Vision - The Lower Levels

Carnegie Mellon University

October 1, 2003

Scott Lenser
James Bruce
Manuela Veloso

What is Vision?

- The process of extracting information from an image.
- Usually, this means identifying the objects contained in the image and their position relative to the camera, or
- The art of throwing out the information you don't want, while keeping the information you do want.

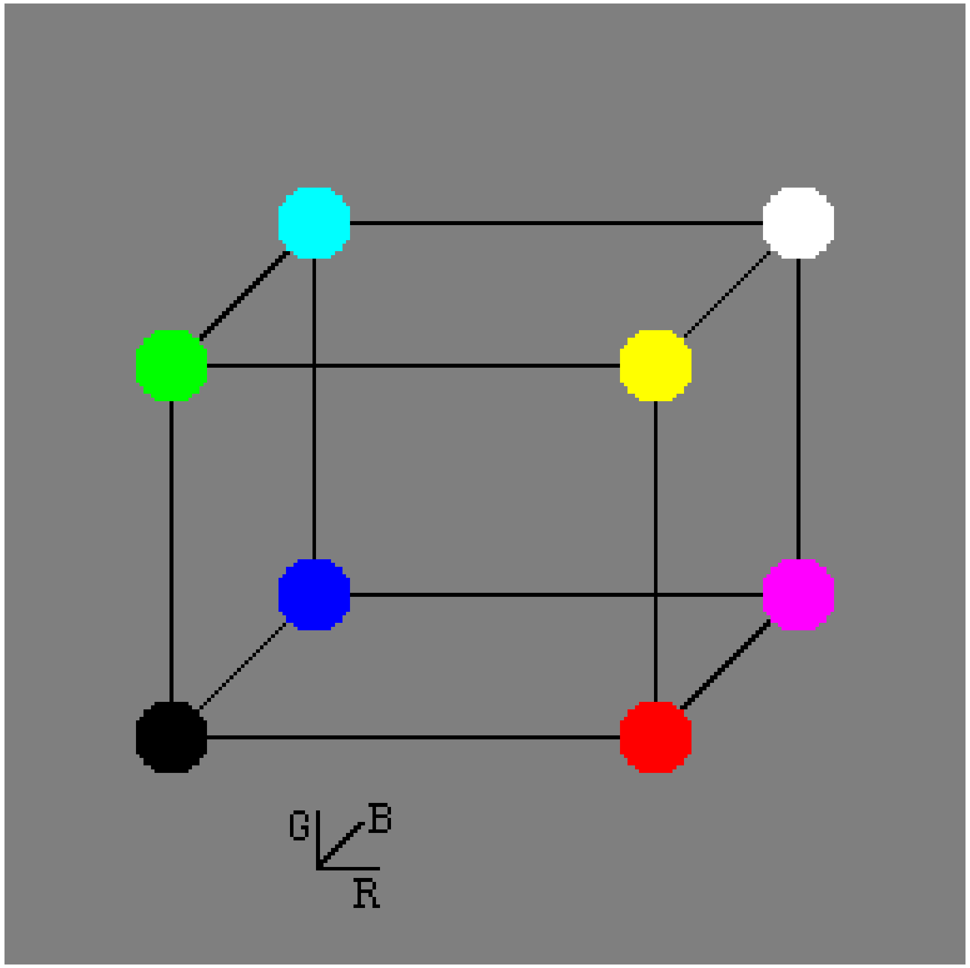
The camera in the AIBOs provides images at 176x144 pixels in the YUV color space.

Color Spaces

Each pixel has a 3-dimensional value. The dimensions of this value are often called channels. This value can be represented in many different ways. Three of the most popular representations are:

- RGB - R=red, G=green, B=blue
- YUV - Y=brightness, UV=color
- HSV - H=hue, S=saturation, V=brightness

Color Spaces - RGB



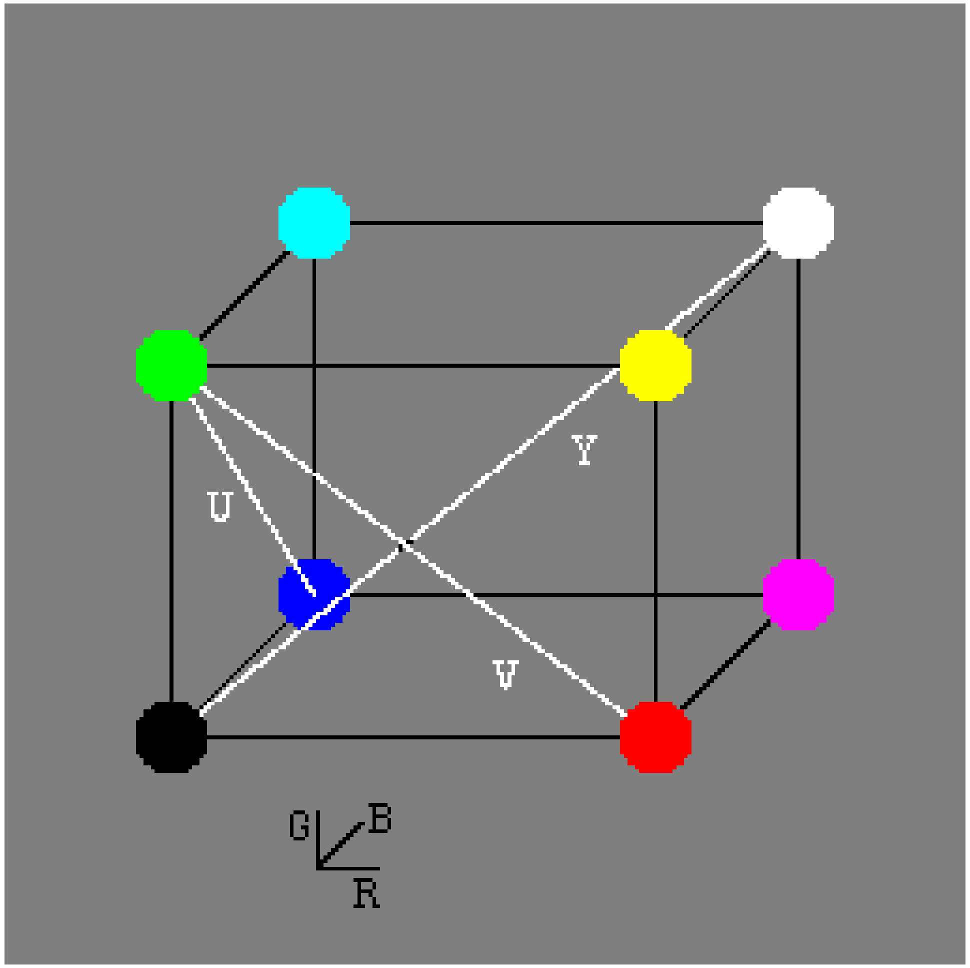
YUV

The AIBO's camera provides images in the YUV or YCrCb color space.

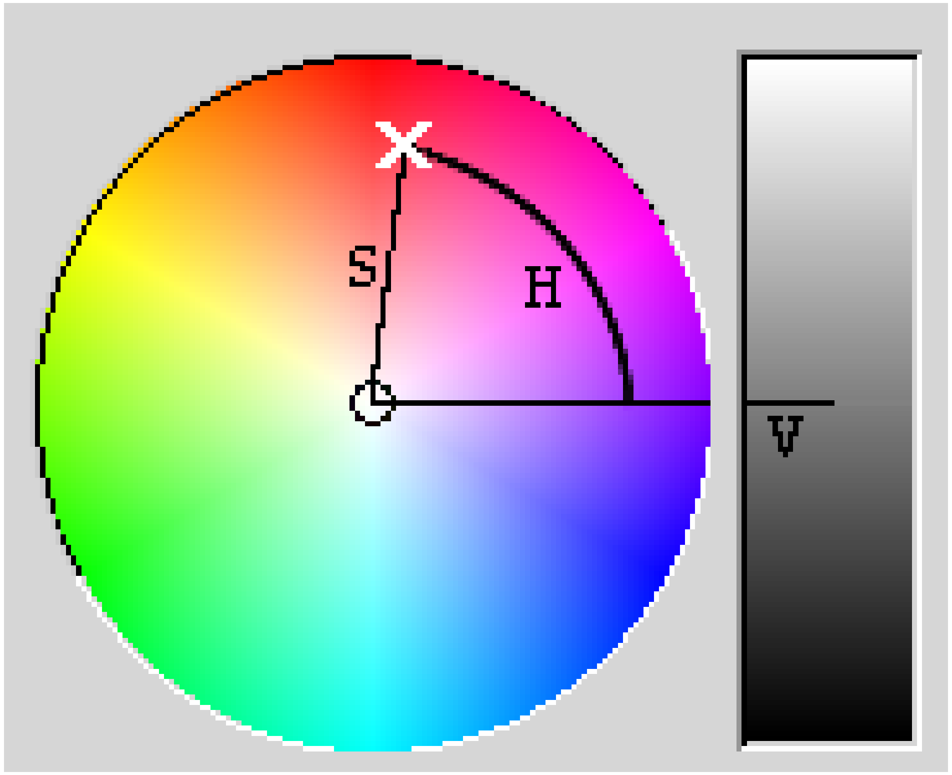
- Y - Luminance (brightness)
- U/Cb - Blueness (Blue vs. Green)
- V/Cr - Redness (Red vs. Green)

Technically, luminance and brightness, U and Cb, and V and Cr are slightly different but the differences seldom matter in robotics applications.

