WINTER WORKSHOP (IMAGE PROCESSING)

# DAY 1

## IMAGE

Pixel is the basic unit of an image just like atoms in matter. Digital image is a collection (2-d array) of pixels. A pixel takes discrete vales for intensity(brightness).

Every pixel can be treated as an object.

## TYPES OF IMAGES

1. GRAYSCALE

here the colours at our disposal are black and white. The combination of these allows us to create many shades of grey.

1. TRUECOLOR(RGB)

Its base colours are red, green and blue. To code the transparency of a colour sometimes a fourth element: alpha (A) is added.

1. BINARY

Like binary in Boolean, the image is composed of either white or black pixel.

1. HSL/HSV

The HSV and HLS decompose colours into their hue, saturation and value/luminance components, which is a more natural way for us to describe colours.

## STORING IMAGES

Each of the colour components has its own valid domains. This brings us to the data type used: how we store a component defines the control we have over its domain.

The smallest data type possible is char, which means one byte or 8 bits. This may be unsigned (so can store values from 0 to 255) or signed (values from -127 to +127).

Although in the case of three components (such as BGR) this already gives 16 million representable colours. **Increasing the size of a component also increases the size of the whole picture in the memory.**

### TEMPLATES

A template is a blueprint or formula for creating a generic class or a function. E.g. Vec3b, uchar.

## CODE

HEADER FILES: contain definitions of functions  
**#include "opencv2/highgui/highgui.hpp**”: - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.

**#include "opencv2/imgproc/imgproc.hpp”**: - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), colour space conversion, histograms, and so on.

**#include "opencv2/core/core.hpp”**: - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

NAMESPACE: provides scope to identifiers hence, prevents name collisions

**using namespace cv;**

**using namespace std;**

### CREATING AN IMAGE

1. CONSTRUCTOR METHOD

Mat a (#ROWS, #COLS, CV\_8UC3, Scalar (B, G, R));

**Mat:** Mat is basically a class with two data parts: the matrix header (containing information such as the size of the matrix, the method used for storing, at which address is the matrix stored, and so on) and a pointer to the matrix containing the pixel values (taking any dimensionality depending on the method chosen for storing) .  
**ROWS:** no. of rows  
**COLS:** no of columns

**CV\_8UC3** means we use unsigned char types that are 8 bit long and each pixel has three of these to form the three channels.

The [**Scalar**](http://docs.opencv.org/modules/core/doc/basic_structures.html#scalar) is a four element short vector. Specify this and you can initialize all matrix points with a custom value.

1. Create () FUNCTION

img.create(#ROWS,#COLS, type);

### READING AN IMAGE

Mat img = imread(filename, 0); //grayscale  
Mat img = imread(filename, 1); //3 channel image

### WRITING AN IMAGE

imwrite( filename, imagename );

CREATING A WINDOW  
namedWindow("NAME OF FILE", WINDOW\_AUTOSIZE);  
namedWindow: it’s a function. it creates a blank window.

### DISPLAYING AN IMAGE

imshow("NAME OF WINDOW", NAME OF MATRIX);

### ACCESSING AN ELEMENT

at() FUNCTION: Returns a reference to the specified array element.template<typename>(position)[index]  
img.at <uchar> ( i, j )  
img.at <Vec3b> ( i, j )[0]  
In case of colour images we have three uchar items per column. This may be considered a short vector of uchar items, that has been baptized in OpenCV with the Vec3b name. To access the n-th sub column we use simple operator [] access.

### WAITKEY

It is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues. If **0** is passed, it waits indefinitely for a key stroke.

### COPYING AN IMAGE

1. clone() FUNCTION

Mat a=B.clone();

1. copyTo() FUNTION  
   A.copyTo(B);

The assignment operator and the copy constructor only copies the header.

