DAY 1

- Introduction to image processing - definition of image : Collection of pixels as a 2D array containing integers as entries, types of images : Colour and grayscale. Note : '0' for black and '255' for white.
- Installed OpenCV from command line using : sudo apt-get install libopencv-dev python-opencv
- Image is a 2-D or 3-D array of pixels.
- GrayScale Image is of 1 Channel and RBG or HSV image is of 3Channel
- In Gray scale image Black is represented by 0 and white by 255.
- Declaration of image class :

  For grayscale :

  Mat <ObjectName>(<rows>,<cols>,CV_<nr_of_bits_for_a_pixel>UC<channels>,Scalar(0));

  Eg: Mat img1(50,50, CV_8UC1,Scalar(0));

  For colour image :

  Eg: Mat img2(50, 50, CV_8UC3, Scalar(0,0,0));
- The basic header files are as follows

```
#include"opencv2/highgui/highgui.hpp"
#include"opencv2/imgproc/imgproc.hpp"
#include"opencv2/core/core.hpp"
```

- Defining a Window

Eg : namedWindow("win", WINDOW_NORMAL); \\WINDOW_AUTOSIZE is also used.

For showing image in window : imshow(“win”, img1);

Use : waitKey(0); to make the window last till any key is entered by the user.

- Reading image from file:

Eg: Mat img = imread(“<img_path>”, <bol>) ; \\bol => ‘1’ for colour and ‘0’ for grayscale

- Image attributes:

img.rows => returns the number of rows of image.

img.cols => returns the number of columns of image.

- Writing an image to Disk Storage.

Eg: imwrite(“<img_path>”, <img_name>);

- Syntax for showing image
  imshow(“win”,img);

- Syntax for specifying display time
  waitKey(0);
  The image stays on screen till the user gives an input.
  For waitKey(time); time->in millisec for which the image stays on screen.

- Traversing an array
  img.at<uchar>(i,j)=255;
  img.at<Vec3b>(i,j)[0]=255;
  where Blue represents 0,Green 1 and Red 2.

- Taking image as an input.
  Mat img=imread(“ “,1/0);
  Where 1 represents color image and 0 represents black & white image

- Tasks:

Display a Yellow Screen.

Chessboard.

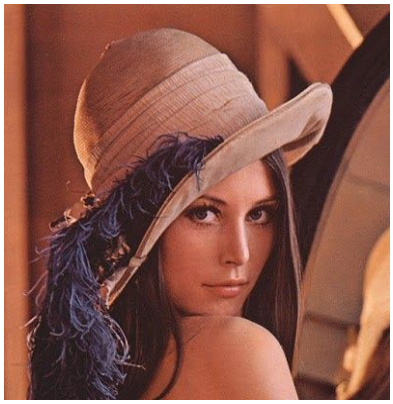
Mirror Image.

Water Reflection.

Yellow Screen:

```
#include”open CV2/highgui/highgui.hpp”
 #include”open CV2/imgproc/imgproc.hpp”
 #include”open CV2/core/core.hpp”
 int main()
 {
 Mat.img(50,50,CV_8UC3,Scalar(0,255,255));
 namedWindow(“win”,WINDO_NORMAL);
 imshow(“win”,img);
 waitKey(0);
 return 0;
}
```

Code To Print a Image in 4th Quadrant and Display its Reflection about x-axis (in 1st quadrant), y-axis (in 3rd quadrant) and about Origin (in 2nd Quadrant):



Source Image ----------> After Processing

Code:

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

using namespace cv;
using namespace std;

int main()
{ int r,c,i,j,p,q;
  Mat img=imread("sample.jpg",1);
    r=2*img.rows;
    c=2*img.cols;
  Mat img2(r,c,CV_8UC3,Scalar(255,255,255));

  for(i=r/2,p=(r/2-1);i<r;i++,p--)
      for(j=c/2,q=(c/2-1);j<c;j++,q--)
          {
                        img2.at<Vec3b>(i,j)=img.at<Vec3b>(i-r/2,j-c/2); //4th Quadrant
          img2.at<Vec3b>(p,q)=img2.at<Vec3b>(i,j);  //2nd Quadrant
          img2.at<Vec3b>(p,j)=img2.at<Vec3b>(p,q);  //1st Quadrant
          img2.at<Vec3b>(i,q)=img2.at<Vec3b>(p,q);  //3rd Quadrant
          }
          imwrite("mirror.png",img2);
          namedWindow("win",WINDOW_NORMAL);
          imshow("win",img2);
          waitKey(0);
          return 0;
}
```

Printing a Chessboard where each box is 10x10 pixel

Code:

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"


using namespace cv;

int main()
{
   Mat img(80,80,CV_8UC1,Scalar(255));
        int i=0,j=0,n=0,k;
        while(n<80)
        {
        for(i=n;i<n+10;i++)
                {
                        k=0;
                        while(k<80)
                        { if(n%20!=0)
                for(j=k-10;j<k;j++)
                                        img.at<uchar>(i,j)=0;
        else
                                for(j=k;j<k+10;j++)
                           x    img.at<uchar>(i,j)=0;


                          k+=20;
                        }
                }
                n+=10;
        }
        namedWindow("win",WINDOW_NORMAL);
   imshow("win",img);
   waitKey(0);
}
```

Image Processing - DAY 2:

DAY 2:

GIST::

Converting Colored Image to Grayscale Image

Creating A Trackbar

Converting an Image to Binary Image

Blurring an Image using different Method
1. Mean Blur

2.Median Blur

3. Gaussian Blur

Image Resizing

Image Rotation

Morphing of two images using weighted mean with a trackbar

Implementation of Vignete Filter

- The day started off with us learning about the different ways to convert a colour to a grayscale image. There are mainly 3 methods:
    - Method 1 - GScale = (R+B+G)/3;
    - Method 2 - GScale = (Max(R,B,G)+Min(R,B,G))/2;

- Method 3 - GScale= = 0.21R+0.72G+0.07B; (Eyes are most sensitive to green colour, Also this is the most accurate way as it takes into account the effect of each colour in the conversion to grayscale.)

