**Winter School of Artificial Intelligence and Robotics (Certified by IEEE, Organized in collaboration with Centre of Excellence in Artificial Intelligence, IIT Kharagpur)**

**Image Processing (IP) Workshop**

**Documentation- Abhranil Chandra**

**(Mentors: Gaurang Mohta, Arnesh Kumar Issar, Sharat Bhat)**

Topics Covered:

1. Introduction to IP, openCV Library.
2. Setup of environment on Ubuntu
3. Types of Images

* A grey-scale image a single channel image in which every pixel contains a number which denotes the intensity.

For e.g an 8 bit image consists of images of intensity 0 to 255, where 255=white, 0=black.

* A coloured image is a three channel image in which each channel represents RGB (or specifically Channel0=Blue, channel1=Green, Channel2=Red)

These three channels are overlapped for a colour image, and the intensity concept is the same.

* A binary image is a two colour image consisting of only black and white

1. Basic image creation or taking input - basic.cpp & readimg.cpp

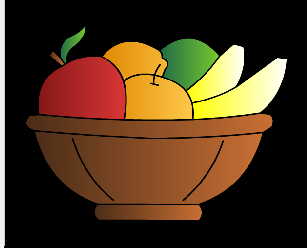
How to open a matrix(Mat) in c++

* An image is stored as a Mat in c++, the data type is Mat
  + A colored Mat can be created by using the following statement
    - Mat img(8000,8000,CV\_8UC3,Scalar(255,255,255));
      * Here 800,800 is height,width.
      * CV is an OpenCV variable.
      * 8U refers to an 8 bit unsigned image.
      * C3 refers that it contains three channels
      * Scalar is for defining the background
  + A grayscale image can be created by using the following statement
    - Mat img2(x,y,CV\_8UC1,Scalar(0));
      * Here only the third and similarly the fourth argument is different
        + C1 says that there is only one channel.
  + A binary image is similar to a grayscale image.

Random points-:

1. To find no. of rows of an image use (image name).rows
2. Similarly for columns use (image name).cols.

To read an image

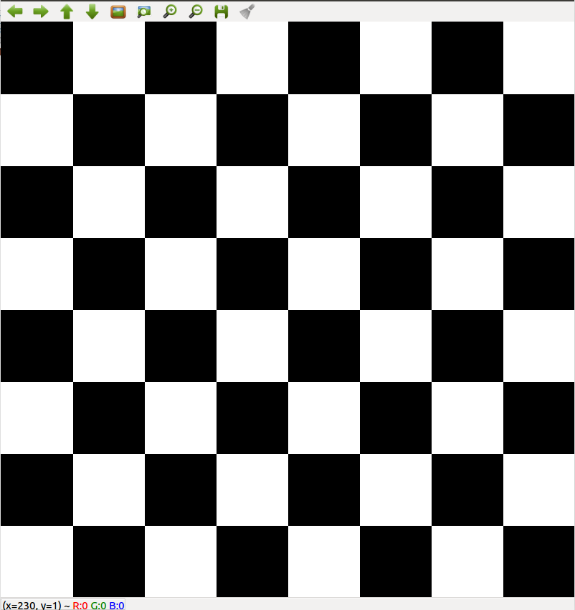
* Use the following command  “ Mat img1=imread”image name/image ",0/1);” for grayscale/colored.
* 

Few statements for displaying the image on the screen

* namedWindow("window\_name",WINDOW\_NORMAL);
  + For creating a window to display the image
* imshow("window\_name",img);
  + To show the image in that window
* waitKey(20000);
  + The time in milli-second for which the window is open. 0 is for infinite time until any key is pressed in the window.