The underlying matrix of an image may be copied using the [clone()](http://docs.opencv.org/modules/core/doc/basic_structures.html#mat-clone) and [copyTo()](http://docs.opencv.org/modules/core/doc/basic_structures.html#mat-copyto) functions.

# DAY 2

## CONVERSIONS

### BGR TO GRAYSCALE

1. Y = (B \* 0.114) +(G \* 0.587) + (R \* 0.299)
2. cvtColor( image, gray\_image, CV\_BGR2GRAY )

a source image (*image*)   
a destination image (*gray\_image*), in which we will save the converted image.   
an additional parameter that indicates what kind of transformation will be performed. In this case we use **CV\_BGR2GRAY**

### GRAYSCALE TO BINAY (thresholding)

This separation is based on the variation of intensity between the object pixels and the background pixels. To differentiate the pixels, we are interested in from the rest (which will eventually be rejected), we perform a comparison of each pixel intensity value with respect to a threshold (determined according to the problem to solve).

Once we have separated properly the important pixels, we can set them with a determined value to identify them (i.e. we can assign them a value of 0(black), 255 (white)

Using TRACKBAR  
*createTrackbar ( trackbarName, windowName, value, count, onChange)***trackbarname** – Name of the created trackbar.  
**winname** – Name of the window that will be used as a parent of the created trackbar.  
**value** – Optional pointer to an integer variable whose value reflects the position of the slider. Upon creation, the slider position is defined by this variable.  
count – Maximal position of the slider. The minimal position is always 0.  
**onChange** – Pointer to the function to be called every time the slider changes position. This function should be prototyped as void Foo(int,void\*); , where the first parameter is the trackbar position and the second parameter is the user data (see the next parameter). If the callback is the NULL pointer, no callbacks are called, but only value is updated.  
**userdata** – User data that is passed as is to the callback. It can be used to handle trackbar events without using global variables.

## HISTOGRAM

Histograms are collected counts of data organized into a set of predefined bins.  
A histogram can keep count not only of colour intensities, but of whatever image features that we want to measure (i.e. gradients, directions, etc.).

### SETMOUSECALLBACK: Sets mouse handler for the specified window

SetMouseCallback(windowName, onMouse, param=None)  
winname – window name  
onMouse – mouse Callback function

void Callbackfunc(int Event, int y, int x, int flags, void \*userdata)

{

if (Event == EVENT\_LBUTTONDOWN)

{

printf("B = %d\n", img.at<Vec3b>(x, y)[0]);

printf("G = %d\n", img.at<Vec3b>(x, y)[1]);

printf("R = %d\n", img.at<Vec3b>(x, y)[2]);

}

}

## KERNEL/ MASK

It is a small [matrix](https://en.wikipedia.org/wiki/Matrix_%28mathematics%29). It is useful for blurring, sharpening, embossing, [edge detection](https://en.wikipedia.org/wiki/Edge_detection), and more. This is accomplished by means of [convolution](https://en.wikipedia.org/wiki/Kernel_%28image_processing%29#Convolution) between a kernel and an [image](https://en.wikipedia.org/wiki/Bitmap_image).

## FILTERS

### SMOOTHING IMAGES

#### MEDIAN FILTER

The median filter run through each element of the signal (in this case the image) and replace each pixel with the **median** of its neighbouring pixels (located in a square neighbourhood around the evaluated pixel).

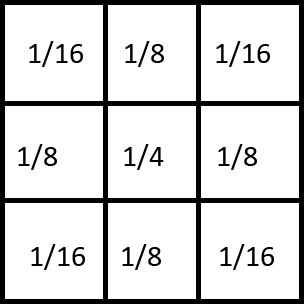
#### MEAN FILTER

The mean filter run through each element of the signal (in this case the image) and replace each pixel with the **mean** of its neighbouring pixels (located in a square neighbourhood around the evaluated pixel).

#### GAUSSIAN FILTER

Gaussian filtering is done by convolving each point in the input array with a Gaussian kernel and then summing them all to produce the output array.

