CMVision and Color Segmentation

CSE398/498 Robocup
19 Jan 05

Announcements

- Please send me your time availability for working in the lab during the M-F, 8AM-8PM time period

Why Color Segmentation?

- Computationally inexpensive (relative to other features)
- “Contrived” colors are easy to track
- Combines with other features for robust tracking

Target Tracking Demo

Color Tracking Demo

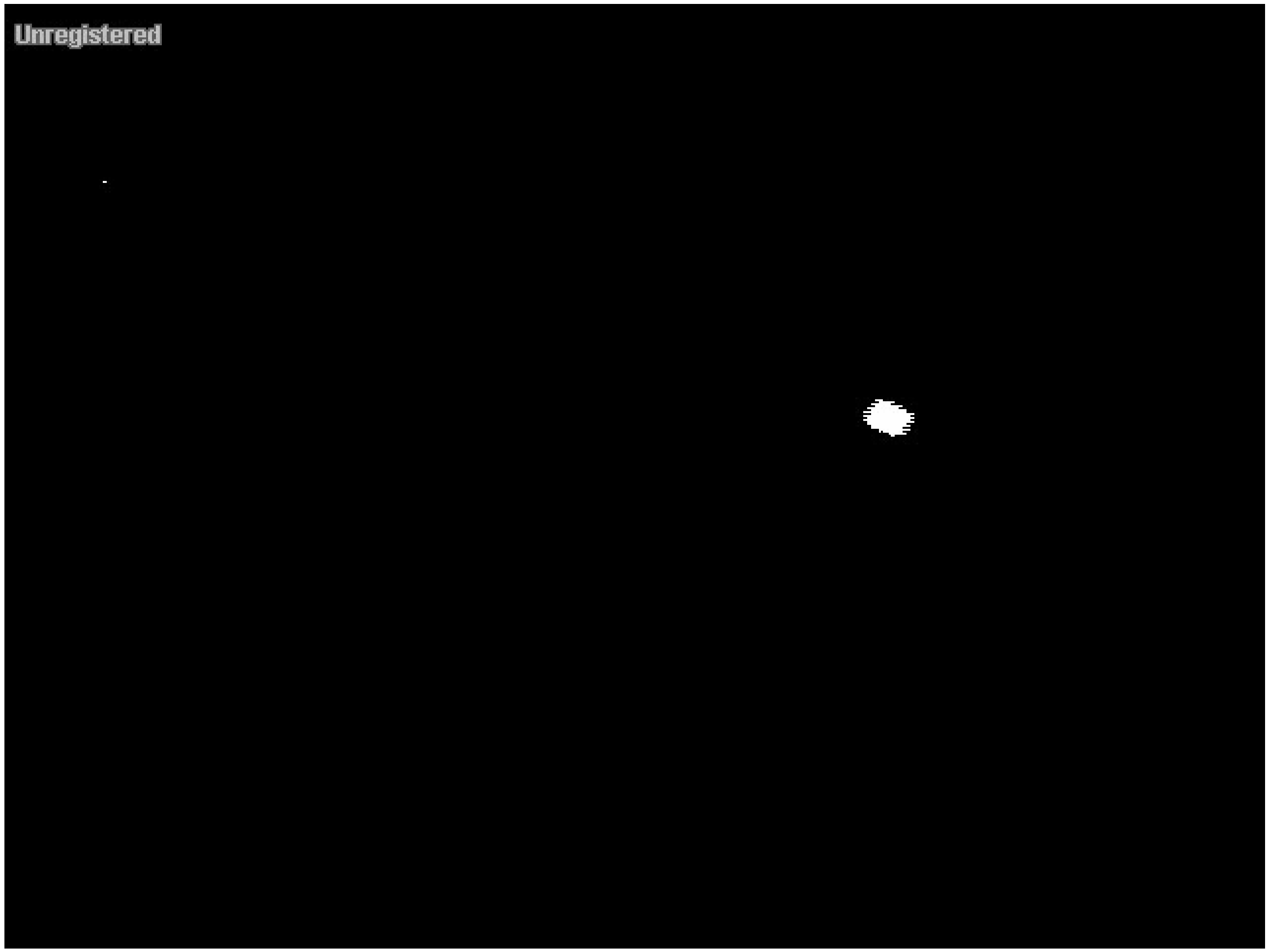


Image Representation

Let's Start with B&W Images

- These are referred to as *grayscale* or *gray level* images
- Corresponds to achromatic or monochromatic light
- Light "devoid" of color
- Also results from equal levels of R-G-B in an image

