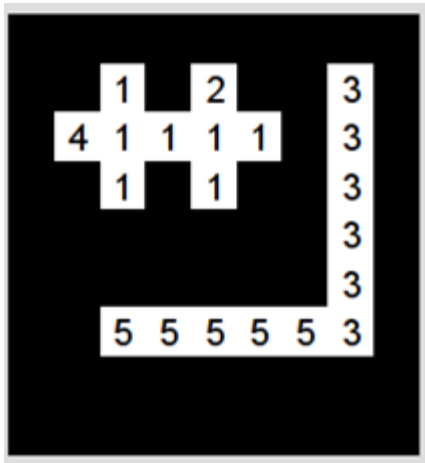


2-pass algorithm for Connected Component Labeling on Binary Images



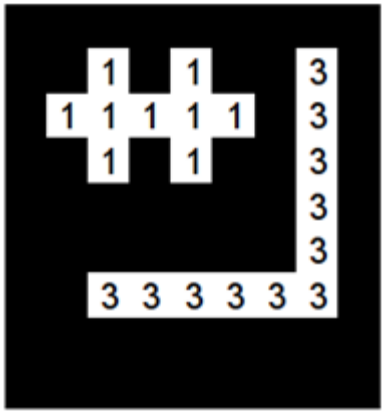
Pass 1:

- If all 4 neighbors are black or unlabeled, assign a new label to current point
- If only one neighbor is white, assign its label to current point
- If more than one of the neighbors are white, assign one of their labels to current point, and note equivalence of their labels



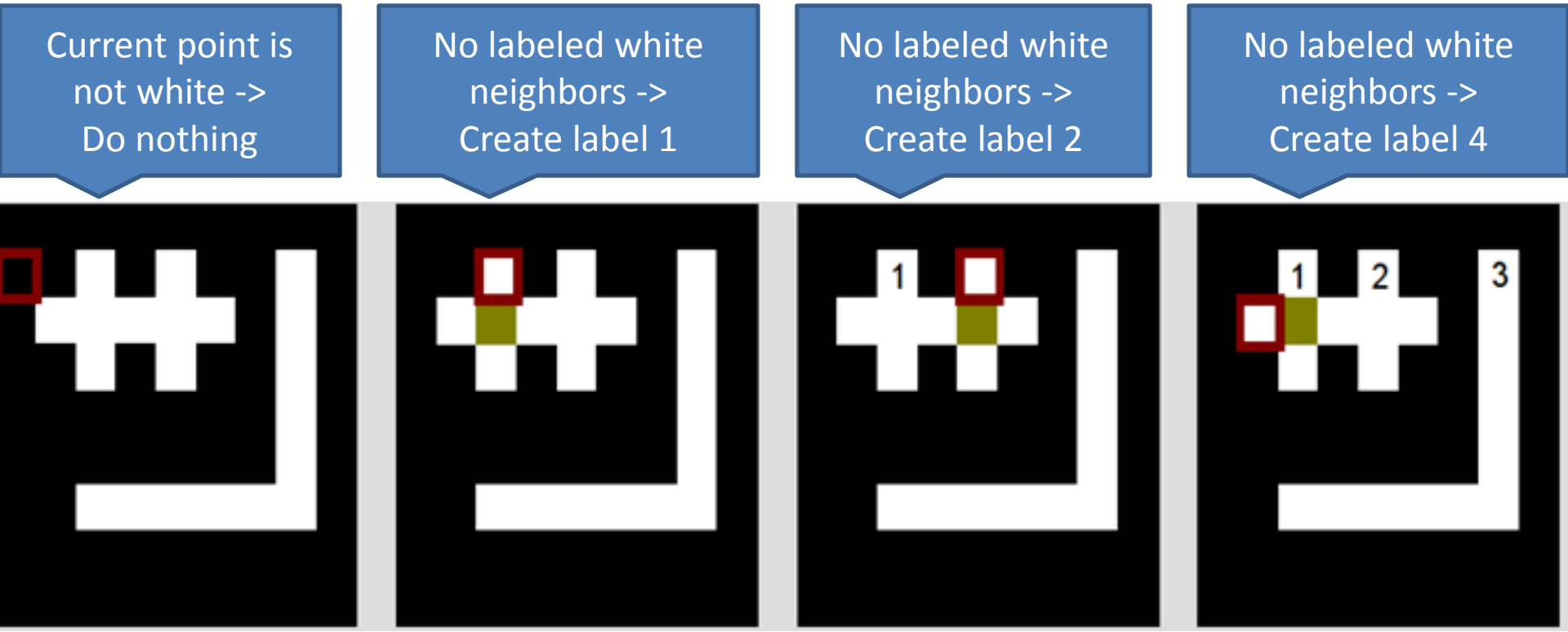
Pass 2:

