Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

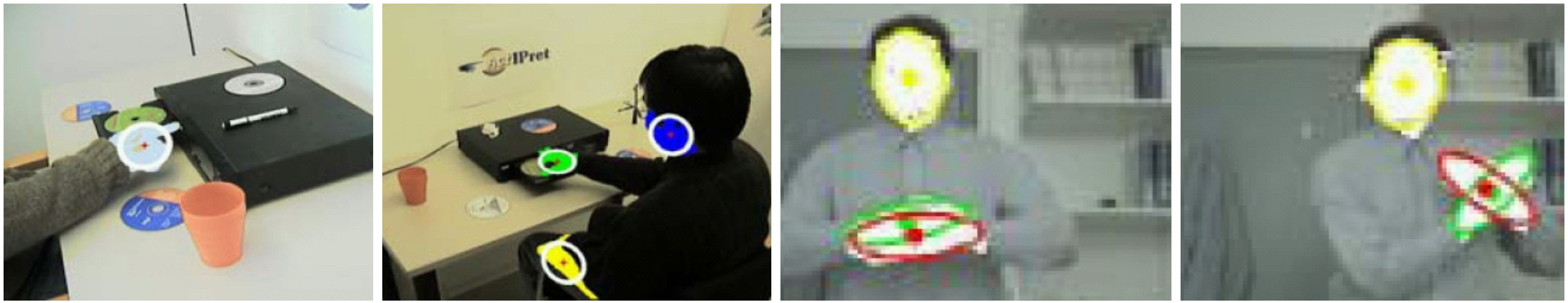
Lecture 9

Segmentation

Region-based Approaches:
Colour-based Segmentation, k-Means Clustering

Colour Segmentation

- Learning the colour distribution
- Pixel-based classification
- Finding connected regions



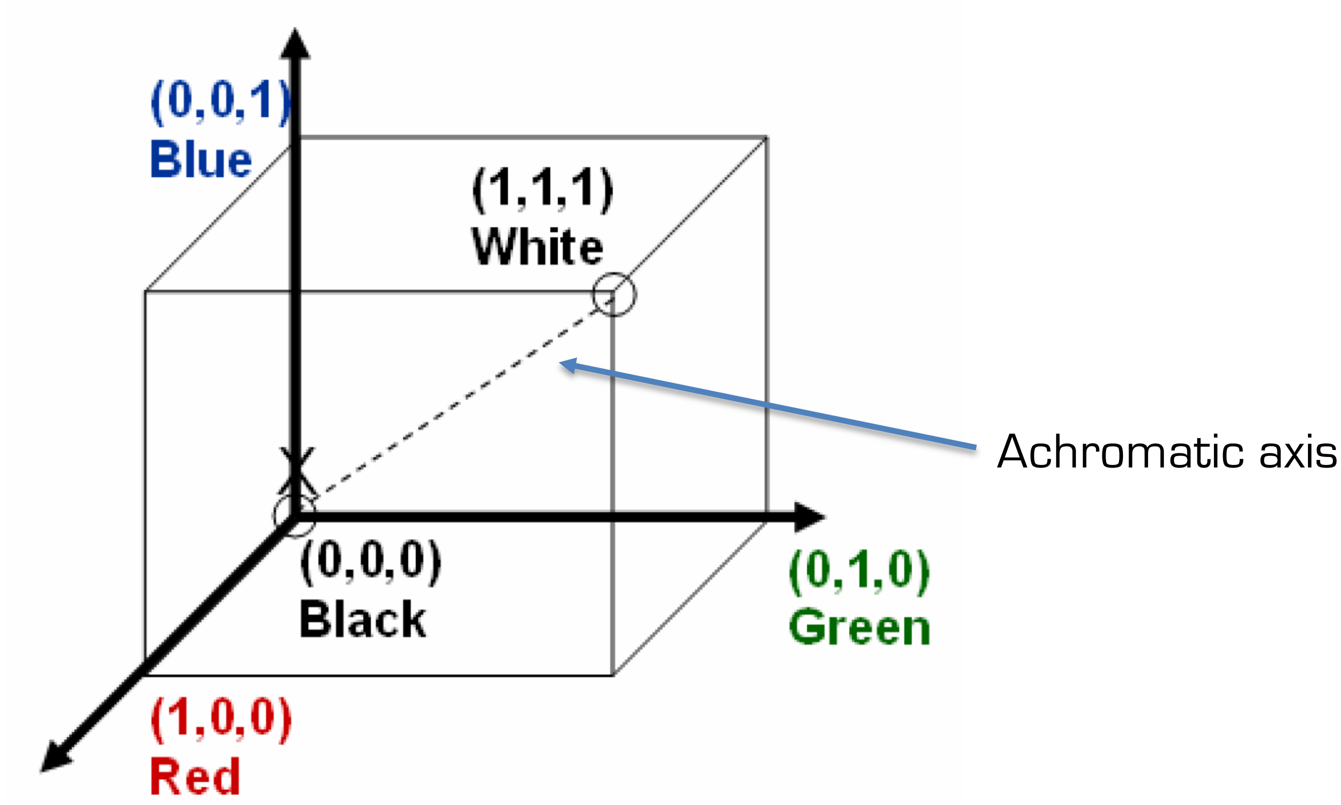
Examples of colour-based segmentation [Argyros]

