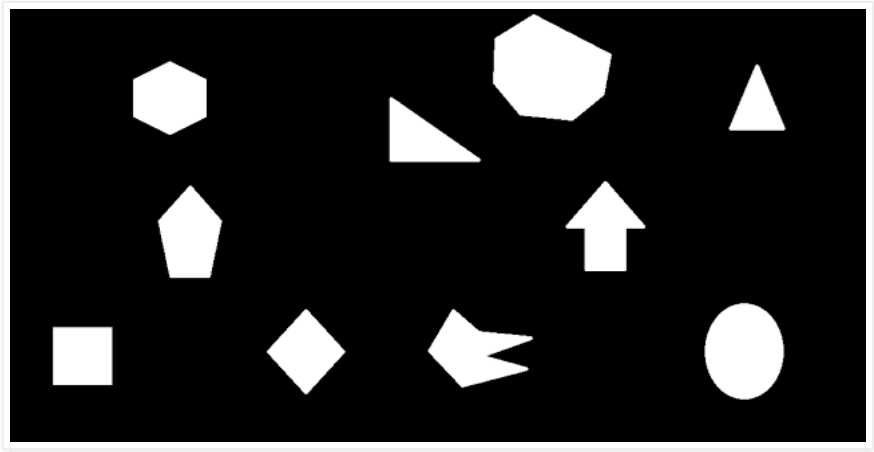
OpenCV Tutorial C++

Shape Detection & Tracking using Contours

In the previous tutorial, we could detect and track an object using color separation. But we could not identify the shape of the object there. In this tutorial, let's see how to identify a shape and position of an object using contours with OpenCV.

Using contours with OpenCV, you can get a sequence of points of vertices of each white patch (White patches are considered as polygons). As example, you will get 3 points (vertices) for a triangle, and 4 points for quadrilaterals. So, you can identify any polygon by the number of vertices of that polygon. You can even identify features of polygons such as convexity, concavity, equilateral and etc by calculating and comparing distances between vertices.

Let's see how this can be done with OpenCV. All you need, is a binary image in which your objects should be white and the background should be black.



Now I am going to identify triangles and quadrilaterals and heptagon in the above image using a C++ application with OpenCV. I'll draw a line along the perimeter of every identified polygon with colors blue for triangle, green for quadrilaterals and red for heptagons. Here is the code.