Code To Convert Image into Grayscale using Three different method in a Menu Based Program:

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <iostream>
using namespace cv;
using namespace std;
Mat img;
int max(int a, int b, int c)
{   if(a>>b && a>>c)
      return a;
   else if(b>>c)
      return b;
   else return c;
}
int min(int a, int b, int c)
{
   if(a<<b && a<<c)
      return a;
   else if(b<<c)
      return b;
   else return c;
}
Mat process(Mat img,int k)
```

```
{
  int gray,r,g,b,i,j;
  int rows=img.rows;
  int cols=img.cols;
  Mat img2(rows,cols,CV_8UC1,Scalar(255));

  for(i=0;i<img.rows;i++)
      for(j=0;j<img.cols;j++)
      {
      b=img.at<Vec3b>(i,j)[0];
      g=img.at<Vec3b>(i,j)[1];
      r=img.at<Vec3b>(i,j)[2];

      if(k==1)
      {
            gray=(r+g+b)/3;
      img2.at<uchar>(i,j)=gray;
      }
      else if(k==2)
      {
      int p=max(r,b,g);
      int q=min(r,b,g);
      gray=(p+q)/2;
      img2.at<uchar>(i,j)=gray;
      }
      else if(k==3)
      {
```

```
      gray=0.21*r+0.72*g+0.07*b;
      img2.at<uchar>(i,j)=gray;
      }
      }
 return img2;
}
int main()
{
  int i=0,g=0,r=0,b=0,gray=0;
  Mat img2;
    Mat img=imread("sample.jpg",1);
      while(i!=10)
      {
            cout<<"What do you want to do with Image \"Sample\" ?
"<<endl
                  <<"  1- Average of R G B "<<endl
                           <<"  2- Max Min Average "<<endl
                           <<"  3- Grayscale configured formula "<<endl
                        <<"  4-Save Processed Image"<<endl<<" 5-Exit!
"<<endl;
            cin>>i;
                  if(i==1 || i==2 || i==3)
                  {
                  img2=process(img,i).clone();
                  namedWindow("win",WINDOW_NORMAL);
                  imshow("win",img2);
                  waitKey(3000);
```

```
                }
                else if(i==4)
                       {
                      imwrite("grayscale.png",img2);
                      cout<<"Image Saved"<<endl;

                        }
      }
      return 0;
 }
```

5

- Then we learnt to create a binary image, by converting all the pixels above a threshold to 255 and below a threshold to 0. We created a program using a trackbar so as to properly observe the changes caused to the images as one changes the threshold value.

Code

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <iostream>
using namespace cv;
using namespace std;

Mat img;
void update_img(int t,void *c)
{
    int i,j,a,b;
    a=img.rows;
    b=img.cols;
```

```
    Mat img1(a,b,CV_8UC1,Scalar(0));
    for(i=0;i<a;i++)
      for(j=0;j<b;j++){
           if(img.at<uchar>(i,j)>t)
                img1.at<uchar>(i,j)=255;
           else if(img.at<uchar>(i,j)<=t)
                img1.at<uchar>(i,j)=0;
      }

    imshow("My_Window",img1);
    waitKey(500);

}

int main()
{
    namedWindow("My_Window",WINDOW_NORMAL);
    int th=127;
    img=imread("Lenna.png",0);
    createTrackbar("change_it","My_Window",&th,255,update_img);
    waitKey(0);
}
```

- A Histogram was created by us that showed the number of pixels that contained a particular value in a grayscale image.

Y axis: number of pixels having values given along X-axis

CODE:

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include<iostream>
using namespace std;
using namespace cv;
```

```
int main()
{
 int a[256],k,i,j;
 Mat img=imread("sample.jpg",0);
int maxr=0;
for(i=0;i<256;i++)
   a[i]=0;
for(k=0;k<256;k++)
{
  for(i=0;i<img.rows;i++)
        for(j=0;j<img.cols;j++)
        {
                if(img.at<uchar>(i,j)==k)
                        a[k]++;
                }
        if(a[k]>maxr)
                maxr=a[k];
}
Mat img2(maxr+2,270,CV_8UC1,Scalar(255));
for(k=0;k<256;k++)
{  cout<<k<<endl<<a[k]<<endl;
 int r=maxr;
 while(a[k]!=0)
        {    img2.at<uchar>(r,k)=0;
                r--;
                a[k]--;
```

```
        }
}
        namedWindow("win",WINDOW_NORMAL);
        imshow("win",img2);
        waitKey(0);
return 0;
}
```

- Then we were taught the difference between changing the contrast (relative brightness of the pixels) and changing the brightness (Absolute brightness of the pixels). The brightness can be changed by adding a particular constant to all pixels whereas the contrast can be changed by multiplying all the pixels with a particular constant.
- We learnt how to create a rotated version of an image. And also to morph two images using a trackbar.

Morphing 2 images.

….

```
Mat img;
Mat img1;
void update_img(int t,void *c)
{
    int i,j,a,b;
    img=imread("index.jpg",1);
    img1=imread("tig.jpg",1);

    a=img.rows;
    b=img.cols;
    Mat img3(a,b,CV_8UC3,Scalar(0,0,0));
    for(i=0;i<a;i++)
       for(j=0;j<b;j++){

img3.at<Vec3b>(i,j)[0]=(img1.at<Vec3b>(i,j)[0]*t+img.at<Vec3b>(i,j)[0
]*(100-t))/100;

img3.at<Vec3b>(i,j)[1]=(img1.at<Vec3b>(i,j)[1]*t+img.at<Vec3b>(i,j)[1
]*(100-t))/100;

img3.at<Vec3b>(i,j)[2]=(img1.at<Vec3b>(i,j)[2]*t+img.at<Vec3b>(i,j)[2
]*(100-t))/100;
    }
    imshow("My_Window",img3);
    waitKey(500);

