
# OpenCV: Modules and Functions

Chih-Yun Pai

# Outline

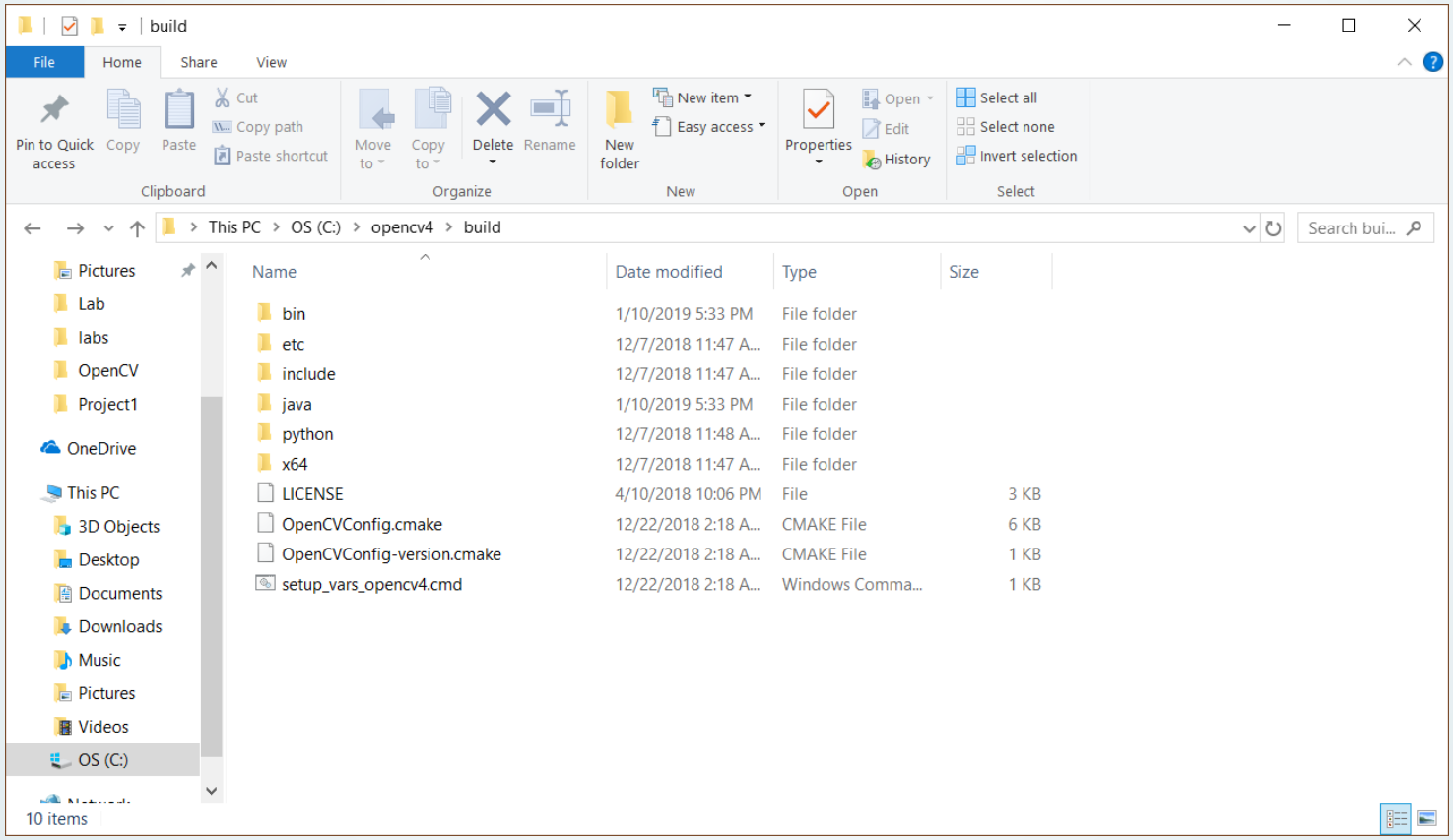
- Introduction
- Setup for the Sample Code
- OpenCV Modules
- Image processing Functions
    - Thresholding
    - Smoothing
    - Edge Detection
    - Hough Transform
    - Color Space
- Application: Face Detection

# Introduction

- OpenCV is a C++ library for **image processing** and **computer vision**.
- It has hundreds of built-in functions.
- **Open Source and Free.**
- It runs on **Windows**, **Linux, and Mac.**
- There are also full interfaces of C, Python and JAVA.

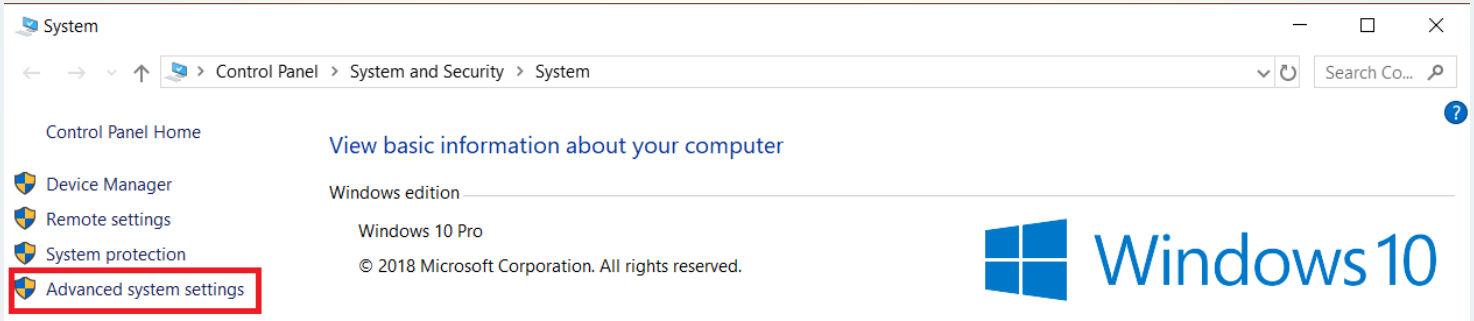
# Setup for the Sample Code

- Windows: (Visual Studio C++)
  - Download OpenCV: https://opencv.org/releases.html
  - Check the location

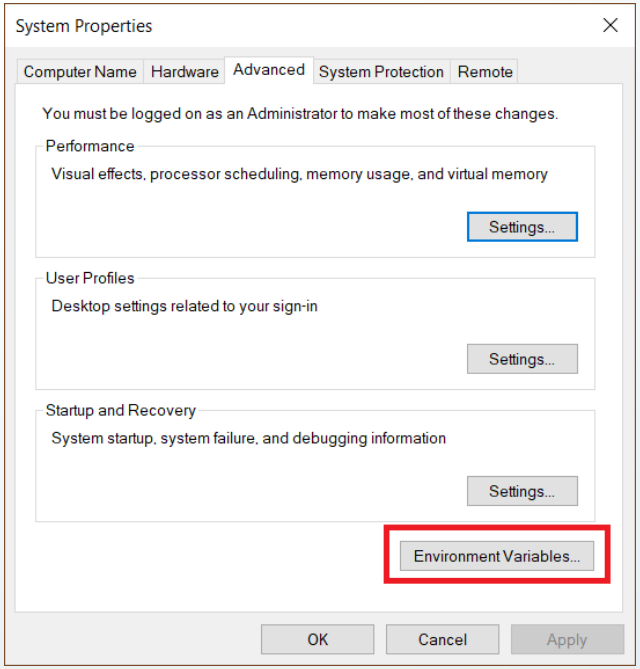


# Setup for the Sample Code

- Windows: (Visual Studio C++)
  - Setup System Properties

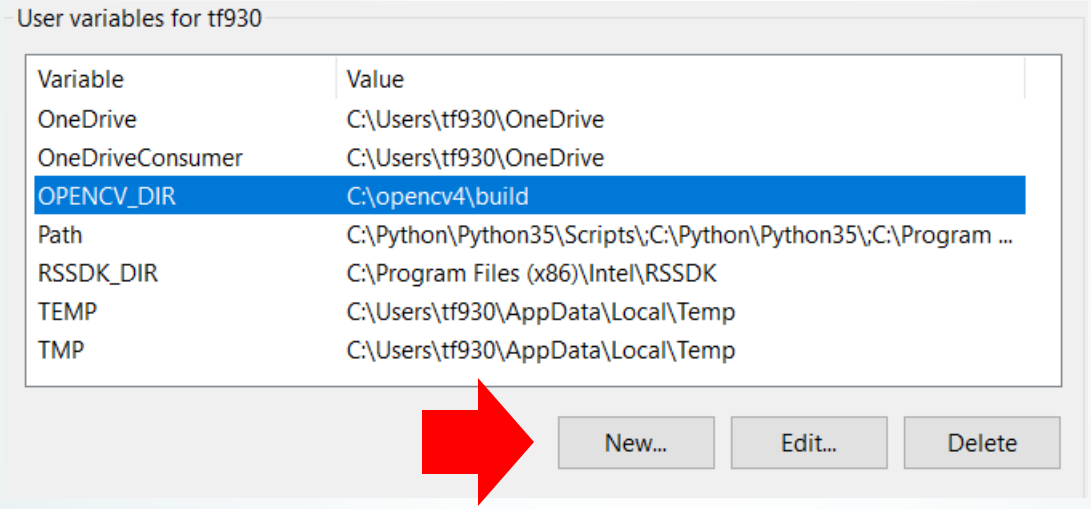


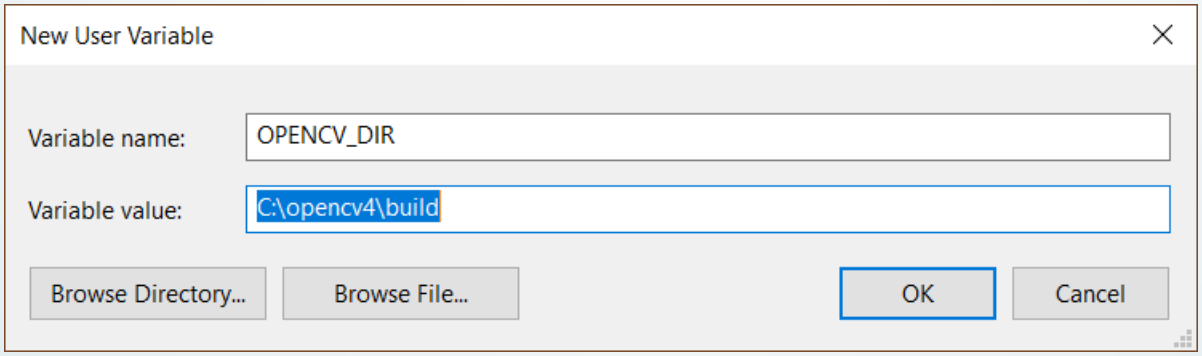

  - Setup Environment Variables



# Setup for the Sample Code

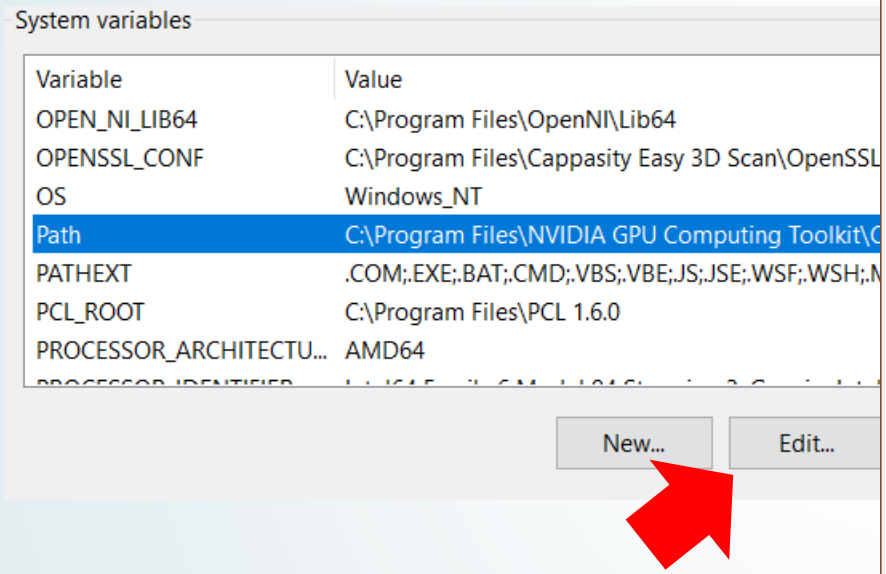
- Windows: (Visual Studio C++)
  - Add user variable

