# DAY 1

First, the window users installed Microsoft Visual Studio and configured it to use OpenCV library for their projects. Linux users were given [commands to install the OpenCV library](http://milq.github.io/install-opencv-ubuntu-debian/). After the configuration of environment for OpenCV, we learnt the following:

1. **Open and Display image**

The function imread(“image path”, parameter) was used to load an image, where parameter can take the value 0 and 1 to open the image in Grayscale and BGR mode respectively.

The function imshow(image) was used to display the image.

1. **Create a simple image**

The function Mat(number of rows, number of columns, CV\_8UC3, Scalar(0,0,0)) was used to create a BGR image. For a Grayscale image, CV\_8UC1 and Scalar(0) are used.

1. **Convert a BGR image to Grayscale**

This can be done in two ways:

1. By taking the average of the intensities corresponding to Blue, Red and Green channel respectively
2. Using the function cvtColor(image, CV\_BGR2GRAY)
3. **Make a tricolour image**
4. **Split mirror image**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <iostream>

using namespace cv;

using namespace std;

int main()

{

Mat img= imread("cat.jpg",1);

Mat img2= Mat(img.rows,img.cols,CV\_8UC3,Scalar(0,0,0));

int k;

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols/2;j++)

{

img2.at<Vec3b>(i,j)[0]=img.at<Vec3b>(i,j)[0];

img2.at<Vec3b>(i,j)[1]=img.at<Vec3b>(i,j)[1];

img2.at<Vec3b>(i,j)[2]=img.at<Vec3b>(i,j)[2];

}

for(int i=0;i<img.rows;i++)

{ k=1;

for(int j=img.cols/2;j<img.cols;j++)

{

img2.at<Vec3b>(i,j)[0]=img.at<Vec3b>(i,(img.cols/2)-k)[0];

img2.at<Vec3b>(i,j)[1]=img.at<Vec3b>(i,(img.cols/2)-k)[1];

img2.at<Vec3b>(i,j)[2]=img.at<Vec3b>(i,(img.cols/2)-k)[2];

k++;

}

}

imshow("Result",img2);

waitKey(0);

}

1. **Create a chessboard**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

using namespace cv;

using namespace std;

int main()

{

Mat img;

img = Mat(512,512,CV\_8UC1);

for(int i=0; i<512;i++){

for(int j=0;j<512;j++)

{ if(((i/64)+(j/64))%2==1)

img.at<uchar>(i,j)=255;

}

}

imshow("Checkerboard",img);

waitKey(0);

}

# DAY 2

1. **Convert grayscale image to binary**

#include "opencv2/opencv.hpp"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

using namespace cv;

using namespace std;

int main()

{

Mat img;

img = imread("dog.jpeg",0);

Mat img2;

img2 = Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

if(img.at<uchar>(i,j)>128)

img2.at<uchar>(i,j)=255;

else

img2.at<uchar>(i,j)=0;

namedWindow("Binary",WINDOW\_AUTOSIZE);

imshow("Binary",img2);

waitKey(0);

}

1. **Size multiplier to increase/decrease the size of the image by a multiplier**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

#include <cmath>

using namespace cv;

using namespace std;

int main()

{

float mag;

cout<<"Enter the maginification required"<<endl;

cin>>mag;

int absmag=int(sqrt(abs(mag)));

Mat img = imread("dog.jpeg");

Mat img2 = Mat(absmag\*img.rows,absmag\*img.cols,CV\_8UC3,Scalar(0,0,0));

if(mag>=0)

{

for(int i=0;i<img2.rows;i++)

for(int j=0;j<img2.cols;j++)

{

img2.at<Vec3b>(i,j)[0]=img.at<Vec3b>(i/absmag,j/absmag)[0];

img2.at<Vec3b>(i,j)[1]=img.at<Vec3b>(i/absmag,j/absmag)[1];

img2.at<Vec3b>(i,j)[2]=img.at<Vec3b>(i/absmag,j/absmag)[2];

}

}

else

{

for(int i=0;i<img2.rows;i++)

for(int j=0;j<img2.cols;j++)