Colour Segmentation

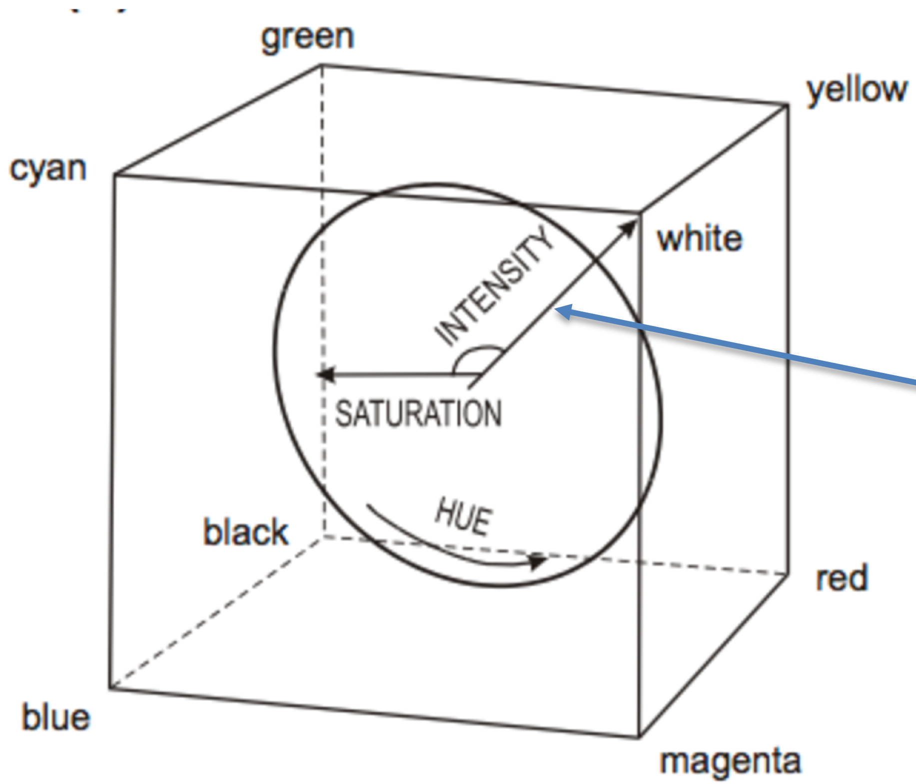
There are many colour representations

RGB	Red, Green, Blue
CMY	Cyan, Magenta, Yellow
YUV	Luminance (Y), Blue minus Luminance (U), Red minus Luminance (V)
YCrCb	Scaled version of YUV
CIE XYZ	Standard reference colour space based on the response of human eye
CIE L*u*V*	Perceptually uniform colour space
CIE L*a*b*	Device independent colour space (all colours perceived by humans)
HSV	Hue, Saturation, Value
HLS	Hue, Luminance, Saturation
HSI	Hue, Saturation, Intensity