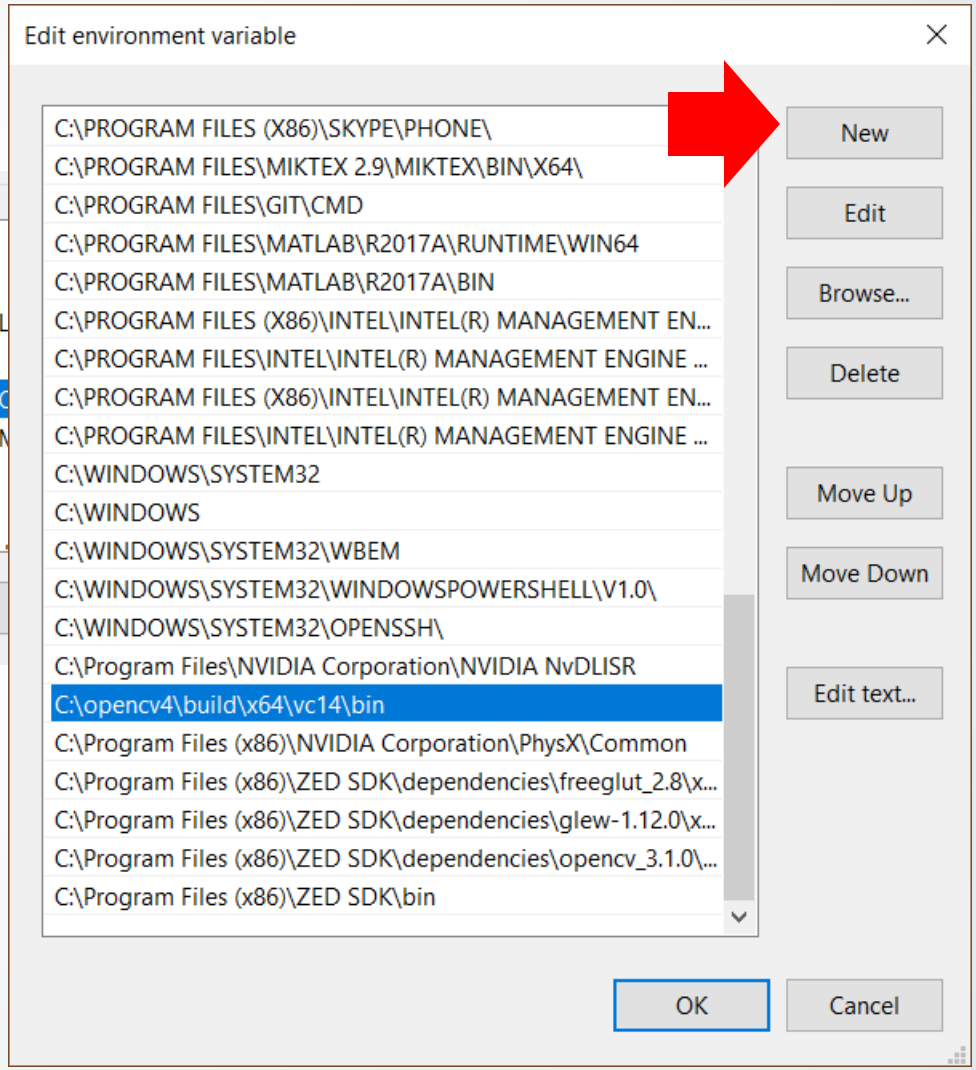


# Setup for the Sample Code

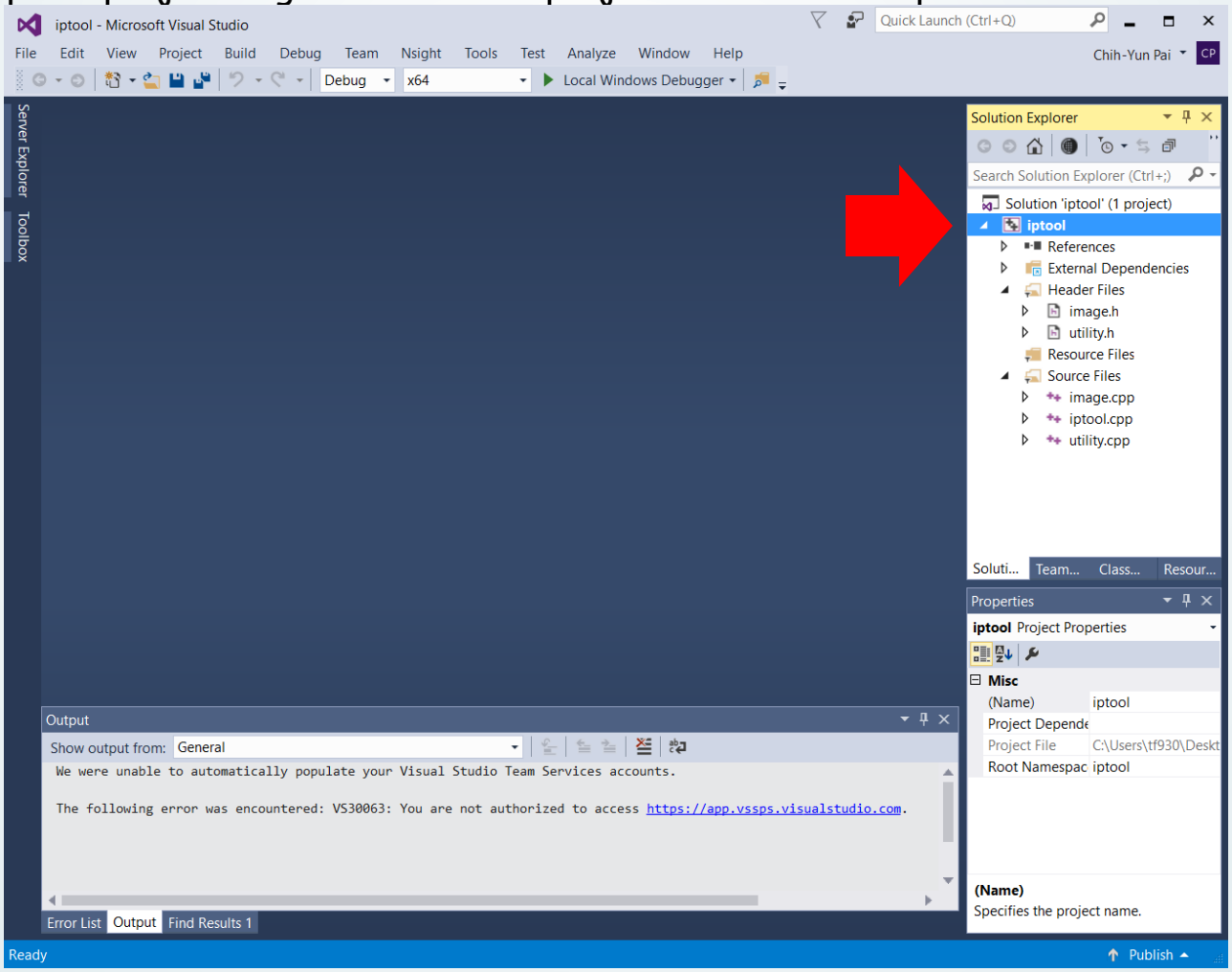
- Windows: (Visual Studio C++)
  - Edit Path in System Variable



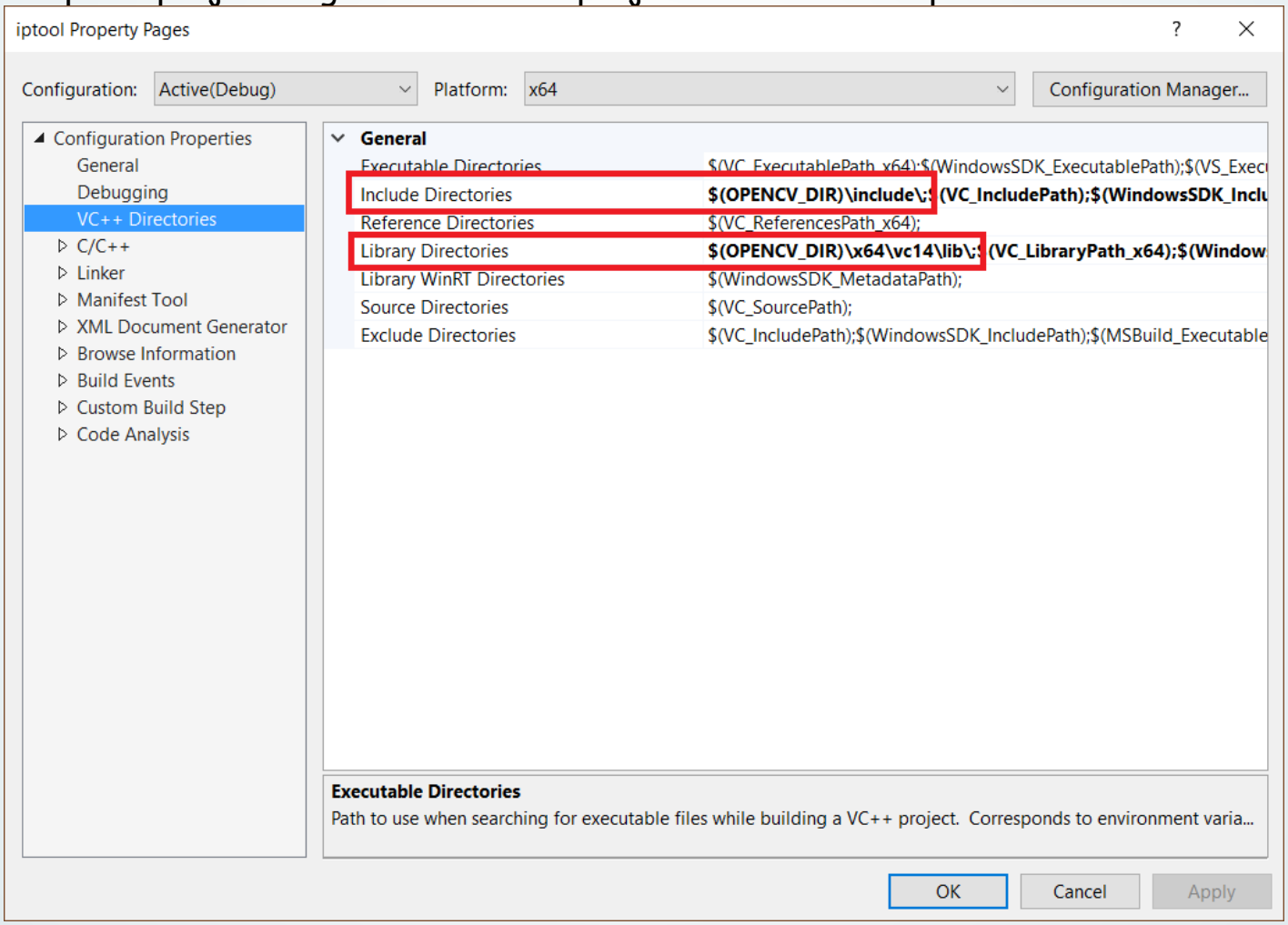
# Setup for the Sample Code

- Windows: (Visual Studio C++)
  - Setup the project: right click on the project and select Properties



