OpenCV Tutorial C++

Color Detection & Object Tracking

Object detection and segmentation is the most important and challenging fundamental task of computer vision. It is a critical part in many applications such as image search, scene understanding, etc. However it is still an open problem due to the variety and complexity of object classes and backgrounds.

The easiest way to detect and segment an object from an image is the color based methods . The object and the background should have a significant color difference in order to successfully segment objects using color based methods.

Simple Example of Detecting a Red Object

In this example, I am going to process a video with a red color object and create a binary video by thresholding the red color. (Red color area of the video is assigned to '1' and other area is assigned to '0' in the binary image so that you will see a white patch wherever the red object is in the original video)

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include <iostream>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

using namespace cv;
using namespace std;

 int main( int argc, char** argv )
 {
   VideoCapture cap(0); //capture the video from web cam

   if ( !cap.isOpened() )  // if not success, exit program
   {
        cout << "Cannot open the web cam" << endl;
        return -1;
   }

   namedWindow("Control", CV_WINDOW_AUTOSIZE); //create a window called "Control"

 int iLowH = 0;
 int iHighH = 179;

 int iLowS = 0;
 int iHighS = 255;

 int iLowV = 0;
 int iHighV = 255;

 //Create trackbars in "Control" window
 cvCreateTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)
 cvCreateTrackbar("HighH", "Control", &iHighH, 179);

 cvCreateTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 - 255)
 cvCreateTrackbar("HighS", "Control", &iHighS, 255);

 cvCreateTrackbar("LowV", "Control", &iLowV, 255); //Value (0 - 255)
 cvCreateTrackbar("HighV", "Control", &iHighV, 255);

   while (true)
   {
      Mat imgOriginal;

      bool bSuccess = cap.read(imgOriginal); // read a new frame from video

       if (!bSuccess) //if not success, break loop
      {
           cout << "Cannot read a frame from video stream" << endl;
```

```
            break;
        }

       Mat imgHSV;

       cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV); //Convert the captured frame from BGR to HSV

       Mat imgThresholded;

       inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded); //Threshold the image

       //morphological opening (remove small objects from the foreground)
       erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );
       dilate( imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );

       //morphological closing (fill small holes in the foreground)
       dilate( imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );
       erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );

       imshow("Thresholded Image", imgThresholded); //show the thresholded image
       imshow("Original", imgOriginal); //show the original image

          if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is pressed, break loop
         {
             cout << "esc key is pressed by user" << endl;
             break;
         }
       }

       return 0;