}

int main()
{
    namedWindow("My_Window",WINDOW_NORMAL);
    int th=0;
    createTrackbar("change_it","My_Window",&th,100,update_img);
    waitKey(0);
    return 0;
}
```

- Scaling up and down an image was taught. Then applying them one after another we could obtain fractional scalings as well. Scaling down leads to a loss of information, thus it is better to scale up first and then scale down.

We use two methods

(i)Scale down and then scale up-Loss of pixels and image quality decreases

(ii)Scale up and then scale down-Better method

For scaling it down what we do is that find the average brightness of n^2 pixels and then provide it to a pixel.

For scale up we take take the average of n^2 pixels and then provide it to m^2 pixels. (m>n)

CODE:

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include<iostream>
using namespace std;
using namespace cv;

Mat scaleup(Mat img1, int ufac)
{ int i,j,p,q;
    Mat
img2(img1.rows*ufac,img1.cols*ufac,CV_8UC3,Scalar(255,255,255));
   for(i=0;i<img1.rows*ufac;i+=ufac)
       for(j=0;j<img1.cols*ufac;j+=ufac)
       {
            for(p=0;p<ufac;p++)
```

```
                for(q=0;q<ufac;q++)

img2.at<Vec3b>(i+p,j+q)=img1.at<Vec3b>(i/ufac,j/ufac);
      }
      return img2;
}

Mat scaledown(Mat img1, int dfac)
{ int i,j,p,q,x[3];
    Mat img2(img1.rows,img1.cols,CV_8UC3,Scalar(255,255,255));

    for(i=0;i<img1.rows;i+=dfac)
      for(j=0;j<img1.cols;j+=dfac)
      {x[0]=0;
           x[1]=0;
           x[2]=0;
           for(p=0;p<dfac;p++)
                {
                     for(q=0;q<dfac;q++)
                    {
                          x[0]+=img1.at<Vec3b>(i+p,j+q)[0];
                          x[1]+=img1.at<Vec3b>(i+p,j+q)[1];
                          x[2]+=img1.at<Vec3b>(i+p,j+q)[2];
                    }
                }
          x[0]/=dfac*dfac;
          x[1]/=dfac*dfac;
          x[2]/=dfac*dfac;
```

```
            img2.at<Vec3b>(i/dfac,j/dfac)[0]=x[0];
            img2.at<Vec3b>(i/dfac,j/dfac)[1]=x[1];
            img2.at<Vec3b>(i/dfac,j/dfac)[2]=x[2];
       }
       return img2;
}
int main()
{
     Mat img1=imread("sample.jpg",1);
      int dfac,ufac;
    cout<<"Scale UP Factor : \n";
    cin>>ufac;
    cout<<"Scale Down Factor : \n";
    cin>>dfac;

     Mat
img3(img1.rows*ufac,img1.cols*ufac,CV_8UC3,Scalar(255,255,255));
      img3=scaleup(img1,ufac).clone();
      Mat
img2(img1.rows*ufac/dfac,img1.cols*ufac/dfac,CV_8UC3,Scalar(255,255,2
55));
      img2=scaledown(img3,dfac).clone();
      imshow("win",img2);
      waitKey(0);
      return 0;
}
```

- Then we learnt about blurs,i.e., Mean, Median, Gaussian blur. The most effective in general is the gaussian blur, however for salt and pepper type disturbed images we use median blur to get the best effects. We were also taught the uses of each blur.

Noise is the irregularities in the image present.To remove it
we employ-
(i)Mean
This is used when the pixels have almost about the same
value of brightness.
Any three cross three matrix is considered and the average
of the brightness is provided to the central pixel.
Ex

12 18 17
15 20 8
10   6   9

(ii)Median
This is used when the pixels have very few pixels having
much much different value of brightness than the rest all.

Ex
12 17   11
14 10   8
220 200 16

(iii)Gaussean Method

This is the best method.Because what we do here is sort of weighted mean because the contribution of a pixel to the central pixel should be different.

Weights of each pixel

1/16 1/8 1/16

1/8 1/4 1/8

1/16 1/8 1/16

Note: For each pixel we apply a isValid() which tells if the pixel actually lies in the picture or not.

CODE:

blur an image using Mean blur.

```
….
Mat img;
int isValid(int a,int b)
{
    if(a>=0 && a<img.rows && b>=0 && b<img.cols)
      return 1;
    else
      return 0;
}
int main()
{
    int i,j,z1=0,z2=0,z3=0,x,y,c;
    img=imread("veg.jpg",1);
    namedWindow("win",WINDOW_NORMAL);
    Mat img1(img.rows,img.cols,CV_8UC3,Scalar(0,0,0));
    for(i=0;i<img.rows;i++){
      for(j=0;j<img.cols;j++){c=z1 = z2 =z3 =0;
           for(x=i-1;x<=i+1;x++){
                    for(y=j-1;y<=j+1;y++){
```

```
                              if(isValid(x,y)==1){c++;
                              z1=z1+img.at<Vec3b>(x,y)[0];
                              z2=z2+img.at<Vec3b>(x,y)[1];
                              z3=z3+img.at<Vec3b>(x,y)[2];
                        }
                        }
                  }
            img1.at<Vec3b>(i,j)[0]=z1/c;
            img1.at<Vec3b>(i,j)[1]=z2/c;
            img1.at<Vec3b>(i,j)[2]=z3/c;
      }}
     imshow("win",img1);
    waitKey(0);
    return 0;
}
```

blur an image using Median blur.

```
….
Mat img;
int isValid(int a,int b)
{
    if(a>=0 && a<img.rows && b>=0 && b<img.cols)
      return 1;
    else
      return 0;
}
int sort(int a[],int n)
{
    int i,j,k;
    for(i=0;i<n;i++)
    {
      k=a[i];
      j=i-1;
      while(j>=0 && a[j]>k)
      {
            a[j+1]=a[j];
            j=j-1;
      }
      a[j+1]=k;
    }
    return a[4];
}
```

```
int main()
{
    int i,j,z1[9],z2[9],z3[9],x,y,c;
    img=imread("Lenna.png",1);
    namedWindow("win",WINDOW_NORMAL);
    Mat img1(img.rows,img.cols,CV_8UC3,Scalar(0,0,0));
    for(i=0;i<img.rows;i++){
      for(j=0;j<img.cols;j++){c=0;
            for(x=i-1;x<=i+1;x++){
                        for(y=j-1;y<=j+1;y++){
                              if(isValid(x,y)==1){
                              z1[c]=img.at<Vec3b>(x,y)[0];
                              z2[c]=img.at<Vec3b>(x,y)[1];
                              z3[c]=img.at<Vec3b>(x,y)[2];
                              c++;
                        }
                        }
                 }
            img1.at<Vec3b>(i,j)[0]=sort(z1,9);
            img1.at<Vec3b>(i,j)[1]=sort(z2,9);
            img1.at<Vec3b>(i,j)[2]=sort(z3,9);
                 img1.at<Vec3b>(i,j)[2]=sort(z3,9);
      }}

    imshow("win",img1);
    waitKey(0);
    return 0;
}
```

blur an image using Gaussian blur.

```
     …...
    Int isValid()...
    int i,j,z1=0,z2=0,z3=0,x,t,t1,y;
    img=imread("Lenna.png",1);
    namedWindow("win",WINDOW_NORMAL);
    Mat img1(img.rows,img.cols,CV_8UC3,Scalar(0,0,0));
    for(i=0;i<img.rows;i++){
      for(j=0;j<img.cols;j++){z1 = z2 =z3=0;
            Mat img2(3,3,CV_8UC3,Scalar(0,0,0));
            for(x=i-1;x<=i+1;x++){
                 for(y=j-1;y<=j+1;y++){
                        if(isValid(x,y)==1){
                              if(x==i && y==j){
```

```
                                    z1=z1+img.at<Vec3b>(x,y)[0]*1/4;
                                    z2=z2+img.at<Vec3b>(x,y)[1]*1/4;
                                    z3=z3+img.at<Vec3b>(x,y)[2]*1/4;
                                }
                                else if((y==j && x!=i)||(x==i && y!=j)){
                                    z1=z1+img.at<Vec3b>(x,y)[0]*1/8;
                                    z2=z2+img.at<Vec3b>(x,y)[1]*1/8;
                                    z3=z3+img.at<Vec3b>(x,y)[2]*1/8;
                                }
                                else{
                                    z1=z1+img.at<Vec3b>(x,y)[0]*1/16;
                                    z2=z2+img.at<Vec3b>(x,y)[1]*1/16;
                                    z3=z3+img.at<Vec3b>(x,y)[2]*1/16;
                                    }
                            }
                        }
                    }
            img1.at<Vec3b>(i,j)[0]=z1;
            img1.at<Vec3b>(i,j)[1]=z2;
            img1.at<Vec3b>(i,j)[2]=z3;
          }}
   imshow("win",img1);
    waitKey(0);
    return 0;
}
```