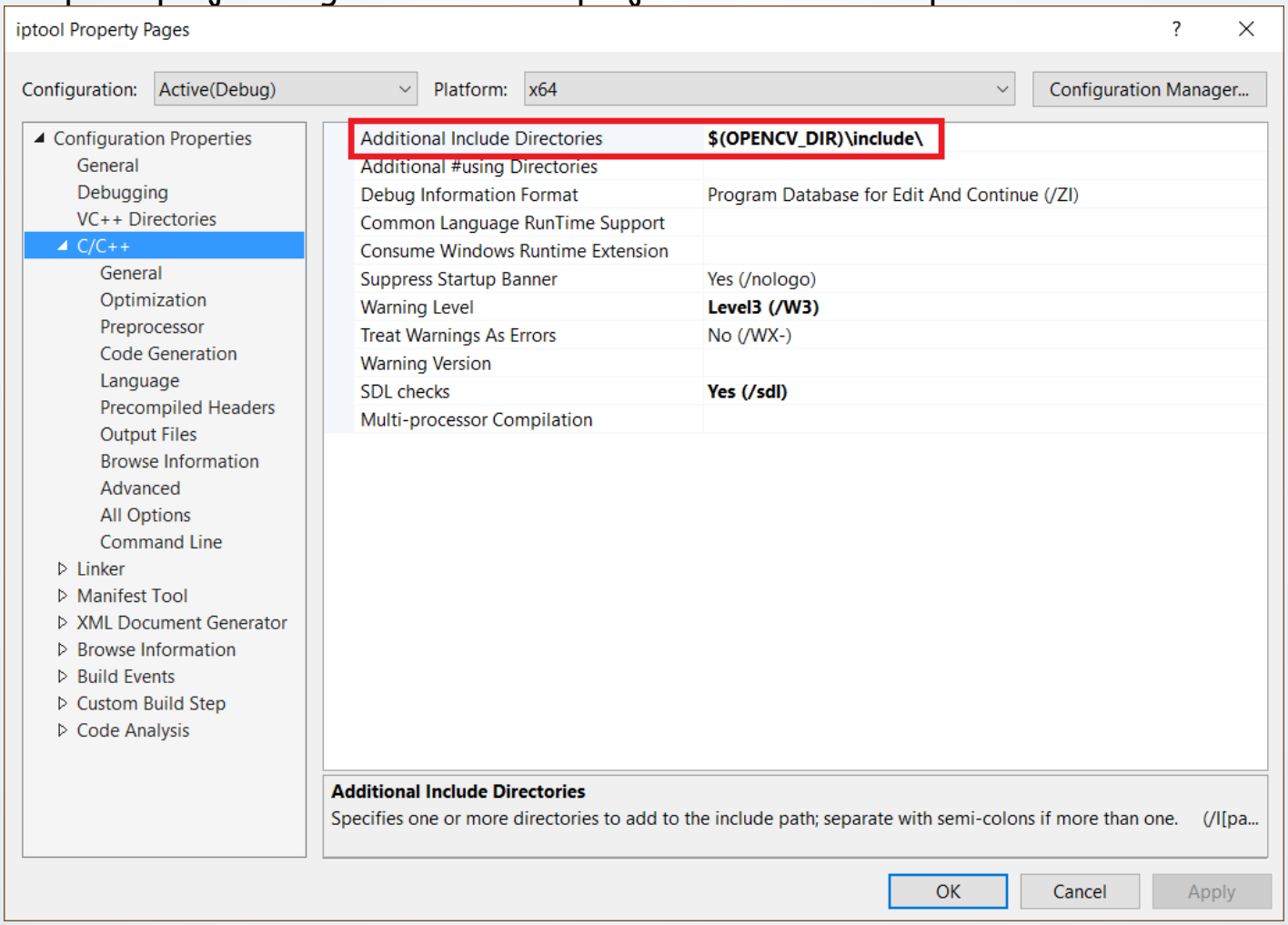
# Setup for the Sample Code

- Windows: (Visual Studio C++)
  - Setup the project: right click on the project and select Properties



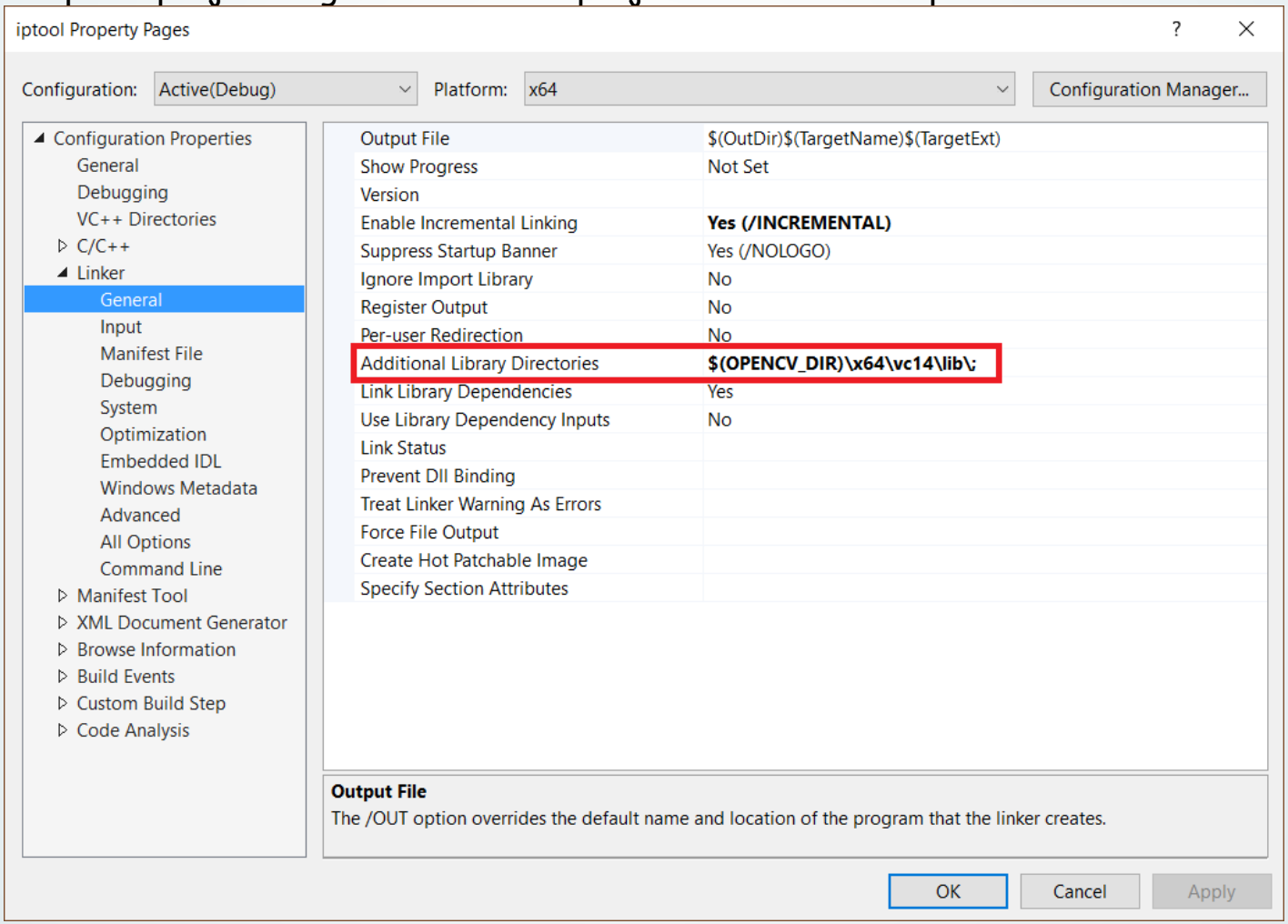
# Setup for the Sample Code

- Windows: (Visual Studio C++)
  - Setup the project: right click on the project and select Properties