To save an image use imwrite(“pic name.extension”,img variable);

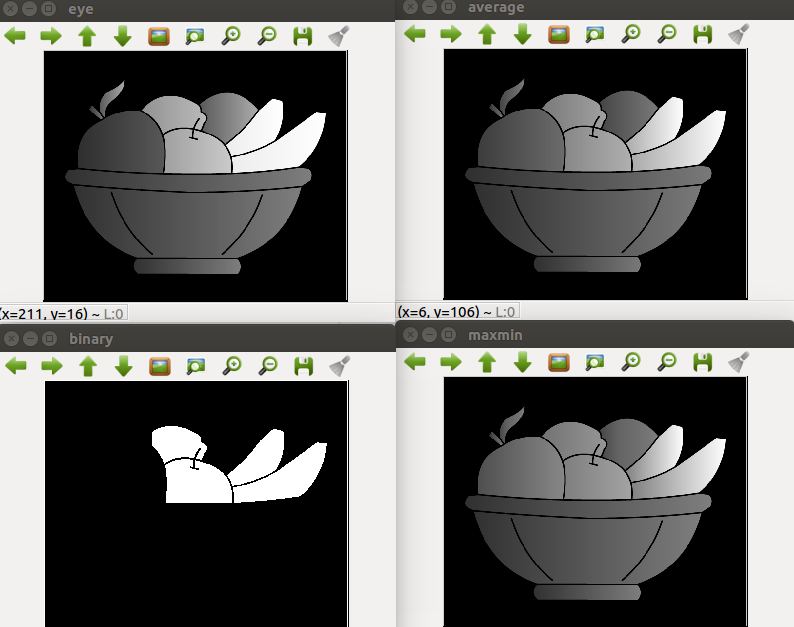
1. Making Basic Patterns like Chess Board - chessboard.cpp



1. Basic Image Manipulation – Flipping - task1.cpp & task2.cpp
2. Types of images and conversion - colTograyscale.cpp

There are three ways to change BRG to grayscale [code](https://drive.google.com/open?id=1YbNxle3d9PEDd6_i_bvydkxzV3vhiL6s)

* 1. R+G+B/3
  2. max(r,g,b)+min(r,g,b)/2
  3. And the best and scientific method is
     1. 0.21R+0.72B+0.07G