Day 3

(1) Started with various methods of implementation of gaussian blur like border, basic wrap round, reflection.
For the corners and edges int the images, there
methods are-
(i)Padding
(ii)Wrap Around
(iii)Reflection
Padding and Reflection are the ones which are used the

most.

(2) Introduction to image segmentation using colour extraction, border detection, etc.
Color Extraction-One of the components of segmentation
In this method what we do is define the maximum and
minimum for the R,G,B. According to that
the one which lie in the range are colored white and the
background is colored as black.For this we make six
trackbars for setting the minimum and the maximum for
each of the shade and we extract the component of the
image in that by changing the minimum and the maximum of
the R,G,B(the one which will lie in the range can be
extracted.)

:: :: Task : Extraction of colours from an image using six trackbars. :: ::

(3) Introduction to edge detection filters : two types - sobel filter and prewitt filter.

Each and every element of a kernel is multiplied by the corresponding elements in a specified matrix. Calculation of mean gradient and comparison with the threshold value should return either '0' or 255.

Prewitt filter

In Prewitt filter the Kernel is multiplied with the values as follows:

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

Sobel filter

In sobel filter the Kernel is multiplied with the values as follows:

$$gy = \frac{1}{8}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \qquad gx = \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Value =sqrt(gy^2+gx^2)

Or approx value=|gx+gy|

(4) To remove noise after edge detection : using erosion - dilation.

Erosion : If there is any black pixel in the kernel, make the centre pixel black. (Used to remove white spots)

Dilation : If there is any white pixel in the kernel, make the centre pixel white. (Used to remove black spots)

(5) Using canny library for edge detection :

Processes done :

(0) Gaussian blur.

(0) Sobel/Prewitt edge detection (By default - sobel filter)

(0) Non-maximum suppression.

(0) Thresholding(L+H).

Note : H/L ratio usually lies between 2 and 3.

CODE:

Color Extraction in RGB

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

using namespace std;
using namespace cv;

Mat img=imread("a.png",1);
int bu,bl,gu,gl,ru,rl;

void convtbinary(int t, void *a)
{
   Mat img2(img.rows,img.cols,CV_8UC1,Scalar(0));
 int i,j,x[3];
 for(i=0;i<img.rows;i++)
        {
        for(j=0;j<img.cols;j++)
        {
        x[0]=img.at<Vec3b>(i,j)[0];
        x[1]=img.at<Vec3b>(i,j)[1];
        x[2]=img.at<Vec3b>(i,j)[2];
```

```
if((x[0]<=bu)&&(x[0]>=bl)&&(x[1]<=gu)&&(x[1]>=gl)&&(x[2]<=ru)&&(x[2]>=
rl))
        {
                img2.at<uchar>(i,j)=255;
        }



        }
        }

        imshow("win2",img2);
        waitKey(500);
}

int main()
{
        namedWindow("win2",WINDOW_NORMAL);

        int th=127;
        createTrackbar("blue_upper","win2",&bu,255,convtbinary);
        createTrackbar("blue_lower","win2",&bl,255,convtbinary);
        createTrackbar("green_upper","win2",&gu,255,convtbinary);
        createTrackbar("green_lower","win2",&gl,255,convtbinary);
        createTrackbar("red_upper","win2",&ru,255,convtbinary);
        createTrackbar("red_lower","win2",&rl,255,convtbinary);

        waitKey(0);
        return 0;
 }
```

CODE: EDGE Detection: Sobel

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

```
#include<math.h>

using namespace std;
using namespace cv;

Mat img=imread("rubik.jpg",0);

int th;

int isValid(int i,int j)
{
 if(i<0||j<0||i>(img.rows-1)||j>(img.cols-1))
        return 0;
 else
        return 1;
}


void edge(int t, void *a)
{
 Mat img2(img.rows,img.cols,CV_8UC1,Scalar(0));
  int gx,gy,gf;
        for(int i=0;i<img.rows;i++)
        for(int j=0;j<img.cols;j++)
        {
        gx=0;
        gy=0;
                for(int p=i-1;p<=i+1;p++)
                for(int q=j-1;q<=j+1;q++)
                        if(isValid(p,q))
                        {

                        if(p==i-1 && q==j-1)
                        { gy+=(-1)*(img.at<uchar>(p,q));
                        gx+=(-1)*(img.at<uchar>(p,q));
                        }
                        else if(p==i-1 && q==j)
                        {
                                gy+=(-2)*(img.at<uchar>(p,q));
```

```
gx+=(0)*(img.at<uchar>(p,q));
}
else if(p==i-1 && q==j+1)
{
	gy+=(-1)*(img.at<uchar>(p,q));
gx+=(1)*(img.at<uchar>(p,q));
}
else if(p==i && q==j-1)
{
	gy+=(0)*(img.at<uchar>(p,q));
gx+=(-2)*(img.at<uchar>(p,q));
}
else if(p==i && q==j)
{
	gy+=(0)*(img.at<uchar>(p,q));
gx+=(0)*(img.at<uchar>(p,q));
}
else if(p==i && q==j+1)
{
	gy+=(0)*(img.at<uchar>(p,q));
gx+=(2)*(img.at<uchar>(p,q));
}
else if(p==i+1 && q==j-1)
{
	gy+=(1)*(img.at<uchar>(p,q));
gx+=(-1)*(img.at<uchar>(p,q));
}
else if(p==i+1 && q==j)
{
	gy+=(2)*(img.at<uchar>(p,q));
gx+=(0)*(img.at<uchar>(p,q));
}
else if(p==i+1 && q==j+1)
{
	gy+=(1)*(img.at<uchar>(p,q));
gx+=(1)*(img.at<uchar>(p,q));
}
}
gx/=4;
```

```
        gy/=4;

        gf=sqrt(gx*gx+gy*gy);

        if(gf>=th)
        {       cout<<gf<<endl;
                        cout<<"the threshold "<<th<<endl;
                img2.at<uchar>(i,j)=255;
        }