- Merge labels which were marked as equivalent in the first pass



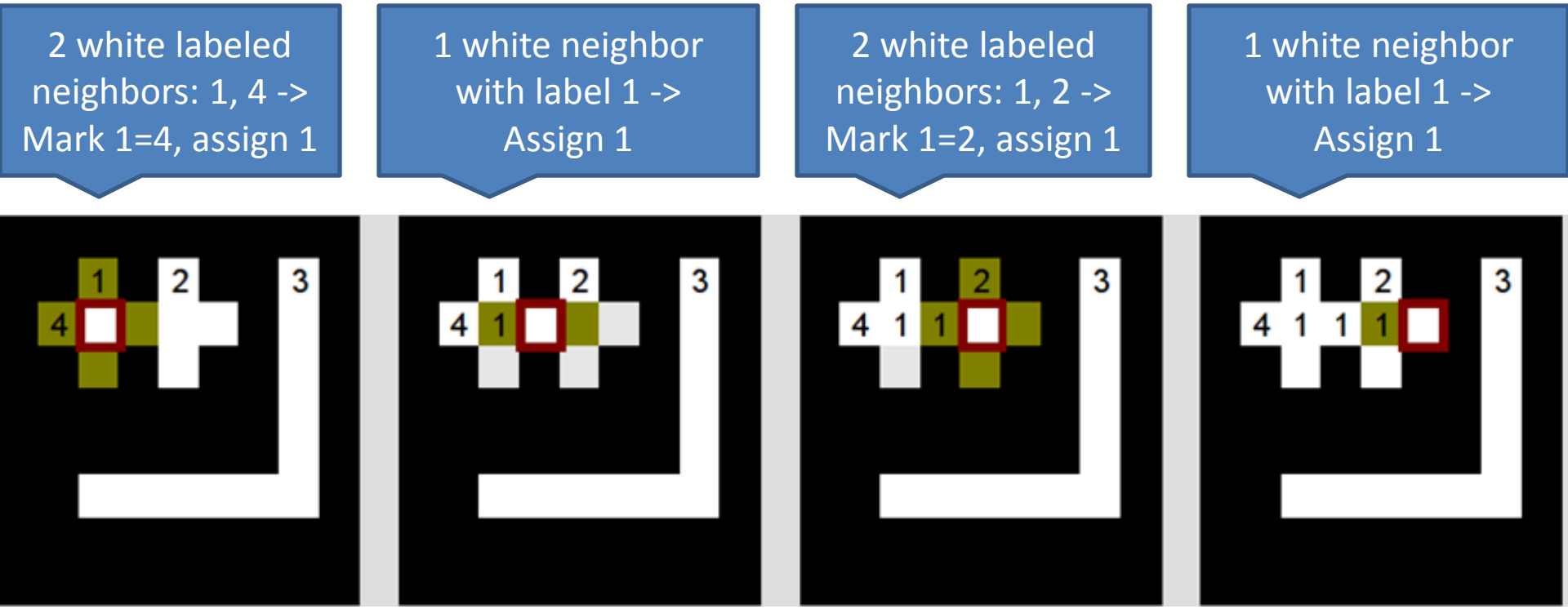
Pass 1:

- If all 4 neighbors are black or unlabeled, assign a new label to current point
- If only one neighbor is white, assign its label to current point
- If more than one of the neighbors are white, assign one of their labels to current point, and note equivalence of their labels