{

img2.at<Vec3b>(i,j)[0]=img.at<Vec3b>((img2.rows-1-i)/absmag,(img2.cols-1-j)/absmag)[0];

img2.at<Vec3b>(i,j)[1]=img.at<Vec3b>((img2.rows-1-i)/absmag,(img2.cols-1-j)/absmag)[1];

img2.at<Vec3b>(i,j)[2]=img.at<Vec3b>((img2.rows-1-i)/absmag,(img2.cols-1-j)/absmag)[2];

}

}

imshow("Magnified",img2);

imshow("Original",img);

waitKey(0);

}

1. **Histogram to show no of pixels vs intensity of pixel for grayscale image**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

using namespace cv;

using namespace std;

int main()

{

Mat img;

img = imread("dog.jpeg",2);

long int arr[256];

float arr2[256];

Mat hist = Mat(800,256\*3,CV\_8UC1,Scalar(255));

for(int i=0;i<256;i++)

arr[i]=0;

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

arr[img.at<uchar>(i,j)]++;

/\*for(int i=0;i<256;i++)

cout<<arr[i]<<' ';\*/

long int max = arr[0];

for(int i=1;i<256;i++)

{

if(arr[i]>max)

max = arr[i];

}

/\*cout<<"max="<<max<<endl;\*/

if(max>800)

for(int i=0;i<256;i++)

arr2[i]=arr[i]\*(800.0/max);

else

for(int i=0;i<256;i++)

arr2[i]=arr[i];

for(int j=0;j<256;j++)

{

int h=799;

for(int k=0;k<arr2[j];k++)

{

hist.at<uchar>(h,j\*3)=0;

hist.at<uchar>(h,j\*3+1)=0;

h--;

}

}

imshow("Histogram",hist);

imshow("Image",img);

waitKey(0);

}

1. **Scatter plot to show the intensity vs no. of pixels of each channel for a BGR image**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

using namespace cv;

using namespace std;

int main()

{

Mat img;

img = imread("stars.jpg",1);

int arr[3][256];

float arr2[3][256];

Mat hist = Mat(800,256\*3,CV\_8UC3,Scalar(255,255,255));

for(int i=0;i<3;i++)

for(int j=0;j<256;j++)

arr[i][j]=0;

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

{

arr[0][img.at<Vec3b>(i,j)[0]]++;

arr[1][img.at<Vec3b>(i,j)[1]]++;

arr[2][img.at<Vec3b>(i,j)[2]]++;

}

int max = arr[0][0];

for(int i=0;i<3;i++)

for(int j=0;j<256;j++)

if(arr[i][j]>max)

max = arr[i][j];

if(max>800)

for(int i=0;i<3;i++)

for(int j=0;j<256;j++)

arr2[i][j]=arr[i][j]\*(800.0/max);

else

for(int i=0;i<3;i++)

for(int j=0;j<256;j++)

arr2[i][j]=arr[i][j];

for(int j=0;j<256;j++)

{

hist.at<Vec3b>(800-arr2[0][j],j\*3)[1]=0;

hist.at<Vec3b>(800-arr2[0][j],j\*3+1)[1]=0;

hist.at<Vec3b>(800-arr2[0][j],j\*3)[2]=0;

hist.at<Vec3b>(800-arr2[0][j],j\*3+1)[2]=0;

hist.at<Vec3b>(800-arr2[1][j],j\*3)[0]=0;

hist.at<Vec3b>(800-arr2[1][j],j\*3+1)[0]=0;

hist.at<Vec3b>(800-arr2[1][j],j\*3)[2]=0;

hist.at<Vec3b>(800-arr2[1][j],j\*3+1)[2]=0;

hist.at<Vec3b>(800-arr2[2][j],j\*3)[1]=0;

hist.at<Vec3b>(800-arr2[2][j],j\*3+1)[1]=0;

hist.at<Vec3b>(800-arr2[2][j],j\*3)[0]=0;

hist.at<Vec3b>(800-arr2[2][j],j\*3+1)[0]=0;