}
```

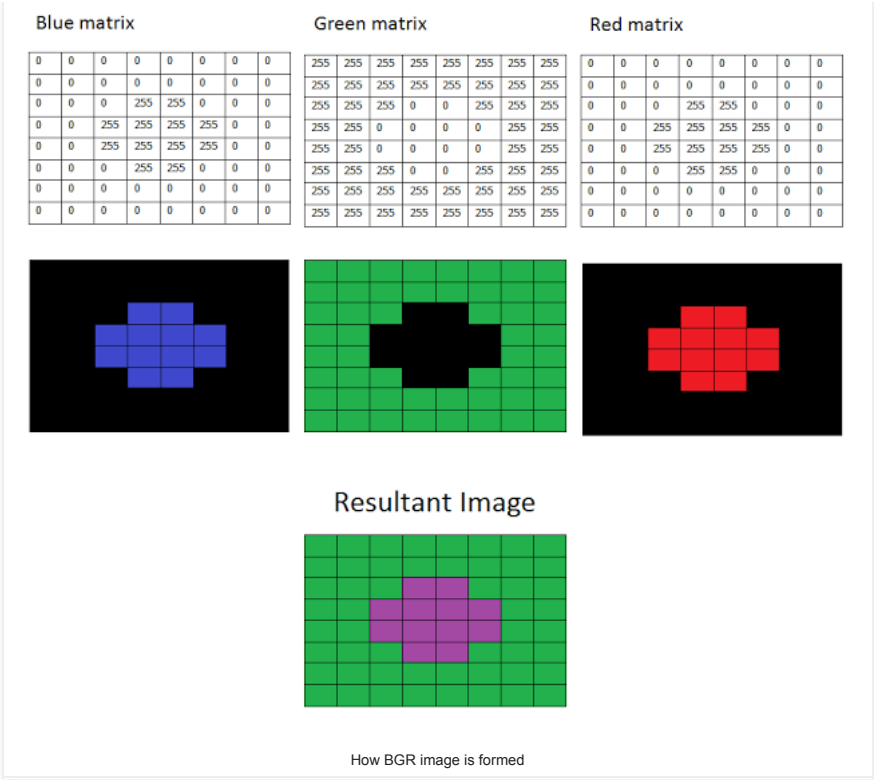
//

You can download this OpenCV visual c++ project from here.

Explanation

OpenCV usually captures images and videos in 8-bit, unsigned integer, BGR format. In other words, captured images can be considered as 3 matrices; BLUE, GREEN and RED (hence the name BGR) with integer values ranges from 0 to 255.

The following image shows how a color image is represented using 3 matrices.



How BGR image is formed

In the above image, each small box represents a pixel of the image. In real images, these pixels are so small that human eye cannot differentiate.

Usually, one can think that BGR color space is more suitable for color based segmentation. But HSV color space is the most suitable color space for color based image segmentation. So, in the above application, I have converted the color space of original image of the video from BGR to HSV image.

HSV color space is also consists of 3 matrices, HUE, SATURATION and VALUE. In OpenCV, value range for HUE, SATURATION and VALUE are respectively 0-179, 0-255 and 0-255. HUE represents the color, SATURATION represents the amount to which that respective color is mixed with white and VALUE represents the amount to which that respective color is mixed with black.

In the above application, I have considered that the red object has HUE, SATURATION and VALUE in between 170-180, 160-255, 60-255 respectively. Here the HUE is unique for that specific color distribution of that object. But SATURATION and VALUE may be vary according to the lighting condition of that environment.

Hue values of basic colors

- Orange 0-22
- Yellow 22- 38
- Green 38-75
- Blue 75-130
- Violet 130-160
- Red 160-179

These are approximate values. You have to find the exact range of HUE values according to the color of the object. I found that the range of 170-179 is perfect for the range of hue values of my object. The SATURATION and VALUE is depend on the lighting condition of the environment as well as the surface of the object.

How to find the exact range of HUE, SATURATION and VALUE for a object is discussed later in this post.

After thresholding the image, you'll see small white isolated objects here and there. It may be because of noises in the image or the actual small objects which have the same color as our main object. These unnecessary small white patches can be eliminated by applying **morphological opening**. **Morphological opening** can be achieved by a erosion, followed by the dilation with the same structuring element.

Thresholded image may also have small white holes in the main objects here and there. It may be because of noises in the image. These unnecessary small holes in the main object can be eliminated by applying **morphological closing**. **Morphological closing** can be achieved by a dilation, followed by the erosion with the same structuring element.

Now let's discuss new OpenCV methods in the above application.

- **void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst);**

Checks that each element of 'src' lies between 'lowerb' and 'upperb'. If so, that respective location of 'dst' is assigned '255' , otherwise '0'. (Pixels with value 255 is shown as white whereas pixels with value 0 is shown as black)

Arguments -

- **InputArray src** - Source image

- **InputArray lowerb** - Inclusive lower boundary (If **lowerb=**Scalar(x, y, z), pixels which have values lower than x, y and z for HUE, SATURATION and VALUE respectively is considered as black pixels in **dst** image)
- **InputArray upperb** - Exclusive upper boundary (If it is **upperb=**Scalar(x, y, z), pixels which have values greater or equal than x, y and z for HUE, SATURATION and VALUE respectively is considered as black pixels in **dst** image)