Colour Segmentation

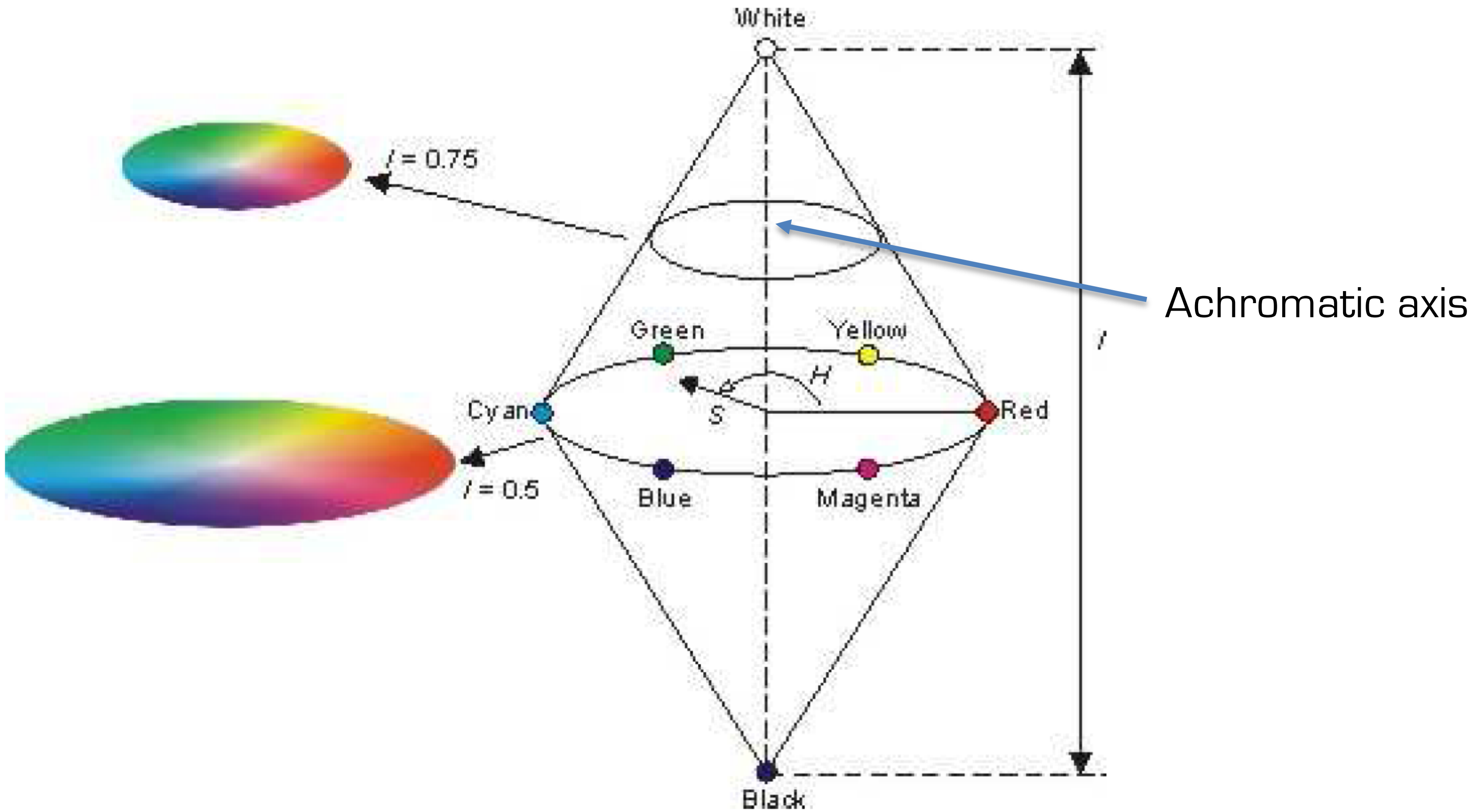


Colour Segmentation

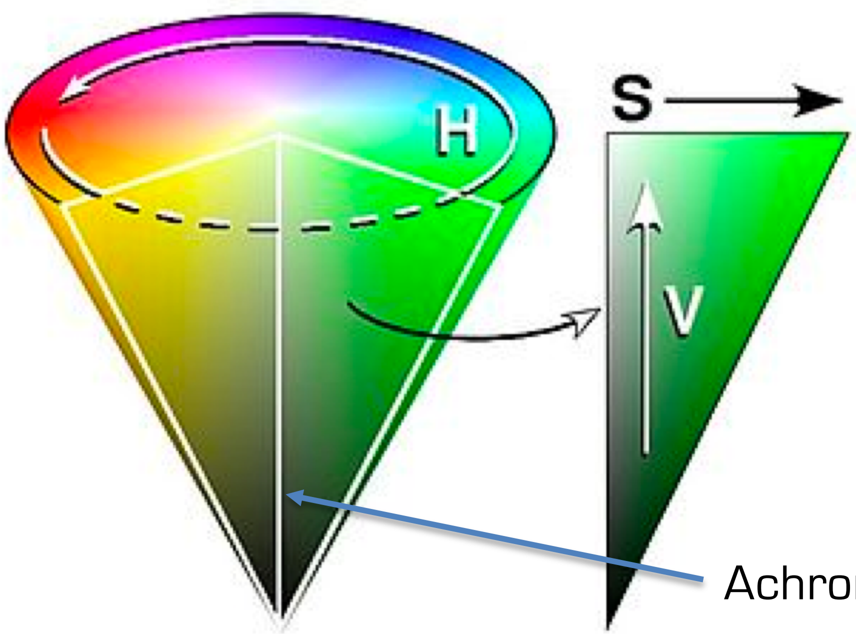


Achromatic axis

Colour Segmentation

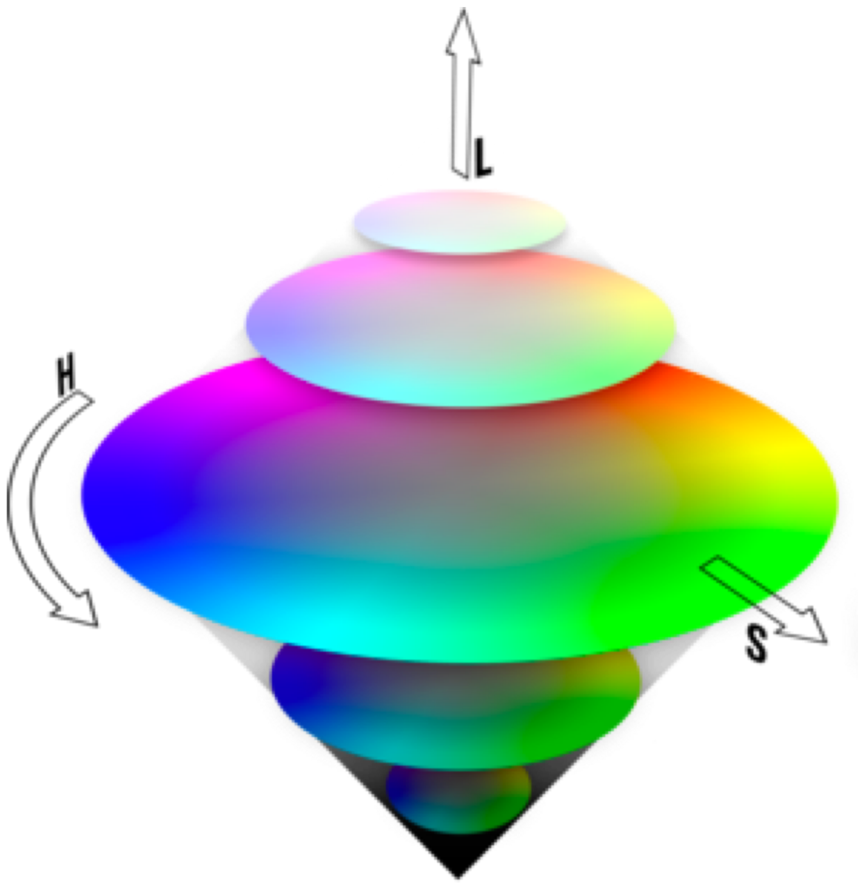


Colour Segmentation



Achromatic axis

Colour Segmentation



Colour Segmentation

Difference between HIS, HLS, and HSV are all based on different definitions of “brightness”

The Generalized Lightness, Hue, and Saturation (GLHS) model defines brightness as follows

$$L\left(\mathbf{c}\right) = w_{\min} \cdot \min\left(\mathbf{c}\right) + w_{\mathrm{mid}} \cdot \mathrm{mid}\left(\mathbf{c}\right) + w_{\max} \cdot \max\left(\mathbf{c}\right)$$

Where $\min\left(\mathbf{c}\right)$, $\mathrm{mid}\left(\mathbf{c}\right)$, and $\max\left(\mathbf{c}\right)$ return minimum, median, and maximum component of a vector $\mathbf{c}$ in the RGB space