Image Representation

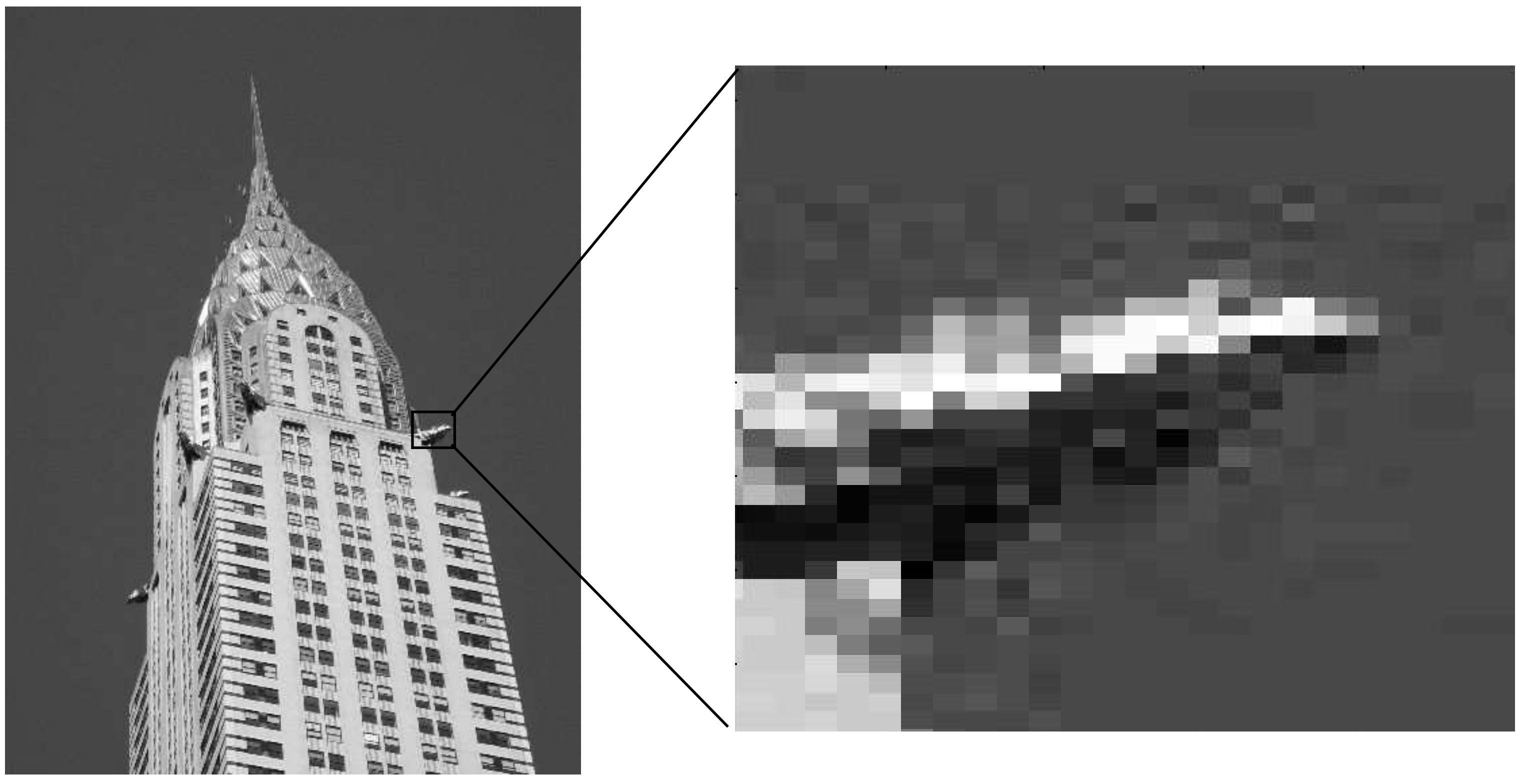
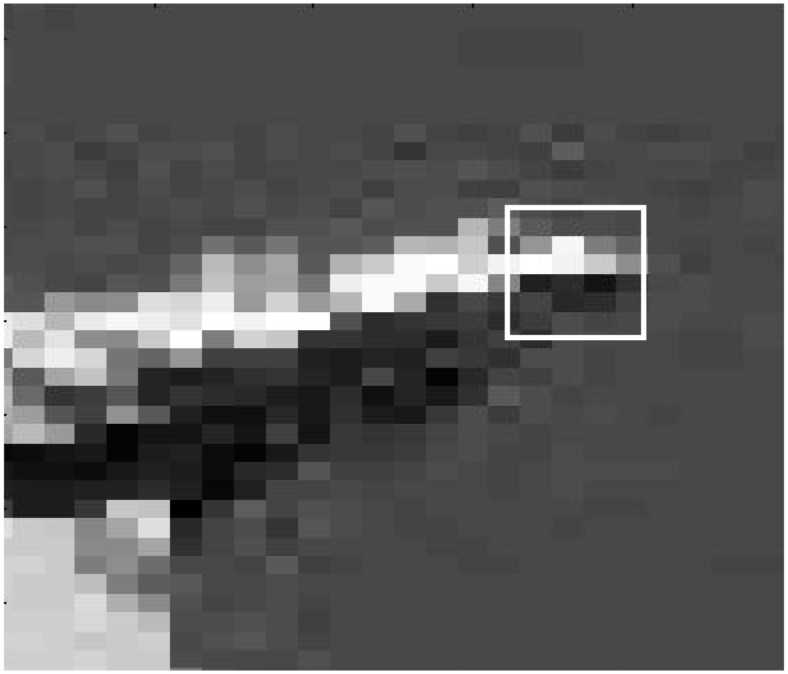


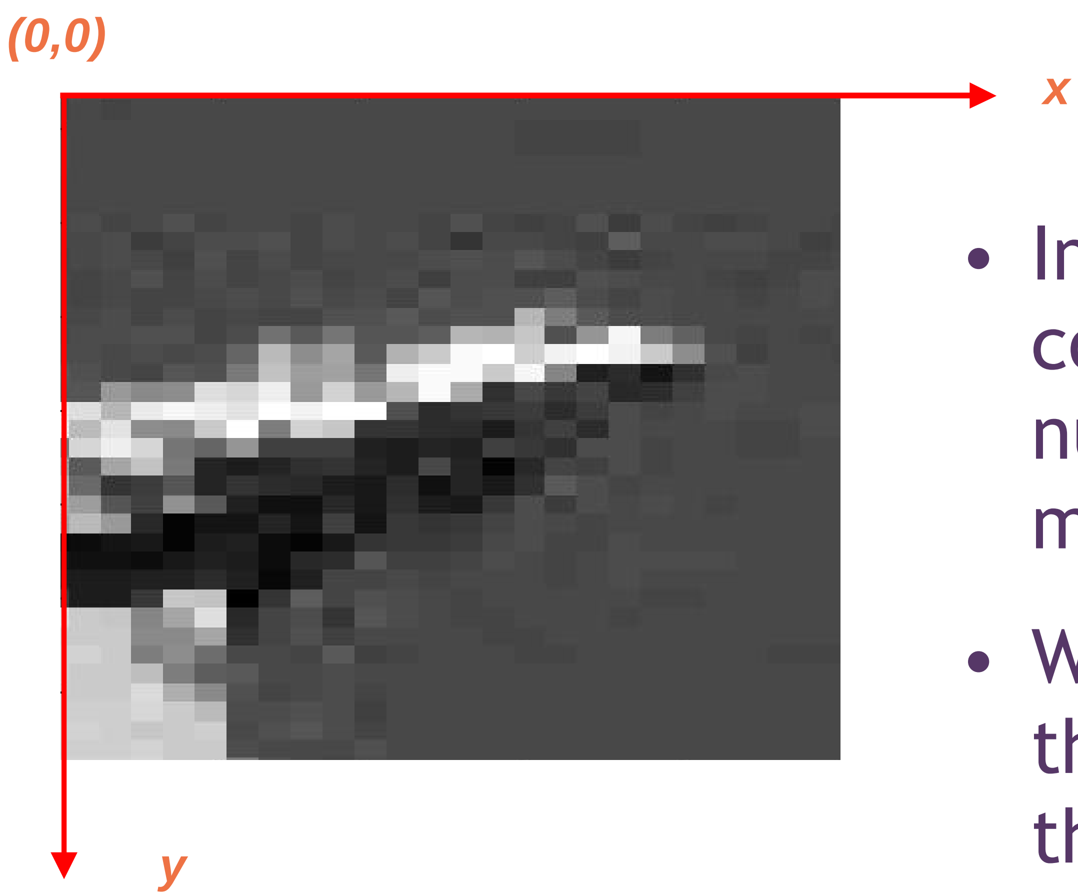
Image Representation



61	29	29	57	199	192
222	200	197	135	167	222
203	203	203	137	137	165
208	208	201	124	142	111
208	203	200	190	127	92
204	201	200	218	173	139

It's just a bunch of NUMBERS!