        }
 imwrite("edgetemp.png",img2);
 cout<<"image saved"<<endl;

 imshow("win",img2);
}


int main()
{
  namedWindow("win",WINDOW_NORMAL);

  th=127;
  createTrackbar("change_it","win",&th,255,edge);

  waitKey(0);
  return 0;
}
```

CODE: ERROSION DILATION

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

using namespace std;
```

```
using namespace cv;

Mat img=imread("edgetemp.png",0);
Mat img4(img.rows,img.cols,CV_8UC1,Scalar(0));
Mat img5(img.rows,img.cols,CV_8UC1,Scalar(0));

int th;

int isValid(int i,int j)
{
 if(i<0||j<0||i>(img.rows-1)||j>(img.cols-1))
        return 0;
 else
        return 1;
}


void errosion()
{

 int c;
        for(int i=0;i<img.rows;i++)
        for(int j=0;j<img.cols;j++)
        {
                for(int p=i-1;p<=i+1;p++)
                for(int q=j-1;q<=j+1;q++)
                        if(isValid(p,q))
                        {
                                if(img.at<uchar>(p,q)==0)
                                        c++;
                        }
                if(c>0)
                {
                 img4.at<uchar>(i,j)=0;
                }
                c=0;
        }
}
```

```
void dilation()
{

 int c;
        for(int i=0;i<img.rows;i++)
        for(int j=0;j<img.cols;j++)
        {
        for(int p=i-1;p<=i+1;p++)
                for(int q=j-1;q<=j+1;q++)
                        if(isValid(p,q))
                        {
                                if(img4.at<uchar>(p,q)==255)
                                        c++;
                        }
        if(c>0)
                {
                 img5.at<uchar>(i,j)=255;
                }
                c=0;
        }
}


int main()
{
 imshow("original",img);
 img4=img.clone();
 errosion();
 img5=img4.clone();
 imshow("after_errosion",img4);
 dilation();
 imshow("after_dilation(after_errosion)",img5);

        waitKey(0);
        return 0;
}
```

Day 4

(1) Started with two types of calling erosion and dilation based on their order of execution in order to remove noice. They are :

(.) Morphological opening. (erosion, then dilation)

(.) Morphological closing. (dilation, then erosion)

(2) Introduction to graphs :

Graph is the combination of the nodes(pixels) and the edges

connecting the node.

Vertice(node) - representation of each pixel.

Edge - line joining two adjacent vertices.

Edge weight - number denoting how difficult is the traversal.

(3) Path traversal methods in graph :

(.) BFS (Breadth First Search)

(.) DFS (Depth First Search)

DFS: In this method we move in only one path ,i.e., starting from

the parent we move from one child to the other and cover

the whole image.
After the path is over, we then backtrack and see if any part had multiple child and then move in that path.Here we do not move along all the children at a time unlike the BFS.
Remember while backtracking the node where we get a unvisited child first we cover that first.
Mark the node Visited and search for Unvisited Nodes.
It can be executed using i-Recursion or ii-using Stacks

Usage : to extract features of an image like number of shapes,etc.

Pseudo code for DFS :

```
DFS(start_node) {
        Mark start_node as visited.

        if (Neighbours of start_node have been visited) return;

        for (each neighbour) DFS(neighbour)
}
```

BFS: In this method what we do is move layer wise ,i.e., starting from the parent we move along all the child of the nodes.
We have to move along all the nodes of a parent, visit all the childs for a layer of node in the order in which they are written.
Implemented Using Queues

(4) Implementation of setMouseCallback to get actions of mouse.

Eg: setMouseCallback("win",onMouse,0);

void onMouse(int event, int y, int x, int flag, void *a) {}

List of events : EVENT_LBUTTONDOWN, EVENT_RBUTTONDOWN, EVENT_LBUTTONUP, EVENT_RBUTTONUP, etc.

(5) Introduction to stacks and queues. Implementation of BFS using queue.

Psuedo Code:

```
push(start_node);

If (queue is not empty) {
        curr = front of queue;
        Pop the last element;

        for (all (v,w) pairs) {
                if (w is not visited)
                        mark w as visited.
                        push w to queue.
        }
}
```

(6) Introduction to linked list.

Syntax :

```
struct node {
        int info;
        struct node* next;
}*head;
```

head.info => returns the integer value of info.

head.next => returns the address of the next node.

(7)

```
void onMouse(int event,int y,int x,int flag,void *a)
{........}
setMouseCallback("window_name",onMouse,0)
```

The setMouseCallback function is used to bring the changes in the image on different actions done by the mouse.

event- This is the one which stores the action of the image.

Y-Stores the y-coordinate

x-Stores the x-coordinate

((Be careful here the x and y axis are different from that of the image))

Horizontal one is x and y is the vertical one.

*setMouseCallback function will be called in the main function.

For the events, for example

EVENT_LBUTTONDOWN-Left click

EVENT_RBUTTONDOWN-Right click

8) Stack: It is an abstract data structure based on LIFO (last in first out). When stack is full, it is called stack overflow. It's functions are:

a. **top**(): initialized as -1, it returns a reference to the top element, i.e. the one last entered in the stack.
b. **push**(**i**): it inserts a new element, i at the top of the stack
c. **pop**(): it removes the top element from the stack
d. **isEmpty**(): it returns boolean true or 1 if the stack is empty or the value of top is -1 and boolean false or 0 otherwise

9) Queue: It is also an abstract data structure which is however based on FIFO (first in first out). It's functions are:

e. isEmpty() – Returns whether the queue is empty
f. size() – Returns the size of the queue
g. front() – Returns a reference to the first element of the queue
h. back() – Returns a reference to the last element of the queue
i. **enQueue**(**i**) – Adds the element 'g' at the end of the queue
j. **deQueue**() – Deletes the first element of the queue

10)

Dijkstra(Shortest path algorithm)

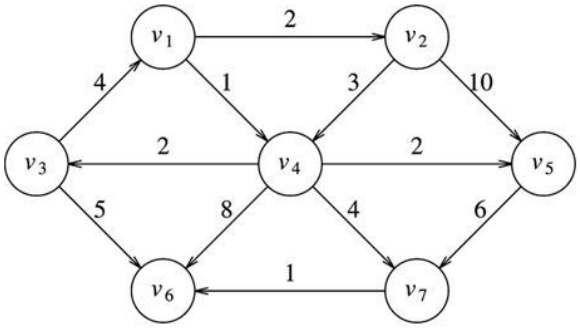
It is the algorithm of finding the shortest path from starting

node to any node according to the nodes. The path with the

minimum weight gives the shortest path.

we create three lists for each node namely open, closed and parent.