}

imshow("Histogram",hist);

imshow("Image",img);

waitKey(0);

}

1. **Using track bars to do adjust threshold in a binary image**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <iostream>

using namespace cv;

using namespace std;

int threshld;

Mat img= imread("cat.jpg",2);

Mat temp = Mat(img.rows,img.cols,CV\_8UC1);

void threshcallback(int, void\*)

{

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

if(img.at<uchar>(i,j)>threshld)

temp.at<uchar>(i,j)=255;

else

temp.at<uchar>(i,j)=0;

imshow("Display",temp);

}

int main()

{

namedWindow("Display",WINDOW\_AUTOSIZE);

createTrackbar("Threshold","Display",&threshld,255,threshcallback);

threshcallback(0,0);

waitKey(0);

}

1. **Colour extractor to extract a particular colour with tolerance track bars**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

# #include <iostream>

using namespace cv;

using namespace std;

int rlow,rhigh,blow,bhigh,glow,ghigh;

Mat img;

Mat temp;

void threshcallback(int, void\*)

{

temp = Mat(img.rows,img.cols,CV\_8UC3,Scalar(255,255,255));

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

{

if(img.at<Vec3b>(i,j)[0]>=blow && img.at<Vec3b>(i,j)[0]<=bhigh)

{

if(img.at<Vec3b>(i,j)[1]>=glow && img.at<Vec3b>(i,j)[1]<=ghigh)

{

if(img.at<Vec3b>(i,j)[2]>=rlow && img.at<Vec3b>(i,j)[2]<=rhigh)

{

temp.at<Vec3b>(i,j)[0]=img.at<Vec3b>(i,j)[0];

temp.at<Vec3b>(i,j)[1]=img.at<Vec3b>(i,j)[1];

temp.at<Vec3b>(i,j)[2]=img.at<Vec3b>(i,j)[2];

}

}

}

}

imshow("Display",temp);

imshow("Display2",img);

}

int main()

{

img = imread("a.png",1);

namedWindow("Display",WINDOW\_AUTOSIZE);

createTrackbar("r-low","Display",&rlow,255,threshcallback);

createTrackbar("r-high","Display",&rhigh,255,threshcallback);

createTrackbar("b-low","Display",&blow,255,threshcallback);

createTrackbar("b-high","Display",&bhigh,255,threshcallback);

createTrackbar("g-low","Display",&glow,255,threshcallback);

createTrackbar("g-high","Display",&ghigh,255,threshcallback);

threshcallback(0,0);

waitKey(0);

}

# DAY 3

1. **Removal of noise/Smoothing images using Median, Mean and Gaussian filters**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

using namespace cv;

using namespace std;

int main(int argc,char \*\*argv)

{

Mat img;

img = imread(argv[1],2);

float ker[3][3];

float num,sum;

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

if(((i+j)%2)==0)

ker[i][j]=1.0;

else

ker[i][j]=2.0;

ker[1][1]=4.0;

for(int i=0;i<img.rows;i++)

{

for(int j=0;j<img.cols;j++)

{ sum=0;

num=0;

int cntr=1;

int cntc=1;

for(int k=0;k<3;k++)

{

for(int l=0;l<3;l++)

{

if(i-cntr<0 || i-cntr>img.rows || j-cntc<0 || j-cntc>img.cols)

continue;

sum = sum + (img.at<uchar>(i-cntr,j-cntc)\*ker[k][l]);

num = num + ker[k][l];

cntc--;

}

cntr--;

}

img.at<uchar>(i,j) = int(sum/num);

}

}

imshow("Noise Removed",img);

waitKey(0);

}

1. **Removal of noise using Erosion and Dilation**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <iostream>

using namespace cv;

using namespace std;

Mat erosion(Mat img)

{

Mat imge = Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

imge.at<uchar>(i,j)=img.at<uchar>(i,j);

int ker[3][3];

for(int i=1;i<img.rows-1;i++)

{

for(int j=1;j<img.cols-1;j++)

{

int cntr=1;

for(int k=0;k<3;k++)

{

int cntc=1;

for(int l=0;l<3;l++)