Colour Segmentation

w_{min} , w_{mid} , w_{max} determine the colour space

$w_{min} + w_{mid} + w_{max} = 1, w_{max} > 0$

HSV $\quad w_{min} = 0, w_{mid} = 0, w_{max} = 1$

HLS $\quad w_{min} = ½ \ , w_{mid} = 0, w_{max} = \ = ½$

HSI $\quad w_{min} = 1/3, w_{mid} = 1/3, w_{max} = 1/3$

Colour Segmentation

$$I = R + G + B$$

$$h = arccos\left(\frac{((R-G)+(R-B))}{2\sqrt{(R-G)^2 + (R-B)(G-B)}} \right)$$

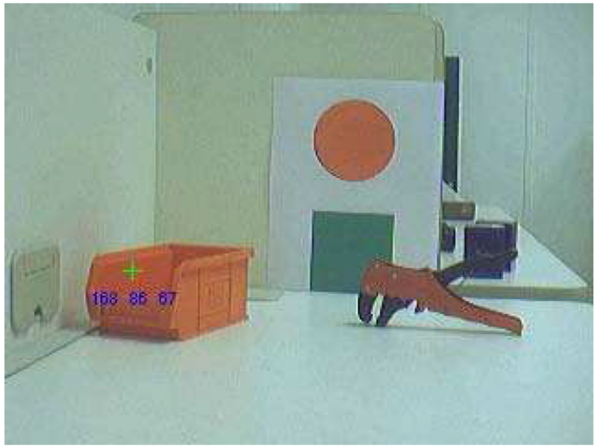
$$H = \begin{cases} h & if \ G \geq B \ and \ not \ (R = G = B) \\ 2\pi - h & if \ G \leq B \\ undefined & if \ (R = G = B) \end{cases}$$

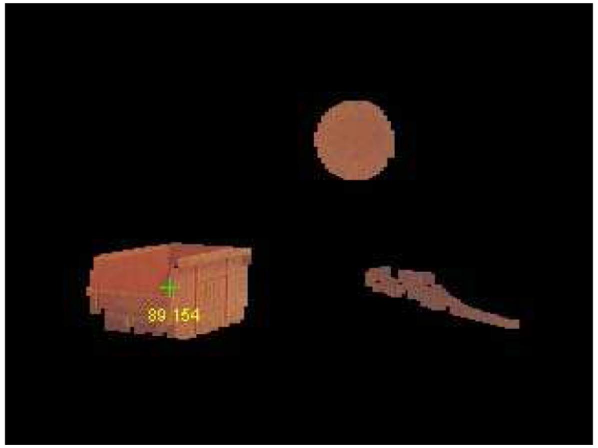
$$S = 1 - \frac{3 \times (min(R, G, B))}{(R + G + B)}$$

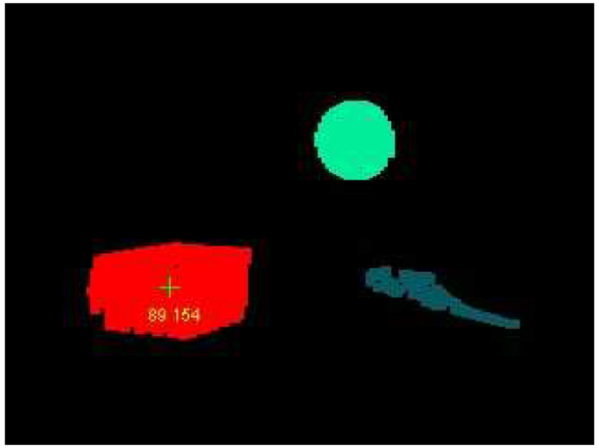
Colour Segmentation

- To segment images based on colour
  - Transform to HSI space
  - Discard I
  - Set minimum and maximum limits (thresholds) of acceptable H and S
  - Need to be careful where H wraps from 0 to 360 (or 180 in if using HLS in OpenCV)
- It may be necessary to smooth the image first
- It will be necessary to perform connected component analysis after the segmentation to label the segmented regions with distinct (mutually-exclusive) labels