//

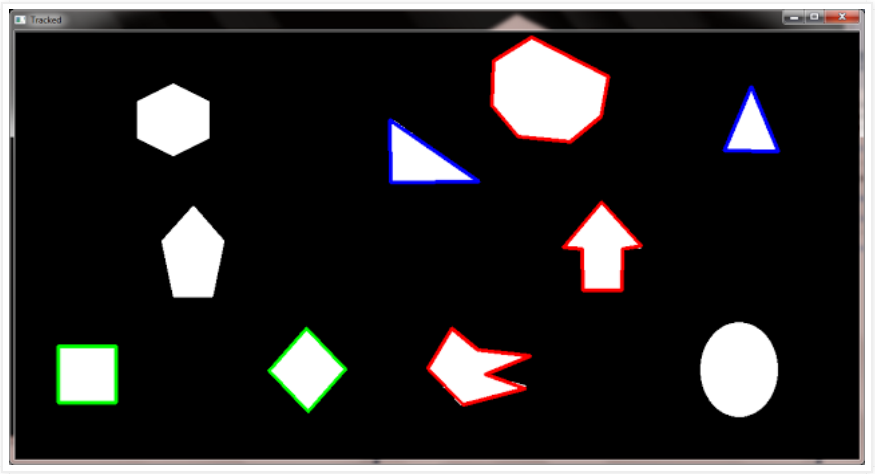
```
#include "stdafx.h"
#include <cv.h>
#include <highgui.h>
using namespace std;

int main()
{

 IplImage* img =  cvLoadImage("C:/Users/SHERMAL/Desktop/FindingContours.png");

 //show the original image
 cvNamedWindow("Raw");
 cvShowImage("Raw",img);

 //converting the original image into grayscale
 IplImage* imgGrayScale = cvCreateImage(cvGetSize(img), 8, 1);
```

```
cvCvtColor(img,imgGrayScale,CV_BGR2GRAY);

//thresholding the grayscale image to get better results
cvThreshold(imgGrayScale,imgGrayScale,128,255,CV_THRESH_BINARY);

CvSeq* contours;  //hold the pointer to a contour in the memory block
CvSeq* result;   //hold sequence of points of a contour
CvMemStorage *storage = cvCreateMemStorage(0); //storage area for all contours

//finding all contours in the image
cvFindContours(imgGrayScale, storage, &contours, sizeof(CvContour), CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE,
cvPoint(0,0));

//iterating through each contour
while(contours)
{
   //obtain a sequence of points of contour, pointed by the variable 'contour'
   result = cvApproxPoly(contours, sizeof(CvContour), storage, CV_POLY_APPROX_DP, cvContourPerimeter(contours)*0.02, 0);

   //if there are 3  vertices  in the contour(It should be a triangle)
  if(result->total==3 )
   {
      //iterating through each point
      CvPoint *pt[3];
      for(int i=0;i<3;i++){
         pt[i] = (CvPoint*)cvGetSeqElem(result, i);
      }

      //drawing lines around the triangle
      cvLine(img, *pt[0], *pt[1], cvScalar(255,0,0),4);
      cvLine(img, *pt[1], *pt[2], cvScalar(255,0,0),4);
      cvLine(img, *pt[2], *pt[0], cvScalar(255,0,0),4);

   }

   //if there are 4 vertices in the contour(It should be a quadrilateral)
   else if(result->total==4 )
   {
      //iterating through each point
      CvPoint *pt[4];
      for(int i=0;i<4;i++){
         pt[i] = (CvPoint*)cvGetSeqElem(result, i);
      }

      //drawing lines around the quadrilateral
      cvLine(img, *pt[0], *pt[1], cvScalar(0,255,0),4);
      cvLine(img, *pt[1], *pt[2], cvScalar(0,255,0),4);
      cvLine(img, *pt[2], *pt[3], cvScalar(0,255,0),4);
      cvLine(img, *pt[3], *pt[0], cvScalar(0,255,0),4);
   }

 //if there are 7  vertices  in the contour(It should be a heptagon)
   else if(result->total ==7  )
   {
      //iterating through each point
      CvPoint *pt[7];
      for(int i=0;i<7;i++){
         pt[i] = (CvPoint*)cvGetSeqElem(result, i);
      }

      //drawing lines around the heptagon
      cvLine(img, *pt[0], *pt[1], cvScalar(0,0,255),4);
      cvLine(img, *pt[1], *pt[2], cvScalar(0,0,255),4);
      cvLine(img, *pt[2], *pt[3], cvScalar(0,0,255),4);
      cvLine(img, *pt[3], *pt[4], cvScalar(0,0,255),4);
      cvLine(img, *pt[4], *pt[5], cvScalar(0,0,255),4);
```

```
            cvLine(img, *pt[5], *pt[6], cvScalar(0,0,255),4);
            cvLine(img, *pt[6], *pt[0], cvScalar(0,0,255),4);
        }

        //obtain the next contour
        contours = contours->h_next;
    }

    //show the image in which identified shapes are marked
    cvNamedWindow("Tracked");
    cvShowImage("Tracked",img);

    cvWaitKey(0); //wait for a key press

    //cleaning up
    cvDestroyAllWindows();
    cvReleaseMemStorage(&storage);
    cvReleaseImage(&img);
    cvReleaseImage(&imgGrayScale);

    return 0;
}
```

//

You can download this OpenCV visual c++ project from here. (The downloaded file is a compressed .rar folder. So, you have to extract it using Winrar or other suitable software)



As you can see, triangles are marked with blue, quadrilaterals are marked with green and heptagons are marked with red. So, now it is obvious that this method is capable of identifying shapes.

Explanation

Here I have converted the original image in to gray scale. It is because this method works only with gray scale image with single channel. To get better results, I threshold the gray-scale image using 'cvThreshold' function. You can use your own way to threshold the image. Then I find all contours in the thresholded image and identify and track all triangles, quadrilaterals and heptagons.

Let's discuss new OpenCV functions, found in this application.