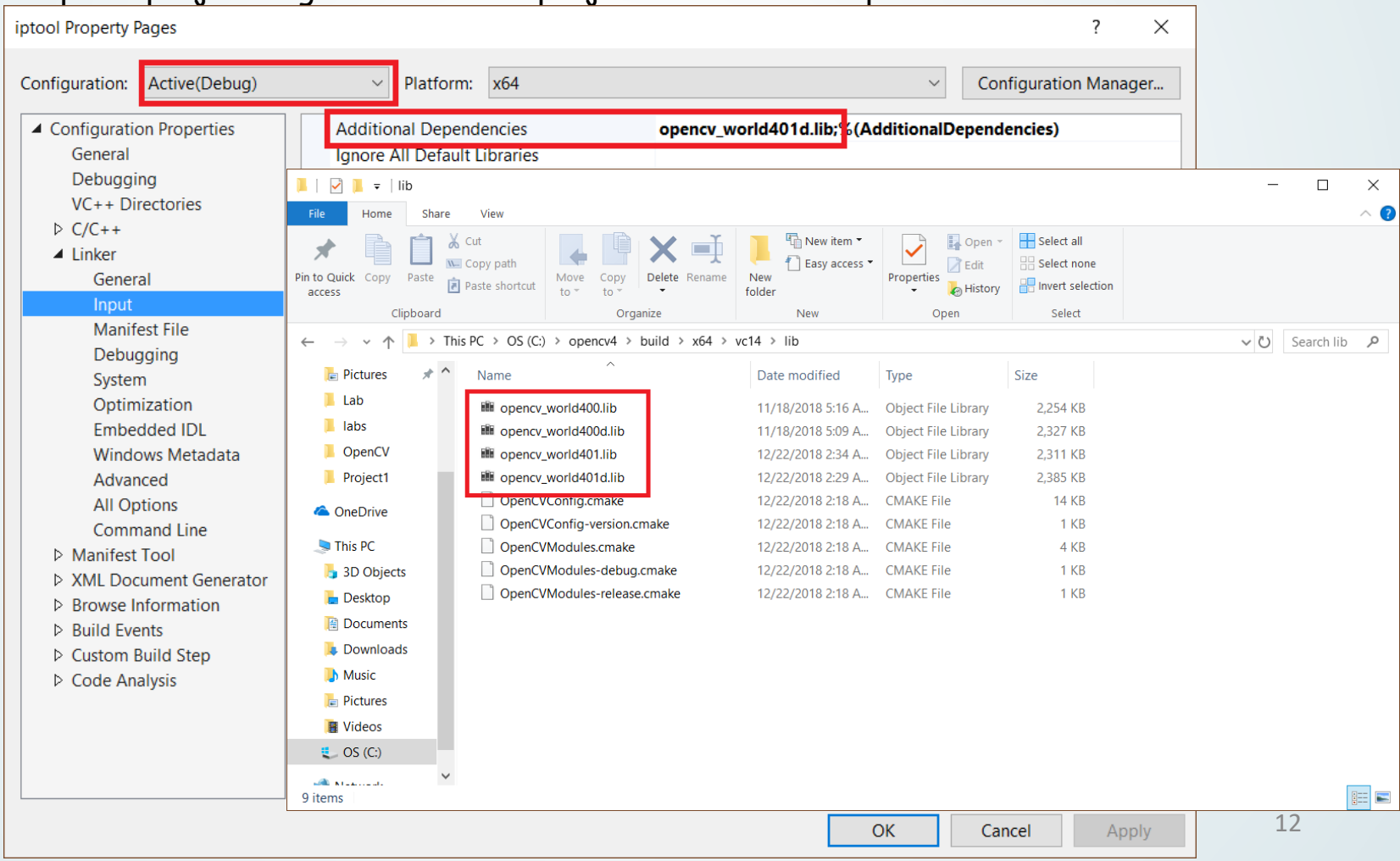
# Setup for the Sample Code

- Windows: (Visual Studio C++)
  - Setup the project: right click on the project and select Properties



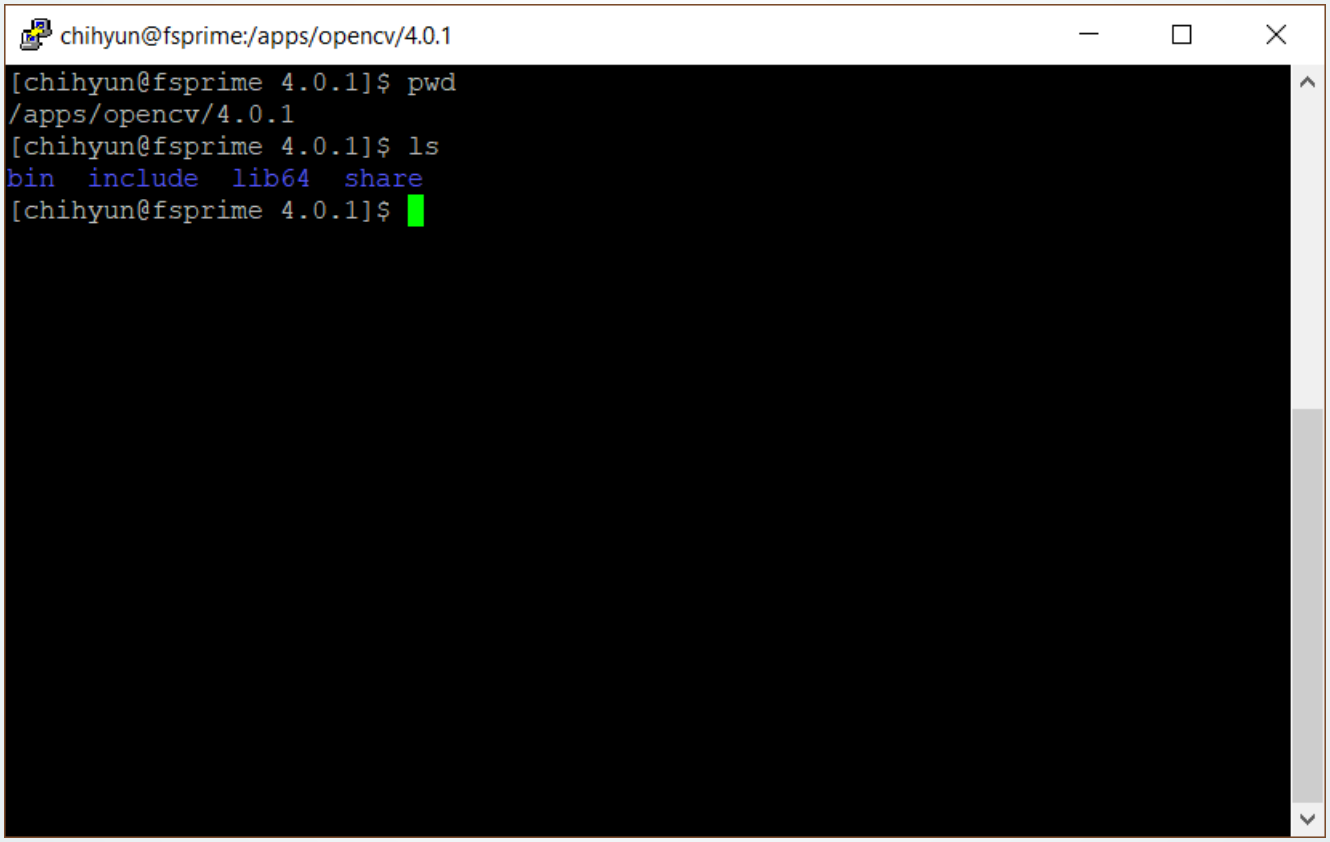
# Setup for the Sample Code

- Windows: (Visual Studio C++)
  - Setup the project: right click on the project and select Properties



# Setup for the Sample Code

- Linux: (FSPrime server)
  - Location of OpenCV 4: /apps/opencv/4.0.1



# Setup for the Sample Code

- Linux: (FSPrime server)
  - Makefile in iptools

```
# opencv library
OPENCVLIBS = -L/apps/opencv/4.0.1/lib64 -lopencv_core -lopencv_highgui -lopencv_imgcodecs -lopencv_imgproc -lopencv_objdetect -Wl,-rpath=/apps/opencv/4.0.1/lib64

# include opencv directory
OPENCVINCLUDE = -I/apps/opencv/4.0.1/include/opencv4
```

```
.cpp.o:
    $(CCC) $(OPENCVLIBS) $(OPENCVINCLUDE) $(INCLUDES) $(CCFLAGS) -c $< -o $@
```

  - Makefile in project

```
# opencv library
OPENCVLIBS = -L/apps/opencv/4.0.1/lib64 -lopencv_core -lopencv_highgui -lopencv_imgcodecs -lopencv_imgproc -lopencv_objdetect -Wl,-rpath=/apps/opencv/4.0.1/lib64

# include opencv directory
OPENCVINCLUDE = -I/apps/opencv/4.0.1/include/opencv4
```

```
DO_EXEC = g++ -std=c++11 $@.cpp $(OPENCVLIBS) $(OPENCVINCLUDE) -o $(BIN_DIR)$@ -L ../lib -l iptools
```

  - Compile with make
  - Run with ./iptool parameters.txt

# OpenCV Modules

- OpenCV is a modular structure library.
- It has several modules, which include:
  - **core**: basic OpenCV data structures.
  - **highgui**: image I/O operations.
  - **imgproc**: image processing functions (e.g., filtering and thresholding).
  - ….. Other modules

# OpenCV Modules

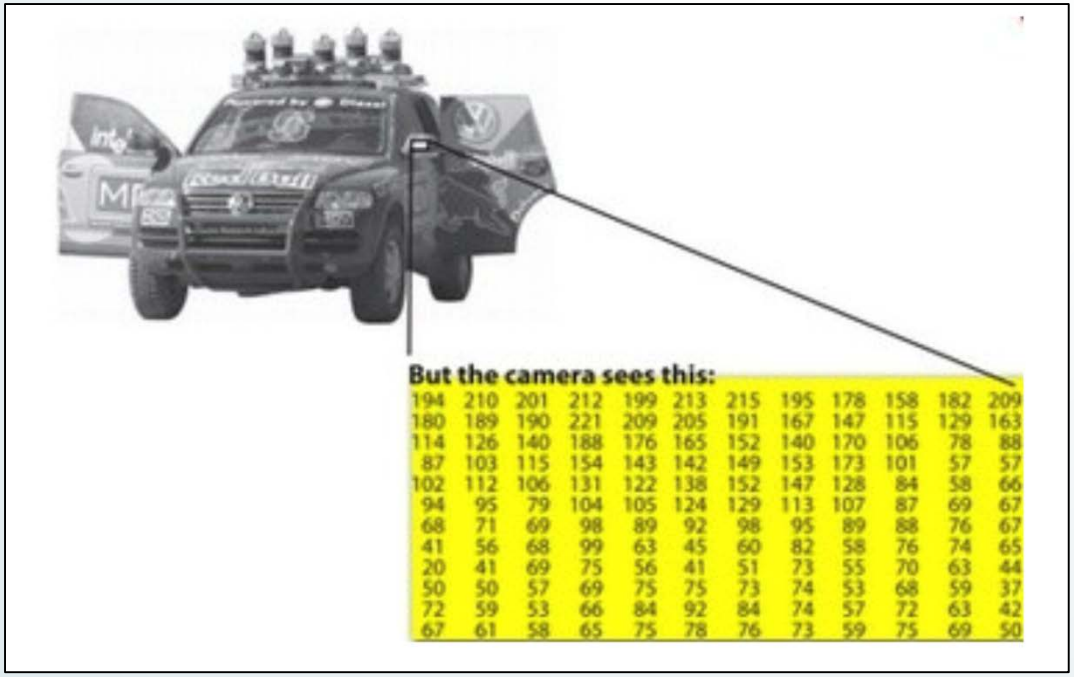
# OpenCV Modules: core Module

- core module defines the basic OpenCV data structures.

- Basic data structures include:
  - **Point:** 2D point (x,y)
  - **Rect:** 2D rectangle object
  - **Vec**: row or column vector
  - **Mat:** Matrix object

# Mat: The basic Image Container

- Mat is the primary data structure in OpenCV
- It is used to store an image as numerical matrix.
- Each cell of the matrix represents the intensity of a specific pixel.




Remember:

*rows is your y coordinate*
*and*
*cols is your x coordinate*

# Mat Data Structure (cont.)

- **Functions:**
  - Mat.**at**<imagetype>(x, y)[channel] – returns the intensity of a pixel at x and y coordinates.
  - Mat.**channels**() - returns the image's number of channels.
  - Mat(**Rect** (x, y, sx, sy)) - returns sub image.
  - Mat.**size**() - returns the SIZE of an image.
  - Mat.**type**() - returns the TYPE of an image.