{

ker[k][l]=img.at<uchar>(i-cntr,j-cntc);

cntc--;

}

cntr--;

}

int min=255;

for(int k=0;k<3;k++)

for(int l=0;l<3;l++)

if(ker[k][l]<min)

min = ker[k][l];

if(min==0)

imge.at<uchar>(i,j)=0;

}

}

return(imge);

}

Mat dilation(Mat imge)

{

Mat imgd = Mat(imge.rows,imge.cols,CV\_8UC1,Scalar(0));

for(int i=0;i<imge.rows;i++)

for(int j=0;j<imge.cols;j++)

imgd.at<uchar>(i,j)=imge.at<uchar>(i,j);

int ker[3][3];

for(int i=1;i<imge.rows-1;i++)

{

for(int j=1;j<imge.cols-1;j++)

{

int cntr=1;

for(int k=0;k<3;k++)

{

int cntc=1;

for(int l=0;l<3;l++)

{

ker[k][l]=imge.at<uchar>(i-cntr,j-cntc);

cntc--;

}

cntr--;

}

int max=0;

for(int k=0;k<3;k++)

for(int l=0;l<3;l++)

if(ker[k][l]>max)

max = ker[k][l];

if(max==255)

imgd.at<uchar>(i,j)=255;

}

}

return(imgd);

}

int main(int argc, char \*\*argv)

{

Mat img;

img = imread(argv[1],2);

Mat imge = erosion(img);

Mat imgd = dilation(imge);

imshow("Original",img);

imshow("Erosion",imge);

imshow("Dilation",imgd);

waitKey(0);

}

1. **Edge detection using Sobel Operator**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

#include <math.h>

using namespace cv;

using namespace std;

Mat img;

int threshld;

void threshcallback(int,void\*)

{

Mat img2 = Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

int matx[3][3] = {{-1,0,1},{-2,0,2},{-1,0,1}};

int maty[3][3] = {{-1,-2,-1},{0,0,0},{1,2,1}};

int gx,gy,g;

for(int i=1;i<img.rows-1;i++)

{

for(int j=1;j<img.cols-1;j++)

{ gy=0,gx=0;

int cntr=1;

//int cntc=1;

for(int k=0;k<3;k++)

{

int cntc=1;

for(int l=0;l<3;l++)

{

gx = gx + (img.at<uchar>(i-cntr,j-cntc)\*matx[k][l]);

gy = gy + (img.at<uchar>(i-cntr,j-cntc)\*maty[k][l]);

cntc--;

}

cntr--;

}

g=sqrt(pow(gx,2)+pow(gy,2));

if(g>threshld)

img2.at<uchar>(i,j)=255;

else

img2.at<uchar>(i,j)=0;

}

}

imshow("Display",img2);

imshow("Original",img);

waitKey(10);

}

int main(int argc,char \*\*argv)

{

img = imread(argv[1],2);

float ker[3][3];

float num,sum;

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

if(((i+j)%2)==0)

ker[i][j]=1.0;

else

ker[i][j]=2.0;

ker[1][1]=4.0;

for(int i=0;i<img.rows;i++)

{

for(int j=0;j<img.cols;j++)

{ sum=0;

num=0;

int cntr=1;

int cntc=1;

for(int k=0;k<3;k++)

{

for(int l=0;l<3;l++)

{

if(i-cntr<0 || i-cntr>img.rows || j-cntc<0 || j-cntc>img.cols)

continue;

sum = sum + (img.at<uchar>(i-cntr,j-cntc)\*ker[k][l]);

num = num + ker[k][l];

cntc--;

}

cntr--;

}

img.at<uchar>(i,j) = int(sum/num);

}

}

namedWindow("Display",WINDOW\_AUTOSIZE);

createTrackbar("Threshold","Display",&threshld,(sqrt(8\*255)),threshcallback);

threshcallback(0,0);

waitKey(0);

}

1. **Canny edge detection**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>

using namespace cv;

using namespace std;

Mat img, img\_gray;

Mat dst, edges;

int threshld;

void canny(int, void\*)