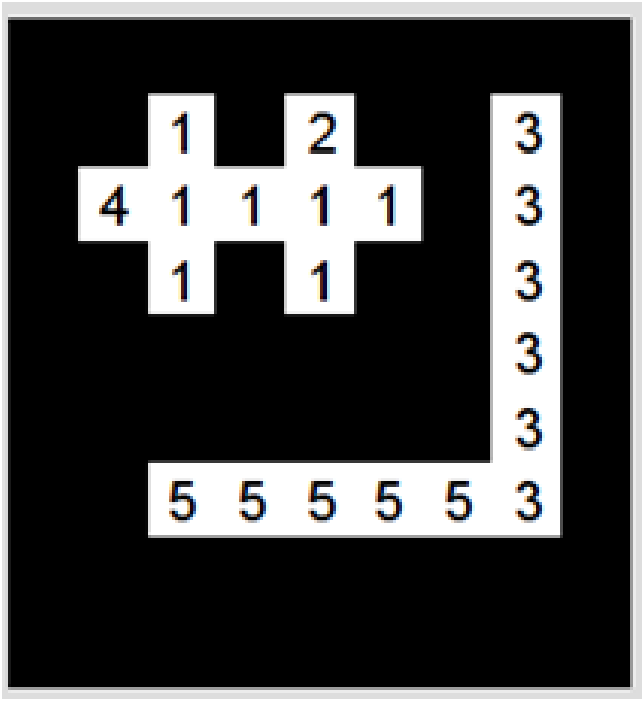


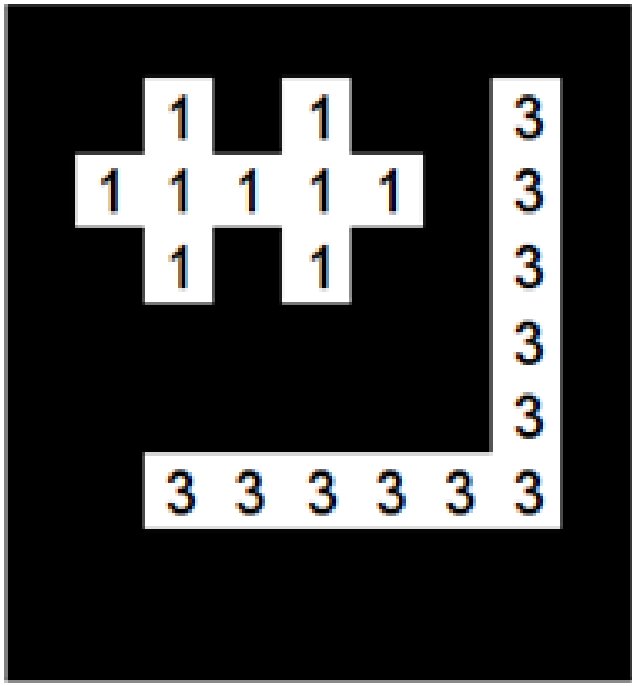
Pass 1:

- If all 4 neighbors are black or unlabeled, assign a new label to current point
- If only one neighbor is white, assign its label to current point
- If more than one of the neighbors are white, assign one of their labels to current point, and note equivalence of their labels

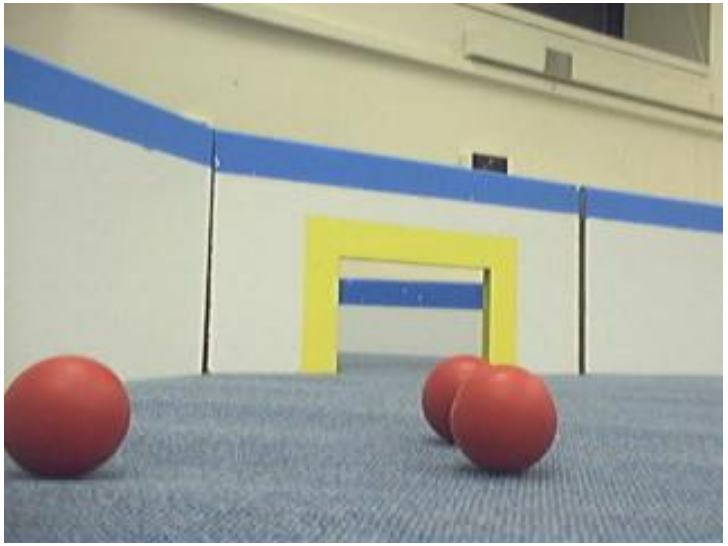


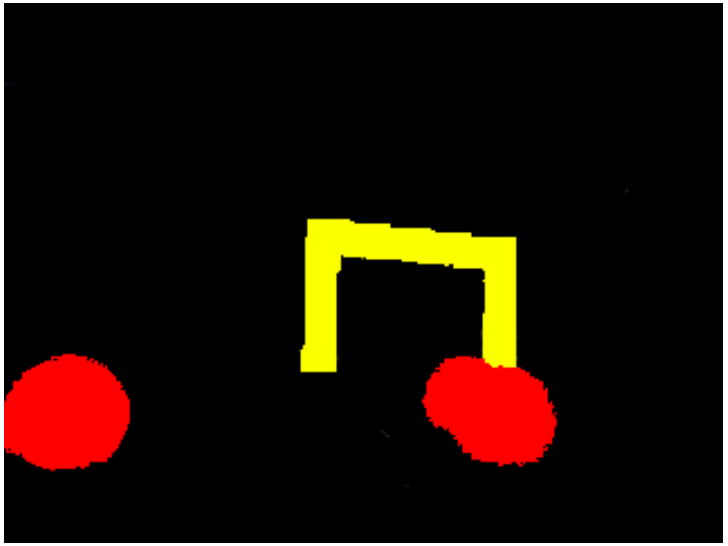
- At the end of the first pass, we have marked labels 1, 2, and 4 as equivalent, and have marked labels 3 and 5 as equivalent
- In the second pass, we replace all 2s and 4s with 1s, and replace all 5s with 3s





Colors aren't always enough for segmenting objects





- Note that at edges of objects, there is a change in pixel value
- Use edge detection for segmenting objects