Dijkstra's Algorithm - Example



v	*known*	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

v	*known*	d_v	p_v
▶ v_1	T	0	0
v_2	F	2	v_1
v_3	F	∞	0
v_4	F	1	v_1
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	3	v_4
▶ v_4	T	1	v_1
v_5	F	3	v_4
v_6	F	9	v_4
v_7	F	5	v_4

v	*known*	d_v	p_v
v_1	T	0	0
▶ v_2	T	2	v_1
v_3	F	3	v_4
v_4	T	1	v_1
v_5	F	3	v_4
v_6	F	9	v_4
v_7	F	5	v_4

Another path planning algorithm - A star, which is a greedy algorithm.

CODE:

1.Identifying no of objects using DFS

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <iostream>
using namespace cv;
using namespace std;

Mat img;
Mat img1;
int isValid(int a,int b)
{
   if(a>=0 && a<img.rows && b>=0 && b<img.cols)
         return 1;
   else
         return 0;
}
int cnt=0;
void DFS(int i, int j)
{
   int x,y;
   img.at<uchar>(i,j)=200;
   img1.at<uchar>(i,j)=200;
   for(x=i-1;x<=i+1;x++){
         for(y=j-1;y<=j+1;y++){
               if(isValid(x,y)==1){
                     if(img.at<uchar>(x,y)<130 ) {
                                 DFS(x,y); }
                     }
         }
   }
   return;
}

int main()
{
   int c=0,i,j;
   img=imread("dfs.jpg",0);
   img1=Mat(img.rows,img.cols,CV_8UC1,Scalar(0));
```

```
   for(i=0;i<img.rows;i++)
   {
        for(j=0;j<img.cols;j++){
            if(img.at<uchar>(i,j)<130){
                DFS(i,j);
                c++;
            }
        }
}
   cout<<"the no of blocks are"<<c;
   namedWindow("win",WINDOW_NORMAL);
   imshow("win",img);
   imshow("win1",img1);
   waitKey(0);
   return 0;
}
```

2.Making different objects different colour

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <iostream>
using namespace cv;
using namespace std;

Mat img;
Mat img1;
int in=50;
int isValid(int a,int b)
{
   if(a>=0 && a<img.rows && b>=0 && b<img.cols)
        return 1;
   else
        return 0;
}
int cnt=0;
void DFS(int i, int j)
{
```

```
    int x,y;
    img.at<uchar>(i,j)=200;
    img1.at<uchar>(i,j)=in;
    for(x=i-1;x<=i+1;x++){
            for(y=j-1;y<=j+1;y++){
                    if(isValid(x,y)==1){
                            if(img.at<uchar>(x,y)<130 )
                                    {
                                            DFS(x,y);
                                    }
                            }
            }
    }
    return;
}

int main()
{
    int c=0,i,j;
    img=imread("dfs.jpg",0);
    img1=Mat(img.rows,img.cols,CV_8UC1,Scalar(0));

    //cout<<"r,c = "<<img1.rows<<","<<img1.cols<<endl;
    for(i=0;i<img.rows;i++)
    {
            for(j=0;j<img.cols;j++){
                    if(img.at<uchar>(i,j)<130){
                            DFS(i,j);
                            c++;
                            in=in+50;
                    }
            }
  }
    cout<<"the no of blocks are"<<c;
    namedWindow("win",WINDOW_NORMAL);
    imshow("win",img);
    imshow("win1",img1);
    waitKey(0);
    return 0;
}
```

3.Paint-fill colour by clicking

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <iostream>
using namespace cv;
using namespace std;

Mat img;

int isValid(int a,int b)
{
   if(a>=0 && a<img.rows && b>=0 && b<img.cols)
         return 1;
   else
         return 0;
}

void DFS(int i, int j,int in)
{
   int x,y;
   img.at<uchar>(i,j)=in;
   for(x=i-1;x<=i+1;x++){
         for(y=j-1;y<=j+1;y++){
               if(isValid(x,y)==1){
                     if(img.at<uchar>(x,y)>240 )
                           {
                                 DFS(x,y,in);

                           }
                     }
         }
   }
   return;
}

void onMouse(int event,int y,int x,int flag,void*c)
{
   if(event==1)
         DFS(x,y,200);
   if(event==2)
         DFS(x,y,50);
```

```
    imshow("win",img);
}

int main()
{
    int c=0,i,j;
    img=imread("shapes.jpg",0);
    namedWindow("win",WINDOW_NORMAL);
    setMouseCallback("win",onMouse,0);
    waitKey(0);
    return 0;
}
```

4.Identifying no of objects using BFS

.....

```
Mat img;
int c = 0;
int isValid(int a,int b)
{
    if(a>=0 && a<img.rows && b>=0 && b<img.cols)
       return 1;
    else
       return 0;
}
void bfs(int i,int j)
{
    int x,y;
    queue gq;
    queue g;
    gq.push(i);
    g.push(j);
    img.at<uchar>(i,j)=200;
    while(!gq.empty() && !g.empty()){

       int v1=gq.front();
       int v2=g.front();
       gq.pop();
       g.pop();
            for(x=v1-1;x<=v1+1;x++){
            for(y=v2-1;y<=v2+1;y++){
```

```
                if(isValid(x,y)==1){
                      if(img.at<uchar>(x,y)<180){
                            img.at<uchar>(x,y)=200;
                gq.push(x);
                g.push(y);
            }
            }
       }
       }
    }
}

int main()
{
    int i,j;
    img=imread("dfs.jpg",0);

    for(i=0;i<img.rows;i++)
    {
       for(j=0;j<img.cols;j++){
            int y =img.at<uchar>(i,j);
            if(y<180){
                bfs(i,j);
                int u =img.at<uchar>(i,j);
                c++;
            }


       }

    }
    cout<<"the no of blocks are"<<c;
    namedWindow("win",WINDOW_NORMAL);
    imshow("win",img);
    waitKey(0);
    return 0;
}
```

Day 5 and 6

1. Hough Transform

Suppose we have a image on which canny has been applied.
Now that book would be black and white and would have deformations like a straight will not be complete, it'll have spaces in between , it would be broken.

To fix that we need to draw a line which passes through maximum number of those points.

To draw that line hough transform is done.

$$x\cos(\theta) + y\sin(\theta) = r$$

That's a general equation of line.
What we do is, for every point on that line after applying canny , we vary theta and get the corresponding value of r and then we plot a graph of θ *vs* *r.*
Now the idea is that which ever line passes through maximum number of points will have the same value of θ and r and hence we'll get some sort of maxima in the graph plotted.

2. That's a hough transform applied on a pentagon, that's we have 5 maximas each giving the value of θ and r and hence the equation of line.
3. Mouse Callback Function