- **OutputArray dst** - Destination image (should have the same size as the **src** image and should be 8-bit unsigned integer, CV_8U)

- **void erode(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue())**

This function erode the source image and stores the result in the destination image. In-place processing is supported. (which means you can use the same variable for the source and destination image). If the source image is multi-channel, all channels are processed independently and the result is stored in the destination image as separate channels.

Arguments -

- **InputArray src -** Source image
- **OutputArray dst** - Destination image (should have the same size and type as the source image)
- **InputArray kernel** - Structuring element which is used to erode the source image
- **Point anchor** - Position of the anchor within the kernel. If it is Point(-1, -1), the center of the kernel is taken as the position of anchor
- **int iterations** - Number of times erosion is applied
- **int borderType** - Pixel extrapolation method in a boundary condition
- **const Scalar& borderValue** - Value of the pixels in a boundary condition if **borderType = BORDER_CONSTANT**

- **void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue());**

This function dilate the source image and stores the result in the destination image. In-place processing is supported. (which means you can use the same variable for the source and destination image). If the source image is multi-channel, all channels are processed independently and the result is stored in the destination image as separate channels.

- **InputArray src** - Source image
- **OutputArray dst** - Destination image (should have the same size and the type as the source image)
- **InputArray kernel** - Structuring element which is used to dilate the source image
- **Point anchor** - Position of the anchor within the kernel. If it is Point(-1, -1), the center of the kernel is taken as the position of anchor
- **int iterations** - Number of times dilation is applied
- **int borderType** - Pixel extrapolation method in a boundary condition
- **const Scalar& borderValue** - Value of the pixels in a boundary condition if **borderType = BORDER_CONSTANT**

- **void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)**

This function convert a source image from one color space to another. In-place processing is supported. (which means you can use the same variable for the source and destination image)

- **InputArray src** - Source image
- **OutputArray dst** - Destination image (should have the same size and the depth as the source image)
- **int code** - Color space conversion code (e.g - COLOR_BGR2HSV, COLOR_RGB2HSV, COLOR_BGR2GRAY, COLOR_BGR2YCrCb, COLOR_BGR2BGRA, etc)
- **int dstCn** - Number of channels in the destination image. If it is 0, number of channels is derived automatically from the source image and the color conversion code.

All other OpenCV methods in the above application have been discussed in early OpenCV tutorials.

Simple Example of Tracking Red objects

In the previous example, I showed you how to detect a color object. In the following example, I'll show you how to track a color object. There are 3 steps involving to achieve this task.

1. Detect the object
2. Find the exact position (x, y coordinates) of the object
3. Draw a line along the trajectory of the object

Here is how it is done with OpenCV / C++.

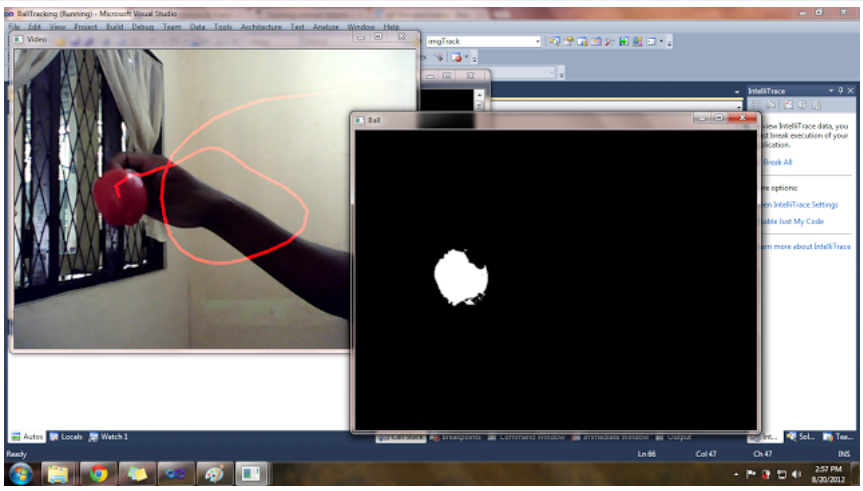
```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include <iostream>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
```