Color Spaces - YUV



Color Spaces - HSV



Circle is a slice of the color cube at a particular brightness.

Color Spaces - Discussion

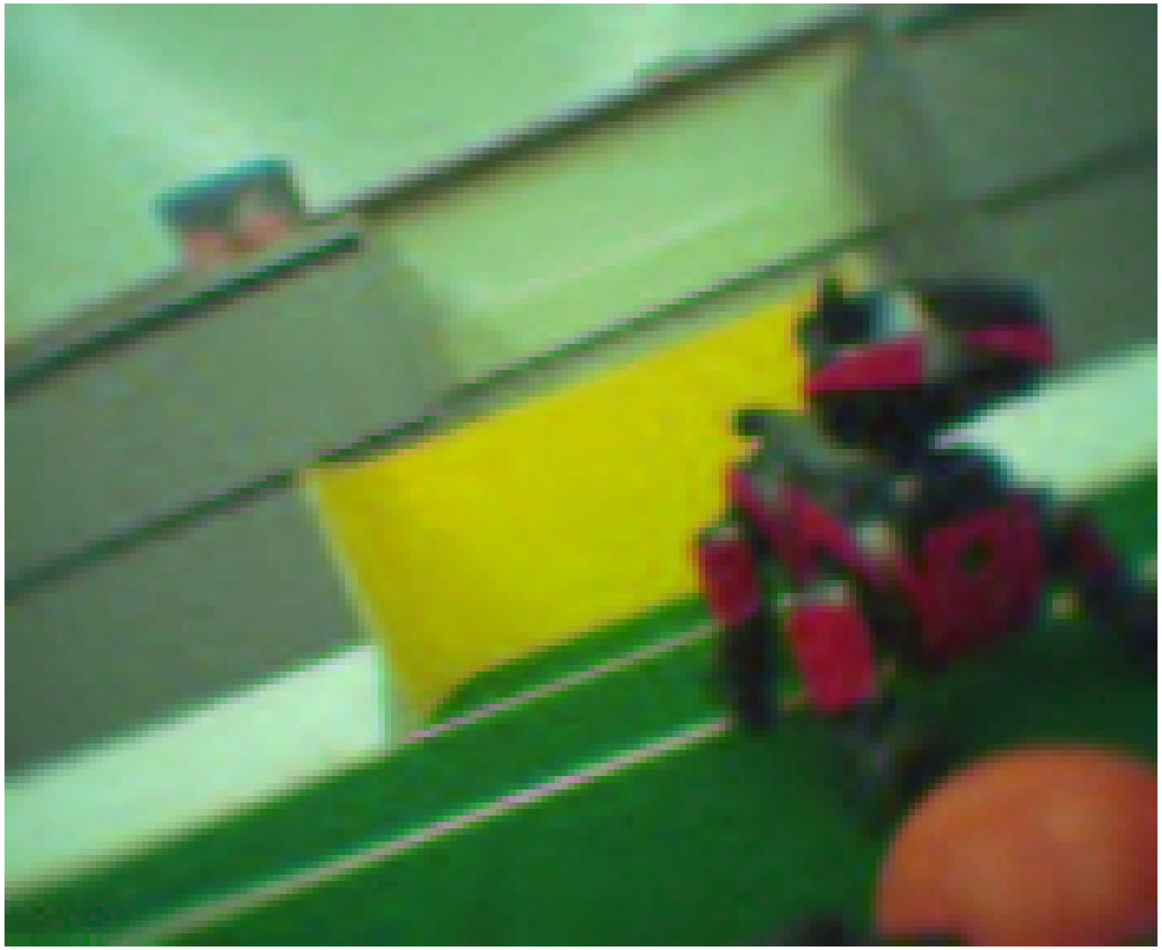
- RGB is provided natively by most image capture cards (frame grabbers). This is the color space used by computer monitors. Its chief disadvantage is that the individual channels.

- YUV is provided natively by many image capture cards. This is the color space used by TVs and JPEGs. One of the easiest color spaces to work with. Channels model human perception of images well. This is the color space we will work with from this point onwards.

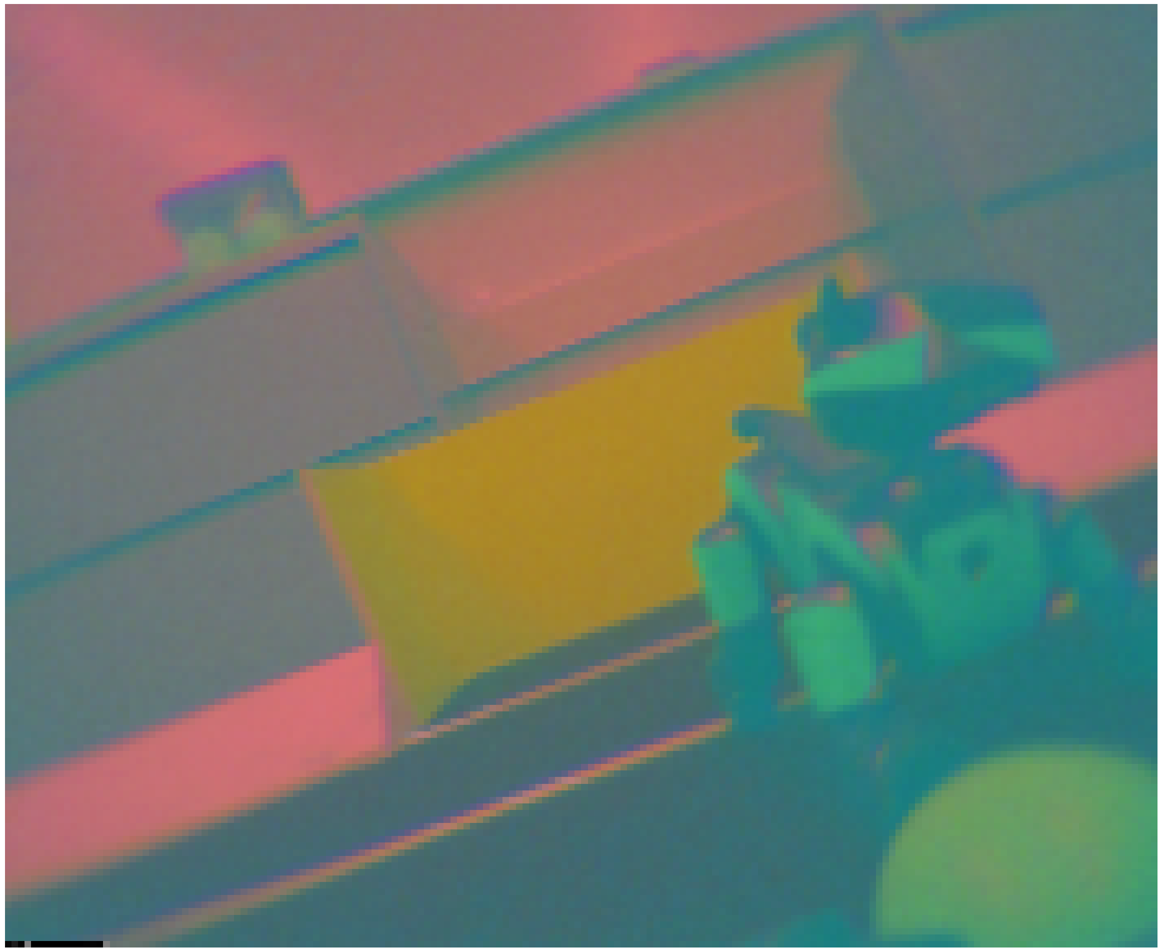
- HSV is almost never provided by image capture cards. Allows extremely simple thresholds to work ok. Numerically unstable for grey tone pixels. Computationally expensive to calculate.