Assuming that an image is 1D, you can notice that the pixel located in the middle would have the biggest weight. The weight of its neighbours decreases as the spatial distance between them and the centre pixel increases.



3X3 GAUSSIAN KERNEL

#### SOBEL FILTER

A method to detect edges in an image can be performed by locating pixel locations where the gradient is higher than its neighbours (or to generalize, higher than a threshold).  
It computes an approximation of the gradient of an image intensity function.

Assuming that the image to be operated is I:

1. We calculate two derivatives:
   1. **Horizontal changes**: This is computed by convolving Iwith a kernel G_{x}with odd size. For example for a kernel size of 3, G_{x}would be computed as:

G_{x} = \begin{bmatrix}
-1 & 0 & +1  \\
-2 & 0 & +2  \\
-1 & 0 & +1
\end{bmatrix} * I

* 1. **Vertical changes**: This is computed by convolving Iwith a kernel G_{y}with odd size. For example for a kernel size of 3, G_{y}would be computed as:

G_{y} = \begin{bmatrix}
-1 & -2 & -1  \\
0 & 0 & 0  \\
+1 & +2 & +1
\end{bmatrix} * I

1. At each point of the image we calculate an approximation of the *gradient* in that point by combining both results above:

G = \sqrt{ G_{x}^{2} + G_{y}^{2} }

# DAY 3

## FILTERS

### ERROSION AND DILATION

It is used for removing noise and finding intensity bumps or holes.

#### ERROSION

The erosion makes the object in white smaller AND object in black bigger.

A kernel run through each element of the image and assigns the centre of the pixel with 0 if any black pixel is encountered.

#### DILATION

The dilatation makes the object in white bigger AND object in white smaller.

A kernel run through each element of the image and assigns the centre of the pixel with 255 if any white pixel is encountered.

### CANNY EDGE DETECTEOR

#### NOISE REDUCTION

The Gaussian filter is used for this purpose.

#### SOBEL FILTER

To Find the intensity gradient of the image.

#### NON-MAXIMUM SUPPRESSION

This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.  
For this, we will traverse perpendicular to the edge(thick) found and find the local maximum value for intensity. This local maximum is then considered to be final edge and the rest is truncated.

HYSTERESIS THRESHOLDING  
Canny uses two thresholds (upper and lower):  
If a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge  
If a pixel gradient value is below the *lower* threshold, then it is rejected.  
If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.

**Canny (source, output, lowThreshold, lowThreshold\*ratio, kernel\_size);**

*detected\_edges*: Source image, grayscale

*detected\_edges*: Output of the detector (can be the same as the input)

*lowThreshold*: The value entered by the user moving the Trackbar

*highThreshold*: Set in the program as three times the lower threshold (following Canny’s recommendation)

*kernel\_size*: We defined it to be 3 (the size of the Sobel kernel to be used internally)

## OBJECT DETECTION (GRAPH THEORY)

## Algorithm for traversing or searching tree or graph data structures

### DEPTH FIRST SEARCH

One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

#### Using RECURSION

void DFS (int v, visited [])

{

    // Mark the current node as visited and print it

    // Recur for all the vertices adjacent to this vertex

}

void main ()

{

    // Mark all the vertices as not visited

    // Call the recursive helper function to print DFS traversal

 }

#### Using STACK

void BFS (int s, visited [])

{

    while (! STACK. Empty ())

    {

        // POP a vertex from STACK and print it

        // Get all adjacent vertices of the POPED vertex

        // If adjacent has not been visited, then mark it visited

         and PUSH it

    }

}

void main ()

{

    // Mark all the vertices as not visited

    // Call the recursive helper function to print DFS traversal

}

# DAY 4

## OBJECT DETECTION

### BREADTH FIRST SEARCH

#### using QUEUE

Breadth – first searches are performed by exploring all nodes at a given depth before proceeding to the next level. This means that all immediate children of nodes are explored before any of the children’s children are considered.