- **cvThreshold(const Mat& src, Mat& dst, double threshVal, double max, int thresholdType)**

applies a fix level threshold to the each element of 'src' array write a value to corresponding array element of 'dst'

Arguements -

- const Mat& src - Source array (This should be single channel)
- Mat& dst - Destination array which has the same size and same type as the 'src'

- double threshVal - Threshold value
- double max - Maximum value to use with 'THRESH_BINARY' and 'THRESH_BINARY_INV' which are thresholding types
- int thresholdType - You can use one of the following for this arguement
  - THRESH_BINARY

    dst(x,y)=max, if src(x,y) > ThreshVal
    dst(x,y)=0, if src(x,y) < ThreshVal
  - THRESH_BINARY_INV

    dst(x,y)=0, if src(x,y) > ThreshVal
    dst(x,y)=max, if src(x,y) < ThreshVal
  - THRESH_TOZERO

    dst(x,y)=src(x,y), if src(x,y) > ThreshVal
    dst(x,y)=0, if src(x,y) < ThreshVal
  - THRESH_TOZERO_INV

    dst(x,y)=0, if src(x,y) > ThreshVal
    dst(x,y)=src(x,y), if src(x,y) < ThreshVal
  - THRESH_TRUNC

    dst(x,y)=threshVal, if src(x,y) > ThreshVal
    dst(x,y)=src(x,y), if src(x,y) < ThreshVal

In the above application, I have used 'THRESH_BINARY', because I want to assign 255 (white) where the objects are located and 0 (black) elsewhere.

- **cvCreateMemStorage(int byteSize)**

Creates memory storage which has the capacity specified by the parameter 'byteSize'. But if byteSize=0, the allocated capacity is the default value(usually 64 Kb)

- **cvFindContours(CvArr* img, CvMemStorage* str, CvSeq** first_contour, int header_size, int mode, int method, CvPoint offset)**

Find all contours in a binary image
Arguments -

- CvArr* img - Source image (This should be 8 bit single channel). All non-zero pixels are considered as 1 and all zero remain zero.
- CvMemStorage* str - Memory blocks to store all obtained contours
- CvSeq** first_contour - store a pointer to the first contour in the memory block, 'str'
- int header_size - size of the sequence header
- int mode - mode of retrieval of contours from the image

  You have to choose one of the following
  - CV_RETR_LIST - Retrieves all of the contours and put them in a list
  - CV_RETR_EXTERNAL - Retrieves only the extreme outer contours
  - CV_RETR_CCOMP - Retrieves all of the contours and organizes them into a two-level hierarchy: on the top level are the external boundaries of the components, on the second level are the boundaries of the holes
  - CV_RETR_TREE - Retrieves all of the contours and reconstructs the full hierarchy of nested contours
- int method - Approximation method

  You have to choose one of the following
  - CV_CHAIN_CODE - Outputs contours in the Freeman chain code
  - CV_CHAIN_APPROX_NONE - Translates all of the points from the chain code into points
  - CV_CHAIN_APPROX_SIMPLE - Compresses horizontal, vertical, and diagonal segments and leaves only their end points
  - CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS - Applies one of the flavors of the Teh-Chin chain approximation algorithm.
  - CV_LINK_RUNS - uses a completely different contour retrieval algorithm by linking horizontal segments of 1's. Only the 'CV_RETR_LIST' retrieval mode can be used with this method.

- CvPoint offset - Offset by which every contour point should be shifted. This is useful when we have set ROI (Region Of Interest) in the image. Normally we set the offset to 'cvPoint(0,0)'

- **cvApproxPoly(const void* src, int header_size, CvMemStorage* storage, int method, double para1, int para2)**

Approximate polygonal curves with specified precision

arguments -

- const void* src - Sequence of points
- int header_size - size of the sequence header
- CvMemStorage* storage - memory block that contains all contours
- int method - Approximation method. (The only method, available to use for this argument is 'CV_POLY_APPROX_DP')
- double para1 - approximation accuracy
- int para2 - Determines whether the single sequence should be approximated or all sequences in the same level or below

- **cvGetSeqElem(const CvSeq* seq, int index)**

Returns a pointer to the element of 'seq' at 'index'