Digital Image Representation



- Images are contiguous blocks of numbers in computer memory
- We will manipulate these numbers to get them into a useful form

Digital Image Representation (cont'd)

- Several properties define the image format
  - Pixel (or spatial) Resolution (e.g. 640x480 pixels)
  - Pixel bit-depth (8-bit unsigned, 16-bit signed, etc.)
  - Frame rate (e.g. 30 Hz)
  - Colorspace (RGB, YCbCr, etc.)
  - Number of planes - 1 for grayscale images, 3 for color
  - Pixel format (planar vs. packed)

    R G B R G B… R G B

    R R … R G G … G B B … B

You MUST know ALL of these or you will have processed GARBAGE!

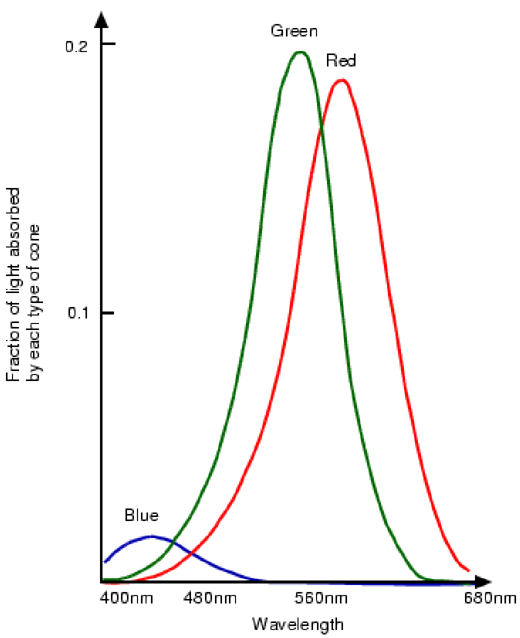
Grayscale Images

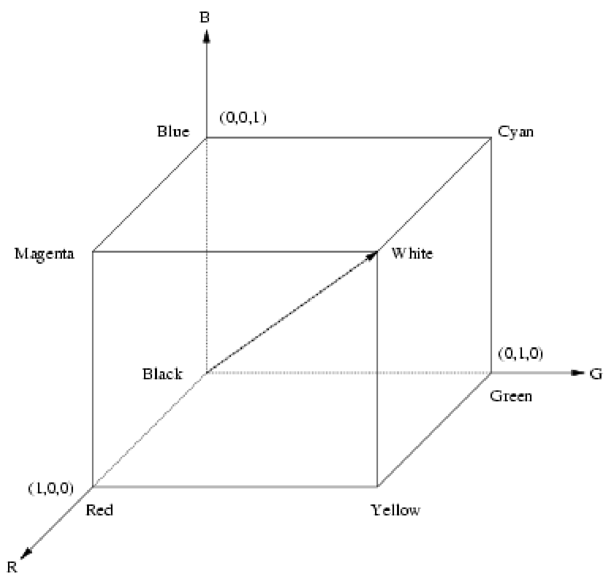
- Corresponds to achromatic or monochromatic light (without color)
- Typically 8-bit unsigned chars with a dynamic range of [0,255]
- One char corresponds to one image pixel

$$0 \leq I(x, y) \leq 255$$

RGB Color Space

- Motivated by human visual system
  - 3 color receptor cells (cones) in the retina with different spectral response curves
- Used in color monitors and most video cameras



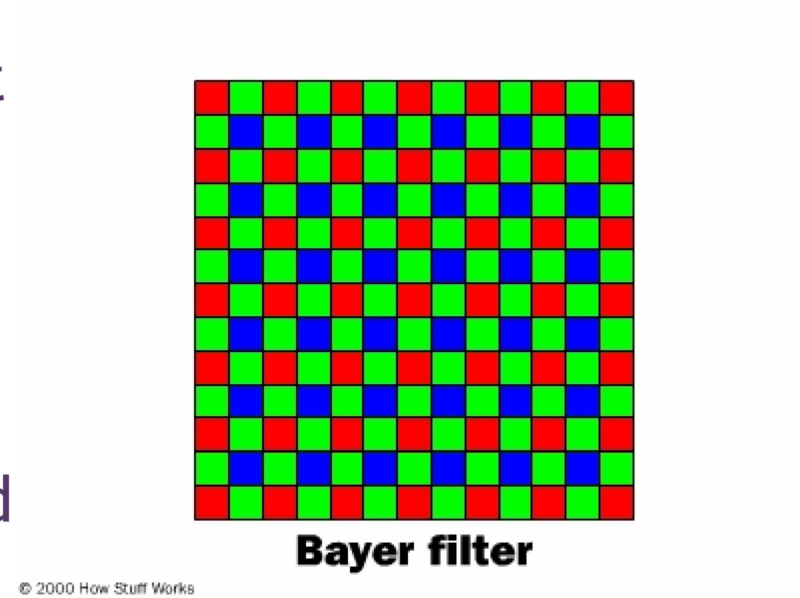


RGB Image Formation in Cameras

- Most video cameras use RGB space
- Expensive variants use 3 CCDs, each with a filter for the respective wavelength of light
- More common variants (like what we will use) have a single CCD
- Q: How do they reproduce color?
- A: A Filter!

The Bayer Filter

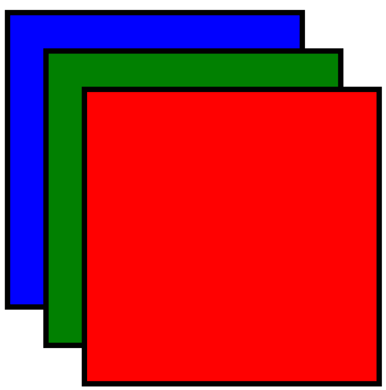
- Based upon the observation that human vision is much more responsive to green light than red or blue
- Half the pixels in the CCD are allocated to green, ¼ to red and ¼ to blue



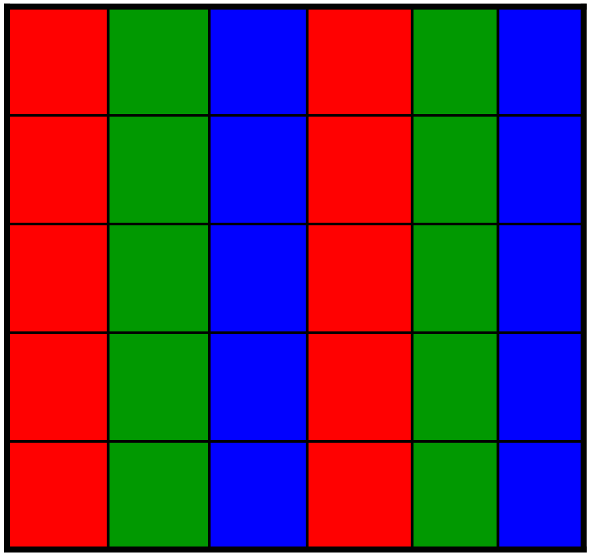
- Color is generated for the whole CCD by interpolating neighbor values
- The image we get has already undergone a "lossy compression"