Image Convolution

- Determines pixel value based on neighboring values (relation described by a kernel matrix)
- Used in blurring, sharpening, edge detection, etc

Kernel

0	1/6	0
1/6	1/3	1/6
0	1/6	0

Source Image (single-chanel, greyscale)

11	4	13	8
7	1	3	9
0	5	10	6
2	15	12	14

Result of convolution

	1/3 + 4/6 + 7/6 + 3/6 + 5/6 = **21/6**	3/3 + 13/6 + 1/6 + 9/6 + 10/6 = **39/6**	
	5/3 + 1/6 + 0/6 + 10/6 + 15/6 = **36/6**	10/3 + 3/6 + 5/6 + 6/6 + 12/6 = **46/6**	

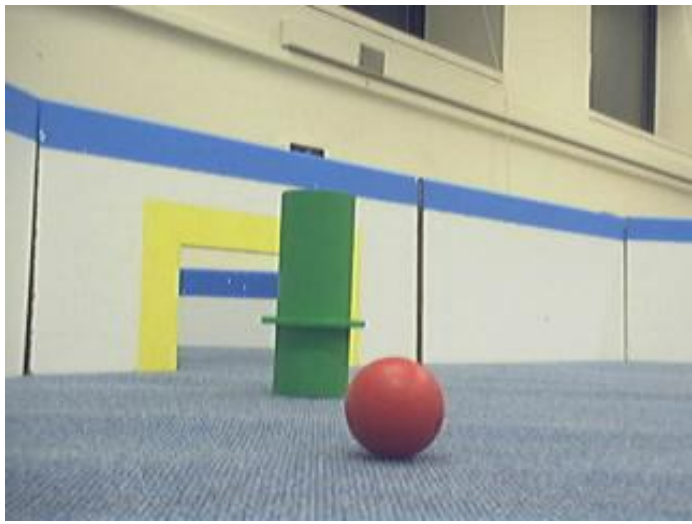
- Various workarounds for determining the edge pixels

```
import java.awt.image.*;
BufferedImage simpleBlur(BufferedImage src) {
    float[] matrix = new float[] {
        0.0f, 1.0f/6, 0.0f,
        1.0f/6, 1.0f/3, 1.0f/6,
        0.0f, 1.0f/6, 0.0f,
    };
    Kernel kernel = new Kernel(3, 3, matrix);
    return new ConvolveOp(kernel).filter(src, null);
}
```

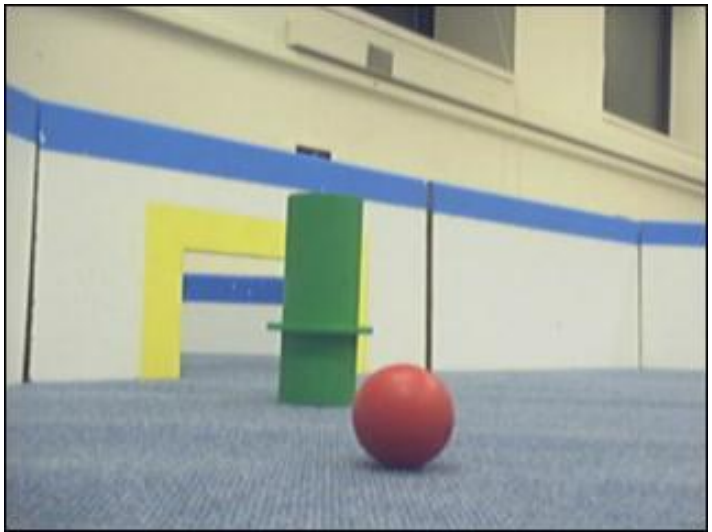
Kernel

0	1/6	0
1/6	1/3	1/6
0	1/6	0

Source Image



Result of convolution (slightly blurred image)

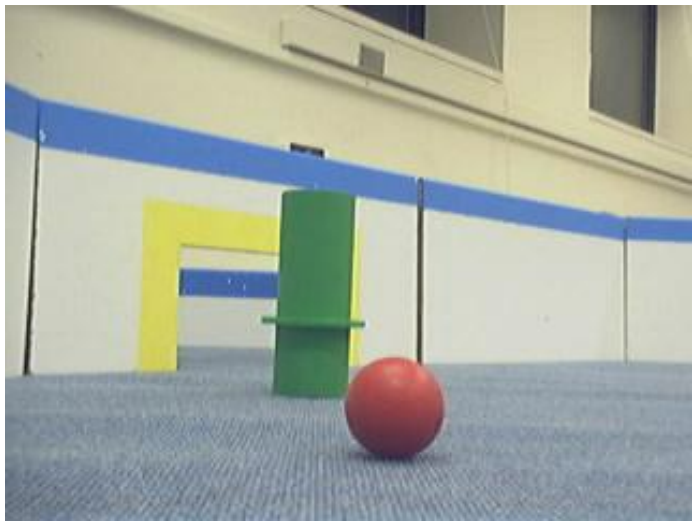


```
import java.awt.image.*;
BufferedImage simpleBlur(BufferedImage src) {
    float[] matrix = new float[] {
        0.0f, 1.0f/6, 0.0f,
        1.0f/6, 1.0f/3, 1.0f/6,
        0.0f, 1.0f/6, 0.0f,
    };
    Kernel kernel = new Kernel(3, 3, matrix);
    return new ConvolveOp(kernel).filter(src, null);
}
```