void BFS (int s, visited [])

{

    while (! queue. Empty ())

    {

        // Dequeue a vertex from queue and print it

        // Get all adjacent vertices of the dequeued vertex

        // If adjacent has not been visited, then mark it visited

         and enqueue it

    }

}

void main ()

{

    // Mark all the vertices as not visited

    // Call the recursive helper function to print BFS traversal

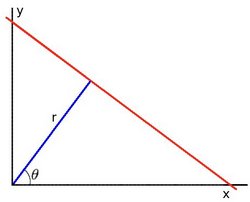
}

## LINE TRANSFORMATION

### HOUGH TRANSFORM: detect lines in an image.

To apply the Transform, first an edge detection pre-processing is desirable.

1. a line in the image space can be expressed with two variables. For example:
   1. In the **Cartesian coordinate system:** Parameters: (m,b).
   2. In the **Polar coordinate system:** Parameters: (r,\theta)



For Hough Transforms, we will express lines in the *Polar system*. Hence, a line equation can be written as:

y = \left ( -\dfrac{\cos \theta}{\sin \theta} \right ) x + \left ( \dfrac{r}{\sin \theta} \right )

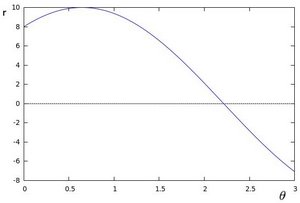
Arranging the terms: r = x \cos \theta + y \sin \theta

1. In general for each point (x_{0}, y_{0}), we can define the family of lines that goes through that point as:

r_{\theta} = x_{0} \cdot \cos \theta  + y_{0} \cdot \sin \theta

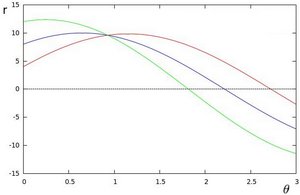
Meaning that each pair (r_{\theta},\theta)represents each line that passes by (x_{0}, y_{0}).

1. If for a given (x_{0}, y_{0})we plot the family of lines that goes through it, we get a sinusoid. For instance, for x_{0} = 8and y_{0} = 6we get the following plot (in a plane \theta- r):



We consider only points such that r > 0and 0< \theta < 2 \pi.

1. We can do the same operation above for all the points in an image. If the curves of two different points intersect in the plane \theta- r, that means that both points belong to a same line. For instance, following with the example above and drawing the plot for two more points: x_{1} = 4, y_{1} = 9and x_{2} = 12, y_{2} = 3, we get:



The three plots intersect in one single point (0.925, 9.6), these coordinates are the parameters (\theta, r) or the line in which (x_{0}, y_{0}), (x_{1}, y_{1})and (x_{2}, y_{2})lay.

1. What does all the stuff above mean? It means that in general, a line can be *detected* by finding the number of intersections between curves. The more curves intersecting means that the line represented by that intersection have more points. In general, we can define a *threshold* of the minimum number of intersections needed to *detect* a line.
2. This is what the Hough Line Transform does. It keeps track of the intersection between curves of every point in the image. If the number of intersections is above some *threshold*, then it declares it as a line with the parameters (\theta, r_{\theta})of the intersection point.

# DAY 5

## CONTOURS

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same colour or intensity.

**findContours (Mat img, contour, hierarchy, mode, method, offset)**

img: input image  
**contour**: array of coordinates of all the boundary points of an object  
**hierarchy**: representation of parent-child relationship between contours   
**mode**: retrieval mode  
**method**: used for contour approximation  
**offset** – Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

### CONTOUR Detected contours. Each contour is stored as a vector of points.

It stores 2-D array of vectors, where each row represents a contour and every element in a row represents coordinates of all the boundary points of an object.