{

//edges = Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

dst = Mat(img.rows,img.cols,CV\_8UC3,Scalar(0,0,0));

blur(img\_gray,edges,Size(3,3));

Canny(edges,edges,threshld,threshld\*3,3);

img.copyTo(dst,edges);

imshow("Edges",edges);

imshow("Canny2",dst);

waitKey(5);

}

int main(int argc, char \*\*argv)

{

img = imread(argv[1],1);

img\_gray = Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

img\_gray.at<uchar>(i,j)=(img.at<Vec3b>(i,j)[0]+img.at<Vec3b>(i,j)[1]+img.at<Vec3b>(i,j)[2])/3;

namedWindow("Canny2",WINDOW\_AUTOSIZE);

createTrackbar("Threshold","Canny2",&threshld,100,canny);

canny(0,0);

waitKey(0);

}

# DAY 4

1. **Basic concepts of Stacks and Queues**
2. **Basic concepts of Graph Theory**
3. **Blob Detection using DFS and BFS**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <iostream>

#include <stack>

using namespace cv;

using namespace std;

Mat img;

struct point

{

int x;

int y;

};

void dfs(struct point p, int \*\*vis)

{

stack<struct point> s;

struct point temp,temp2;

s.push(p);

while(!s.empty())

{

temp2=s.top();

s.pop();

for(int i=-1;i<2;i++)

for(int j=-1;j<2;j++)

{ if(temp2.x+i<0 || temp2.x+i>=img.rows || temp2.y+j<0 || temp2.y+j>=img.cols)

continue;

if(img.at<uchar>(temp2.x+i,temp2.y+j)==255 && vis[temp2.x+i][temp2.y+j]==0)

{

temp.x=temp2.x+i;

temp.y=temp2.y+j;

vis[temp.x][temp.y]=1;

s.push(temp);

}

}

}

}

int main()

{

img=imread("binary.png",2);

struct point p;

int \*\*vis=new int\*[img.rows];

for(int i=0;i<img.rows;i++)

vis[i]=new int[img.cols];

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

vis[i][j]=0;

int count =0;

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

{

if(vis[i][j]==0 && img.at<uchar>(i,j)==255)

{ vis[i][j]=1;

p.x=i;

p.y=j;

dfs(p,vis);

count++;

}

}

Mat img\_dfs = Mat(img.rows,img.cols,CV\_8UC3,Scalar(0,0,0));

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

if(vis[i][j]==1)

img\_dfs.at<Vec3b>(i,j)[0]=255;

imshow("Blob Detected",img\_dfs);

imshow("Original",img);

waitKey(0);

}

1. **Hough Transform**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <iostream>

#include <stack>

#include <cmath>

#define pi 3.14

using namespace cv;

using namespace std;

int main(int argc, char \*\*argv)

{

Mat img = imread(argv[1],2);

Mat img2 = Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

int rmax=sqrt(pow(img.rows,2)+pow(img.cols,2));

int \*\*r\_t = new int \*[rmax];

for(int i=0;i<rmax;i++)

r\_t[i]= new int[181];

for(int r=0;r<rmax;r++)

for(int t=0;t<181;t++)

r\_t[r][t]=0;

for(int i=0;i<img.rows;i++)

{

for(int j=0;j<img.cols;j++)

{

if(img.at<uchar>(i,j)==255)

{

for(int t=0;t<181;t++)

{

int r=abs((i\*cos(pi/180\*t)+j\*sin(pi/180\*t)));

r\_t[r][t]++;

}

}

}

}

int thresh=80;

for(int r=0;r<rmax;r++)

for(int t=0;t<181;t++)

{

if(r\_t[r][t]>thresh)

{

int max=r\_t[r][t];

int flag=0;

for(int cntr=-8;cntr<9;cntr++)

for(int cntt=-8;cntt<9;cntt++)

{

if(r+cntr<0 && r+cntr>=img.rows && t+cntt<0 && t+cntt>=img.cols)

continue;

if(r\_t[r+cntr][t+cntt]>max)

flag=1;

}

if(flag==0)