- For more information, consult Poynton's Colour FAQ:
  http://www.engineering.uiowa.edu/~aip/Misc/ColorFAQ.html

Example Image - RGB

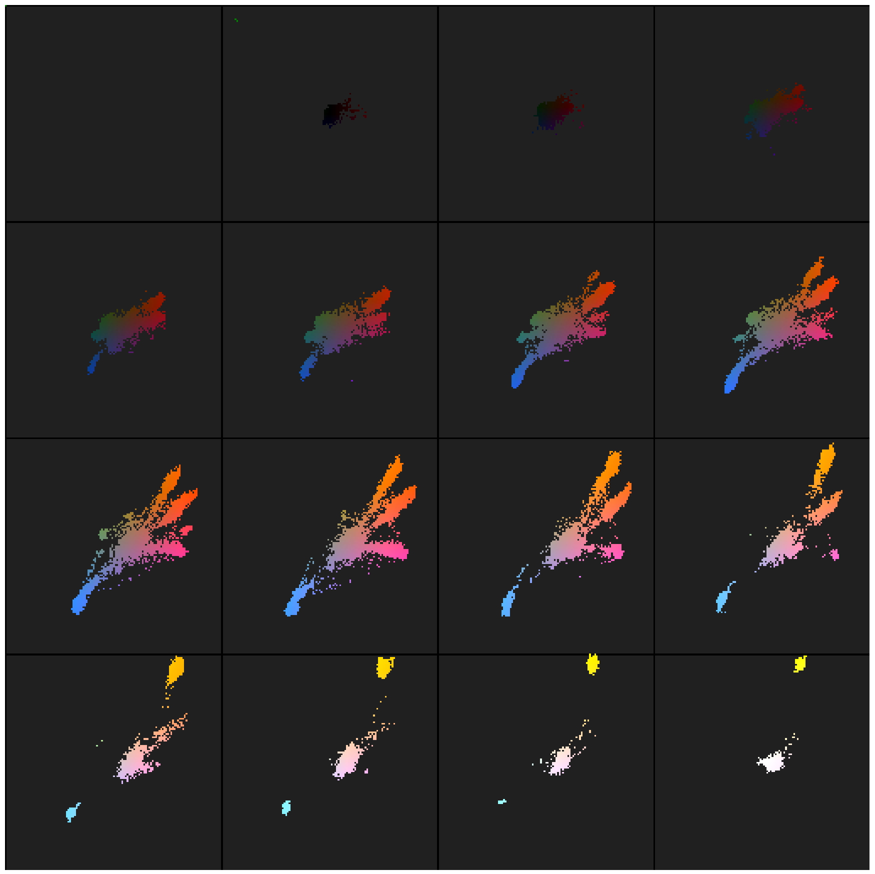


Example Image - Raw



R = Y, G = U, B = V

Color Image Histogram



Each square is a UV plane in YUV space. The upper left is $Y = 0..15$, with increasing Y ranges to the right, and then the next row. The lower right is $Y = 240..255$.

Vision Overview

Vision in CMPack is divided into two parts:

- low-level vision Performs bottom-up processing of image and provides summaries of important features of image.

- high-level vision Performs top-down processing of image. Uses expectations of objects that might be in image and features provided by low-level vision to find objects of interest and estimate their properties.

Low-level Vision Overview

Low level vision is responsible for summarizing the important features of the image. We will cover the color segmentation approach used by CMPack in detail. It consists of the following main stages:

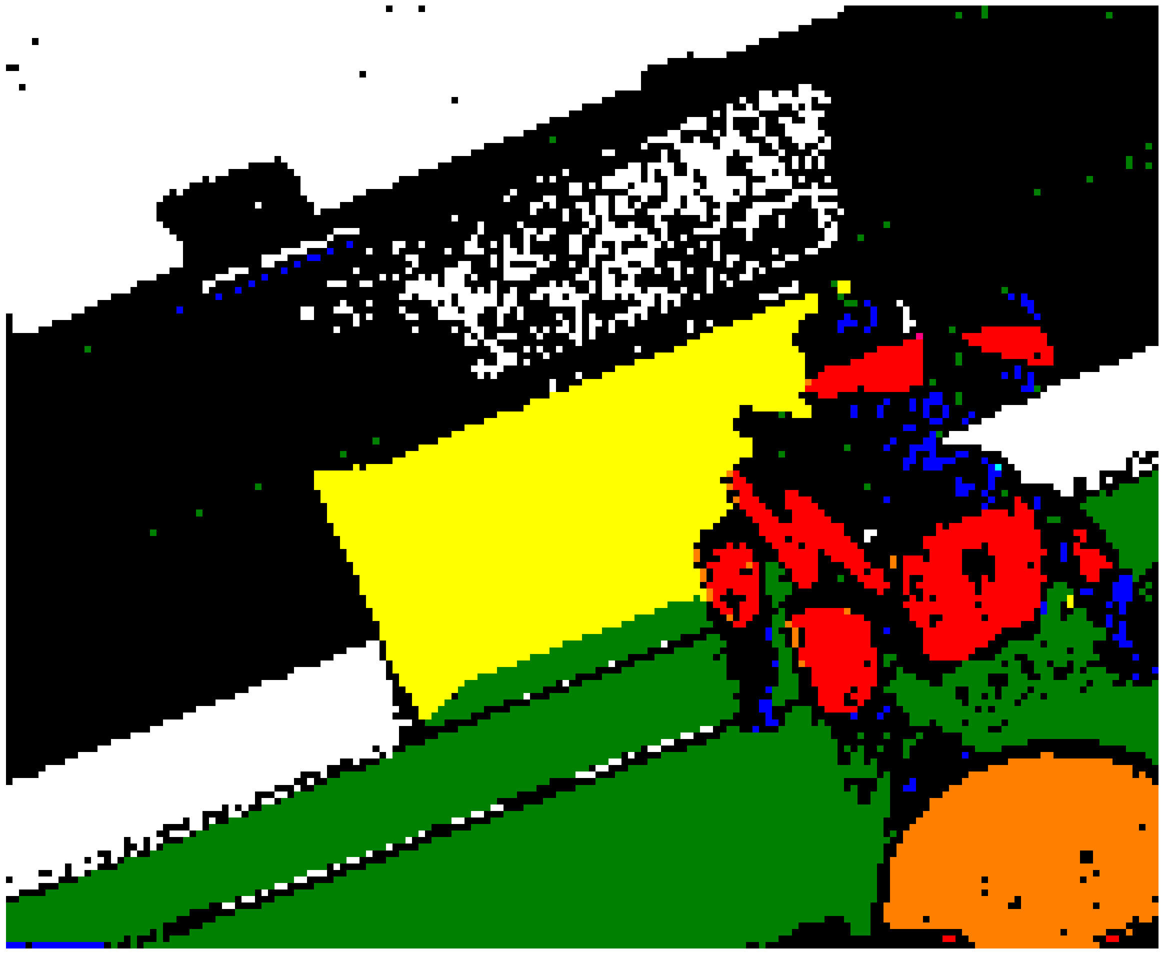
- Segment image into symbolic colors.
- Run length encode image.
- Find connected components.
- Join nearby components into regions.

Each stage reduces the amount of information that will be processed further. The output of all stages is available to the high-level vision so that the high-level vision can access the exact level of detail needed to perform its task.

Color Segmentation

- The object of color segmentation is to map from raw camera pixels to a member (c) of the class of symbolic colors (C), i.e. to create a function $F : y, u, v \rightarrow c \in C$.
- This reduces the amount of information per pixel from 256^3 to $|C|$. $|C|$ is 9 for our current system.
- This step reduces the amount of information that needs to be processed further roughly by 1.8M times.