RGB Image Format

- Images pixels can be either *planar* or *packed* format
- Planar format separates the colors into three contiguous arrays in memory
- Packed alternate R->G->B->R->… in memory

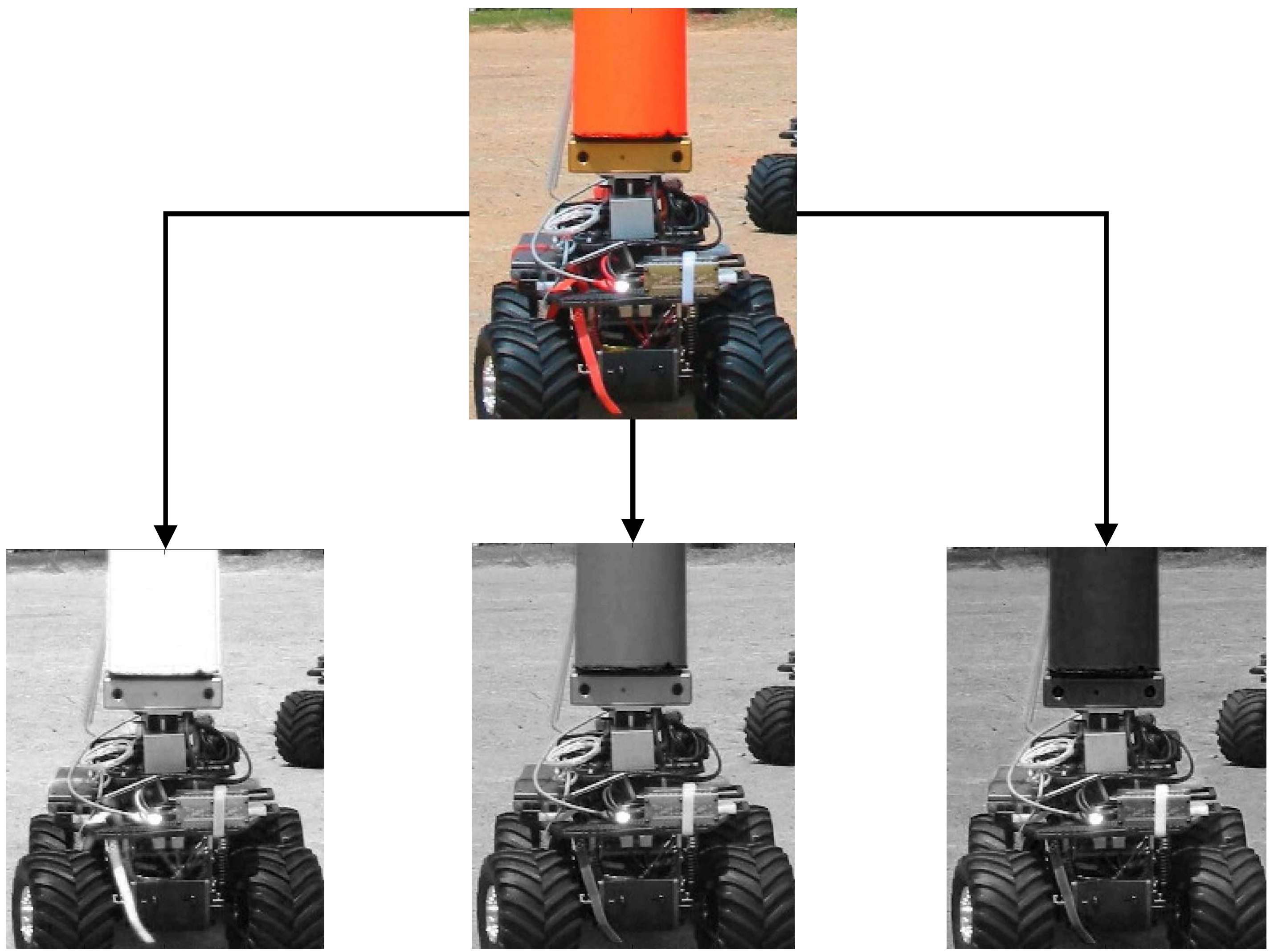


Planar



Packed

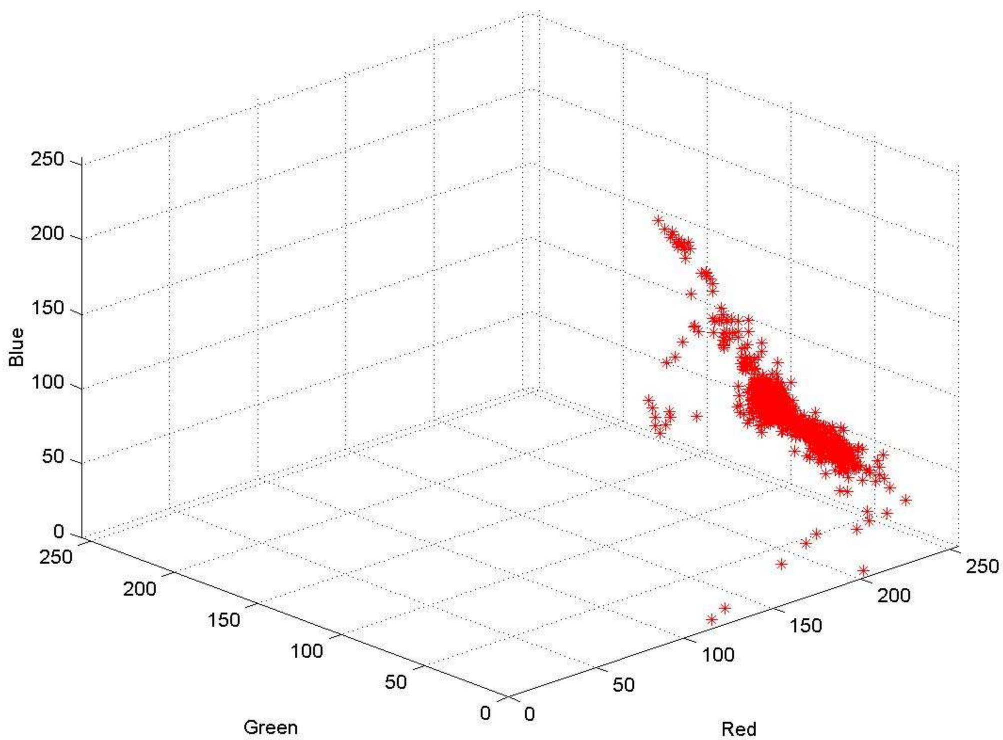
Representing Colors in an RGB Image



Red *Green* *Blue*

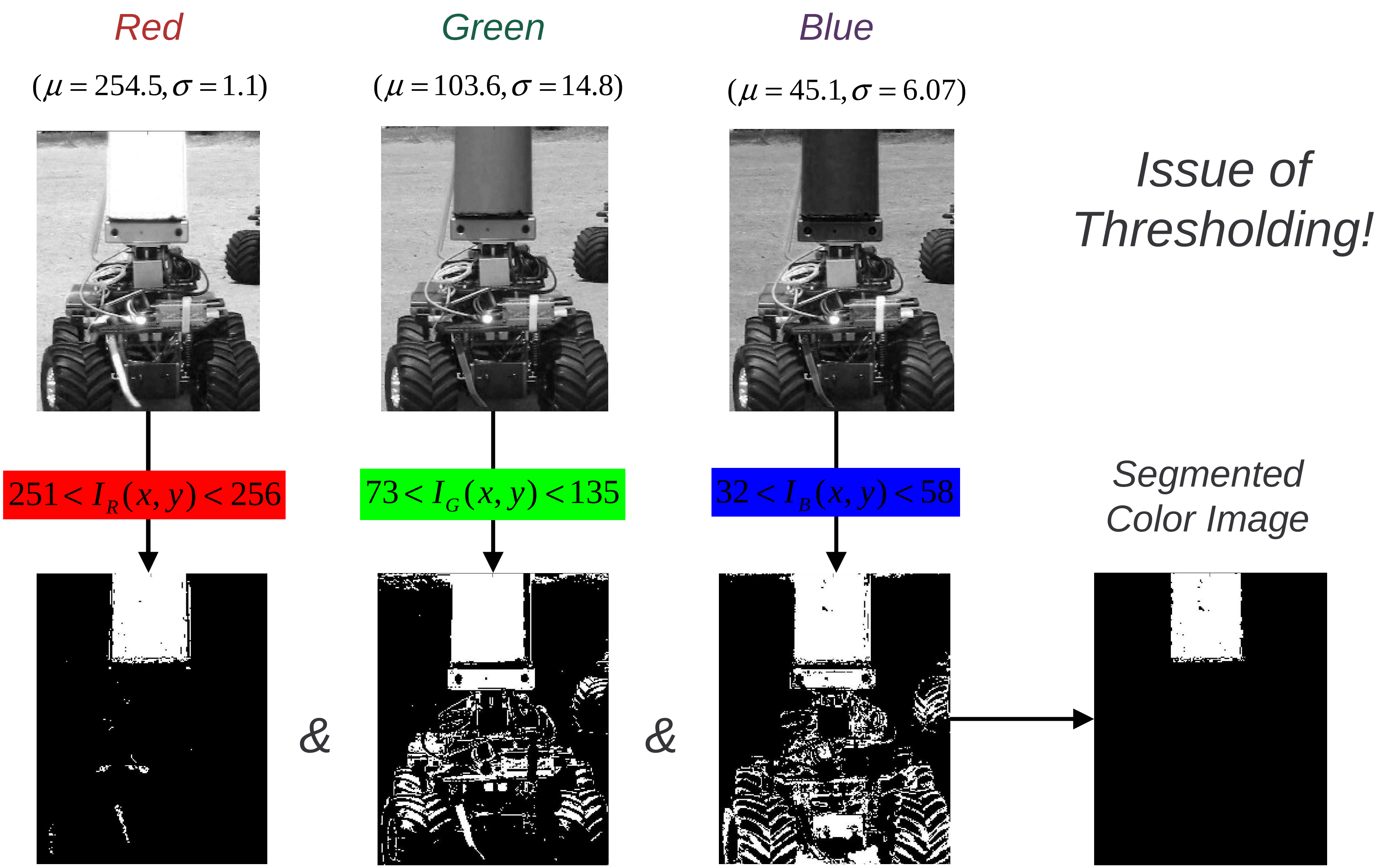
How do we segment a “single” color?

- We need to model is mathematically *a priori*
- In other words, the robot needs models of colors it is looking for in its memory



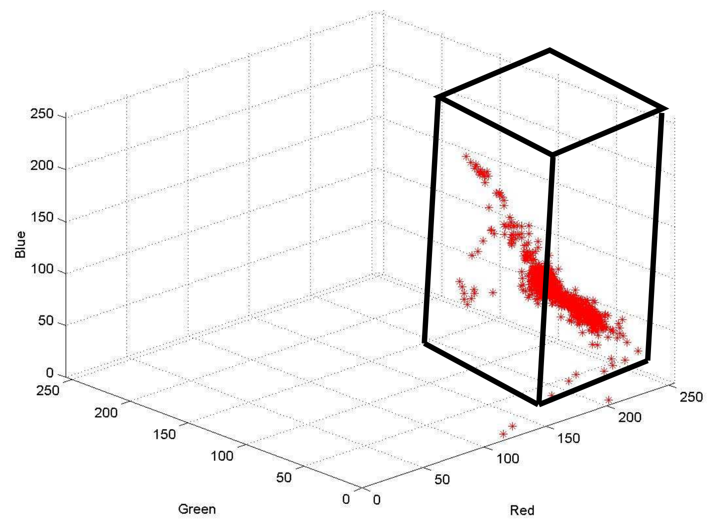
Sample set for orange hat

Simple RGB Color Segmentation



Segmentation Issues

- The approach surrounds the color with a prism
- This captures the color, but also many other colors that are not of interest
- Remember, each POINT represents a unique color



Implementation is Important!

- Recall that we “only” have a 567 MHz, so the implementation is important
- What’s wrong with the following code segment (the RGB pixel values are imR, imG, imB respectively):

```
if(imR<=rMax && imR>=rMin && imG<=gMax && imG>=gMin && imB<=bMax && imB>=bMin)
        x=1;
else
        x=0;
```

Conditional Branch is a control hazard!
Could result in a flushed pipeline!!!

- Better would be:

```
x = imR<=rMax && imR>=rMin && imG<=gMax && imG>=gMin && imB<=bMax && imB>=bMin;
```