{

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

if((int)(i\*cos(pi/180\*t)+j\*sin(pi/180\*t)-r)==0)

{

if(img.at<uchar>(i,j)==255)

img2.at<uchar>(i,j)=255;

}

}

}

}

imshow("Lines Detected",img2);

waitKey(0);

}

# DAY 5

1. **Introduction to video processing**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>

using namespace cv;

using namespace std;

int main()

{

VideoCapture cap(0);

Mat src;

while(1)

{

cap>>src;

imshow("hi",src);

waitKey(10);

}

}

1. **HSV Colour Space**
2. **Contour Detection and Approximation of the contour into a polygon**

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <vector>

using namespace cv;

using namespace std;

int main()

{

Mat img = imread("try.png",1);

Mat img\_gray = Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

img\_gray.at<uchar>(i,j)=(img.at<Vec3b>(i,j)[0]+img.at<Vec3b>(i,j)[1]+img.at<Vec3b>(i,j)[2])/3;

Mat edges;

Mat dst = Mat(img.rows,img.cols,CV\_8UC3,Scalar(0,0,0));

blur(img\_gray,edges,Size(3,3));

Canny(edges,edges,100,100\*3,3);

img.copyTo(dst,edges);

imshow("Canny",dst);

vector< vector< Point > > contours;

vector< Vec4i > hierarchy;

findContours(edges,contours,hierarchy,CV\_RETR\_TREE,CV\_CHAIN\_APPROX\_NONE);

Mat final= Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

for(int i=0;i<contours.size();i++)

for(int j=0;j<contours[i].size();j++)

{

final.at<uchar>(contours[i][j].y,contours[i][j].x)=255;

}

imshow("Contours",final);

Mat poly= Mat(img.rows,img.cols,CV\_8UC1,Scalar(0));

for(int i=0;i<contours.size();i++)

approxPolyDP(contours[i],contours[i],10,true);

/\*for(int i=0;i<contours.size();i++)

for(int j=0;j<contours[i].size();j++)

{

poly.at<uchar>(contours[i][j].y,contours[i][j].x)=255;

}\*/

RNG rng(12345);

for(int i=0;i<contours.size();i++)

{

Scalar color = Scalar(rng.uniform(0,255),rng.uniform(0,255),rng.uniform(0,255));

drawContours(poly,contours,i,color,2,8,hierarchy,0,Point());

}

imshow("Polygon",poly);

waitKey(0);

}

1. **Problem Statement 1: To implement the Fill Bucket tool in Paint**

#include "opencv2/opencv.hpp"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stack>

using namespace cv;

using namespace std;

typedef struct

{

int x;

int y;

}point;

Mat img=imread("shapes.jpg",2) ;

void dfs(point p)

{

stack<point> s;

point temp,temp2;

int i,j;

int \*\*vis=new int\*[img.rows];

for(int i=0;i<img.rows;i++)

vis[i]=new int[img.cols];

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

vis[i][j]=0;

s.push(p);

while(!s.empty())

{

temp2=s.top();

s.pop();

for(int i=-1;i<2;i++)

for(int j=-1;j<2;j++)

{ if(temp2.x+i<0 || temp2.x+i>=img.rows || temp2.y+j<0 || temp2.y+j>=img.cols)

continue;

if(img.at<uchar>(temp2.x+i,temp2.y+j)==255 && vis[temp2.x+i][temp2.y+j]==0)

{

temp.x=temp2.x+i;

temp.y=temp2.y+j;

vis[temp.x][temp.y]=1;

s.push(temp);

}

}

}

for(int i=0;i<img.rows;i++)

for(int j=0;j<img.cols;j++)

if(vis[i][j]==1)

img.at<uchar>(i,j)=100;

imshow("Display",img);

}

void CallbackFunc(int event,int y,int x, int flags, void\* userdata)

{

if(event==EVENT\_LBUTTONDOWN)

{

point p;

p.x=x;

p.y=y;

dfs(p);

}

}

int main()

{

point p;

imshow("Display",img);

setMouseCallback("Display",CallbackFunc,NULL);

waitKey(0);