Kernel

0	1/6	0
1/6	1/3	1/6
0	1/6	0

Source Image



Result of convolution applied 20 times (more blurred image)

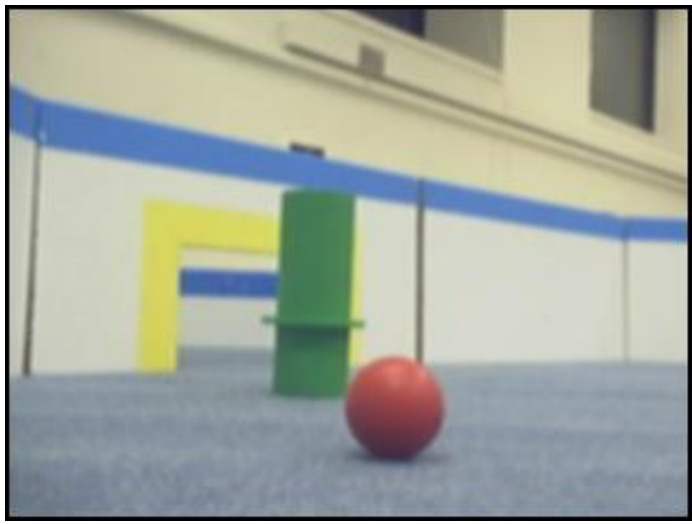
Gaussian Blur

- A common preprocessing step before various operations (edge detection, color classification before doing connected component labeling, etc)

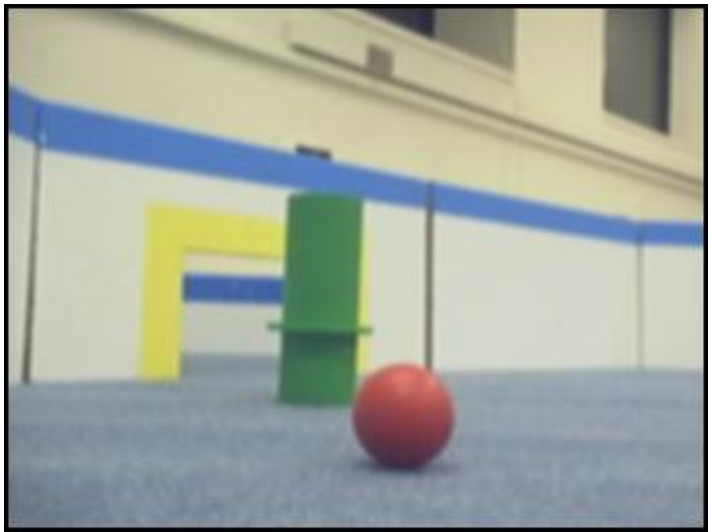
Kernel

2/159	4/159	5/159	4/159	2/159
4/159	9/159	12/159	9/159	4/159
5/159	12/159	15/159	12/159	5/159
4/159	9/159	12/159	9/159	4/159
2/159	4/159	5/159	4/159	2/159