- So the segmentation can be reduced to a series of logical operations

But we have Many colors to segment…

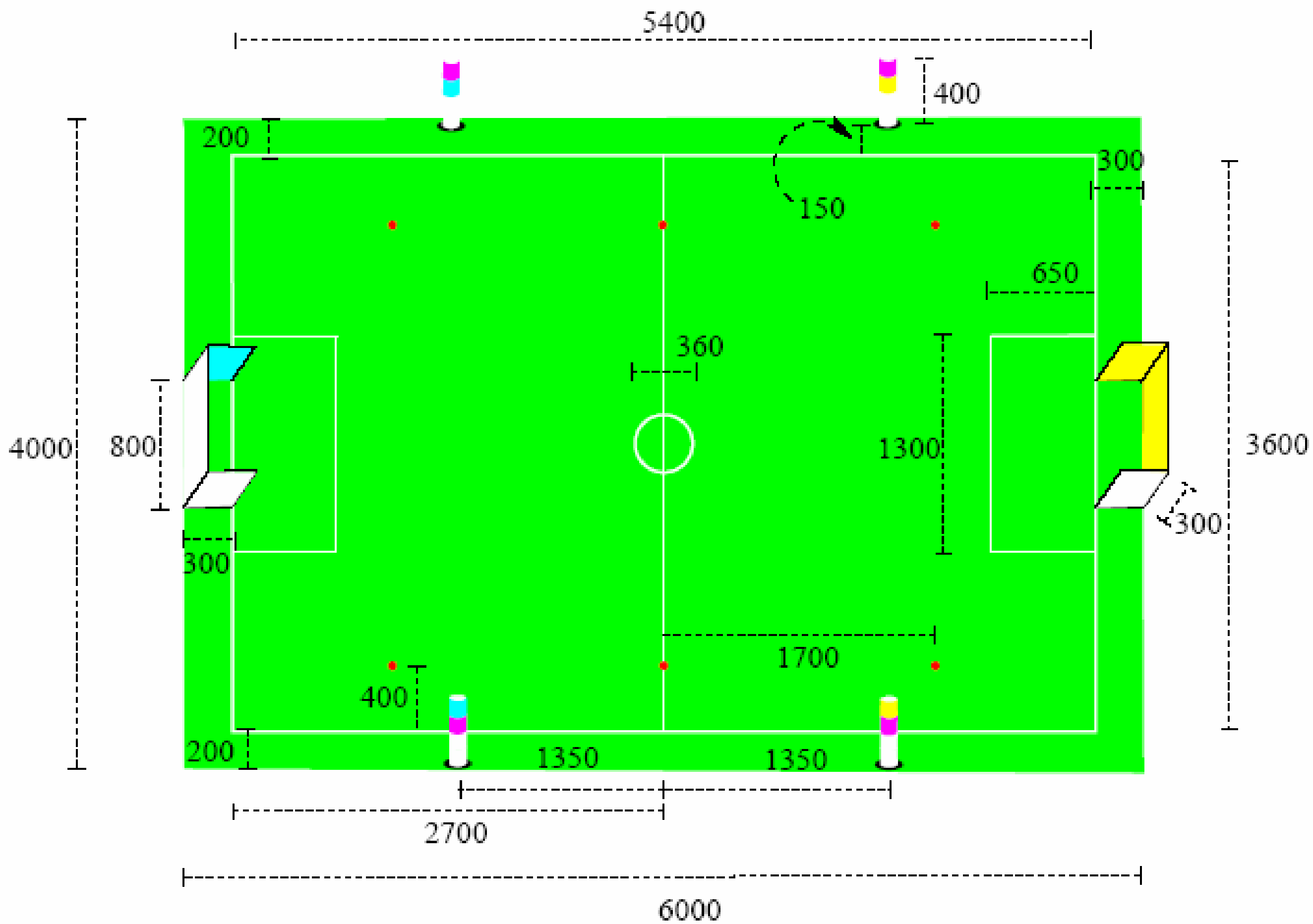


Figure 1: Field dimensions in mm.

* *www.robocup.org*

CMVision Color Segmentation

- James Bruce *et al*, IROS 2000
- The main ideas:
  - Use lookup tables (LUT) to store colors
  - Since color membership is based on binary logical operations, represent colors at the bit level
  - For an integer based LUT, this allows the segmentation of up to 32 colors in parallel
  - Since the LUTs are small, they will can be contained in the cache for improved performance

CMVision Color Segmentation (cont'd)

```
x = imR<=rMax && imR>=rMin && imG<=gMax && imG>=gMin && imB<=bMax && imB>=bMin;
```

- We want to convert this into a LUT. Assume for now that the pixel depth is 4 bits
- Let's say the valid range of colors for a ball are:

$$0 \le red \le 6$$
$$8 \le green \le 9$$
$$3 \le blue \le 15$$

- We can write these as the following LUTs:

```
int inRed[16] =     {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0};
int inGreen[16] =   {0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0};
int inBlue[16] =    {0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1};
```

CMVision Color Segmentation (cont'd)

- Now we can express

```
x = imR<=rMax && imR>=rMin && imG<=gMax && imG>=gMin && imB<=bMax && imB>=bMin;
```

as:

```
x = inRed[imR] && inGreen[imG] && inBlue[imB]
```

- This is the whole point of LUTs – increase speed at the cost of memory
- Notice that testing whether an image pixel is a member of a color requires only a single bit (0/1) representation
- Use this to embed multiple colors in the LUT and segment them in parallel

CMVision Color Segmentation (cont'd)

- Lets consider two colors:

```
int inRed1[16] =      {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0};
int inGreen1[16] =    {0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0};
int inBlue1[16] =     {0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1};
int inRed2[16] =      {0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0};
int inGreen2[16] =    {0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0};
int inBlue2[16] =     {0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0};
```

- We can combine these into a single LUT

```
int inRed[16] =       {1,1,1,1,1,3,3,0,0,0,0,0,0,0,0,0};
int inGreen[16] =     {0,0,0,0,0,0,2,2,3,3,0,0,0,0,0,0};
int inBlue[16] =      {0,0,0,1,1,1,3,3,3,3,3,3,3,1,1,1};
```

CMVision Color Segmentation (cont'd)

- Lets consider two colors:

```
int inRed1[16] =      {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0};
int inGreen1[16] =    {0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0};
int inBlue1[16] =     {0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1};
int inRed2[16] =      {0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0};
int inGreen2[16] =    {0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0};
int inBlue2[16] =     {0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0};
```

- We can combine these into a single LUT

```
int inRed[16] =       {01,01,01,01,01,11,11,00,00,00,00,00,00,00,00,00};
int inGreen[16] =     {00,00,00,00,00,00,10,10,11,11,00,00,00,00,00,00};
int inBlue[16] =      {00,00,00,01,01,01,11,11,11,11,11,11,11,01,01,01};
```

The first color is embedded in the LSB.

The next color is in the next bit

CMVision Color Segmentation (cont'd)

- Now we can express

```
x = inRed[imR] && inGreen[imG] && inBlue[imB]
```

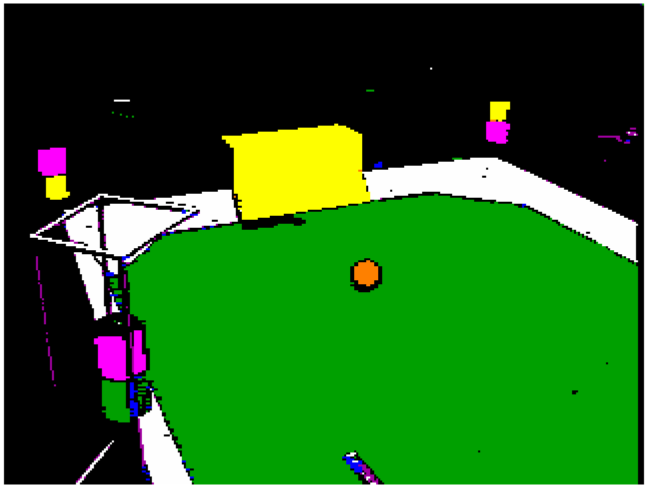
as:

```
x = inRed[imR] & inGreen[imG] & inBlue[imB]
```

- Note that the logical operations are now done at the BIT level
- Thus, we test a pixel against *n* colors (for an *n*-bit word) in parallel!
- The only negative is that since we are representing colors by prisms, it will be difficult to find that many that don't overlap.