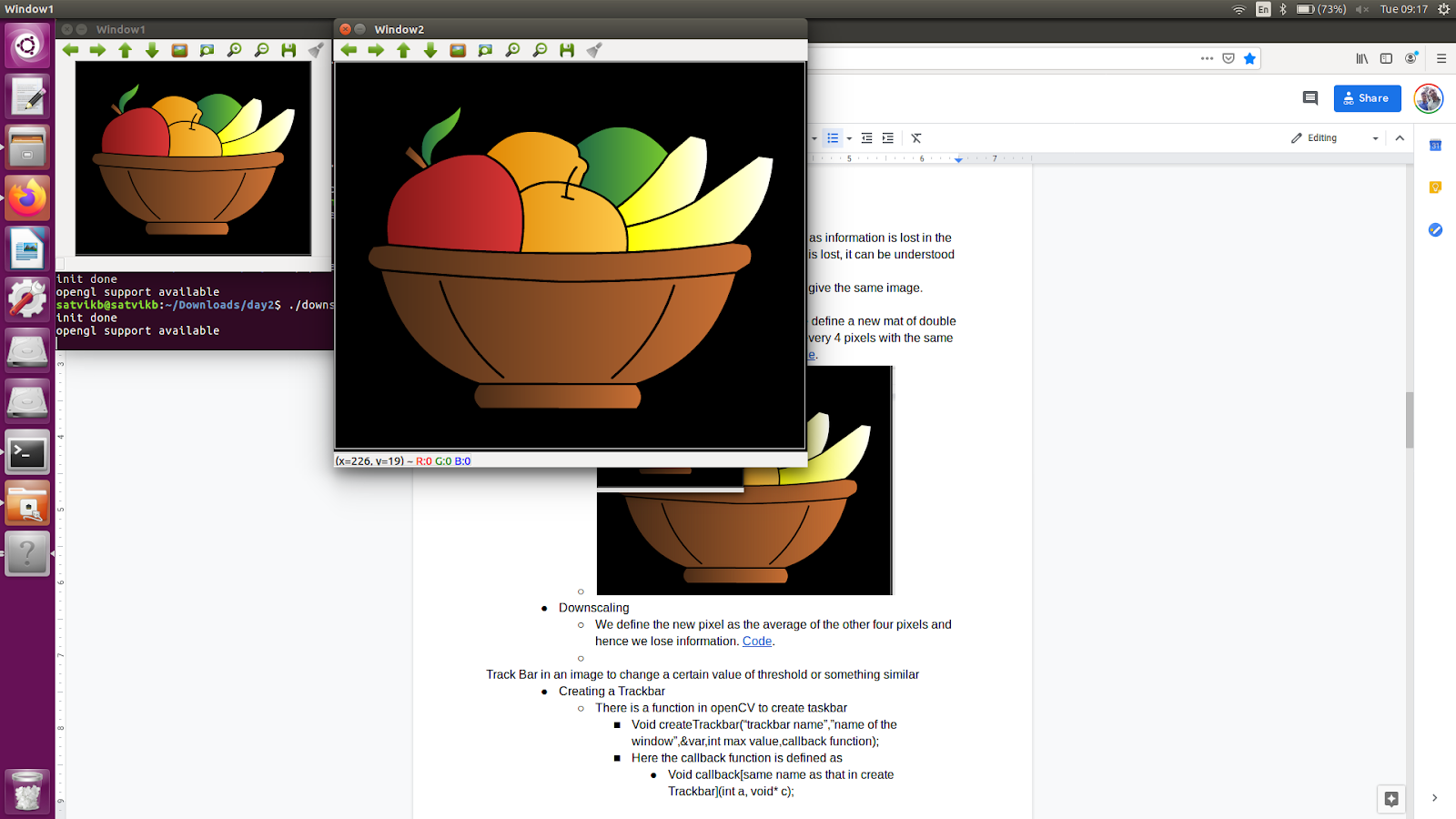


Conversion of Grayscale to binary

Code is same as the above few more statements are added to show the conversion.

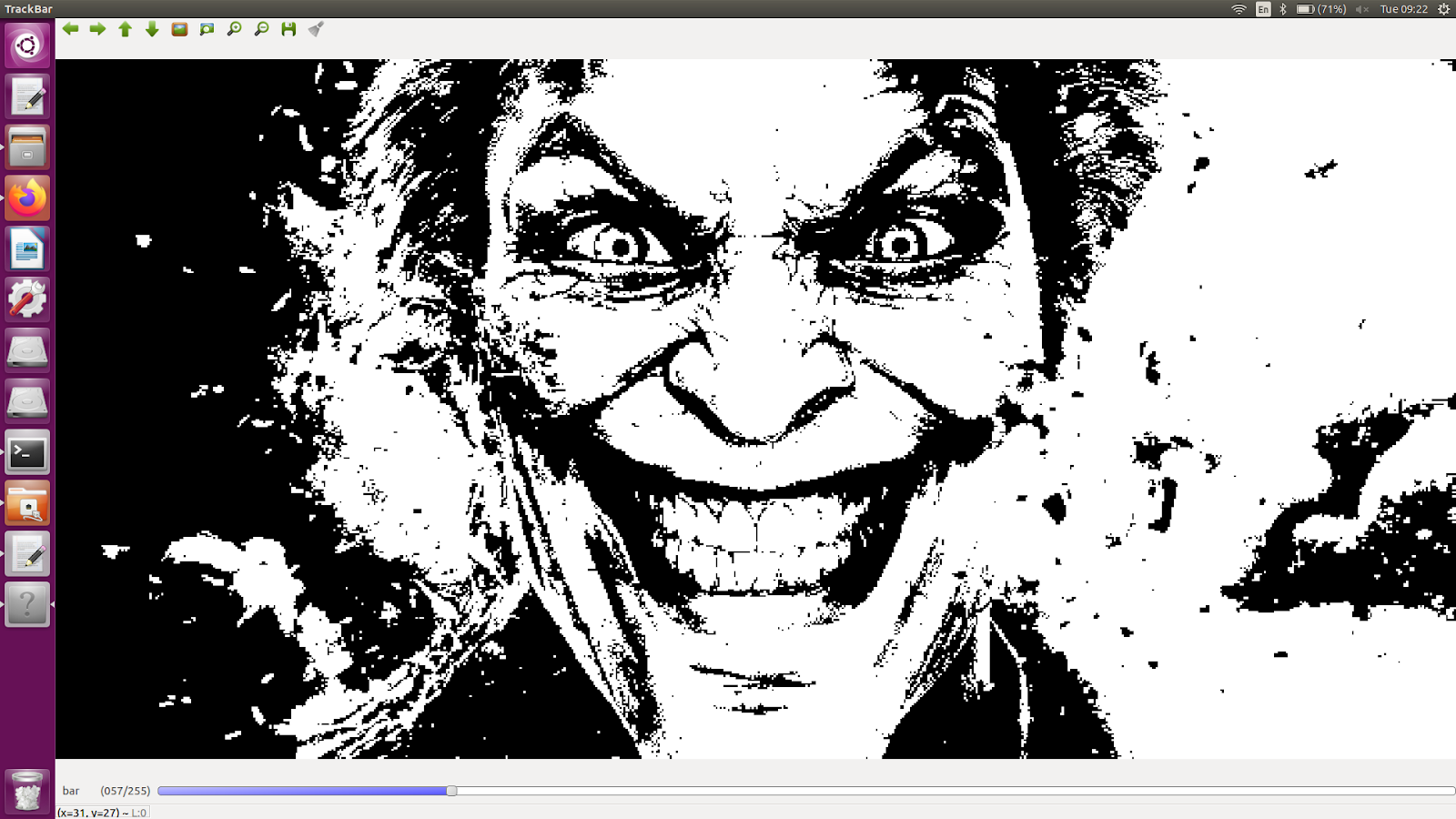
1. Binerization of Image - bineizationByTrackbar.cpp
2. Resizing of Image - downscale.cpp & upscale.cpp