Goal of Segmentation

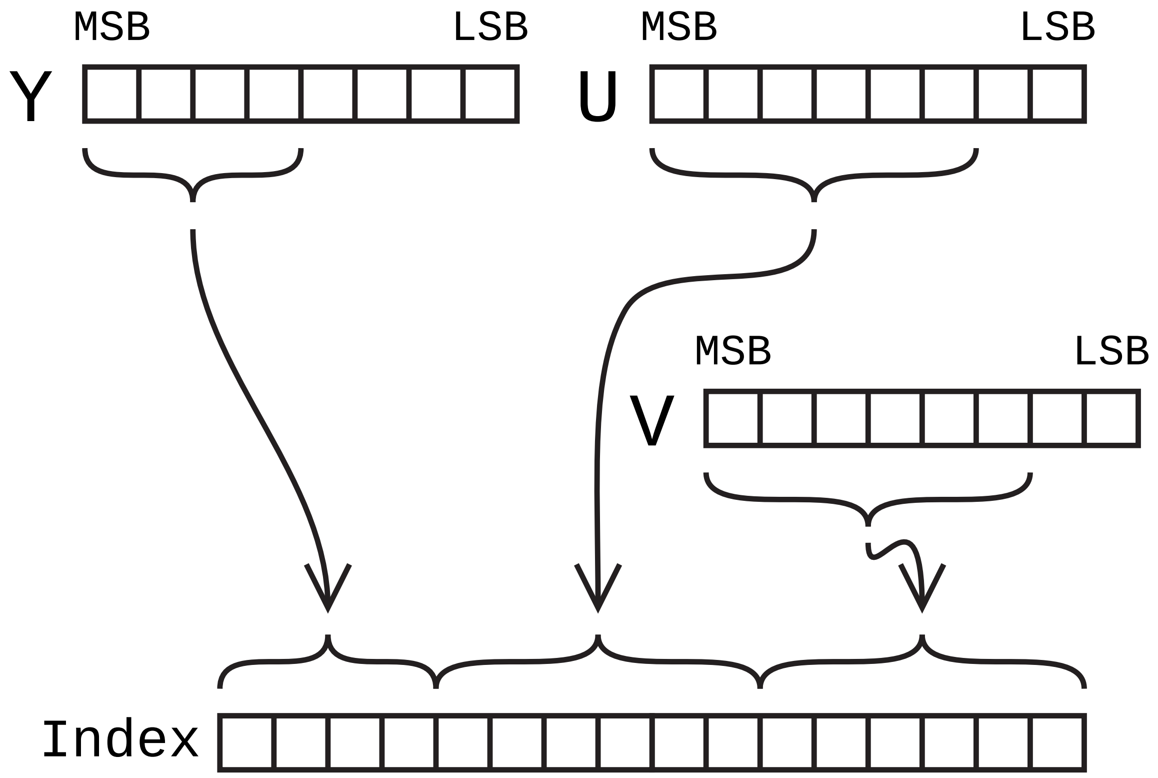


Color Class Thresholds

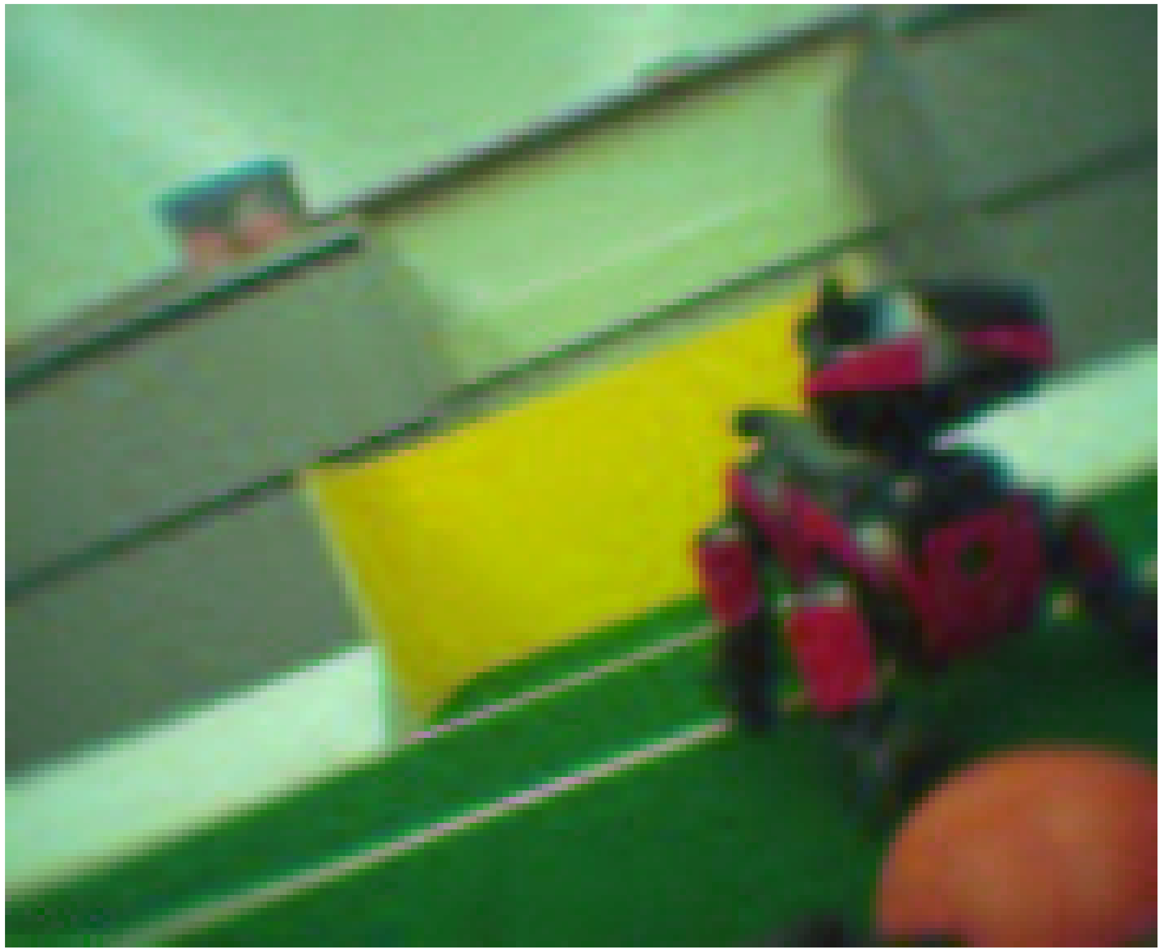
- Create mapping function from YUV color to symbolic color class.
- The map is a 64KB lookup table using sub-sampled YUV color as index.
- The index uses 4 bits of Y, 6 bits of U, and 6 bits of V.
- Each entry of the table has the symbolic color class (either a color or miscellaneous background color).

Index Calculation

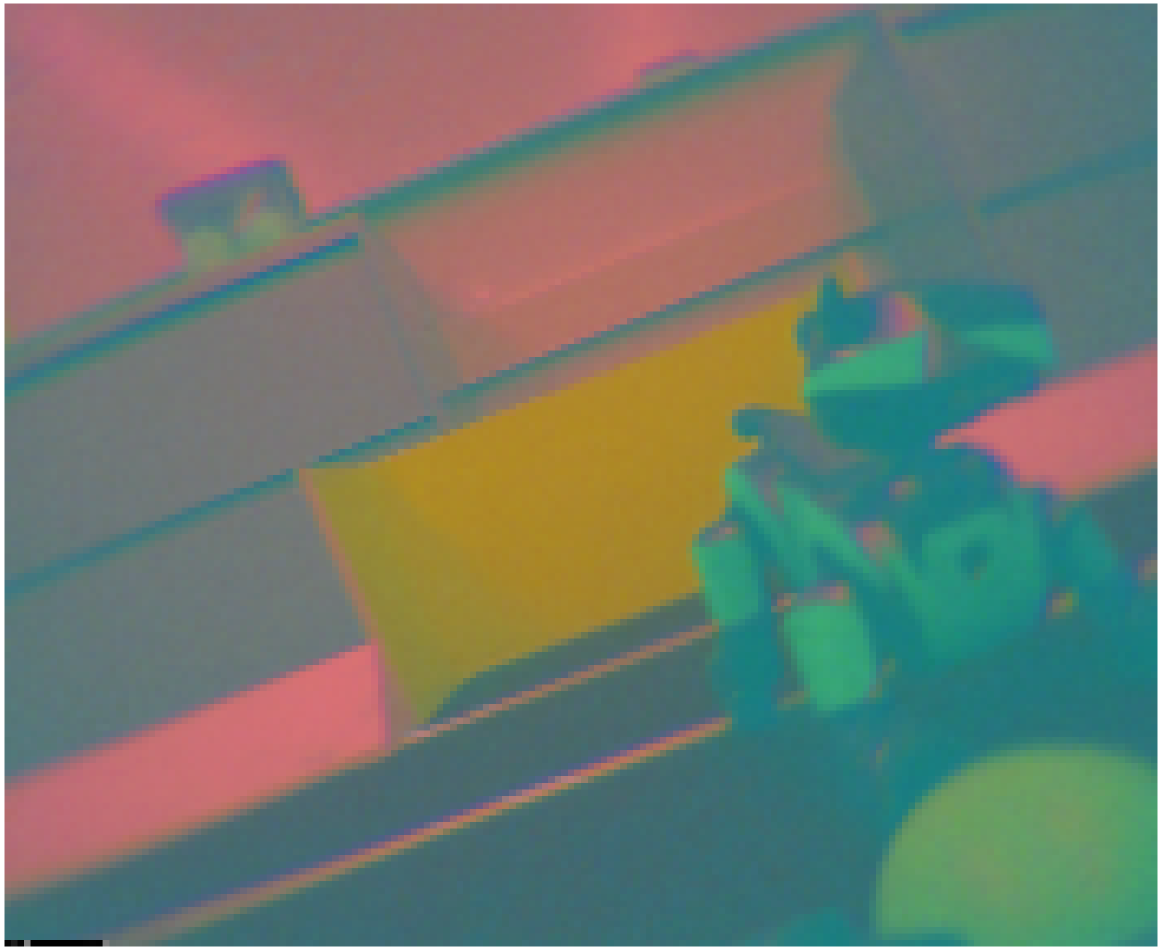
Graphically, the calculation of the index for the color class thresholds look like this:



Example Image - RGB

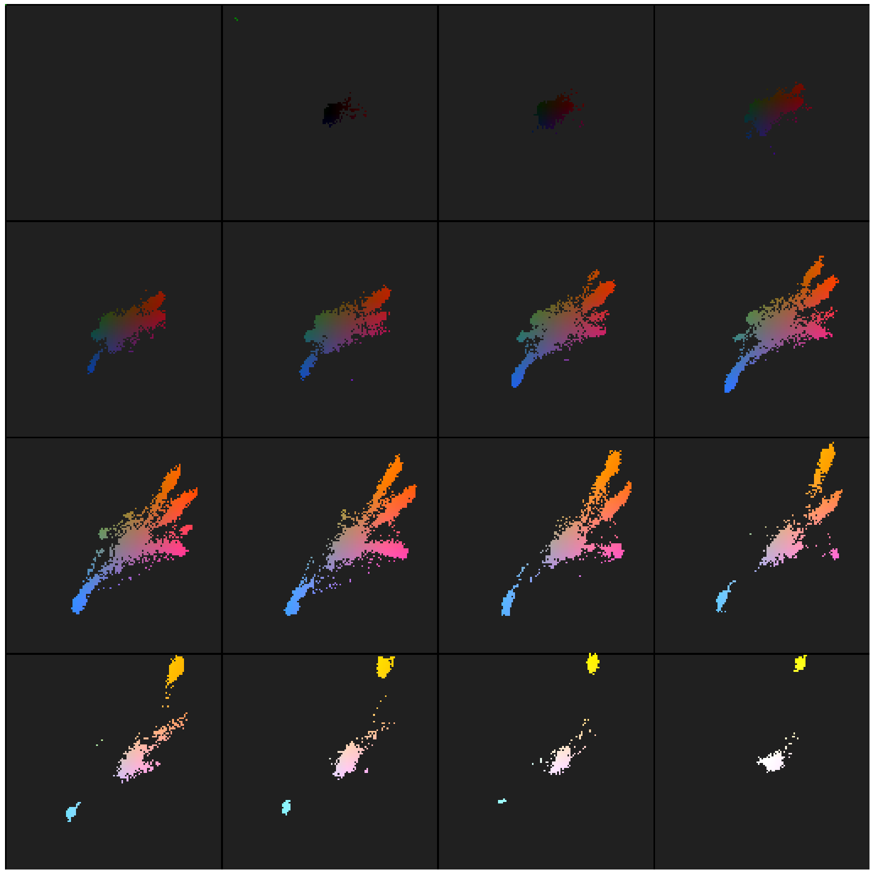


Example Image - Raw



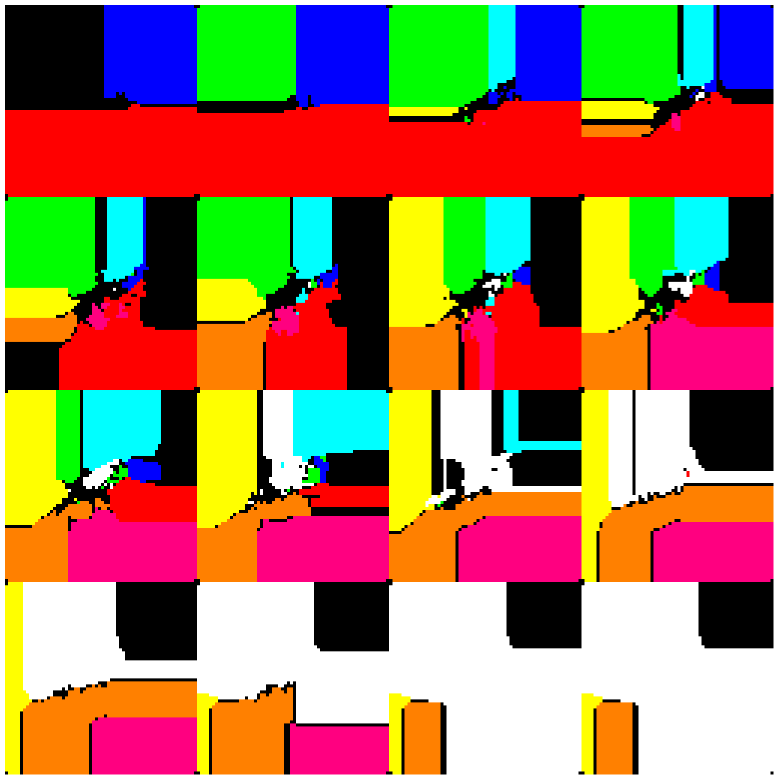
R = Y, G = U, B = V

Color Image Histogram



Each square is a UV plane in YUV space. The upper left is $Y = 0..15$, with increasing Y ranges to the right, and then the next row. The lower right is $Y = 240..255$.

Color Class Thresholds



This is just like the color histogram, but each location shows the class that a pixel at that location in the color space will be classified as. In this view, each class has a representative color.

Color Class Thresholds

The threshold table is generated using the following steps:

- Take example images with camera.
- Hand label the correct color class of each pixel for classification.
- Learn threshold map from YUV color to symbolic color.

Color Threshold Learning - Overview

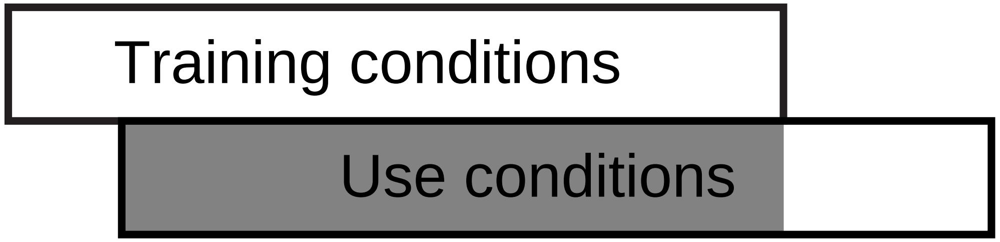
- Objective is to take about .8M hand-labelled example pixels and use them to label 64K grid cells in threshold table.
- Need a model for influence of each of the .8M examples on each grid cell.
- Influence model should have each example influence multiple grid cells. This improves the generalization of the learned thresholds.
- Need to perform .8M * 64K = 102.4B influence calculations in general case. Need to find efficient way to process examples and cells.

Why Generalization is Needed

Generalization is needed because:

- Training examples never cover all of the possible variations in lighting conditions within one environment.
- Training examples are never taken in exactly the same environment as where the thresholds are used.

This is represented graphically below:



Grey section shows correctly classified pixels without generalization.

Color Threshold Learning - Overview

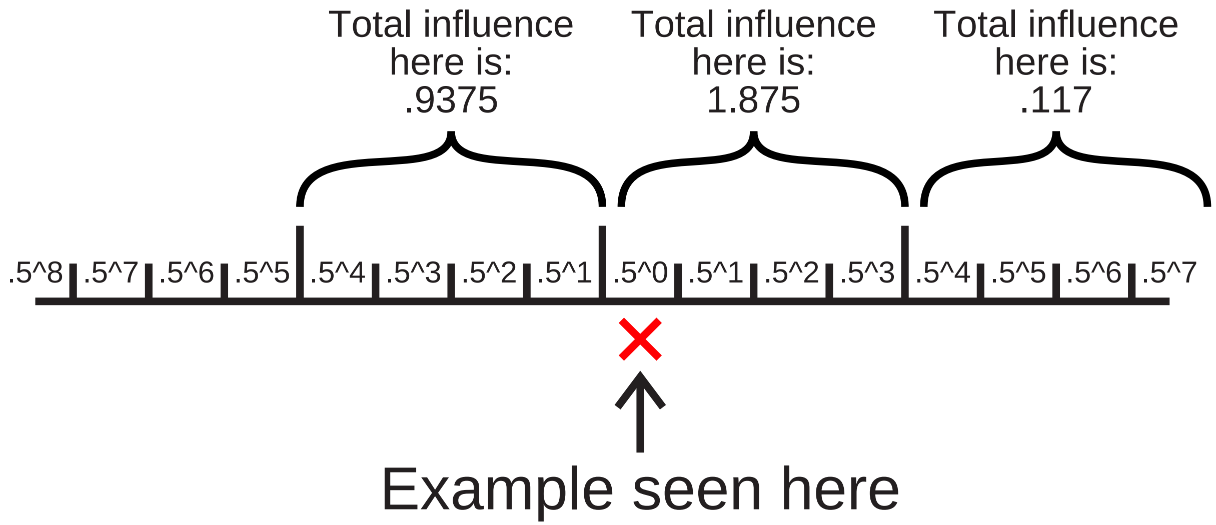
Our approach:

- Allow samples to potentially influence every cell for maximum generality.
- Stop generalization only when negative examples are detected.
- Use exponential model to simplify calculations.