Colour Segmentation





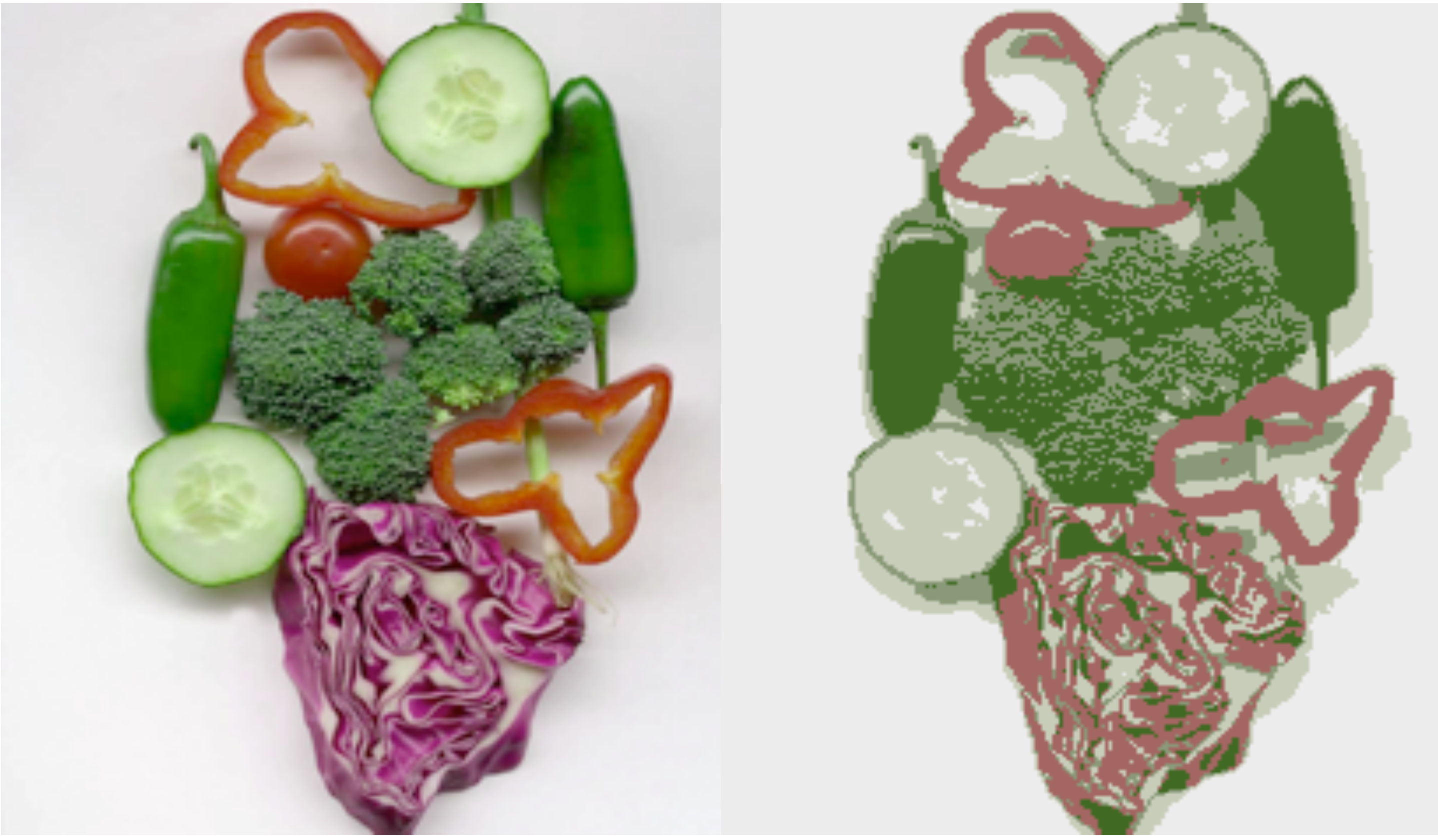


k-Means Clustering

- Identify significant colours in images
  - Concise descriptions
  - Object tracking
- How do we find the best colours?
  - k-means clustering
  - Creates k clusters of pixels
  - Cluster in feature-space (e.g. hue, hue-saturation)
  - k is known in advance
- Unsupervised learning

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

k-Means Clustering



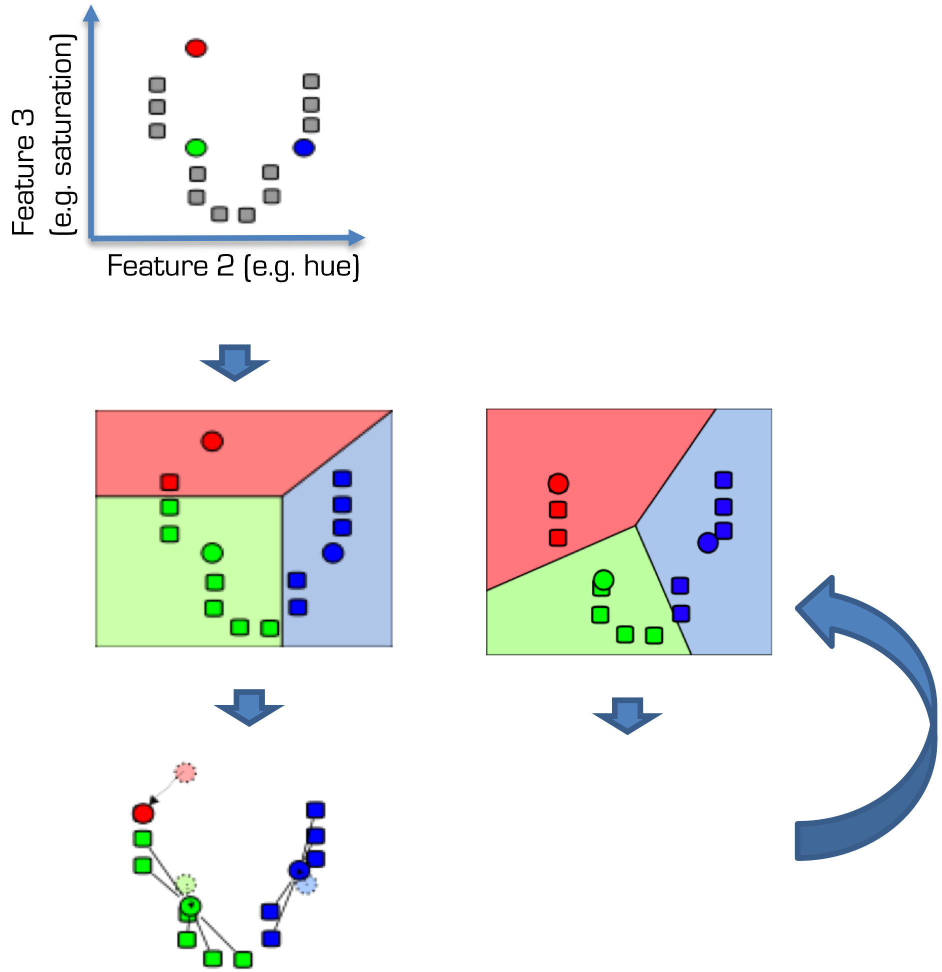
Credit: Markus Vincze, Technische Universität Wien

k-Means Clustering

1. Select randomly k start points (centroids, nodes, centers)

2. Assign all points to the closest centroid (in the feature space)

3. Calculate new centroid (mean values) for each cluster



Iterate until all centroids remain stable

Illustrations from: http://en.wikipedia.org/wiki/K-means_clustering

Credit: Markus Vincze, Technische Universität Wien

k-Means Clustering

Goal: group data to minimise the variance in the data $\mathbf{x}$ of a cluster $\mathbf{c}$