CMVision Segmentation Example

Raw Image

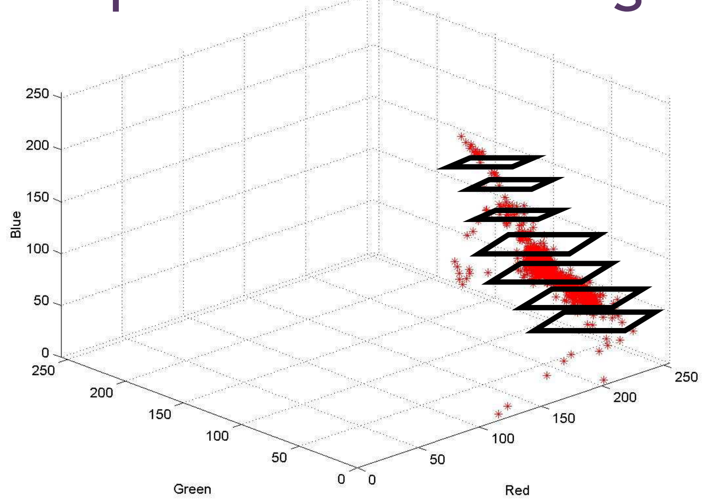


Segmented Image

* *http://www-2.cs.cmu.edu/~jbruce/cmvision/*

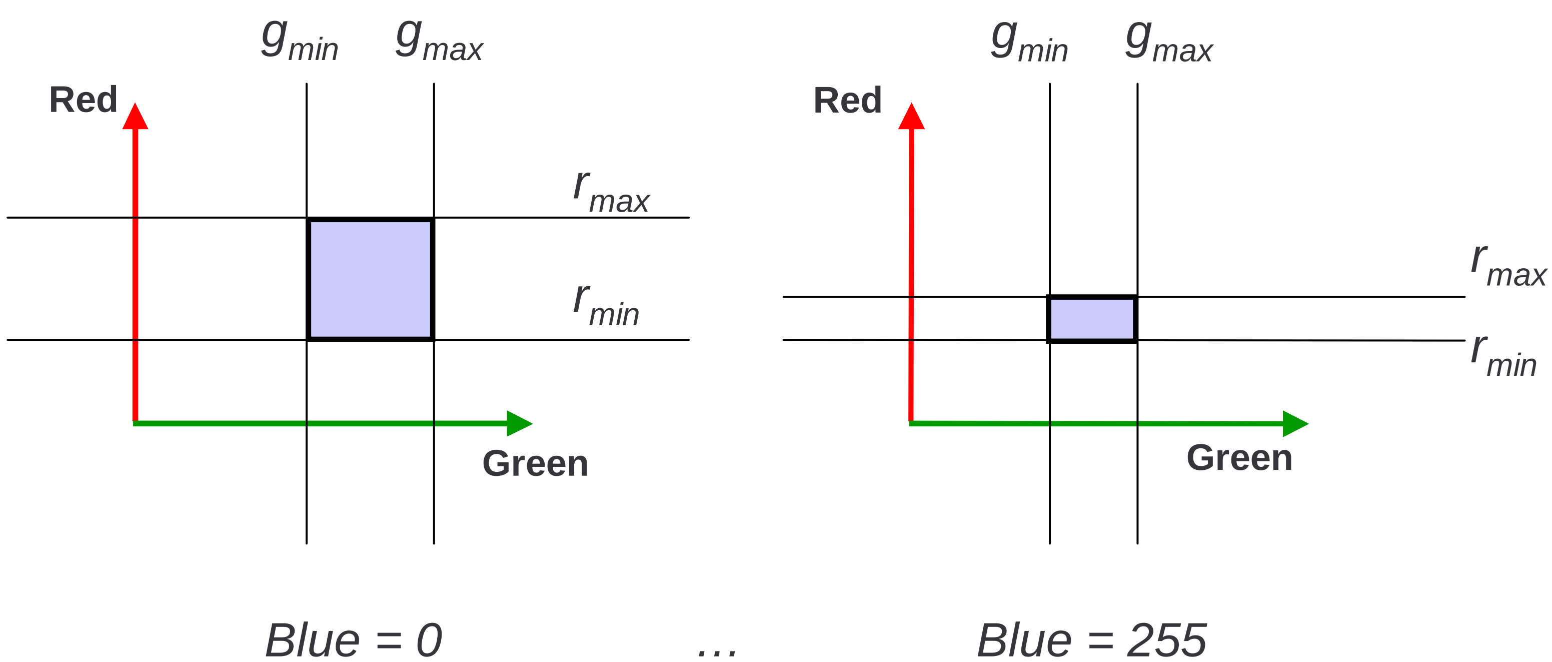
An Alternate Segmentation Approach 1

- Bound the color with a rectangle at a color/grayscale level
- Much less conservative in that it lets in less “invalid” pixels, but still conservative
- Fast implementations employ bit-based LUT to segment multiple colors in a single pass



A Layered Bounding Rectangle Approach

- Example: For each level of blue, bound the red & green levels from above and below:



2D LUT

- We will now have 2, two-dimensional LUTs:

```
int blueRed[16][16] =    {{1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0},...,
                          {0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0}};

int blueGreen[16][16] = {{0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1},...
                          {0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0}};
```

- Our test now becomes

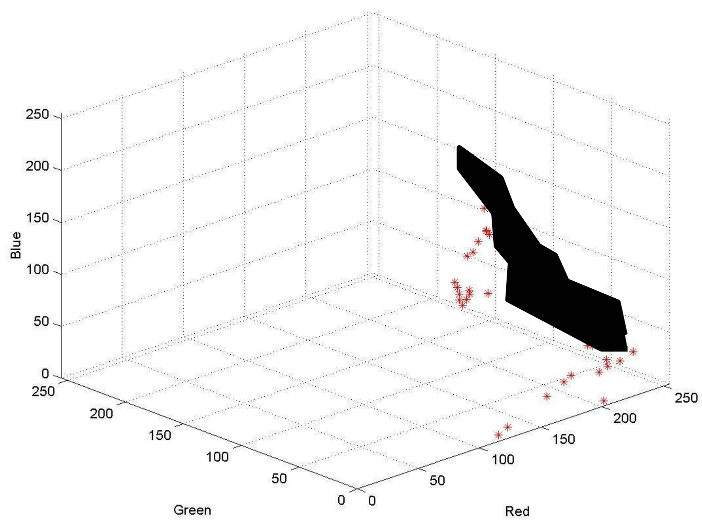
```
x = blueRed[imB][imR] & blueGreen[imB][imG]
```

  where we again use a bitwise representation for color membership

- Only negative is the growth of the LUT by *O(n)* – but still small enough to be very fast