Scaling of an image

* Upscaling and Downscaling
  + Downscaling is generally not beneficial as information is lost in the process but in upscaling no information is lost, it can be understood by seeing the method of scaling.
  + Upscaling a downscaled image will not give the same image.
* Upscaling
  + For eg. if we are upscaling by 2 then we define a new mat of double rows and columns and then we define every 4 pixels with the same value as the corresponding 1 pixel. [Code](https://drive.google.com/open?id=182oTit_2irJS88w4ASjxtFJr47NoM25R).
  + 
* Downscaling
  + We define the new pixel as the average of the other four pixels and hence we lose information. [Code](https://drive.google.com/open?id=1XcmaL9MD02NH50WN6NX9_gjvA_CW8o17).
  + 

1. Taskbar and Functionalities - trackbar.cpp

Track Bar in an image to change a certain value of threshold or something similar

* Creating a Trackbar
  + There is a function in openCV to create taskbar
    - Void createTrackbar(“trackbar name”,”name of the window”,&var,int max value,callback function);
    - Here the callback function is defined as
      * Void callback[same name as that in create Trackbar](int a, void\* c);
  + This will create a track bar through which we can dynamically change a parameter for image definition.
* Here are few sample codes
  + [Single trackbar](https://drive.google.com/open?id=1uzSVxRnEEs9Ec58YvnM5r8szhXf7DhZ_).
    - 
  + [Multiple trackbar](https://drive.google.com/open?id=1Een9H53FsTPe-Msq5nmomrE760ijMTtQ).
    - 

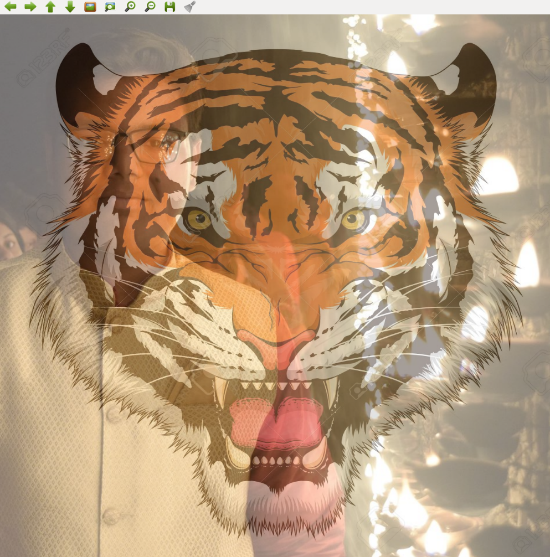
1. Segmentation of Image - segmentation.cpp
2. Image Rotation - imgrotation.cpp

It is just a mathematical transformation of pixel coordinates to an inclined system.