### HIERARCHY

Sometimes objects are in different locations. But in some cases, some shapes are inside other shapes. Just like nested figures. In this case, we call outer one as **parent** and inner one as **child**. This way, contours in an image has some relationship to each other.

So each contour has its own information regarding what hierarchy it is, who is its child, who is its parent etc. OpenCV represents it as an array of four values: **[Next, Previous, First\_Child, Parent]  
Next** denotes next contour at the same hierarchical level.  
**Previous** denotes previous contour at the same hierarchical level.  
**First\_Child** denotes its first child contour.  
**Parent** denotes index of its parent contour.

### RETRIEVAL MODE

#### RETR\_LIST This is the simplest of the four flags (from explanation point of view). It simply retrieves all the contours, but doesn't create any parent-child relationship. Parents and kids are equal under this rule, and they are just contours. ie they all belongs to same hierarchy level.

#### RETR\_EXTERNAL If you use this flag, it returns only extreme outer flags. All child contours are left behind. We can say, under this law, Only the eldest in every family is taken care of. It doesn't care about other members of the family.

#### RETR\_CCOMP

#### RETR\_TREE And this is the final guy, Mr. Perfect. It retrieves all the contours and creates a full family hierarchy list. It even tells, who is the grandpa, father, son, grandson and even beyond...

### METHOD

#### CV\_CHAIN\_APPROX\_NONE

It stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighboUrs, that is, max(abs(x1-x2),abs(y2-y1))==1.

#### CV\_CHAIN\_APPROX\_SIMPLE

It compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.

**approxPolyDP (**InputArray **curve**, OutputArray **approxCurve**, double **epsilon**, bool **closed)**

Approximates a polygonal curve(s) with the specified precision.

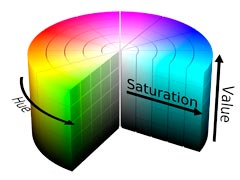
InputArray **curve** – input vector of a 2-D point stored in a Mat

OutputArray **approxCurve** – Result of the approximation.

**epsilon** – Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation.  
**closed** – If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.

## HSV/HSL Colour space

HSL stands for *hue*, *saturation*, and *lightness* (or *luminosity*). HSV stands for *hue*, *saturation*, and *value*.  
HUE: a colour or shade.  
SATURATION: It is determined by intensity. When a colour's saturation level is reduced to 0, it becomes a shade of grey.  
LUMINOSITY: It refers to how much white (or black) is mixed in the colour.



For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255].

## COLOUR TRANSFORMATION

**cv2.cvtColor(input\_image, flag)**

**flag:** it determines the type of conversion.  
For BGR → Grayscale cv2.COLOR\_BGR2GRAY.   
For BGR → HSV cv2.COLOR\_BGR2HSV.

## VIDEO PROCESSING

Video signal is basically any sequence of time varying images. A still image is a spatial distribution of intensities that remain constant with time, whereas a time varying image has a spatial intensity distribution that varies with time. Video signal is treated as a series of images called frames.

### CAPTURE VIDEO FROM CAMERA

**vid. VideoCapture vid(0);  
VideoCapture vid("C:\\Users\\Public\\Videos\\Sample Videos\\Wildlife.wmv");  
vid.read();**  
  
To capture a video, you need to create a **VideoCapture** object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera.

**VideoCapture** : Class for video capturing from video files, image sequences or cameras. **read() :** returns a bool (True/False). If frame is read correctly, it will be True. So you can check end of the video by checking this return value.

void main ()

{

vid. VideoCapture vid(0); //store a video  
vid.read();

while(1)

{

     // Capturing video frame by frame

     // display the resulting frame

}

}

# DAY 6

## CASCADE CLASSIFICATION (OBJECT DETECTION)

First, a classifier is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size.

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a “1” if the region is likely to show the object (i.e., face/car), and “0” otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily “resized” in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

The word “cascade” in the classifier name means that the resultant classifier consists of several simpler classifiers (stages) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed.