```
setMouseCallback("win",onMouse,0);

void onMouse(int event,int y,int x, int flag ,void *a)
{
int z;
if(event==1)                    // Left mouse click
z=f(x,y,75);
else if(event==2)               // Right mouse click
z=f(x,y,125);
imshow("win",img1);
}
```

Program to fill the white objects in a black & white image with different shades of grey using mouse click

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

using namespace cv;
using namespace std;

Mat img=imread("newblop.png",0);
Mat img1(img.rows,img.cols,CV_8UC1,Scalar(0));
int isvalid(int a, int b)
{
if(a>=0&&a<img.rows&&b>=0&&b<img.cols)
return 1;
else return 0;
}
int dfs(int a, int b , int c)
{
int m,n,p;
if(img1.at<uchar>(a,b)<240)
return 0;
else
img1.at<uchar>(a,b)=c;
for(m=a-1;m<=a+1;m++)
for(n=b-1;n<=b+1;n++)
if(isvalid(m,n))
p=dfs(m,n,c);
return 1;
}
```

```
void onMouse(int event,int y,int x, int flag ,void *a)
{
int z;
if(event==1)
z=dfs(x,y,75);
else if(event==2)
z=dfs(x,y,125);
imshow("win",img1);
}

int main()
{
int i,j;
for(i=0;i<img.rows;i++)
for(j=0;j<img.cols;j++)
img1.at<uchar>(i,j)=img.at<uchar>(i,j);
namedWindow("win",WINDOW_NORMAL);
setMouseCallback("win",onMouse,0);
waitKey(0);
}
```

4. **Contour Detection**

Contours are basically an outline representing or bounding the shape or form of something eg. in this case similar pixels.

There is an in built function for this *findContours .*

findContours(image,contours,hierarchy,mode,method);
Where:
Contours : vector<vector<Point>>
Hierarchy : vector<Vec 4i>
Mode : This is of various types:
A. RETR_EXTERNAL (only external contours)
B.RETR_LIST (provides list of contours without hierarchy)

C.RETR_COMP (only the outer and it's two children are considered,rest external)
D.RETR_TREE (provides list of contours with hierarchy)
E.RETR_FLOODFILL

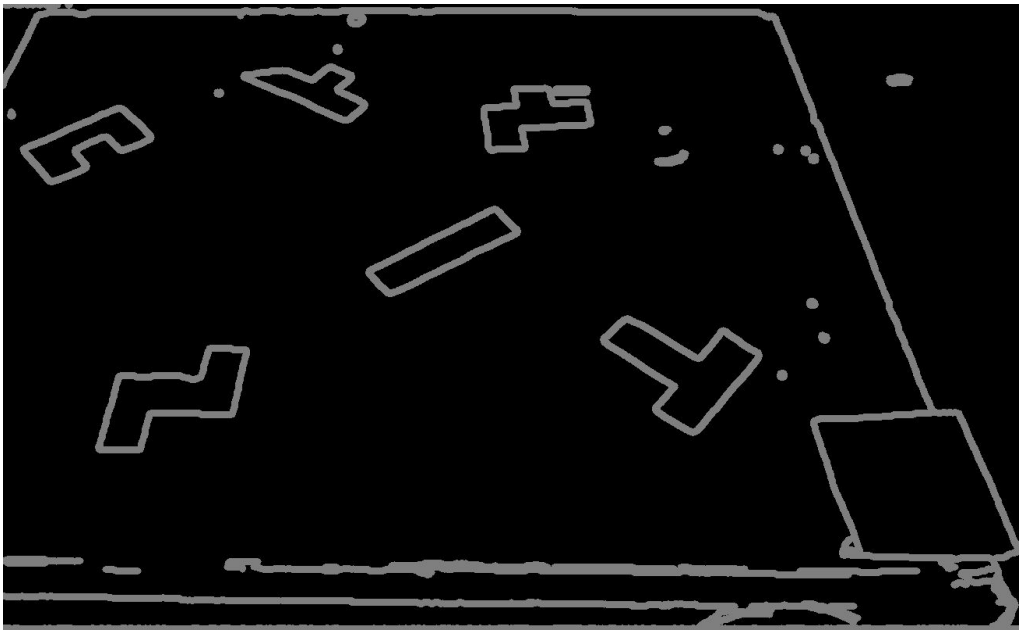


Image on which contour has been applied, we can clearly see the boundaries of various objects.

3. **HSV system of colours**

It's a different system than RGB to define a pixel.
It's better as in RGB same colour might not look same every time and at every place…
HSV system takes care of that.
We saw during the color extraction that we had to make six different track bars so as to extract different colors.
Hence it become very hectic to work on RGB.
Therefore, a new system of HSV is defined.
H-Hue- Degree on the HSV cylinder
S-Saturation- Percentage of the color in the pixel
V-Value- Darkness or the lightness

H : Hue
V : Value
S : Saturation

b' = b/255
r' = r/255
g' = g/255

V = max{b',g',r'}

S = (v-min{r',g',b'})/v if v ≠ 0
= 0 otherwise

H = 60(g' - b')/(V - min{r',g',b'}) if V=r'
= 120 + 60(b' - r')/(V - min{r',g',b'}) if V=g'
= 240 + 60(r' - g')/(V - min{r',g',b'}) if V=b'

There is an in built function for this

cutColour(input_img,output_img,BGR2HSV);

4. <u>Template Matching</u>

A template is a part of a given image.
In template matching we take that template an and try to find it's best match in the given Image by traversing it over the image.

What we do is we take the template or a big kernel of it and then we traverse it across the image, we do so by taking the sq. of the difference of colour intensities of the corresponding pixels and then adding them up, the kernel of the image with the least difference is the best match.