# Mat Data Structure (cont.)

- Return code for the datatype

| | C1 | C2 | C3 | C4 | C(5) | C(6) | C(7) | C(8) |
|---|---|---|---|---|---|---|---|---|
| CV_8U | 0 | 8 | 16 | 24 | 32 | 40 | 48 | 56 |
| CV_8S | 1 | 9 | 17 | 25 | 33 | 41 | 49 | 57 |
| CV_16U | 2 | 10 | 18 | 26 | 34 | 42 | 50 | 58 |
| CV_16S | 3 | 11 | 19 | 27 | 35 | 43 | 51 | 59 |
| CV_32S | 4 | 12 | 20 | 28 | 36 | 44 | 52 | 60 |
| CV_32F | 5 | 13 | 21 | 29 | 37 | 45 | 53 | 61 |
| CV_64F | 6 | 14 | 22 | 30 | 38 | 46 | 54 | 62 |

Source: https://stackoverflow.com/questions/10167534/how-to-find-out-what-type-of-a-mat-object-is-with-mattype-in-opencv

# highgui Module: Image I/O Operations

- OpenCV provides simple and useful ways to read and write images.

- Examples

```
//Read an image
Mat image = imread( <filename>)

//Write an image
imwrite( <string filename> , image );

//Output image to window
imshow( <window name> , <image Mat to show> );

//pause program for input
key = waitKey( 0 );
```

# Example Code

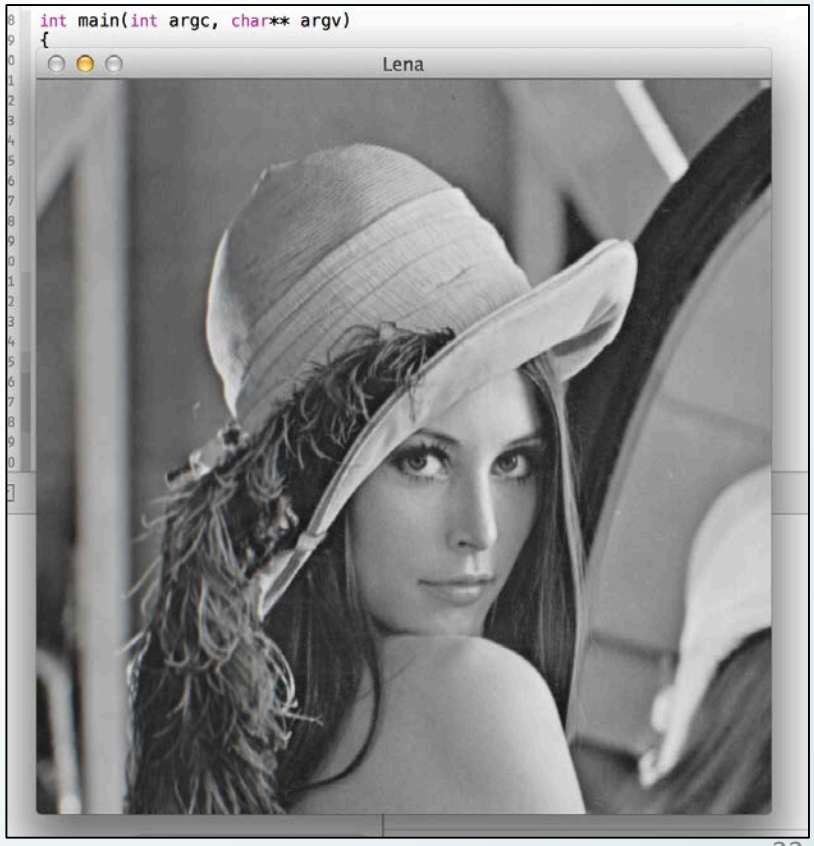
```
//header of modules
#include <opencv2/opencv.hpp>

// OpenCV uses cv namespace
using namespace cv;

int main(int argc, char* argv[ ]){
   Mat image = imread(argv[1]);

   namedWindow("Lena");
   imshow("Lena",image);
   waitKey(0);
   imwrite("lena_copy.jpg",image);
   return 0;
}
```

**This program will load and show an image**



# imageproc Module

- Image processing module has many functions such as:
  - Thresholding
  - Image smoothing
  - Edge detections
  - Hough Transform
  - Color space conversion
  - Histogram modifications
  - and so on…

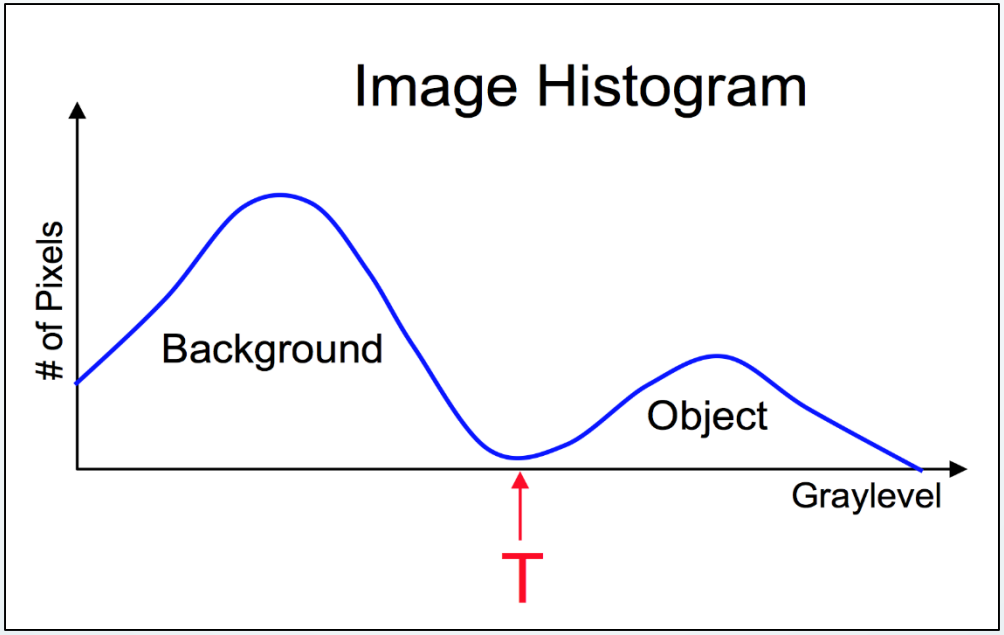
# Thresholding

- Background:
  - It's a point operator.
  - The simplest method of segmentation.
  - Reject those pixels above or below some value (i.e., threshold) while keeping the others.

# Thresholding (cont.)

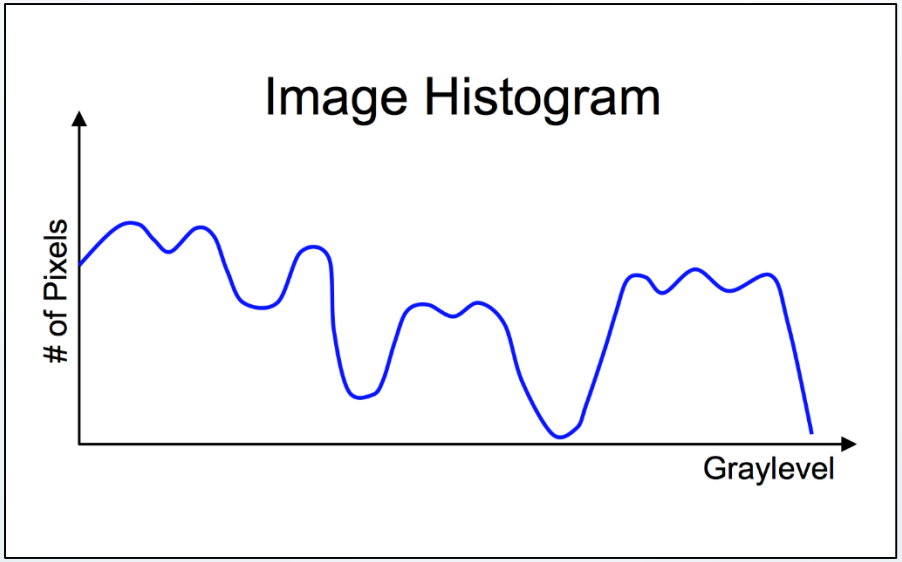
**Global Thresholding** =

Choose threshold **T** that separates object from background.



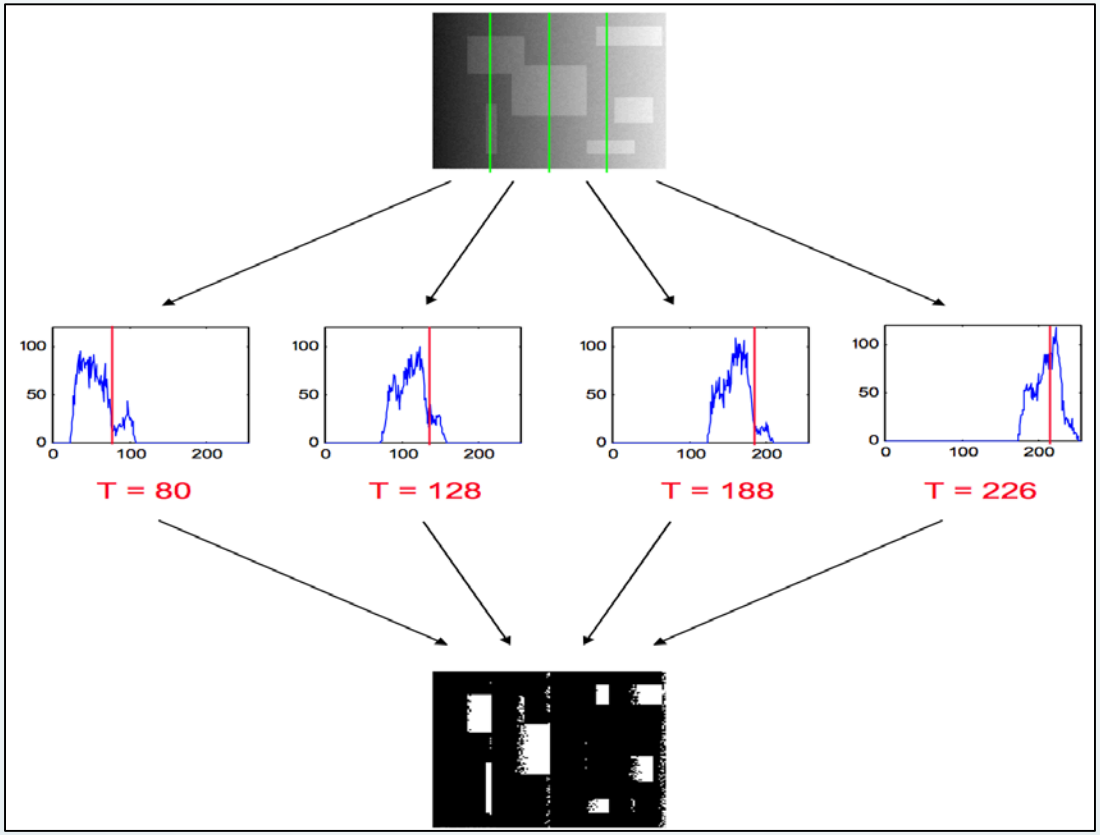
# Thresholding (cont.)