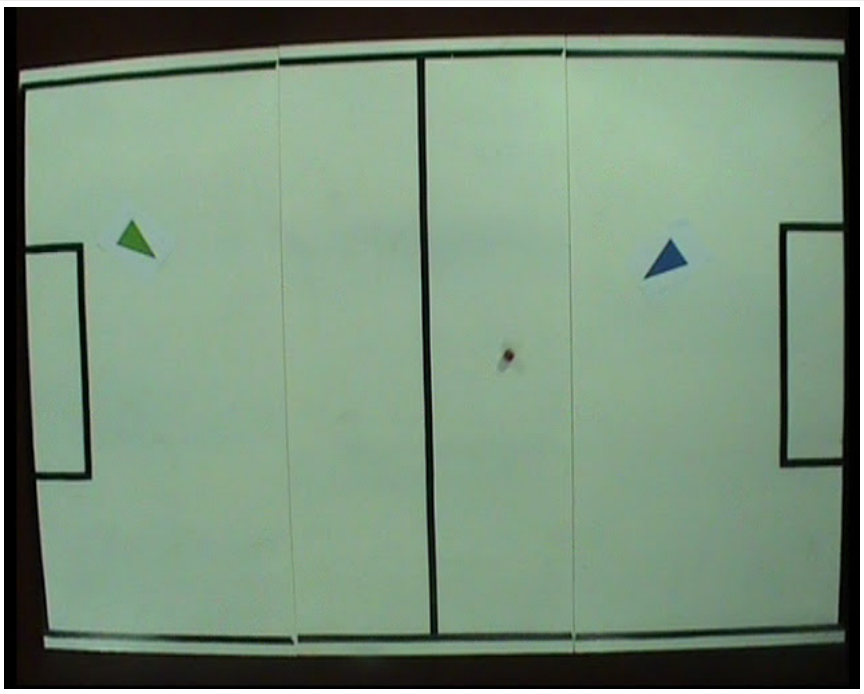
- **cvReleaseMemStorage(CvMemStorage** storage)**

Deallocate memory blocks which have been allocated by 'cvCreateMemStorage()' function

Real World Example

The above example is not really useful in practical situation. Usually, there are lots of noises in an image such as irregular lighting, shadows, camera irregularities and etc. So, above application as it is, cannot be used to identify shapes in a real image. It should be modified to cope with these noises. And images usually have 3 channels (BGR color). So, it should be converted into grey-scale which has only one channel.

Here is a real world image of an arena of a robot soccer, taken from a camera.



Here, we are going to detect and mark the perimeter of each triangle in the image with a blue line. Let's see the modified OpenCV c++ application which accomplish the above task.

//

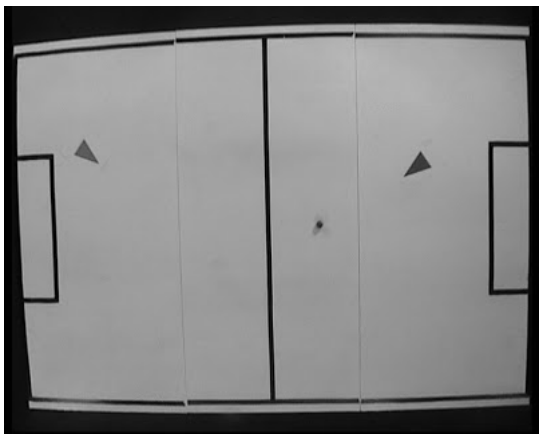
```
#include "stdafx.h"
```

```
#include <cv.h>
#include <highgui.h>
using namespace std;

int main()
{

 IplImage* img =  cvLoadImage("C:/Users/SHERMAL/Desktop/DetectingContours.jpg");

 //show the original image
 cvNamedWindow("Original");
 cvShowImage("Original",img);

 //smooth the original image using Gaussian kernel to remove noise
cvSmooth(img, img, CV_GAUSSIAN,3,3);

 //converting the original image into grayscale
 IplImage* imgGrayScale = cvCreateImage(cvGetSize(img), 8, 1);
 cvCvtColor(img,imgGrayScale,CV_BGR2GRAY);

 cvNamedWindow("GrayScale Image");
 cvShowImage("GrayScale Image",imgGrayScale);

 //thresholding the grayscale image to get better results
 cvThreshold(imgGrayScale,imgGrayScale,100,255,CV_THRESH_BINARY_INV);

 cvNamedWindow("Thresholded Image");
 cvShowImage("Thresholded Image",imgGrayScale);

 CvSeq* contour;  //hold the pointer to a contour
 CvSeq* result;   //hold sequence of points of a contour
 CvMemStorage *storage = cvCreateMemStorage(0); //storage area for all contours

 //finding all contours in the image
 cvFindContours(imgGrayScale, storage, &contour, sizeof(CvContour), CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0));

 //iterating through each contour
 while(contour)
 {
  //obtain a sequence of points of the countour, pointed by the variable 'countour'
  result = cvApproxPoly(contour, sizeof(CvContour), storage, CV_POLY_APPROX_DP, cvContourPerimeter(contour)*0.02, 0);

  //if there are 3 vertices  in the contour and the area of the triangle is more than 100 pixels
  if(result->total==3 && fabs(cvContourArea(result, CV_WHOLE_SEQ))>100 )
  {
   //iterating through each point
   CvPoint *pt[3];
   for(int i=0;i<3;i++){
    pt[i] = (CvPoint*)cvGetSeqElem(result, i);
   }

   //drawing lines around the triangle
   cvLine(img, *pt[0], *pt[1], cvScalar(255,0,0),4);
   cvLine(img, *pt[1], *pt[2], cvScalar(255,0,0),4);
   cvLine(img, *pt[2], *pt[0], cvScalar(255,0,0),4);

  }

  //obtain the next contour
  contour = contour->h_next;
 }

 //show the image in which identified shapes are marked
 cvNamedWindow("Tracked");
 cvShowImage("Tracked",img);

 cvWaitKey(0); //wait for a key press

 //cleaning up
```