* 1. Seeing a [sample code](https://drive.google.com/open?id=1ICnqjwihTE-CiAiVvwE8r6BhsTEhlPKC) can help understanding, here a track bar is used for different angles.
     1. This code is for rotating around the centre but with a little more logic.
     2. Here is the [code](https://drive.google.com/open?id=1U197HP8Y5om3ULXrubVW0M08i2vHcbqf) for that.
     3. 

1. Morphing of Images - morphing.cpp

* Morphing is a special effect in motion pictures and animations that changes one image or shape into another through a seamless transition.
  + This can be achieved in image processing by fading the pixels of one image and making the pixels of the other image more prominent. Just like the section formula.
    - Here is [a code](https://drive.google.com/open?id=11tvdr1VDFhAY-5w887Nwt-hlMSFkh5so) for the morphing of two images.



1. Histogram and Histogram Equalization - histogram.cpp & YET TO DO

Histogram of an image

* It is a way to determine whether an image is dark or light, or of high or low contrast.
  + It is the shape of the histogram that help us determine the property.
  + The histogram consists on the axis the pixel value and on the other the frequency of that pixel value.
    - Here is a [code](https://drive.google.com/open?id=1U7bRxBLbE4e6anjCyLtoGA6EdnIDI8JT) as an example of a histogram.
* There is a concept of equalisation of image which used to almost equalise the frequency of every pixel value.
  + The formulae for equalisation is
    - new= old\*No. Of pixels less than old\*255/total pixels.

1. Kernel and Kernel Operations

Kernel

* It is an odd number matrix which can be imposed on any image to perform any function, function like blurs, erosion and dilation, edge detection.
* Kernel can be weighted and different processes can be performed.
  1. Blurs
     1. Mean Blur - kernel\_mean.cpp

It is the type of blur in which the central pixel is designated the mean of all the pixel values in the kernel. [Code](https://drive.google.com/open?id=1CJHi9PflTOkkTjHjXxV3pEFBTz9e7la_).

* + 1. Median Blur - kernel\_median.cpp

It is the type of blur in which the median of the surrounding elements of the central pixel is assigned to the central pixel, used to remove salt pepper noise. [Code](https://drive.google.com/open?id=1k7ZbMUDC2_8hTGQbEyBJQMw0q6w2_IxH).

* + 1. Gaussian Blur – gaussian\_blur.cpp (ISSUE)

It is the type of blur in which every element of the kernel is weighted, this type of blur is very effective

While applying the blur the boundary lines are left out, to prevent so

Padding is done.

Padding Types

* Reflective
  + Make the new pixels same as that of the neighbouring pixels
* Wrap around
  + As the name suggests the first of left is pasted in right new row and similarly for the other three.
* 0 padding
  + All the new pixels set to 0.
  1. Erosion & Dilation - dilation\_erosion.cpp

These both functions are functions of binary functions

Erosion decreases the number of white pixels.

Whereas dilation increases the white pixels.

Code for [dilation and erosion](https://drive.google.com/open?id=1qoiioIq03OxxNA04nImz9_HW17ge-kgc).

* 1. Edge Detection

One of the basic way is to dilate the image and then subtract the dilated from original image to get the boundaries.

The other is intensity derivative that is if there is intense change then it is a boundary.

* + 1. Preweitt Filter - edge.cpp
    2. Sobel Filter - edge.cpp
    3. Edge Optimization through Canny Algo - edge\_optimisation\_Canny.cpp

There is also a function in OpenCV for edge detection known as

canny

Void Canny(input img,output img,double tl, double thu(generally

3\*tl),kernel side size).