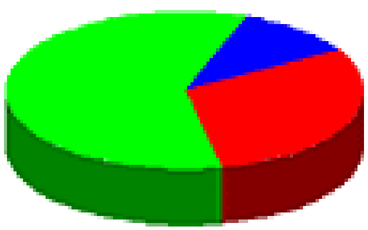
Alternate Segmentation Approach 2

- Bound the color with a three-dimensional solid
- Best color representation
- Requires a 3D LUT, which for even an 8-bit LUT depth is > 16 MB



YCbCr Color Space

- Human eye more responsive to brightness changes than color changes
- Separates *luma* ("brightness") from the *chroma* ("color") channels
- Basis for US television signal (related to YUV/YIQ formats)
  - Allows for the transmission of B&W images
- Image format for Aibos



"Greyscale"
*Y= 0.30*R+0.59*G+0.11*B*

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.082 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

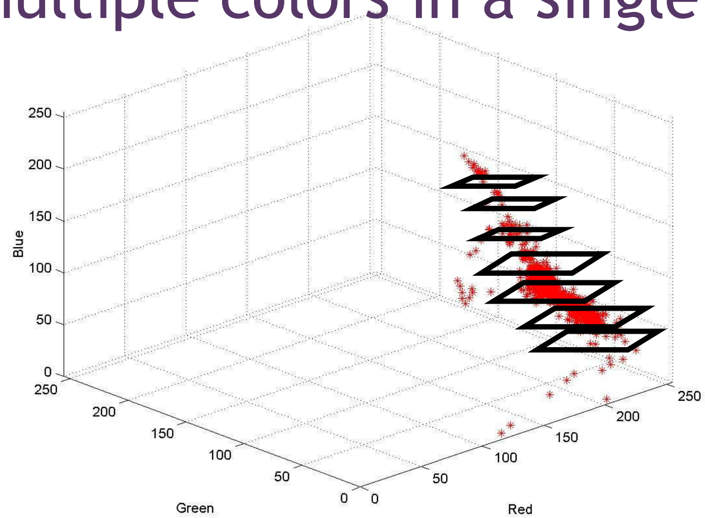
* One possible conversion.

YIQ Image Format

- Images can be either *planar* or *packed* format, but normally is packed
- Alternates U1->Y1->V1->Y2->U2->Y3->V2->Y4
- Every 2 Y pixels share a Cb and Cr
- Sub-sampled horizontally
- 4 bytes/2 pixels vs. 6 bytes for RGB24
- Separation of the luminance helps in color segmentation (sometimes)

An Alternate Segmentation Approach 1

- Bound the color with a rectangle at a color/grayscale level
- Much less conservative in that it lets in less “invalid” pixels, but still conservative
- Fast implementations employ bit-based LUT to segment multiple colors in a single pass



Summary

- Colors are easily segmented from images
- Need to be characterized *a priori*
- Color is the *perception* of reflected light in a scene
- Perception is strongly tied to illumination levels
- Formats of interest for us are RGB and YCbCr
- Often combined with other feature detectors for robust tracking
- Efficient implementation is important
- Tradeoffs between speed, memory use and accurate color representation: "There is no free lunch"

Next Time…

- Review of edge detection for line segmentation

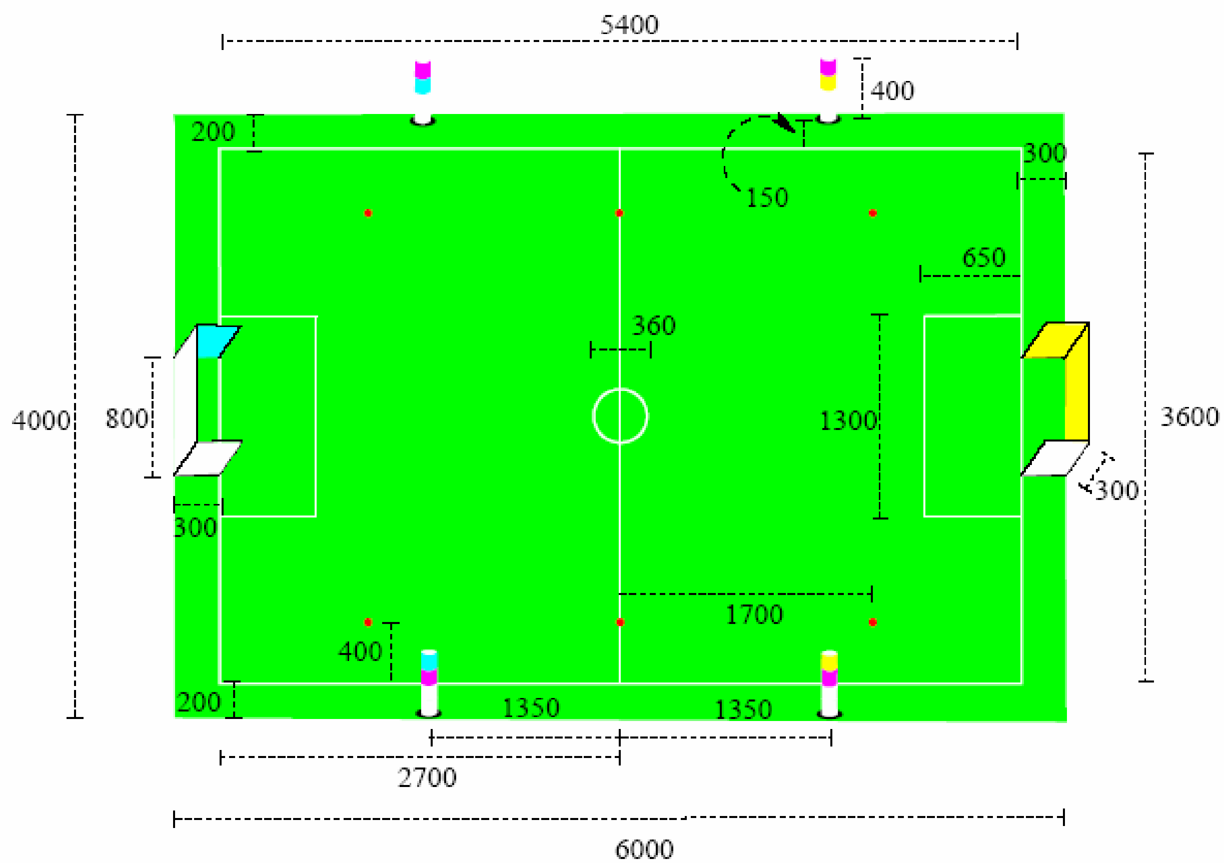


Figure 1: Field dimensions in mm.

* www.robocup.org