```
using namespace cv;
using namespace std;

 int main( int argc, char** argv )
 {
   VideoCapture cap(0); //capture the video from webcam

   if ( !cap.isOpened() )  // if not success, exit program
   {
        cout << "Cannot open the web cam" << endl;
        return -1;
   }

   namedWindow("Control", CV_WINDOW_AUTOSIZE); //create a window called "Control"

 int iLowH = 170;
 int iHighH = 179;

 int iLowS = 150;
 int iHighS = 255;

 int iLowV = 60;
 int iHighV = 255;

 //Create trackbars in "Control" window
 createTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)
 createTrackbar("HighH", "Control", &iHighH, 179);

 createTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 - 255)
 createTrackbar("HighS", "Control", &iHighS, 255);

 createTrackbar("LowV", "Control", &iLowV, 255);//Value (0 - 255)
 createTrackbar("HighV", "Control", &iHighV, 255);

 int iLastX = -1;
 int iLastY = -1;

 //Capture a temporary image from the camera
 Mat imgTmp;
 cap.read(imgTmp);

 //Create a black image with the size as the camera output
 Mat imgLines = Mat::zeros( imgTmp.size(), CV_8UC3 );;


   while (true)
   {
      Mat imgOriginal;

      bool bSuccess = cap.read(imgOriginal); // read a new frame from video



       if (!bSuccess) //if not success, break loop
      {
           cout << "Cannot read a frame from video stream" << endl;
           break;
      }

  Mat imgHSV;

  cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV); //Convert the captured frame from BGR to HSV

  Mat imgThresholded;

  inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded); //Threshold the image

  //morphological opening (removes small objects from the foreground)
  erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );
  dilate( imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );

  //morphological closing (removes small holes from the foreground)
  dilate( imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );
  erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );

  //Calculate the moments of the thresholded image
  Moments oMoments = moments(imgThresholded);

  double dM01 = oMoments.m01;
  double dM10 = oMoments.m10;
```

```
 double dArea = oMoments.m00;

 // if the area <= 10000, I consider that the there are no object in the image and it's because of the noise, the area is not zero
 if (dArea > 10000)
 {
  //calculate the position of the ball
  int posX = dM10 / dArea;
  int posY = dM01 / dArea;

  if (iLastX >= 0 && iLastY >= 0 && posX >= 0 && posY >= 0)
  {
   //Draw a red line from the previous point to the current point
   line(imgLines, Point(posX, posY), Point(iLastX, iLastY), Scalar(0,0,255), 2);
  }

  iLastX = posX;
  iLastY = posY;
 }

 imshow("Thresholded Image", imgThresholded); //show the thresholded image

 imgOriginal = imgOriginal + imgLines;
 imshow("Original", imgOriginal); //show the original image

      if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is pressed, break loop
     {
         cout << "esc key is pressed by user" << endl;
         break;
     }
  }