[Code](https://drive.google.com/open?id=18VgRyGwuTJ8Kj1LiN0zsxV_bhNVYDY--).

* 1. Hough Line Transform - hough\_line.cpp & houghline.cpp

Hough Line Transformation

* It is a way to find the line and its equation.
* It is obtained by using the r,theta for of line, and then plotting it for number of lines or vertices.
* The [code](https://drive.google.com/open?id=1NysKRl6c8SG39hUJpQMtIFTXUZEJ4ESm) is for hough line.
  1. Hough Circle Transform - YET TO DO

Similar concept is for Hough circle which indeed is not 2-D but 3-D

curve.

1. Basic Data Structures & Programming Paradigms – No Code

Introduction to Stacks, Queues, Recursion and Graphs

* Stacks
  + These are Last In First Out type of data structure in c++, where the STL of stacks helps us to store data in a stack and further use it. Stacks are generally used in DFS.
* Queues
  + Queues are First In First Out type of data structure, it also has a different header file for its functions. Queues are used in BFS generally.

A **priority queue** is a special type of **queue** in which each element is associated with a **priority** and is served according to its **priority**.

* + Recursion
  + Calling of a function again and again is called recursion.
* Graphs
  + A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and edges are lines or arcs that connect any two nodes in the graph.

1. Path Detection & Optimization
   1. BFS – Breadth First Search - BFS.cpp

**Breadth-first search** (**BFS**) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key’), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

In image processing it is used to find out no. of obstacles in the image.

For more information please refer to the [link](https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/).

* 1. DFS - Depth First Search - DFS.cpp

**Depth-first search** (**DFS**) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

In Image Processing, it used to find a path(not the shortest), to an end point from a starting point.

For more information please refer to the [link](https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/).

* 1. Djikstra Algorithm – Djikstra.cpp & Djikstra1.cpp (ISSUE)

It is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.

As this method has no direction sense,hence, it is not an efficient method but accurate.

A better way is A\* algorithm.

For more information on dijkstra [click here](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/).

* 1. A\* Algorithm - AStar.cpp

It is an efficient algorithm as it also takes in consideration the distance from the destination

For detailed information [click here](https://www.geeksforgeeks.org/a-search-algorithm/).

This method although is time efficient but takes a lot of processing and hence another method known as rrt is more efficient in both the ways

* 1. RRT (Rapidly Generating Random Tree) – NO CODE

It is an algorithm designed to efficiently search nonconvex, high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem.

For more information [click here](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree).

* 1. RRT\* - NO CODE

Further optimisation of RRT

1. Operations on Videos - Video.cpp & Video1.cpp

Videos are taken as different frames and then these frames are treated as stable images and processing is done.

1. Contour Detection - contourThrWebcam.cpp

Video segmentation

* It is the process through which we can segment out a certain object (color) from the video feed. It is similar to that of the segmentation of a still image.
  + For doing this first take a test still of the video the apply segmentation on it with the help of trackbars and then apply the limits in the code of video as trackbars can not be used in video
* Certain problems are faced in this process like the lighting conditions on the object. To overcome this problem we use HSV filters for image.

1. Harris Corner Detection – NO CODE

* In computer vision, usually we need to find matching points between different frames of an environment. Why? If we know how two images relate to each other, we can use *both* images to extract information of them.
* When we say **matching points** we are referring, in a general sense, to *characteristics* in the scene that we can recognize easily. We call these characteristics **features**.
* **So, what characteristics should a feature have?**
  + It must be *uniquely recognizable*

**Types of Image Features**

To mention a few:

* Edges
* **Corners** (also known as interest points)
* Blobs (also known as regions of interest )

In this tutorial we will study the *corner* features, specifically.

**Why is a corner so special?**

* Because, since it is the intersection of two edges, it represents a point in which the directions of these two edges *change*. Hence, the gradient of the image (in both directions) have a high variation, which can be used to detect it.

**How does it work?**

* Let's look for corners. Since corners represents a variation in the gradient in the image, we will look for this "variation".
* Consider a grayscale image I. We are going to sweep a window w(x,y) (with displacements u in the x direction and v in the y direction) I and will calculate the variation of intensity.