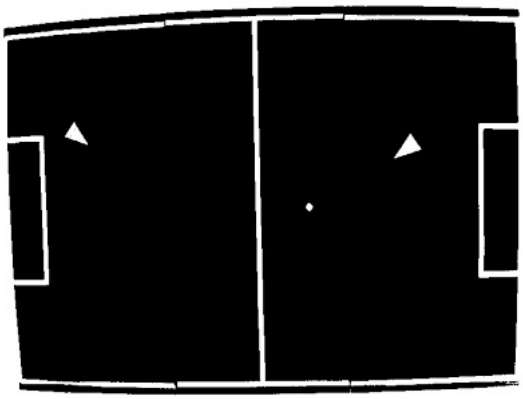
```
 cvDestroyAllWindows();
 cvReleaseMemStorage(&storage);
 cvReleaseImage(&img);
 cvReleaseImage(&imgGrayScale);

 return 0;
}
```

//

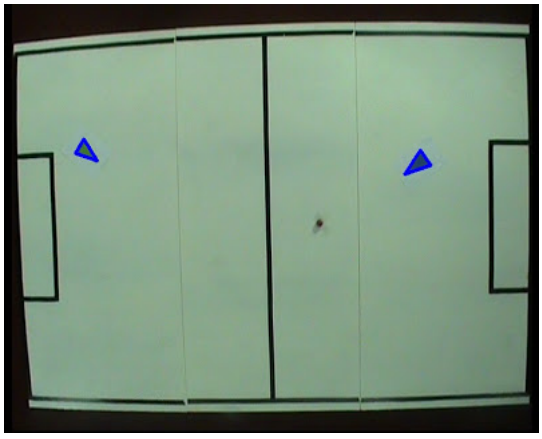
You can download this OpenCV visual c++ project from here. (The downloaded file is a compressed .rar folder. So, you have to extract it using Winrar or other suitable software)



Gray scale Image



Thresholded Image



In the same way, any shapes with any sizes can be detected with OpenCV.

Explanation

To reduce the noise level of the original image, I have smoothed the original image with a Gaussian kernel.
Further you can change the 5th argument of **cvApproxPoly()** function to cope with the noise. In the above example, I have used **cvContourPerimeter(contour)*0.02** as the 5th argument of **cvApproxPoly()**. You can try **cvContourPerimeter(contour)*0.01** or **cvContourPerimeter(contour)*0.04** or any other value and see the difference of the output yourselves.
Still there may be very small triangles, formed due to the noise. Therefore all triangles with areas less than 100 pixels are filtered out.

Here are the new OpenCV functions, found in the above example.

- **cvContourArea(const CvArr* contour, CvSlice slice)**

Calculate the area enclosed by sequence of contour points.

  - const CvArr* contour - array of vertices of the contour
  - CvSlice slice - starting and ending point of the contour. 'CV_WHOLE_SEQ' will take the whole contour to calculate the area

The orientation of contour affects the area sign. So, this function may return a negative value. So, it should be used fabs() function to get the absolute value.

- **fabs(double x)**

This function returns the absolute value of any floating point number. (This is a C function, not a OpenCV function)

Tracking two Triangles in a Video

Here I am going to track the two triangles in a video. The blue triangle is marked with red and the green triangle is marked with blue.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include <cv.h>
#include <highgui.h>
using namespace std;

IplImage* imgTracking=0;

int lastX1 = -1;
int lastY1 = -1;

int lastX2 = -1;
int lastY2 = -1;

void trackObject(IplImage* imgThresh){
      CvSeq* contour;  //hold the pointer to a contour
      CvSeq* result;    //hold sequence of points of a contour
      CvMemStorage *storage = cvCreateMemStorage(0); //storage area for all contours

       //finding all contours in the image
      cvFindContours(imgThresh, storage, &contour, sizeof(CvContour), CV_RETR_LIST,
CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0));