- Contain information in cluster as far as possible
- N data points
- K clusters

Cluster center

Data

$$\mathbf{c}^*, \boldsymbol{\delta}^* = \underset{\mathbf{c},\boldsymbol{\delta}}{\arg\min} \tfrac{1}{N} \sum_{j}^{N} \sum_{i}^{K} \delta_{ij} \left(\mathbf{c}_i - \mathbf{x}_j \right)^2$$

Binary assignment of x_j to c_i

Credit: Markus Vincze, Technische Universität Wien

k-Means Clustering

Initialize cluster centers: $\mathbf{c}^t$, $t = 0$

Repeat

Assign all points to nearest node/centre

$$\boldsymbol{\delta}^t = \underset{\boldsymbol{\delta}}{\operatorname{argmin}} \tfrac{1}{N} \sum_j^N \sum_i^K \delta_{ij} \left(\mathbf{c}_i^{t-1} - \mathbf{x}_j \right)^2$$

Re-calculate the centers as means of points

$$\mathbf{c}^t = \underset{\mathbf{c}}{\operatorname{argmin}} \tfrac{1}{N} \sum_j^N \sum_i^K \delta_{ij}^t \left(\mathbf{c}_i - \mathbf{x}_j \right)^2$$

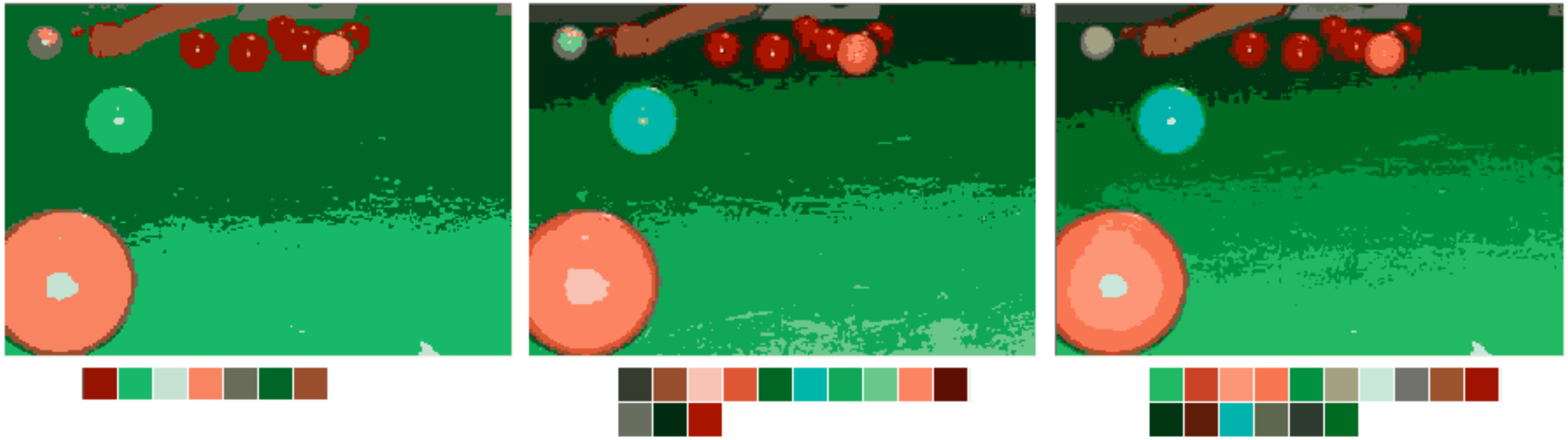
$t = t + 1$

Until no more new points are assigned

Credit: Markus Vincze, Technische Universität Wien

k-Means Clustering

- Different values of k (10, 15 & 20 random exemplars):



- Not all clusters end up with patterns
- More exemplars generally gives a more faithful representation

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

k-Means Clustering

- How many exemplars?
- Search for the number of clusters that gives the highest confidence, e.g. using Davies-Bouldin index
  - measures cluster separation

$$DB = 1/k \ \Sigma_{1..k} \ \max_{i \neq j} ((\Delta i + \Delta j) / \delta i.j)$$

Average distance to the cluster centres, a measure of cluster scatter (should be a small as possible)

Distance between the clusters (should be as large as possible)