Source Image



Result of convolution with gaussian kernel



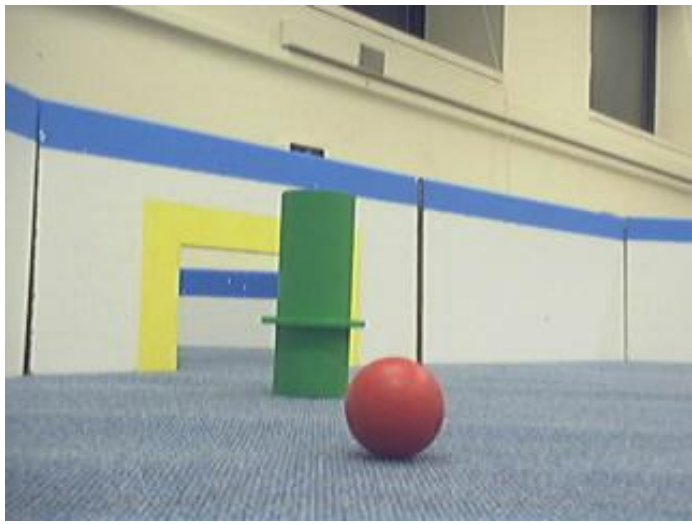
Detecting Horizontal Edges

- Use the Sobel operator G_x

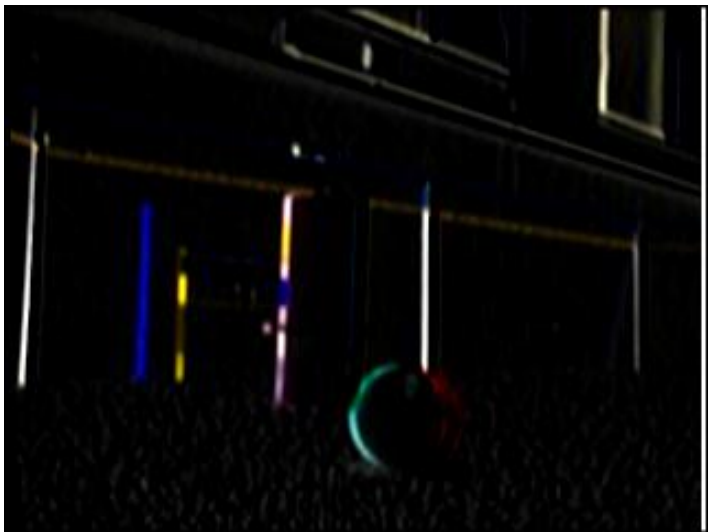
Kernel

-1	-2	-1
0	0	0
1	2	1

Source Image



Result of convolution with sobel operator G_x



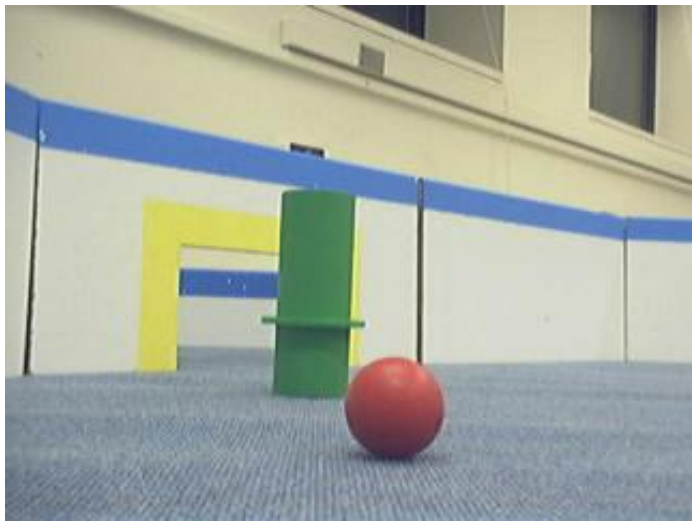
Detecting Vertical Edges

- Use the Sobel operator G_y

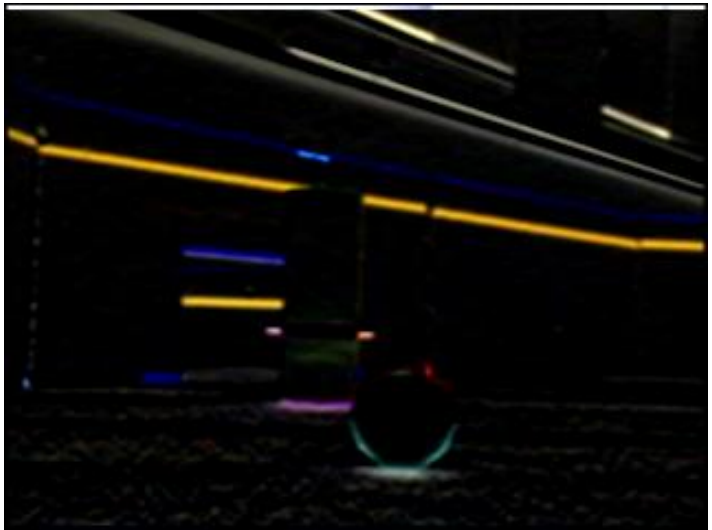
Kernel

-1	-2	-1
0	0	0
1	2	1

Source Image



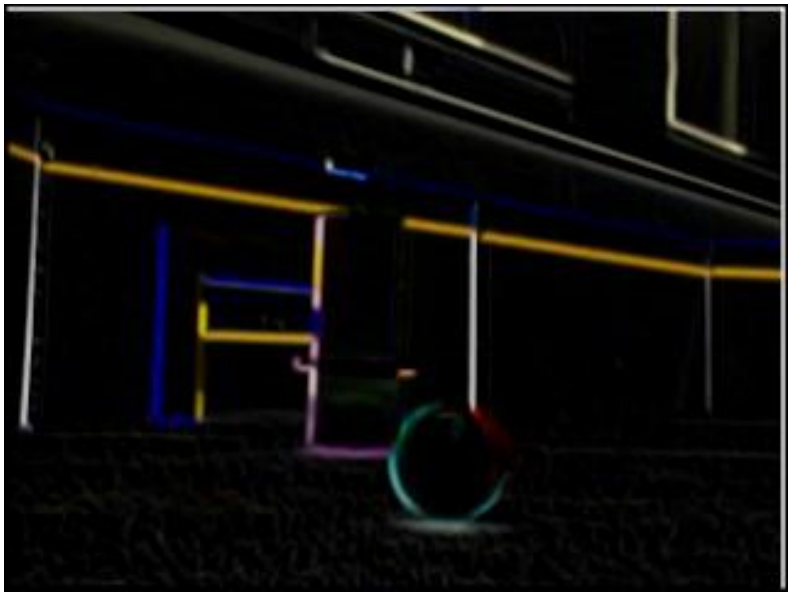