- The Simple global thresholding is not always possible.
- What if the image histogram due to noise or large intensities variations looks like:



# Thresholding (cont.)

- **Local thresholding:** divide the image into regions and perform thresholding in each region.



# Thresholding (cont.)

- **Other Methods:**
  - Adaptive Thresholding.
  - Otsu's Thresholding.
  - Region growing
  - Graph cut
  - and so on…

# OpenCV imageproc Module:
## Threshold Function

- void threshold( src_img, dst_img, thresh, maxVal, thresold_type);

  **Parameters:**

  - **src** – source image.
  - **dst** – Destination image.
  - **thresh** - a user specified threshold.
  - **maxValue** – The new pixel intensity
  - **thresholdType** – Thresholding type.

## Example:

```
threshold( src_gray, dst, 100, 255,
THRESH_BINARY );
```

# OpenCV imageproc Module: Threshold Function

- Below is a list of threshold_types in OpenCV:

- **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_BINARY_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases}$$

- **THRESH_TRUNC**

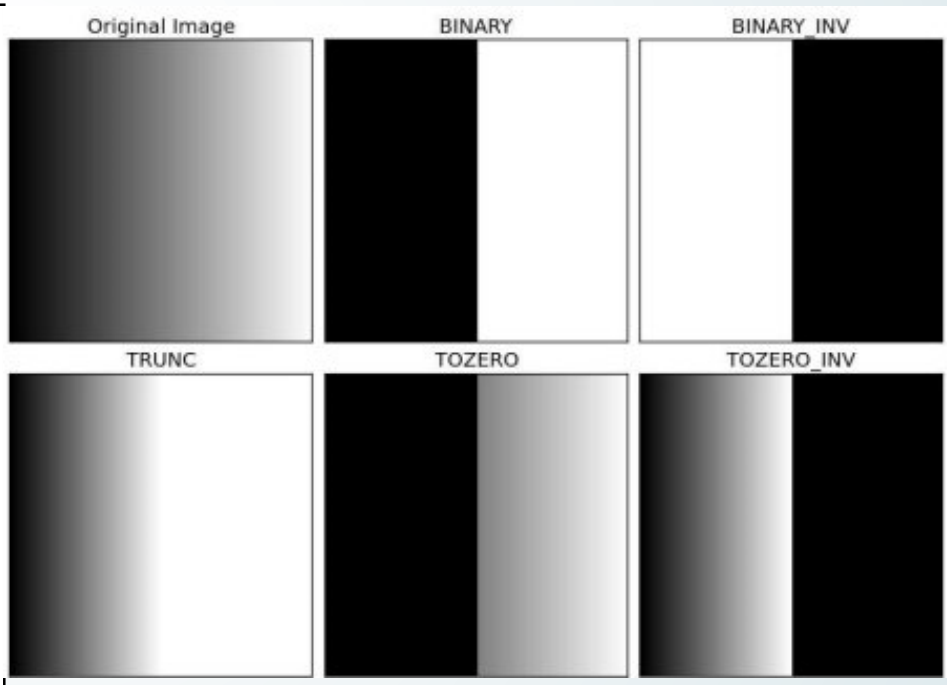
$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$



# Image Smoothing

- Image smoothing suppress image noise or undesired image fluctuations.
- Three smoothing functions in OpenCV:
  - Uniform Smoothing
  - Gaussian Smoothing
  - Median Smoothing

# Uniform Smoothing

- Average smoothing slides a kernel (filter) across the image and replaces each pixel with an average of its neighborhood

- Example:

- blur(src, dst, ksize);
  - src - input image
  - dst - output image
  - ksize - blurring kernel size (e.g., 3x3)

# Gaussian Smoothing

- Gaussian smoothing is the result of blurring an image by a Gaussian function
- Example:
- GaussianBlur( src, dst, ksize, sigmaX, sigmaY );
  - src - input imag
  - .dst - output image
  - ksize - Gaussian kernel width and height can differ but they both must be positive and odd. 0 for computing from Sigma
  - sigmaX - Gaussian kernel standard deviation in X direction
  - sigmaY - Gaussian kernel standard deviation in Y direction

# Median Smoothing

- The median filter works by moving through the image and replacing each value with the median value of neighboring pixels.
- Very effective at removing 'salt and pepper' noise.
- Example:
  - medianBlur( src, dst, ksize );
    - src - input image
    - dst - output image
    - ksize - kernel size;  it must be odd

# Example

Original

Blur

medianBlur

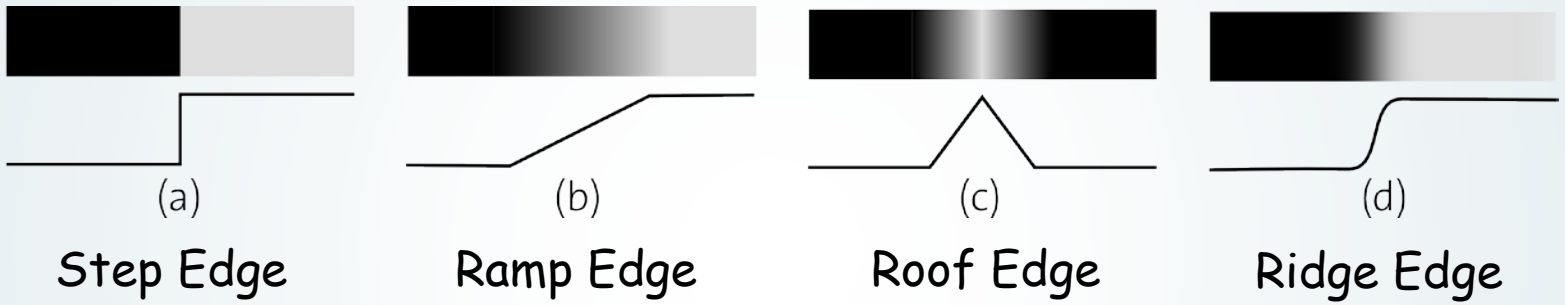
GussianBlur

# Edge Detections

- Another way for image segmentation.
- Convert a 2D image into a set of curves (i.e., edges) and background.
- Edges are significant local intensity changes in an image.
- Edge Types include: step edge, ramp edge, ridge edge, and roof edge.

# Edge Detections

- Examples of edge types:

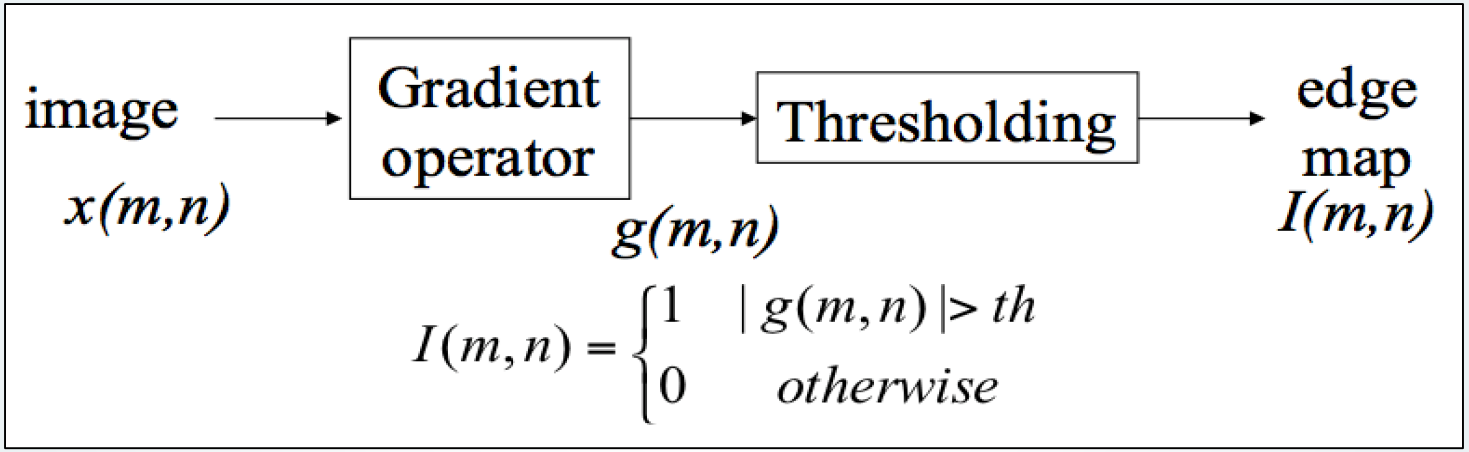



- Gradient Operators to detect edges.

change in the pixel value ⟶ large gradient

# Edge Detections

- Gradient Operator:




- The gradient of the image at every point is computed using:

$g_x(m,n) = x(m,n) * H_1 \rightarrow g_x$

$g_y(m,n) = x(m,n) * H_2 \rightarrow g_y$

# Edge Detections

Where I(m,n) denote an arbitrary image location and $H_1$ and $H_2$ are masks (i.e., gradient operators) to measure the gradient of the image in two orthogonal directions

- Then, we compute the gradient magnitude:

$$g(m,n) = \sqrt{{g_x}^2(m,n) + {g_y}^2(m,n)}$$

- Examples of gradient operators or masks:

$$\text{Sobel} : H_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad H_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

# Edge Detections

$$\text{Prewitt}: \quad H_1 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad H_2 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\text{Roberts}: \quad H_1 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \qquad H_2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

# OpenCV imageproc Module:
## Edge Detections

- void Sobel(src, dst, ddepth, xorder, yorder ksize)

  Parameters:

  – **src** – input image.
  – **dst** – output image.
  – **ddepth** – output image depth.
  – **xorder & yorder** – x direction or y direction.
  – **ksize** – size of the Sobel kernel (H); it must be odd value.