Exponential Model

The weight due to each sample is a sum of terms of the form α^d where d is the Manhattan distance in YUV space from the example to the location within the grid cell. α controls the strength of generalization. We use $\alpha = .5$ which strongly emphasizes local examples.

The model is shown graphically in one dimension below:


Total influence here is: .9375
Total influence here is: 1.875
Total influence here is: .117
.5^8
.5^7
.5^6
.5^5
.5^4
.5^3
.5^2
.5^1
.5^0
.5^1
.5^2
.5^3
.5^4
.5^5
.5^6
.5^7
Example seen here

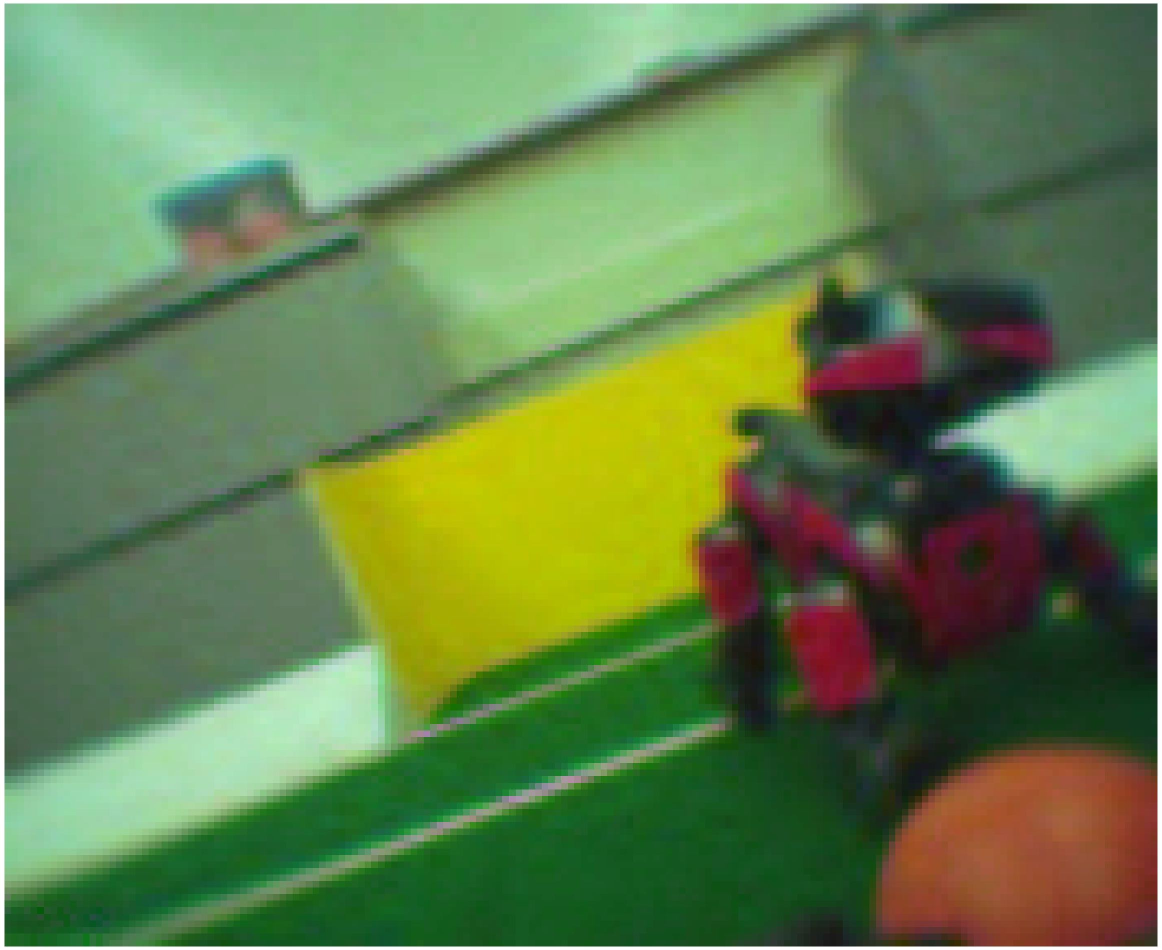
Color weights

- The total weight for each color class (every color and the background class) is calculated for each cell.
- Each color class is calculated independently.

Decision Rule

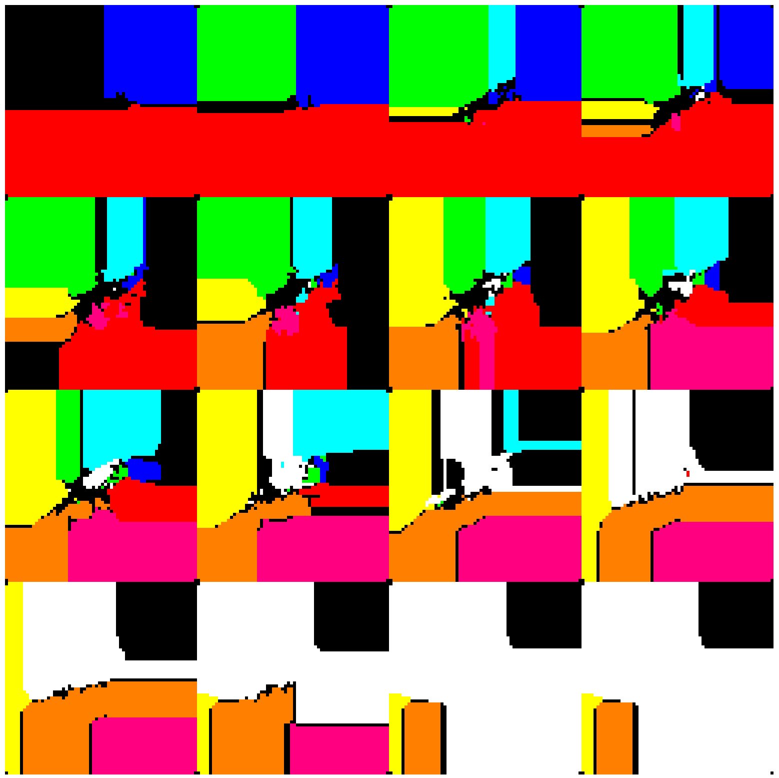
- The color class for each cell is determined by looking at the weight totals for each cell separately.
- Each color class weight is multiplied by a constant M_c. This parameter allows for the correction of bias introduced by a high/low number of examples of a particular color.
- The total weight for the cell is calculated w_t.
- The color class with the most weight is chosen for the cell if $\frac{w_c}{w_t} \geq C_c$. The parameter C_c is a confidence threshold which can be used to adjust the relative frequency of false positives and false negatives.
- If no color class passes this threshold, the cell is labelled with the background color class.