Result of convolution with sobel operator G_y

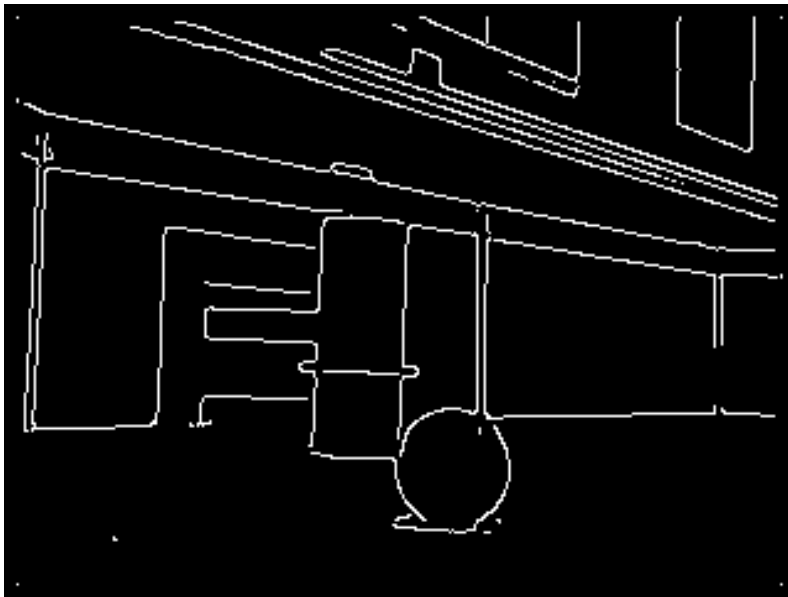


Edge Detection

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

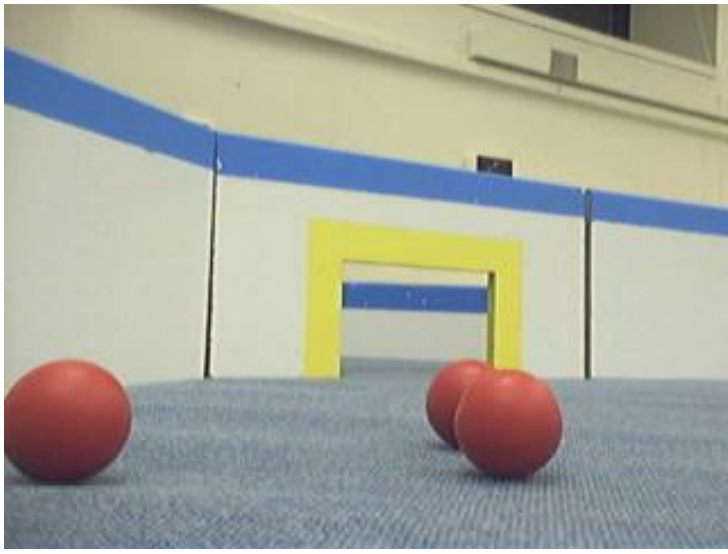
- Get matrices representing horizontal edges (G_x) and vertical edges (G_y) (via convolution with sobel operator)
- Then, assign each pixel squareroot(value in the horizontal edge squared + value in vertical edge squared)
- Use more elaborate preprocessing and postprocessing to get nicer results