}

# DAY 6

1. **Problem Statement 2: To count the number of goals of each team in the video clip of RoboSoccer match**

#include "opencv2/opencv.hpp"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stack>

using namespace cv;

using namespace std;

Mat src,match;

//int sat,val,hue,err;

/\*void threshCallback(int,void\*)

{

ball= ball-ball;

for(int i=0;i<src.rows;i++)

for(int j=0;j<src.cols;j++)

{

if(src.at<Vec3b>(i,j)[0]>=hue-err && src.at<Vec3b>(i,j)[0]<=hue+err)

{

if(src.at<Vec3b>(i,j)[1]>=sat-err && src.at<Vec3b>(i,j)[1]<=sat+err)

{

if(src.at<Vec3b>(i,j)[2]>=val-err && src.at<Vec3b>(i,j)[2]<=val+err)

{

ball.at<Vec3b>(i,j)[0]=src.at<Vec3b>(i,j)[0];

ball.at<Vec3b>(i,j)[1]=src.at<Vec3b>(i,j)[1];

ball.at<Vec3b>(i,j)[2]=src.at<Vec3b>(i,j)[2];

}

}

}

}

imshow("Dsiplay",ball);

//imshow("Dsiplay2",src);

}\*/

/\*void CallbackFunc(int event,int y,int x, int flags, void\* userdata)

{

if(event==EVENT\_LBUTTONDOWN)

{

int h=src.at<Vec3b>(x,y)[0];

int s=src.at<Vec3b>(x,y)[1];

int v=src.at<Vec3b>(x,y)[2];

cout<<h<<" "<<s<<" "<<v<<endl;

cout<<x<<" "<<y<<endl;

}

}\*/

int main()

{

VideoCapture cap("new.mp4");

int gr=0,gl=0;

int flagl=0,flagr=0,detected=0;

while(1)

{

cap>>match;

cvtColor(match,src,CV\_BGR2HSV);

int detected=0;

//ball = Mat(src.rows,src.cols,CV\_8UC3,Scalar(0,0,0));

//cvtColor(ball,ball,CV\_BGR2HSV);

//namedWindow("Display",WINDOW\_AUTOSIZE);

/\*createTrackbar("Hue","Display",&hue,255,threshCallback);

createTrackbar("Sat","Display",&val,255,threshCallback);

createTrackbar("Val","Display",&sat,255,threshCallback);

createTrackbar("Err","Display",&err,255,threshCallback);\*/

//imshow("Display",src);

//setMouseCallback("Display",CallbackFunc,NULL);

for(int i=200;i<600;i++)

for(int j=0;j<70;j++)

{

if(src.at<Vec3b>(i,j)[0]>=8-45 && src.at<Vec3b>(i,j)[0]<=8+45 && src.at<Vec3b>(i,j)[1]>=234-45 && src.at<Vec3b>(i,j)[1]<=234+45 && src.at<Vec3b>(i,j)[2]>=224-45 && src.at<Vec3b>(i,j)[2]<=224+45)

{

detected=1;

//cout<<"detected"<<endl;

}

}

if(detected==1)

{

if(flagl==0)

{

gl++;

cout<<"goal icremented"<<endl;

flagl=1;

}

else

{

//cout<<"continued"<<endl;

continue;

}

}

else

{

//cout<<"not detected"<<endl;

flagl=0;

}

for(int i=290;i<550;i++)

for(int j=1056;j<1070;j++)

{

if(src.at<Vec3b>(i,j)[0]>=8-20 && src.at<Vec3b>(i,j)[0]<=8+20 && src.at<Vec3b>(i,j)[1]>=234-20 && src.at<Vec3b>(i,j)[1]<=234+20 && src.at<Vec3b>(i,j)[2]>=224-20 && src.at<Vec3b>(i,j)[2]<=224+20)

{

if(flagr==0)

{

gr++;

flagr=1;

}

else

continue;

}

else

flagr=0;

}

cout<<gl<<" "<<gr<<endl;

imshow("Match",match);

waitKey(70);

}

}