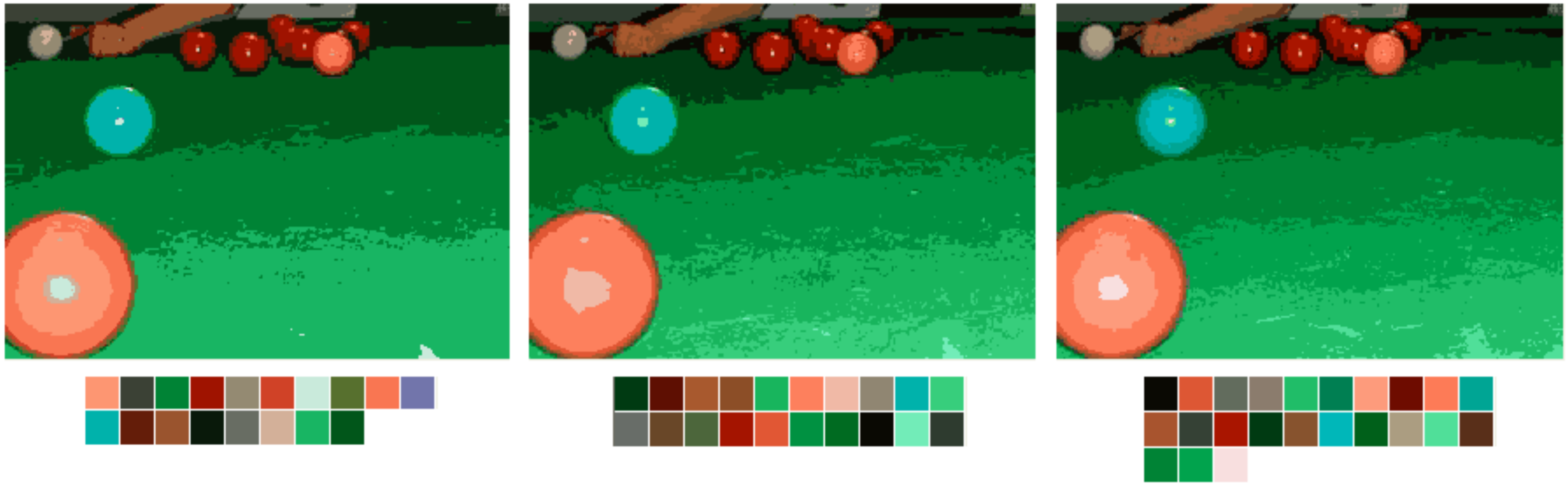
Choose worst case

  - The best clustering scheme minimizes the Davies–Bouldin index
  - Does not take into account the cluster size
    - Does not work well where there are some large clusters and some small clusters

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

k-Means Clustering

- Using random exemplars gives non-deterministic results (30 random exemplars each time):



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Demos

The following code is taken from the colourSegmentation project in the lectures directory of the ACV repository

See:

```
colourSegmentation.h
colourSegmentationImplementation.cpp
colourSegmentationApplication.cpp
```

```
/*
  Example use of openCV to perform colour segmentation

  The user must interactively select the colour sample that will form the basis of the segmentation.
  The user can also adjust the hue and saturation tolerances on that sample.
  -------------------------------------------------------------------------------------------------
  Application file

  David Vernon
  1 June 2017
*/

#include "colourSegmentation.h"

// Global variables to allow access by the display window callback functions

Mat inputBGRImage;
Mat inputHLSImage;
int hueRange             = 10; // default range
int saturationRange      = 10; // default range
Point2f sample_point;
int number_of_sample_points;

char* input_window_name       = "Input Image";
char* segmented_window_name   = "Segmented Image";
```

```
int main() {

   int end_of_file;
   bool debug = false;
   char filename[MAX_FILENAME_LENGTH];
   int max_hue_range = 180;
   int max_saturation_range = 128;
   Mat outputImage;

   FILE *fp_in;

   if ((fp_in = fopen("../data/colourSegmentationInput.txt","r")) == 0) {
     printf("Error can't open input colourSegmentationInput.txt\n");
     prompt_and_exit(1);
   }

   printf("Example of how to use openCV to perform colour segmentation.\n\n");

   do {
      end_of_file = fscanf(fp_in, "%s", filename);

      if (end_of_file != EOF) {

         inputBGRImage = imread(filename, CV_LOAD_IMAGE_UNCHANGED);
         if(inputBGRImage.empty()) {
            cout << "can not open " << filename << endl;
            prompt_and_exit(-1);
         }

         CV_Assert(inputBGRImage.type() == CV_8UC3 ); // make sure we are dealing with a colour image
```

```
         printf("Click on a sample point in the input image.\n");
         printf("When finished with this image, press any key to continue ...\n");

         /* Create a window for input and display it */
         namedWindow(input_window_name, CV_WINDOW_AUTOSIZE );
         setMouseCallback(input_window_name, getSamplePoint);    // use this callback to get the colour components of the sample point
         imshow(input_window_name, inputBGRImage);

         /* convert the BGR image to HLS to facilitate hue-saturation segmentation */
         cvtColor(inputBGRImage, inputHLSImage, CV_BGR2HLS);

         /* Create a window for segmentation based on hue and saturation thresholding */
         namedWindow(segmented_window_name, CV_WINDOW_AUTOSIZE );
         resizeWindow(segmented_window_name,0,0); // this forces the trackbar to be as small as possible (and to fit in the window)
         createTrackbar( "Hue Range", segmented_window_name, &hueRange,        max_hue_range,        colourSegmentation);
         createTrackbar( "Sat Range", segmented_window_name, &saturationRange, max_saturation_range, colourSegmentation);

         /* display a zero output */
         outputImage = Mat::zeros(inputBGRImage.rows, inputBGRImage.cols, inputBGRImage.type());
         imshow(segmented_window_name, outputImage);

         /* now wait for user interaction - mouse click to change the colour sample or trackbar adjustment to change the thresholds */
         number_of_sample_points = 0;
         do {
            waitKey(30);
         } while (!_kbhit());

         getchar(); // flush the buffer from the keyboard hit

         destroyWindow(input_window_name);
         destroyWindow(segmented_window_name);
      }
   } while (end_of_file != EOF);

   fclose(fp_in);

   return 0;
}
```

```
/*
  Example use of openCV to perform colour segmentation

  The user must interactively select the colour sample that will form the basis of the segmentation.
  The user can also adjust the hue and saturation tolerances on that sample.
  -------------------------------------------------------------------------------------------------
  Implementation file

  David Vernon
  1 June 2017
*/
#include "colourSegmentation.h"

void colourSegmentation(int, void*) {

   extern Mat inputBGRImage;
   extern Mat inputHLSImage;
   extern int hueRange;
   extern int saturationRange;
   extern Point2f sample_point;
   extern char* segmented_window_name;
   extern int number_of_sample_points;

   Mat segmentedImage;
   int row, col;

   int hue;
   int saturation;
   int h;
   int s;

   bool debug = false;
```

```
   /* now get the sample point */
   if (number_of_sample_points == 1) {

      segmentedImage = inputBGRImage.clone();

      hue        = inputHLSImage.at<Vec3b>((int)sample_point.y,(int)sample_point.x)[0]; // note order of indices
      saturation = inputHLSImage.at<Vec3b>((int)sample_point.y,(int)sample_point.x)[2]; // note order of indices

      if (debug) {
         printf("Sample point (%f, %f) Hue: %d  Saturation: %d\n", sample_point.y, sample_point.x, hue, saturation); // note order of indices
         printf("Hue range: %d  Saturation range: %d\n", hueRange, saturationRange);                               // note order of indices
      }

      /* now perform segmentation */
      for (row=0; row < inputBGRImage.rows; row++) {
         for (col=0; col < inputBGRImage.cols; col++) {

            h = inputHLSImage.at<Vec3b>(row,col)[0];
            s = inputHLSImage.at<Vec3b>(row,col)[2];

            /* Note: 0 <= h <= 180 ... NOT as you'd expect: 0 <= h <= 360  */
            if ((((h >= hue     - hueRange) && (h <= hue     + hueRange)) ||
                 ((h >= hue+180 - hueRange) && (h <= hue+180 + hueRange)) ||
                 ((h >= hue-180 - hueRange) && (h <= hue-180 + hueRange)))
                 &&
                 ((s >= (saturation - saturationRange)) && (s <= (saturation + saturationRange)))) {
               segmentedImage.at<Vec3b>(row,col)[0] = inputBGRImage.at<Vec3b>(row,col)[0];
               segmentedImage.at<Vec3b>(row,col)[1] = inputBGRImage.at<Vec3b>(row,col)[1];
               segmentedImage.at<Vec3b>(row,col)[2] = inputBGRImage.at<Vec3b>(row,col)[2];
            }
            else {
               segmentedImage.at<Vec3b>(row,col)[0] = 0;
               segmentedImage.at<Vec3b>(row,col)[1] = 0;
               segmentedImage.at<Vec3b>(row,col)[2] = 0;
            }
         }
      }
      imshow(segmented_window_name, segmentedImage);
   }

   if (debug) printf("Leaving colourSegmentation() \n");

}
```

```
void getSamplePoint( int event, int x, int y, int, void* ) {

   extern char*   input_window_name;
   extern Mat     inputBGRImage;
   extern Point2f sample_point;
   extern int     number_of_sample_points;
   Mat            inputImageCopy;
   int crossHairSize = 10;

   if (event != EVENT_LBUTTONDOWN) {
      return;
   }
   else {
      number_of_sample_points = 1;
      sample_point.x = (float) x;
      sample_point.y = (float) y;

      inputImageCopy = inputBGRImage.clone();

      line(inputImageCopy,Point(x-crossHairSize/2,y),Point(x+crossHairSize/2,y),Scalar(0, 255, 0),1, CV_AA); // Green
      line(inputImageCopy,Point(x,y-crossHairSize/2),Point(x,y+crossHairSize/2),Scalar(0, 255, 0),1, CV_AA);

      imshow(input_window_name, inputImageCopy); // show the image with the cross-hairs

      colourSegmentation(0, 0); // Show the segmented image for new colour sample and current thresholds
   }
}
```