E(u,v)=∑x,yw(x,y)[I(x+u,y+v)−I(x,y)]2

where:

* + w(x,y) is the window at position (x,y)
  + I(x,y) is the intensity at (x,y)
  + I(x+u,y+v) is the intensity at the moved window (x+u,y+v)
* Since we are looking for windows with corners, we are looking for windows with a large variation in intensity. Hence, we have to maximize the equation above, specifically the term:

∑x,y[I(x+u,y+v)−I(x,y)]2

* Using *Taylor expansion*:

E(u,v)≈∑x,y[I(x,y)+uIx+vIy−I(x,y)]2

* Expanding the equation and cancelling properly:

E(u,v)≈∑x,yu2I2x+2uvIxIy+v2I2y

* Which can be expressed in a matrix form as:

E(u,v)≈[uv](∑x,yw(x,y)[I2xIxIyIxIyI2y])[uv]

* Let's denote:

M=∑x,yw(x,y)[I2xIxIyIxIyI2y]

* So, our equation now is:

E(u,v)≈[uv]M[uv]

* A score is calculated for each window, to determine if it can possibly contain a corner:

R=det(M)−k(trace(M))2

where:

* + det(M) = λ1λ2
  + trace(M) = λ1+λ2

a window with a score R greater than a certain value is considered a "corner"

1. H.S.V. Scale and Conversion to RGB – NO CODE

* HSV Scale

HSV scale or Hue Saturation Value scale is a colour scale like RGB scale which can be used for filtering out certain colours even if the lighting condition changes. This scale stores the colour information in hue and the amount of darkness in value and the saturation in saturation. If the saturation is 0 then the image is a grayscale image.

Converting RGB to HSV

* As explained HSV we change all the RGB values to HSV values by the following method :
* https://lh3.googleusercontent.com/1bic455-8mTJWFB0WDMq2UYKvSqfvU1nC-tGImomKCUpTJdG9rGiYXdha7wnwj_ZqKFjcIe3f7615Dj24jLKlreta4zjqOeHv8wdr28zzH7-yGxkn8BbxB1ZqjCcXPkBCK1nUiWh

Then

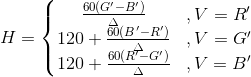
https://lh4.googleusercontent.com/3_OKMEwkv00W5Ke6viwANJdE8wwApUpYhxnzLgbJ4iGMFf3fGzfX9TZcnSrZOMLU9piKJb9YDpHdvcvUvm21FvQ1aY-lJgzyqgqBTdVFjogPkozMC6C4cLiXjOUu8yyXS7SBK-vu

https://lh5.googleusercontent.com/-ZxMBhBbnTgAGZR2P6MD69z2lGZyjzGxf-_lTgaoFylPNb6qQZ418yrgi9rKwGCv2zWPhVG-izVDS221fKhxrBv_C5-EGr52bKnrEZZ2_h8Wnv5b1Nh3vIK7WyFf-zgHVvAXgdlG

Now

https://lh4.googleusercontent.com/n6gwyaBxhx6zjvVgVBomTL29NMnPCzrKuJB74e_pHmcha2W2eShsTyevXgzk8GDU8Oq9bTkh7K2NWvPou1yj3ZVATAMQ9ikv85AvlhsmYX17tCFiNIlC53rtwSmh3FguTvRaOKvM

Then,



1. Template Match - YET TO DO

* When searching for a small part in an image, one important function is template matching and it is possible in IP by pixel matching`.
* This is done by image rotation and resizing of the object image.

1. Arduino and Implementing C++ codes in Arduino Code

* We were told the functioning of an arduino, it’s installation(all the connection), all the steps for a fully functional bot.
* We were told some basic c++ codes for arduino, and we used it in our codes for successful working of the bot.

1. Conclusion & Future Enhancement

At the end we were given a problem in which we have to control a bot using the red screen on our phones and moving it should move the bot.

We used segmentation, contour detection for the problem solution.