Example Image - RGB



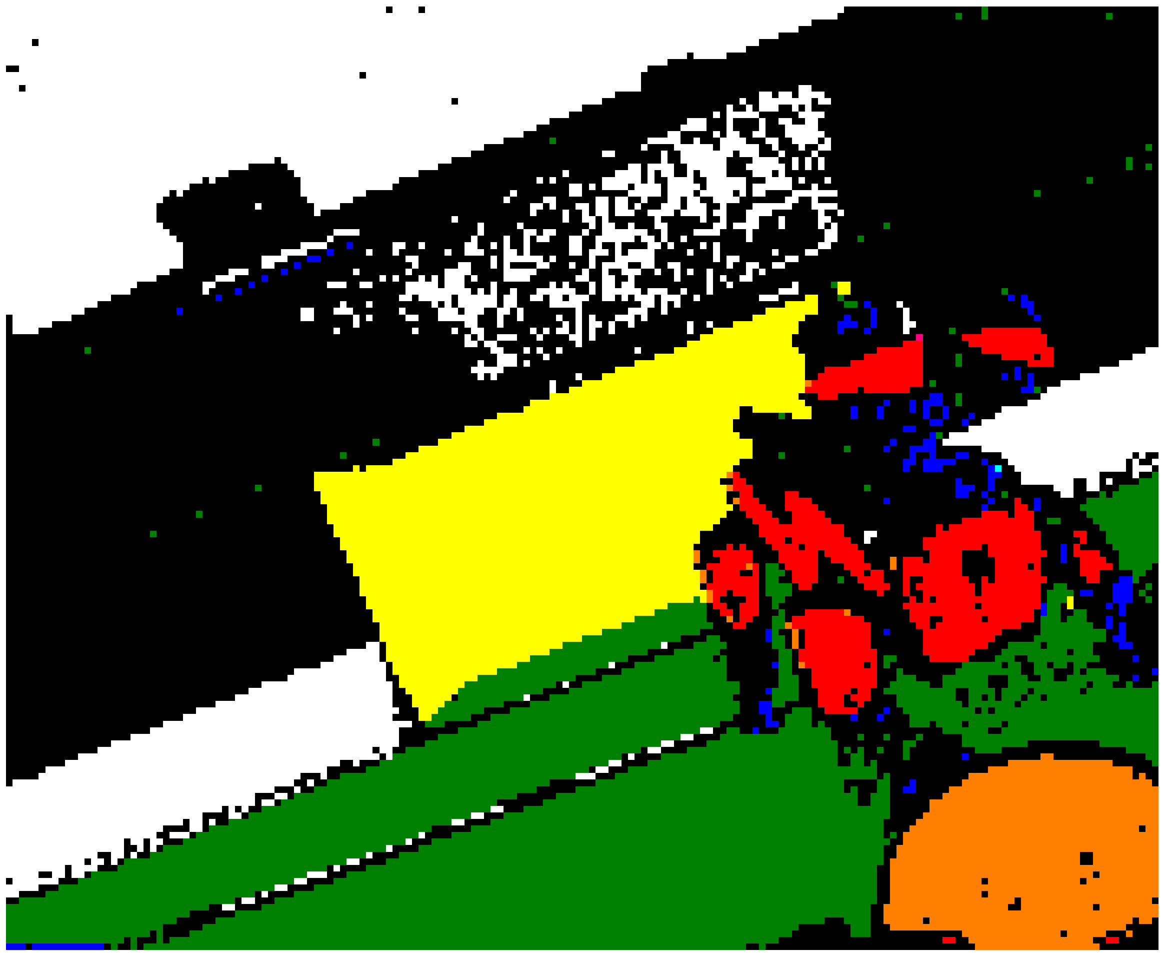
Example Image - Raw

Color Class Thresholds



This is just like the color histogram, but each location shows the class that a pixel at that location in the color space will be classified as. In this view, each class has a representative color.

Example Image - Classified



Color Segmentation - Limitations

Color segmentation provides a lot but there are some things it can't do:

- Correctly segment YUV pixels that have the same value into different symbolic color classes.
- Generate smoothly contoured regions from noisy images.

Color Segmentation - Abilities

Color segmentation provides the following abilities:

- Quickly eliminate a vast quantity of mostly redundant information.
- Provide a representation of the image that is well suited for further processing.
- Differentiate between YUV pixels that have very similar values.

Regions

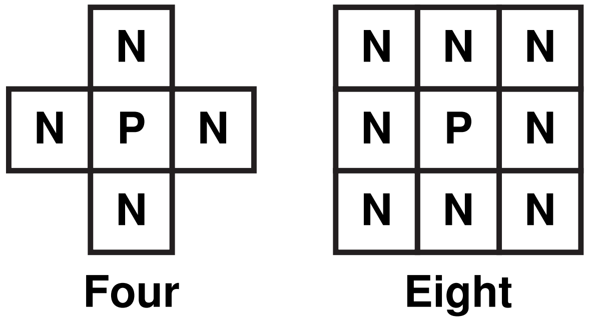
All we really have now is pixels with a guess at which color they might be.

We need a notion of regions. In the case of a class thresholded image these are connected areas of the same color.

Connectedness

Forming regions has to start with a notion of connectedness. This defines what a region contains and where it stops. If two neighboring pixels have the same color, we consider those to be in the same region.

- Four connectedness: The neighbors are the pixels either directly above or directly to the side of a pixel.
- Eight connectedness: The neighbors are all pixels that touch the center pixel, including the diagonally adjacent pixels.



The choice between the two is somewhat arbitrary. We use four connectedness on our robots.

Run Length Encoding

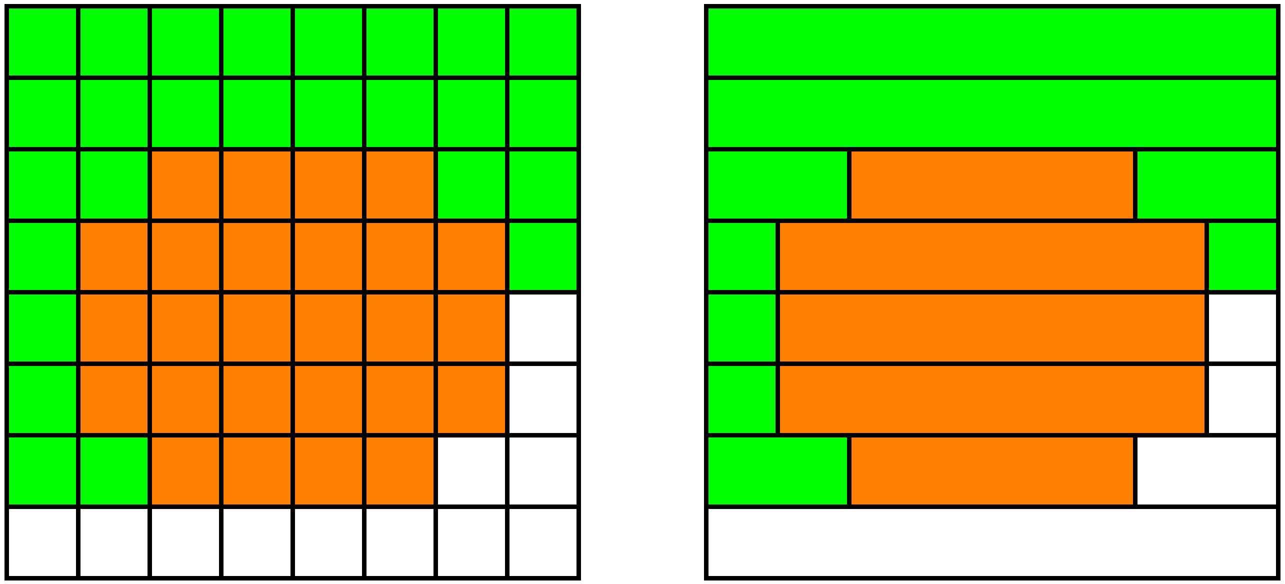
We run length encode (RLE) our classified image before extracting connected regions.

In an RLE image, instead of storing values for each pixel (val), we store a value followed by the number of repeat values $(val, length)$. Each run also stores the x,y location on the image at which it starts. Background colored runs are not encoded to save processing time in later steps.

This step reduces the amount of information required to represent an image down from 203K down to about 3K, a decrease of about 67 times. This drastically reduces the amount of processing needed in further steps.

RLE and Connected Regions

Compressing runs also has the nice side benefit of handling left and right connectedness, so we only need to worry about vertical connections when building regions.



Left: Original image, Right: RLE version of image

Union Find

The name of the algorithm we want for finding connected regions is Union Find.

- Find refers to the operation of finding the name of the region that a particular run is a part of.
- Union refers to the operation of joining two regions into one larger region.

Tree-Based Union Find

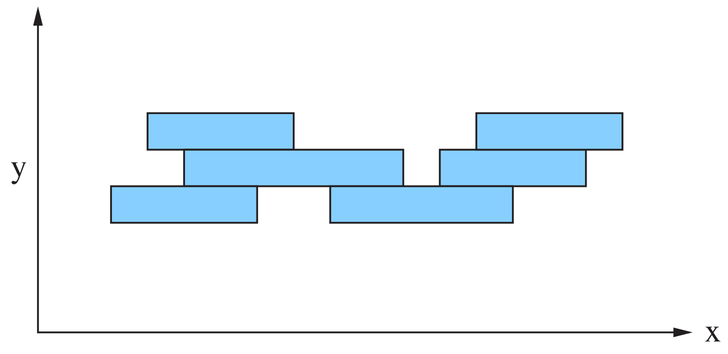
The unique region IDs are the uppermost, leftmost run in a region. Call this the representative element of a region.

Each run points to its “parent”, which is another run in the same region above it in the image.