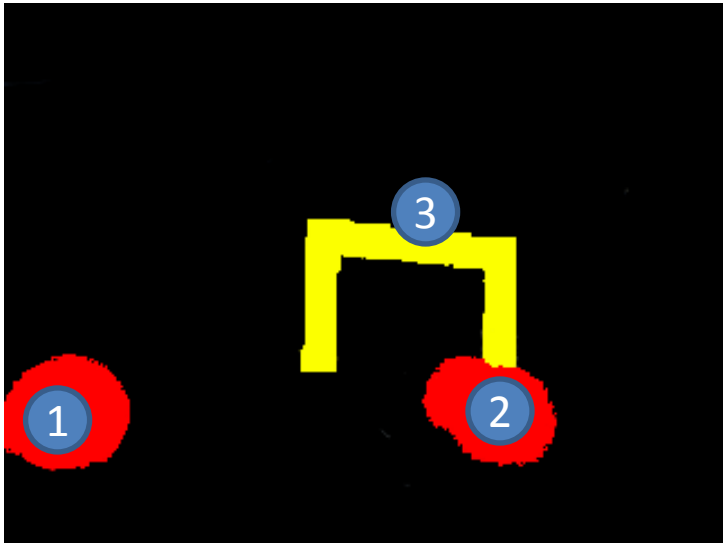


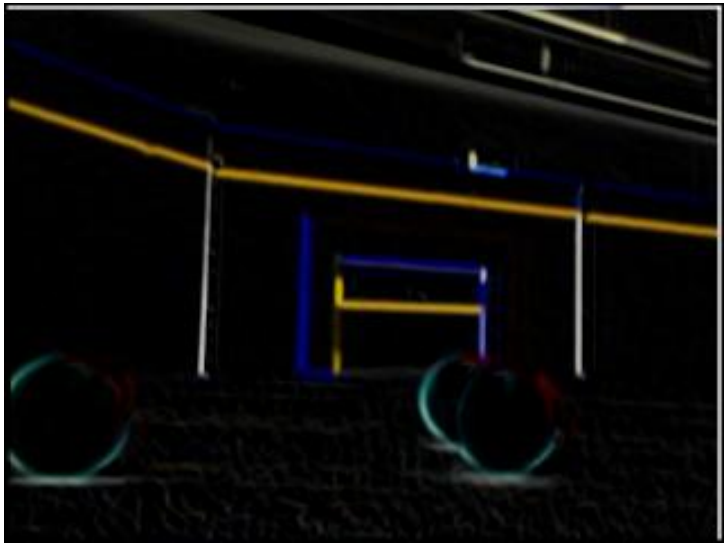


Segmenting Objects

- Group same-colored regions using connected component labeling, and use edges to segment objects further?
  - Fit lines or curves to edges?





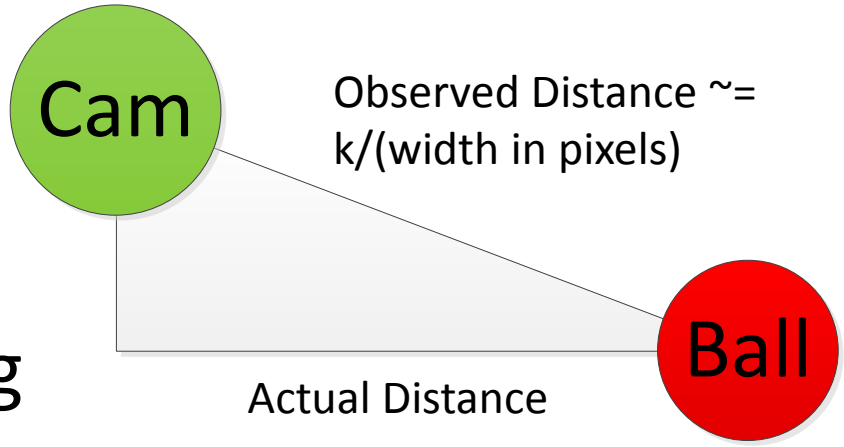


Classifying Objects as Goals or Balls

- Consider the shape: rounded boundary, vs straight boundary
- Consider the region around the center: red/yellow or not?
- Consider special cases: goals with balls in the middle, goals observed at an angle, etc

Estimating Distances to Objects

- Note that all balls have the same size. Likewise with goals, wall heights, etc
- By making some measurements and using some trig, you can estimate distance to objects from your image data



Testing Advice

- Keep a collection of images which you can use unit tests on
- Test detection of balls and goals from different angles, and arrange in various ways
- Make sure to test your vision code (especially color detection) in different lighting conditions

Other Resources

- Connected Component Labeling: http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm
- Edge Detection: http://www.pages.drexel.edu/~weg22/can_tut.html
- Various lectures from previous years also have info on camera details, performance optimizations, stereo vision, rigid body motion, etc

http://web.mit.edu/6.186/2010/lectures/vision.pdf
http://web.mit.edu/6.186/2007/lectures/vision/maslab-vision.ppt
http://web.mit.edu/6.186/2006/lectures/Vision.pdf
http://web.mit.edu/6.186/2005/doc/basicvision.pdf
http://web.mit.edu/6.186/2005/doc/morevision.pdf
http://courses.csail.mit.edu/6.141/spring2008/pub/lectures/Vision-I-Lecture.pdf