<u>CODE FOR TEMPLATE MATCHING: BY all the 4 Method/Mode</u>

```
/*
Template Matching
Source img-> coins.jpg
Template img -> temp.jpg
*/

#include "opencv2/imgcodecs.hpp"
```

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

using namespace std;
using namespace cv;

bool use_mask;
Mat img; Mat templ; Mat mask; Mat result;
const char* image_window = "Source Image";
const char* result_window = "Result window";

int match_method;
int max_Trackbar = 5;

void MatchingMethod( int, void* );

int main()
{ img = imread("coins.jpg", IMREAD_COLOR );
  templ = imread("temp.jpg", IMREAD_COLOR );

  if(img.empty() || templ.empty())
  {
        cout << "Can't read one of the images" << endl;
        return -1;
  }
  namedWindow( image_window, WINDOW_AUTOSIZE );
  namedWindow( result_window, WINDOW_AUTOSIZE );
  const char* trackbar_label = "Method: \n 0: SQDIFF \n 1: SQDIFF NORMED \n
2: TM CCORR \n 3: TM CCORR NORMED \n 4: TM COEFF \n 5: TM COEFF NORMED";
  createTrackbar( trackbar_label, image_window, &match_method,
max_Trackbar, MatchingMethod );

  MatchingMethod( 0, 0 );

  waitKey(0);
  return 0;
}

void MatchingMethod( int, void* )
{
  Mat img_display;
  img.copyTo( img_display );
```

```
        int result_cols =  img.cols - templ.cols + 1;
        int result_rows = img.rows - templ.rows + 1;

        result.create( result_rows, result_cols, CV_32FC1 );

        bool method_accepts_mask = (CV_TM_SQDIFF == match_method ||
match_method == CV_TM_CCORR_NORMED);
        if (use_mask && method_accepts_mask)
            { matchTemplate( img, templ, result, match_method, mask); }
        else
            { matchTemplate( img, templ, result, match_method); }
        normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );
        double minVal; double maxVal; Point minLoc; Point maxLoc;
        Point matchLoc;

        minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );
        //! [best_match]

        //! [match_loc]
        /// For SQDIFF and SQDIFF_NORMED, the best matches are lower values. For
all the other methods, the higher the better
        if( match_method  == TM_SQDIFF || match_method ==
TM_SQDIFF_NORMED )
            { matchLoc = minLoc; }
        else
            { matchLoc = maxLoc; }
        //! [match_loc]

        rectangle( img_display, matchLoc, Point( matchLoc.x + templ.cols ,
matchLoc.y + templ.rows ), Scalar::all(0), 2, 8, 0 );
        rectangle( result, matchLoc, Point( matchLoc.x + templ.cols , matchLoc.y +
templ.rows ), Scalar::all(0), 2, 8, 0 );

        imshow( image_window, img_display );
        imshow( result_window, result );

        return;
       }
```

```
	// Sample program for template matching

 #include "opencv2/highgui/highgui.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;
int main()
{
	Mat img, templ,result, mask,fimg;
	img=imread("coins.jpg",1);
	templ=imread("coins1.jpg",1);
	img.copyTo(fimg);

	int result_cols =  img.cols - templ.cols + 1;
	int result_rows = img.rows - templ.rows + 1;
	result.create( result_rows, result_cols, CV_8UC3);

	matchTemplate( img, templ, result, CV_TM_SQDIFF);

	double minVal, maxVal; Point minLoc, maxLoc;

	minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );

	rectangle( fimg, minLoc, Point( minLoc.x + templ.cols , minLoc.y +
templ.rows ), (100,50,105), 2);

	cout<<minLoc;
	cout<<fimg.rows<<" "<<fimg.cols;
```

```
        imshow("final",fimg);
        imshow("template",templ);
        waitKey(0);
}
```

5. Corner Detection

It's basic principle was based on the fact that at a corner the intensities of colour would change by a large amount in both x and y direction.

Astar Algorithm for pathfinding:

```
function A*(start, goal)
    // The set of nodes already evaluated
    closedSet := {}

    // The set of currently discovered nodes that are not evaluated yet.
    // Initially, only the start node is known.
    openSet := {start}

    // For each node, which node it can most efficiently be reached from.
    // If a node can be reached from many nodes, cameFrom will eventually contain the
    // most efficient previous step.
    cameFrom := an empty map

    // For each node, the cost of getting from the start node to that node.
    gScore := map with default value of Infinity

    // The cost of going from start to start is zero.
    gScore[start] := 0

    // For each node, the total cost of getting from the start node to the goal
    // by passing by that node. That value is partly known, partly heuristic.
    fScore := map with default value of Infinity


    // For the first node, that value is completely heuristic.
    fScore[start] := heuristic_cost_estimate(start, goal)
```

```
    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        closedSet.Add(current)

        for each neighbor of current
            if neighbor in closedSet
                continue                // Ignore the neighbor which is already evaluated.

            if neighbor not in openSet      // Discover a new node
                openSet.Add(neighbor)

            // The distance from start to a neighbor
            //the "dist_between" function may vary as per the solution requirements.
            tentative_gScore := gScore[current] + dist_between(current, neighbor)
            if tentative_gScore >= gScore[neighbor]
                continue                // This is not a better path.

            // This path is the best until now. Record it!
            cameFrom[neighbor] := current
            gScore[neighbor] := tentative_gScore
            fScore[neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)

    return failure

function reconstruct_path(cameFrom, current)
    total_path := [current]
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.append(current)
    return total_path
```

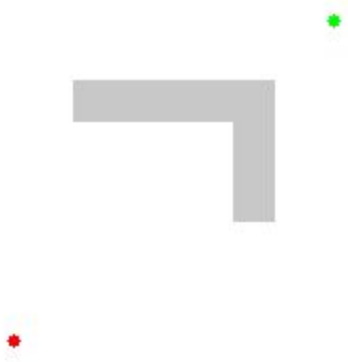


Illustration of A* search for finding path from a start node to a goal node in a robot motion planning problem. The empty circles represent the nodes in the *open set*, i.e., those that remain to be explored, and the filled ones are in the closed set. Color on each closed node indicates the distance from the start: the greener, the farther. One can first see the A* moving in a straight line in the direction of the goal, then when hitting the obstacle, it explores alternative routes through the nodes from the open set.

RRT Algorithm

For a general configuration space C, the algorithm in pseudocode is as follows:

Algorithm BuildRRT
Input: Initial configuration q_{init}, number of vertices in RRT K, incremental distance Δq)
Output: RRT graph G

G.init(q_{init})
for k = 1 **to** K
$q_{rand} \leftarrow$ RAND_CONF()
$q_{near} \leftarrow$ NEAREST_VERTEX(q_{rand}, G)
$q_{new} \leftarrow$ NEW_CONF(q_{near}, q_{rand}, Δq)
G.add_vertex(q_{new})
G.add_edge(q_{near}, q_{new})
return G

- "←" is a shorthand for "changes to". For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the value that follows.

In the algorithm above, "**RAND_CONF**" grabs a random configuration q_{rand} in C. This may be replaced with a function "**RAND_FREE_CONF**" that uses samples in C_{free}, while rejecting those in C_{obs} using some collision detection algorithm.

"**NEAREST_VERTEX**" is a function that runs through all vertices v in graph G, calculates the distance between q_{rand} and v using some measurement function thereby returning the nearest vertex.
"**NEW_CONF**" selects a new configuration q_{new} by moving an incremental distance Δq from q_{near} in the direction of q_{rand}. (According to [4] in holonomic problems, this should be omitted and q_{rand} used instead of q_{new}.)