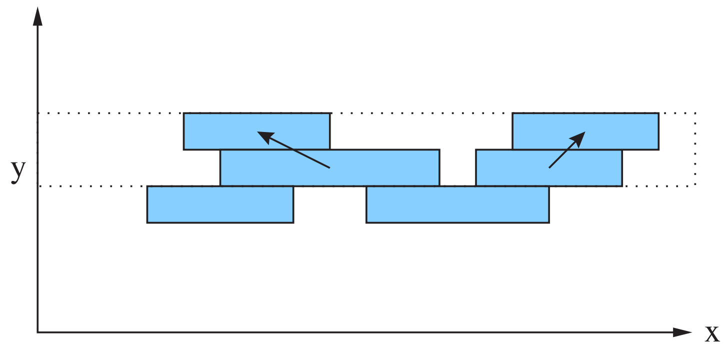
To find out the ID of a run, we follow up the parent pointers until we reach a root node with no parent, which is the representative element. This is the find operation.

To merge two regions when we detect overlap, we run a find on both regions, and set one of the representative elements parent to point to the other. Now all find operations on either region will reach the same representative element. This is the union operation.

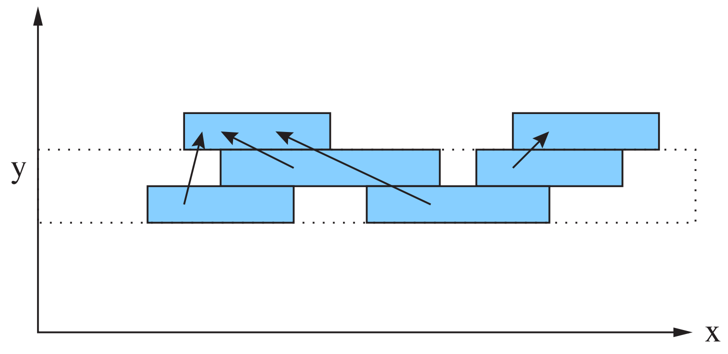
Tree-Based Union Find



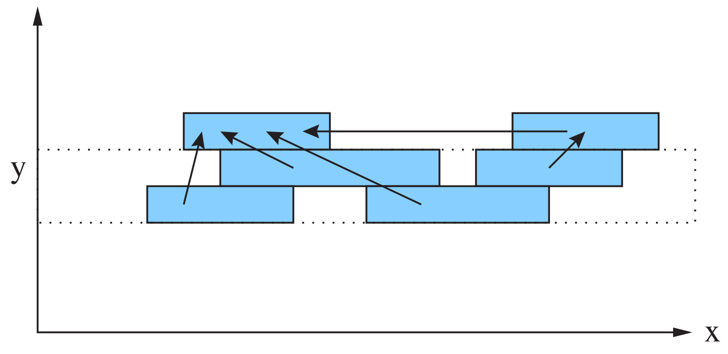
1: Runs start as a fully disjoint forest



2: Scanning adjacent lines, neighbors are merged



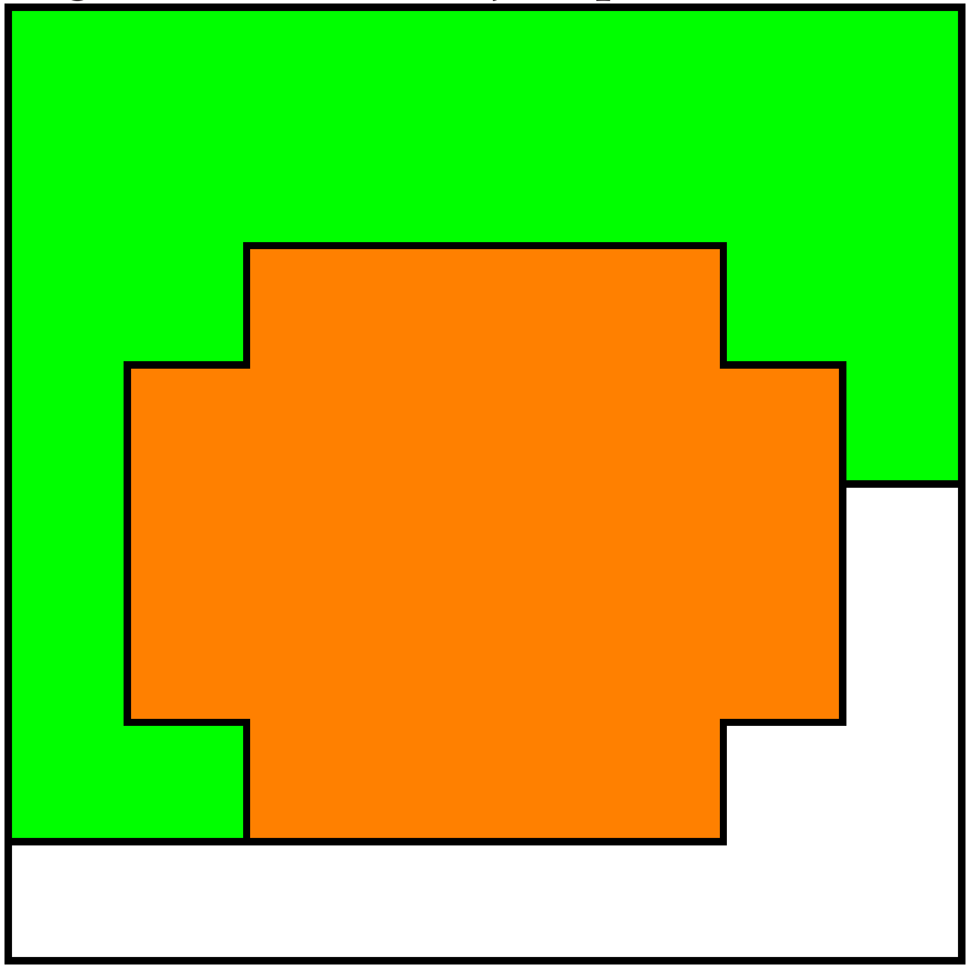
3: New parent assignments are to the furthest parent



4: If overlap is detected, latter parent is updated

What do we have now?

All the runs point to a unique parent identifying that region. So we have an image full of regions rather than just pixels.



What good is it?

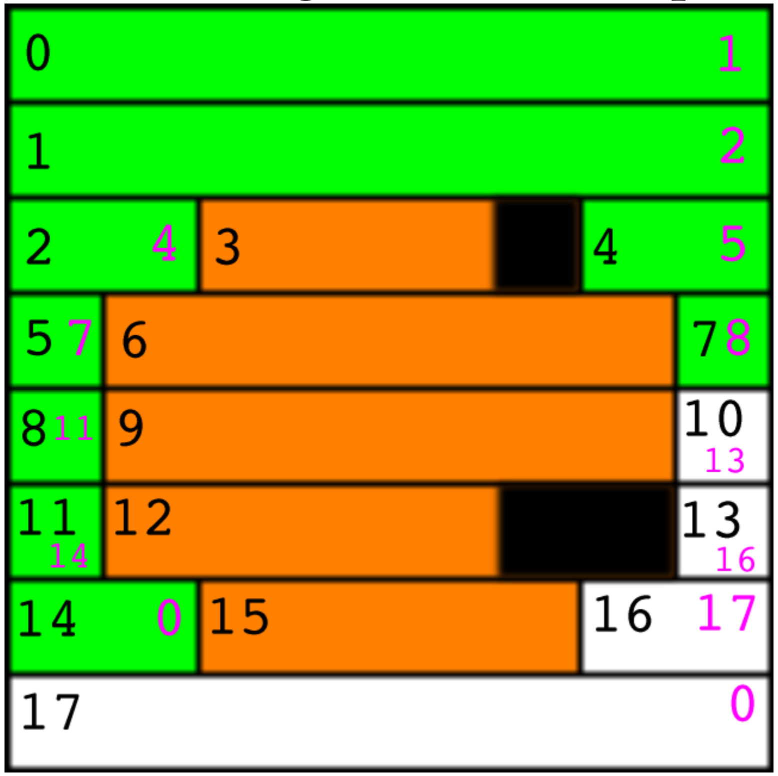
With another pass, we can gather statistics and features about the region:

- centroid: mean location
- bounding box: maximum and minimum in x and y
- area: occupied pixels
- average color: mean color of region pixels

Using these features, we can write concise and fast object detectors.

Representation details

Runs are numbered consectively (black numbers in picture). Background runs are not encoded. Each run has a next pointer to the next run in the same region (magenta numbers, shown only for the green and white regions). The last run in each region has a next pointer of 0.



References

- Low-level vision.
  CMVision Color Vision Library:
  http://www.cs.cmu.edu/~jbruce/cmvision/