  return 0;
}
```

//

You can download this OpenCV visual c++ project from here.



Object Tracking

Explanation

In this application, I use moments to calculate the position of the center of the object. We have to calculate 1st order spatial moments around x-axis and y-axis and the 0th order central moments of the binary image.

0th order central moments of the binary image is equal to the white area of the image in pixels.

- X coordinate of the position of the center of the object = 1st order spatial moment around x-axis / 0th order central moment
- Y coordinate of the position of the center of the object = 1st order spatial moment around y-axis / 0th order central moment

If there are 2 or more objects in the image, we cannot use this method. And noise of the binary image is also should be at minimum level to get accurate results.

In the above application, I considered that if the white area of the binary image is less than or equal to 10000 pixels, there are no objects in the image because my object is expected to have an area more than 10000 pixels.

Now, let's discuss new OpenCV methods that can be found in the above application.

- **Moments moments(InputArray array, bool binaryImage=false)**

This OpenCV function calculates all of the spatial moments up to the third order and returns a **Moments** object with the results.

  - **InputArray array** - Single channel image
  - **bool binaryImage** - If this is true, all non zero pixels are considered as ones when calculating moments.

- **void line(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0)**

This function draws a line between two points on a given image

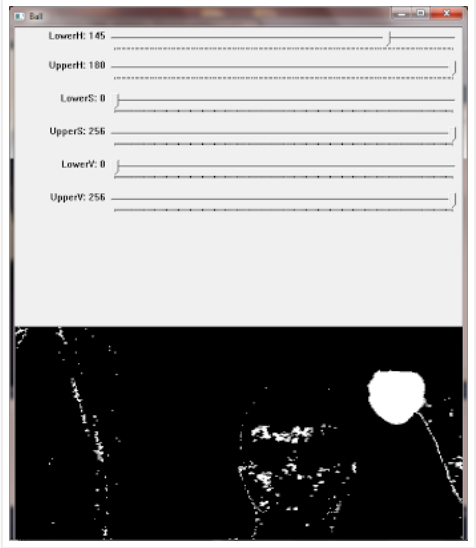
  - **Mat& img** - image which you want to draw the line
  - **Point pt1** - First point of the line segment
  - **Point pt2** - Other point of the line segment
  - **const Scalar& color** - Color of the line (values of Blue, Green and Red colors respectively)
  - **int thickness** - Thickness of the line in pixels

- **static MatExpr zeros(Size size, int type)**

This function returns a black image (with pixels with zero values) with a given size and type.

  - **Size size** - Size of the required image (Size(No of columns, No of rows))
  - **int type** - Type of the image (e.g - CV_8UC1, CV_32FC4, CV_8UC3, etc)

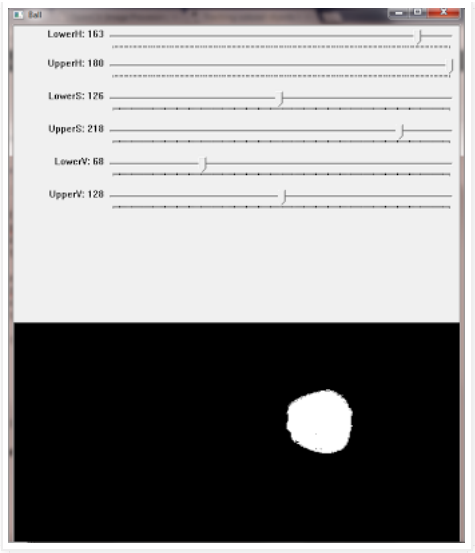
How to Find Exact Range for 'Hue', 'Saturation' and 'Value' for a Given Object



Finding the optimum HUE, SATURATION and VALUE ranges for an object is a 4 step process.

1. Track bars should be placed in a separate window so that ranges for HUE, SATURATION and VALUE can be adjusted. And set the initial ranges for HUE, SATURATION and VALUE as 0-179, 0-255 and 0-255 respectively. So, we will see a complete white image in the 'Control' window.
2. First, adjust 'LowH' and 'HighH' track bars so that the gap between 'LowH' and 'HighH' is minimized. Here you have to be careful that white area in 'Ball' window that represents the object should not be affected, while you are trying to minimize the gap.
3. Repeat the step 2 for 'LowS' and 'HighS' trackbars
4. Repeat the step2 for 'LowV' and 'HighV' trackbars

Now you can find the optimum HUE, SATURATION and VALUE ranges for the object. It is 163-179, 126-217 and 68-127 in my case as you can see in the below picture.



Next Tutorial : Object Detection & Shape Recognition using Contours

Previous Tutorial : Rotate Image & Video

Posted by Shermal Fernando

+18 Recommend this on Google

Is This Helpful : Yes (9) No (0)

63 comments:

gino0717 June 13, 2013 at 11:11 AM

Thanks for sharing this tutorial.

Reply

Yonas Teodros June 16, 2013 at 1:59 AM

thanks man

Reply

hika moryu June 18, 2013 at 8:19 AM

Thanks for the tutorial, I want to ask how to calculate the number of moving vehicle from the contour?
1 vehicle will detect in some frame...
Thanks before, anyone could help? please...

Reply

Anonymous August 6, 2013 at 9:58 PM

Really awesome tutorial, i really want to ask about how to handle mouse click events and mouse movements with my finger using object tracking . thanks in advance!!

Reply

sonofben August 24, 2013 at 12:36 AM

Guys, I'm looking for an updated object recognition API that has the ability to count objects such as cars, as well as facial recognition. Any suggestions?

Reply

budi santosa September 5, 2013 at 3:57 PM