Demos

The following code is taken from the kMeansClustering project
in the lectures directory of the ACV repository

See:

```
kMeansClustering.h
kMeansClusteringImplementation.cpp
kMeansClusteringApplication.cpp
```

```
/*
  Example use of openCV to perform k-means clustering
  -------------------------------------------------
  Implementation file

  David Vernon
  3 June 2017
*/

#include "kMeansClustering.h"

void kMeansClustering(int, void*) {

   extern Mat image;
   extern int k;
   extern char* outputWindowName;

   Mat result_image;
   int iterations;
   iterations = 5;

   if (k < 1)  // the trackbar has a lower value of 0 which is invalid
      k = 1;
```

```
/************************************************************************************************/
/*
 * This code is provided as part of "A Practical Introduction to Computer Vision with OpenCV"
 * by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014.  All rights reserved.
 */

   CV_Assert( image.type() == CV_8UC3 );
   // Populate an n*3 array of float for each of the n pixels in the image
   Mat samples(image.rows*image.cols, image.channels(), CV_32F);
   float* sample = samples.ptr<float>(0);
   for(int row=0; row<image.rows; row++)
      for(int col=0; col<image.cols; col++)
         for (int channel=0; channel < image.channels(); channel++)
            samples.at<float>(row*image.cols+col,channel) =
                        (uchar) image.at<Vec3b>(row,col)[channel];
   // Apply k-means clustering to cluster all the samples so that each sample
   // is given a label and each label corresponds to a cluster with a particular
   // centre.
   Mat labels;
   Mat centres;
   kmeans(samples, k, labels, TermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 1, 0.0001),
      iterations, KMEANS_PP_CENTERS, centres );
   // Put the relevant cluster centre values into a result image
   result_image = Mat( image.size(), image.type() );
   for(int row=0; row<image.rows; row++)
      for(int col=0; col<image.cols; col++)
         for (int channel=0; channel < image.channels(); channel++)
            result_image.at<Vec3b>(row,col)[channel] = (uchar) centres.at<float>(*(labels.ptr<int>(row*image.cols+col)), channel);
 /************************************************************************************************/

   imshow(outputWindowName, result_image);
}
```

Reading

R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

Section 5.3.1 K-means and mixtures of Gaussian

A. Hanbury, “The Taming of the Hue, Saturation, and Brightness Colour Space”, Proc. Computer Vision Winter Workshop (CVWW), Austria, 2002.