     //iterating through each contour
     while(contour)
     {
      //obtain a sequence of points of the countour, pointed by the variable 'countour'
     result = cvApproxPoly(contour, sizeof(CvContour), storage, CV_POLY_APPROX_DP,
cvContourPerimeter(contour)*0.02, 0);

     //if there are 3 vertices  in the contour and the area of the triangle is more than 100 pixels
```

```
        if(result->total==3 && fabs(cvContourArea(result, CV_WHOLE_SEQ))>100 )
        {
            //iterating through each point
            CvPoint *pt[3];
            for(int i=0;i<3;i++){
                pt[i] = (CvPoint*)cvGetSeqElem(result, i);
            }

             int posX=( pt[0]->x + pt[1]->x + pt[2]->x )/3;
             int posY=( pt[0]->y + pt[1]->y + pt[2]->y )/3;

             if(posX > 360 ){
                if(lastX1>=0 && lastY1>=0 && posX>=0 && posY>=0){
                   // Draw a red line from the previous point to the current point
                   cvLine(imgTracking, cvPoint(posX, posY), cvPoint(lastX1, lastY1), cvScalar(0,0,255), 4);
                }

                lastX1 = posX;
                lastY1 = posY;
             }
             else{
                if(lastX2>=0 && lastY2>=0 && posX>=0 && posY>=0){
                   // Draw a blue line from the previous point to the current point
                    cvLine(imgTracking, cvPoint(posX, posY), cvPoint(lastX2, lastY2), cvScalar(255,0,0), 4);
                 }

                  lastX2 = posX;
                  lastY2 = posY;
              }
           }

           //obtain the next contour
           contour = contour->h_next;
       }

       cvReleaseMemStorage(&storage);
}


int main(){
   //load the video file to the memory
   CvCapture *capture =    cvCaptureFromAVI("E:/Projects/Robot/IESL Robot/robot/a.avi");

   if(!capture){
      printf("Capture failure\n");
      return -1;
   }

   IplImage* frame=0;
   frame = cvQueryFrame(capture);
   if(!frame) return -1;

   //create a blank image and assigned to 'imgTracking' which has the same size of original video
   imgTracking=cvCreateImage(cvGetSize(frame),IPL_DEPTH_8U, 3);
   cvZero(imgTracking); //covert the image, 'imgTracking' to black

   cvNamedWindow("Video");

   //iterate through each frames of the video
   while(true){

      frame = cvQueryFrame(capture);
      if(!frame) break;
      frame=cvCloneImage(frame);

      //smooth the original image using Gaussian kernel
      cvSmooth(frame, frame, CV_GAUSSIAN,3,3);

     //converting the original image into grayscale
     IplImage* imgGrayScale = cvCreateImage(cvGetSize(frame), 8, 1);
```

```
    cvCvtColor(frame,imgGrayScale,CV_BGR2GRAY);

    //thresholding the grayscale image to get better results
    cvThreshold(imgGrayScale,imgGrayScale,100,255,CV_THRESH_BINARY_INV);

    //track the possition of the ball
    trackObject(imgGrayScale);

    // Add the tracking image and the frame
    cvAdd(frame, imgTracking, frame);

    cvShowImage("Video", frame);

    //Clean up used images
    cvReleaseImage(&imgGrayScale);
    cvReleaseImage(&frame);

    //Wait 10mS
    int c = cvWaitKey(10);
    //If 'ESC' is pressed, break the loop
    if((char)c==27 ) break;
  }

  cvDestroyAllWindows();
  cvReleaseImage(&imgTracking);
  cvReleaseCapture(&capture);

  return 0;
}
```

//

You can download this OpenCV visual c++ project from here. (The downloaded file is a compressed .rar folder. So, you have to extract it using Winrar or other suitable software)

Explanation

You already know how to obtain 3 vertices of a triangle with OpenCV. Averaging those 3 vertices gives you the center point of the triangle. So, it is easy to track triangles in a video.
Then, how do you identify two similar triangles separately? Here I have used a simple trick. I know that the green triangle always is in the left side of the video and the blue triangle is in the right side of the video. So, if the x coordinate of a triangle is more than (frame width)/2 = 360, then it is the blue triangle, otherwise it is the green triangle.

Next Tutorial :

Previous Tutorial : Object Detection & Tracking Using Color

Posted by Shermal Fernando

45 comments:

Cherry January 29, 2013 at 1:47 AM

thank you so much! This helped me a lot =)