# **OpenCV imageproc Module:**
## Edge Detections

- Example:

Original Image

Horizontal Edge

Vertical Edge

# Hough Transform

- It is a transform for direct object recognition (e.g., lines, circles).
- Apply on edge detected image.
- Key Idea: edges VOTE for the possible model

# Parameter Space

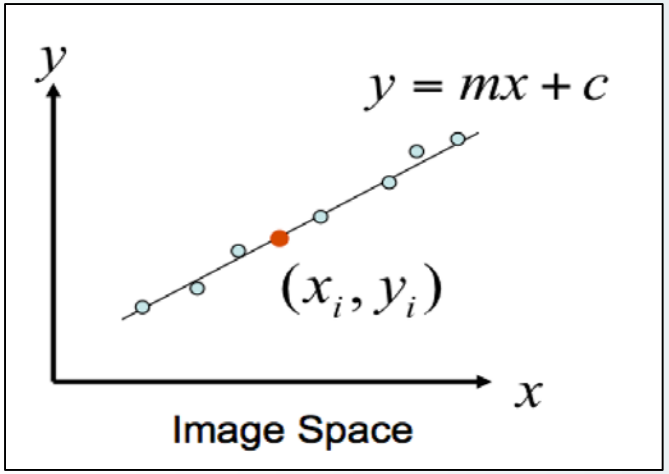
Equation of Line: $y = mx + c$

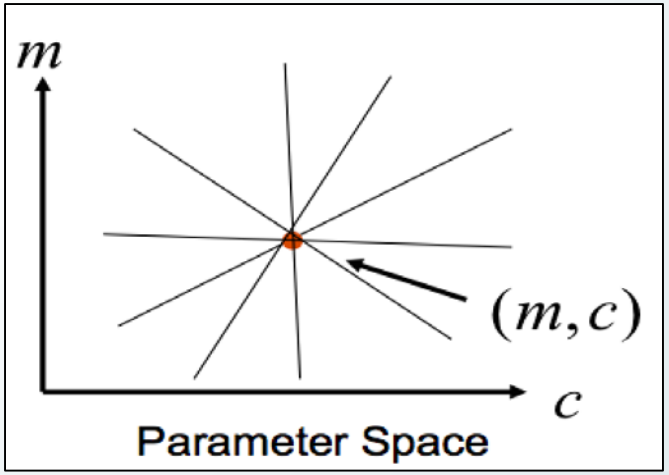
Find: (m,c)

Consider point: $(x_i , y_i)$

$y_i = mx_i + c$ or $c = -mx_i + y_i$

Parameter space also called Hough Space



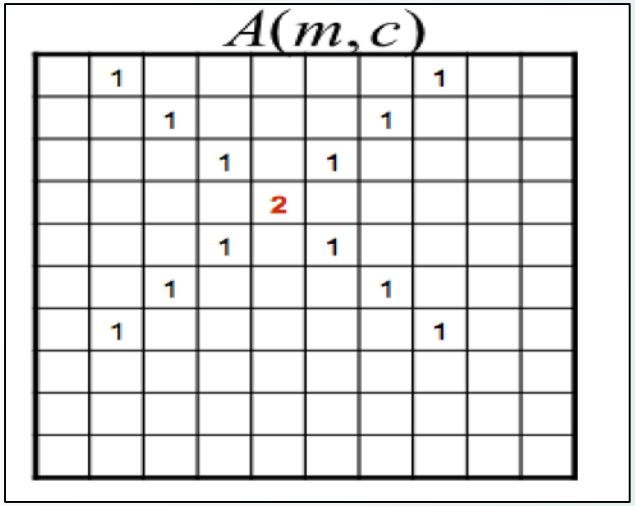
# Line Detection

Algorithm:

- Quantize Parameter Space **(m,c)**
- Create Accumulator Array **A(m,c)**
- Set **A(m,c) = 0** for all m,c
- For each image edge **$(x_i, y_i)$** increment:

  **A(m,c) = A(m,c) + 1**
- If **(m,c)** lies on the line:

  $c = -mx_i + y_i$
- Find local maxima in **A(m,c)**



# OpenCV imageproc Module: Hough Transform – Lines

- void HoughLines(src, out_lines, rho, theta, thresh)

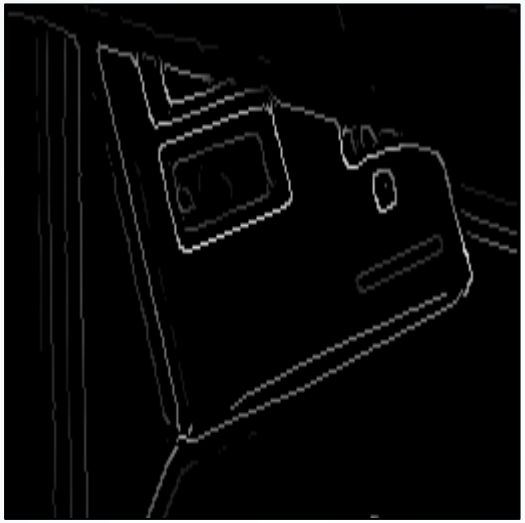
  Line described by $\text{rho} = x\cos\theta + y\sin\theta$

  Parameters:
  - **src** – input image. It should be the edge map.
  - **out_lines** – output vector of lines.
  - **rho** – distance resolution of the accumulator in pixels.
  - **theta**– angle resolution of the accumulator in radians.
  - **thresh** – accumulator threshold parameter. Only those lines are returned that get enough votes (> thresh).
- Example:

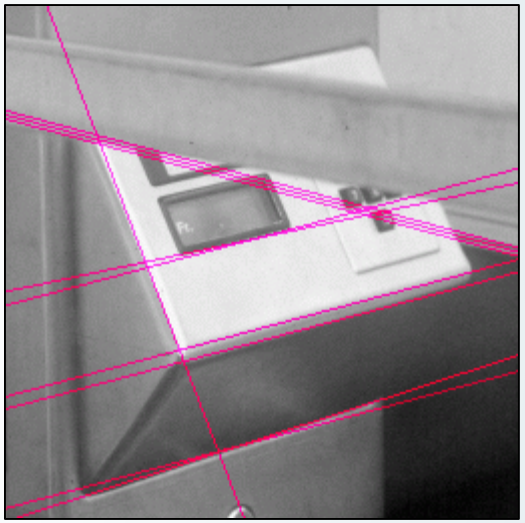
  *HoughLines( src, lines, 1, CV_PI/180, 100 );*

# Example

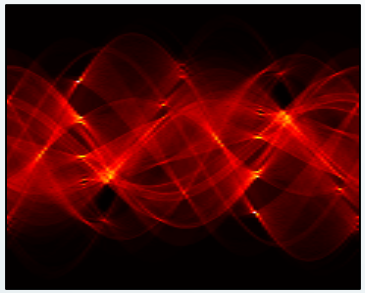
Original



Edge Detection



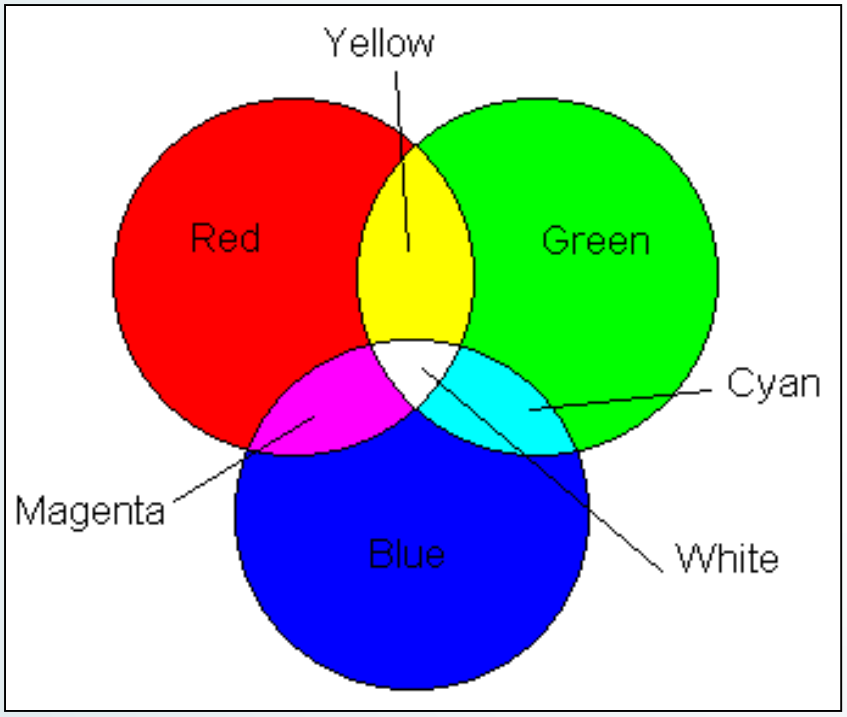
Found Lines

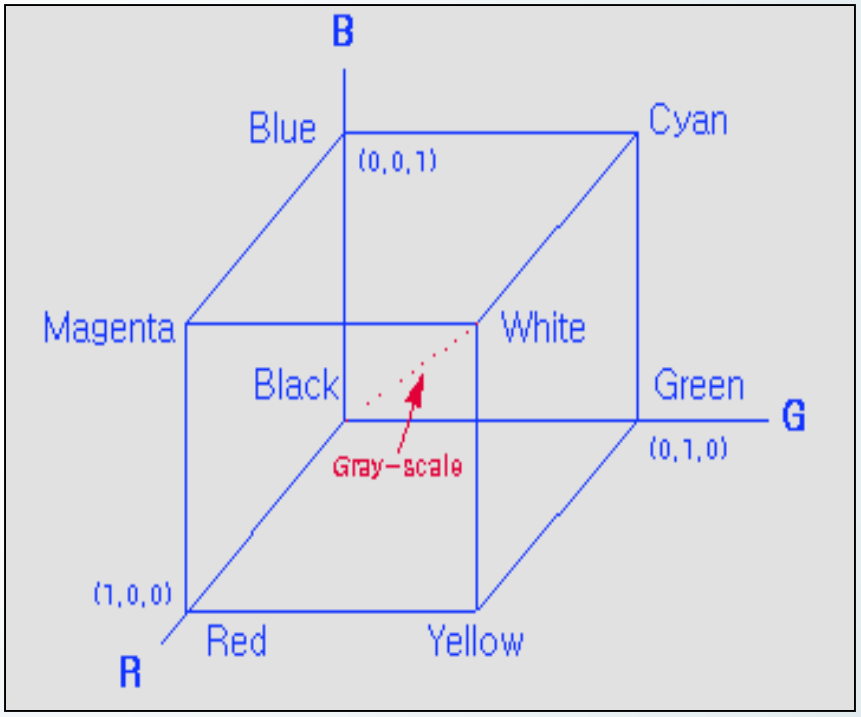


Parameter Space

# Color Space

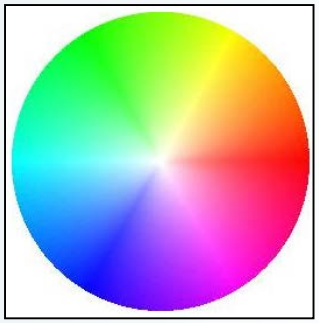
## RGB:

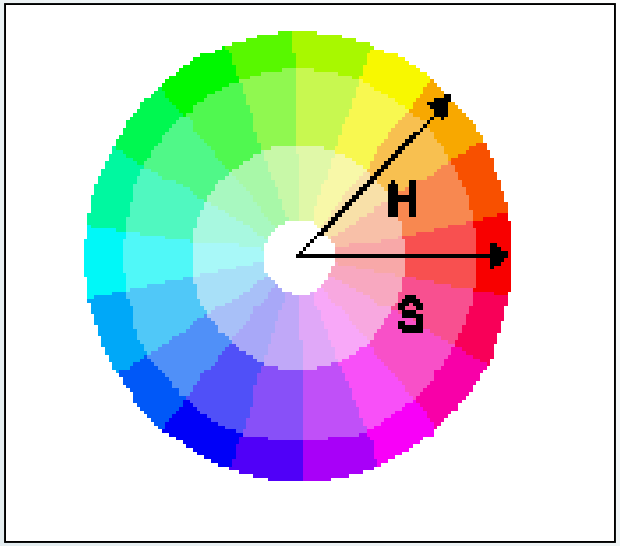


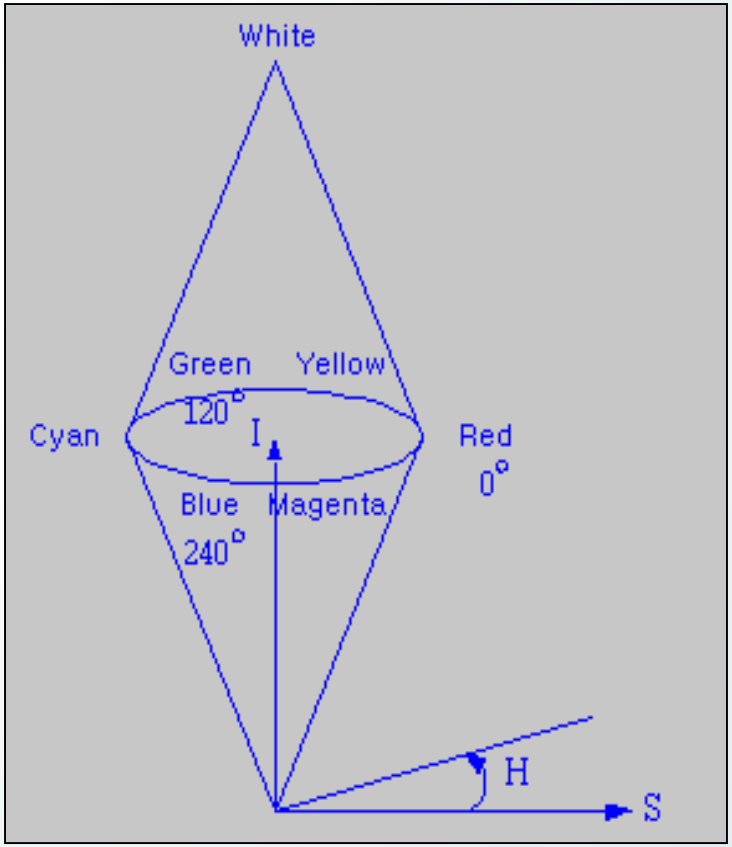



# Color Space

## HSI:

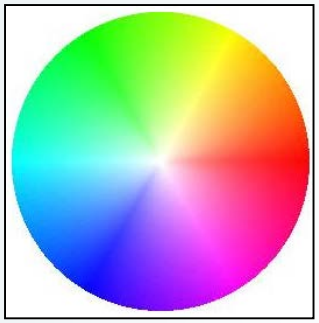


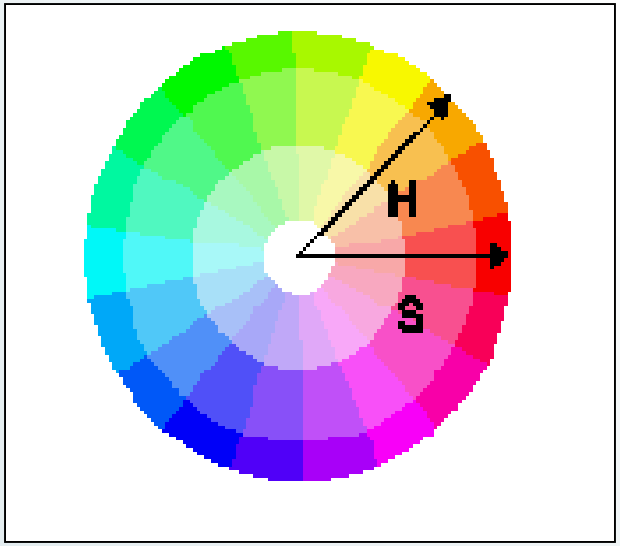


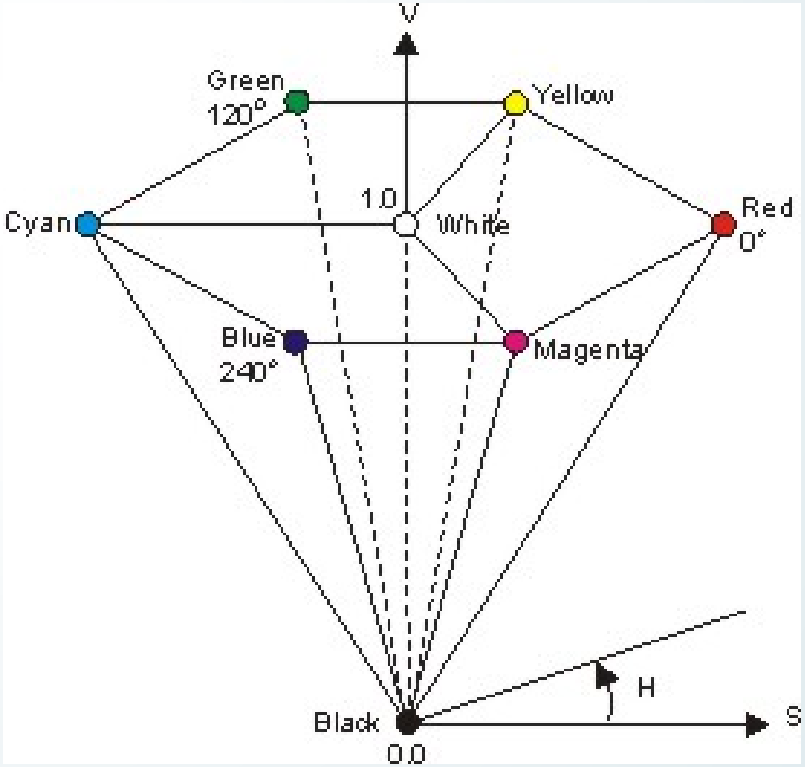



# Color Space

## HSV:





# Color Spaces in OpenCV

- Examples of color spaces in OpenCV:
  - **BGR (i.e., RGB)** - 3 channels color (blue, green, red).
  - **HSV** – Hue, Saturation, and Value; 3 channels.
  - **GRAYSCALE** - Single channel.
- The output value range is depend on the input.

# Color Spaces in OpenCV

- OpenCV function: void cvtColor(src, dst, code)
  - Parameters:
    - **src**: input image.
    - **dst**: output image.
    - **Code:** color space conversion code (e.g., COLOR_BGR2GRAY, COLOR_BGR2HSV)
- Example:
  - cvtColor(src, graysrc, COLOR_BGR2GRAY);
  - cvtColor(src, hsvsrc, COLOR_BGR2HSV);

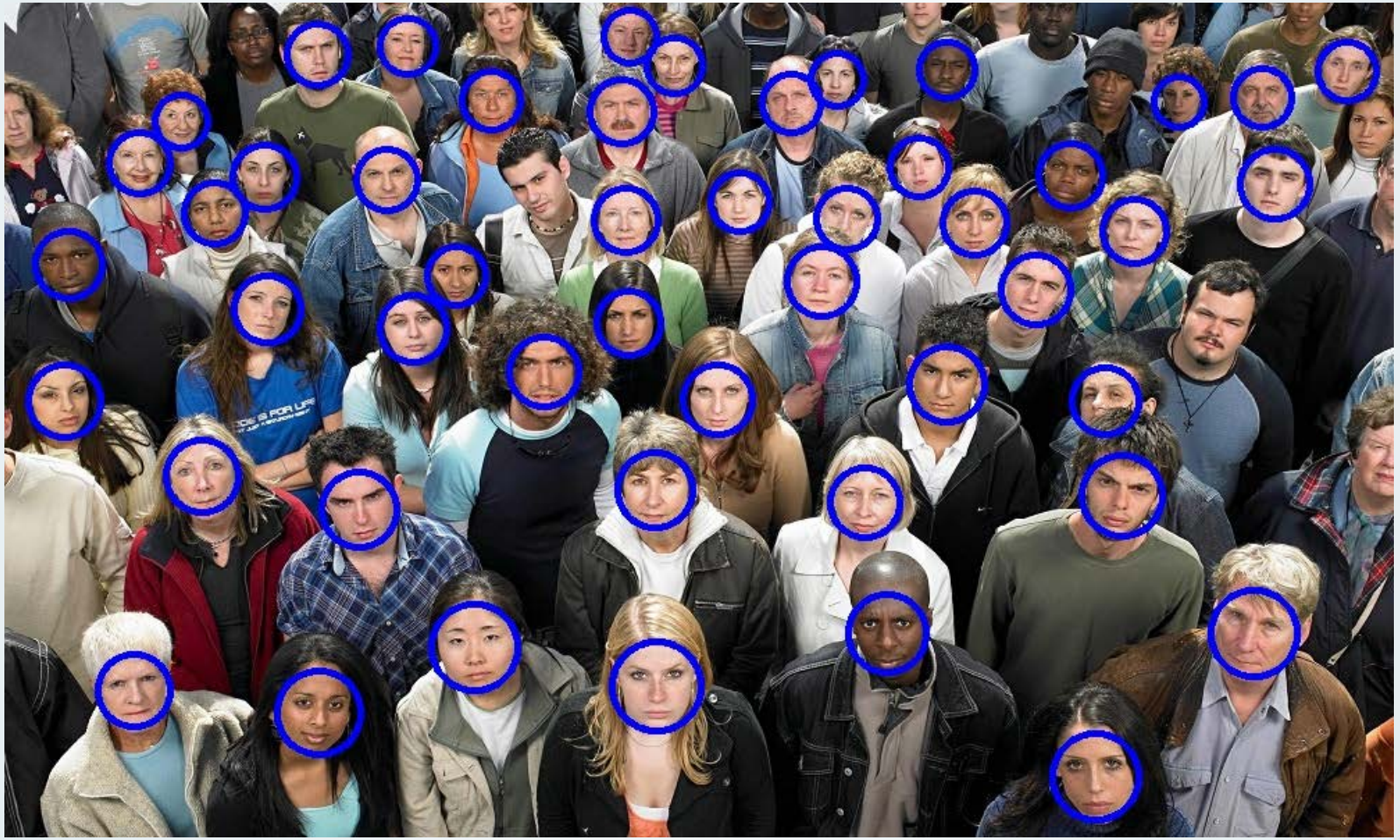
# Example

RGB

HSV

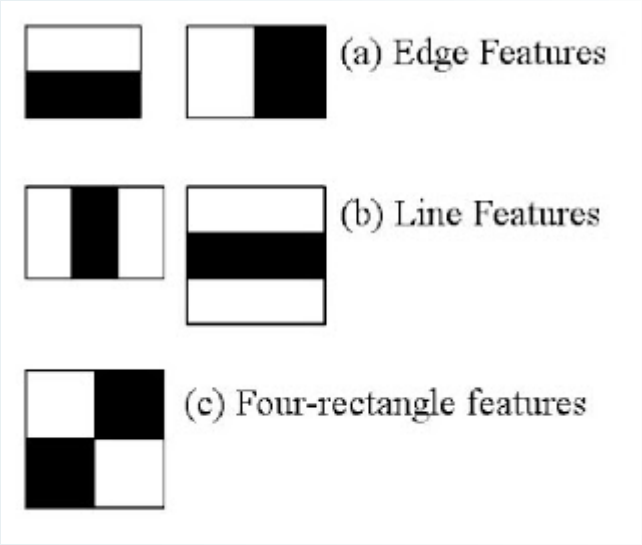
Gray

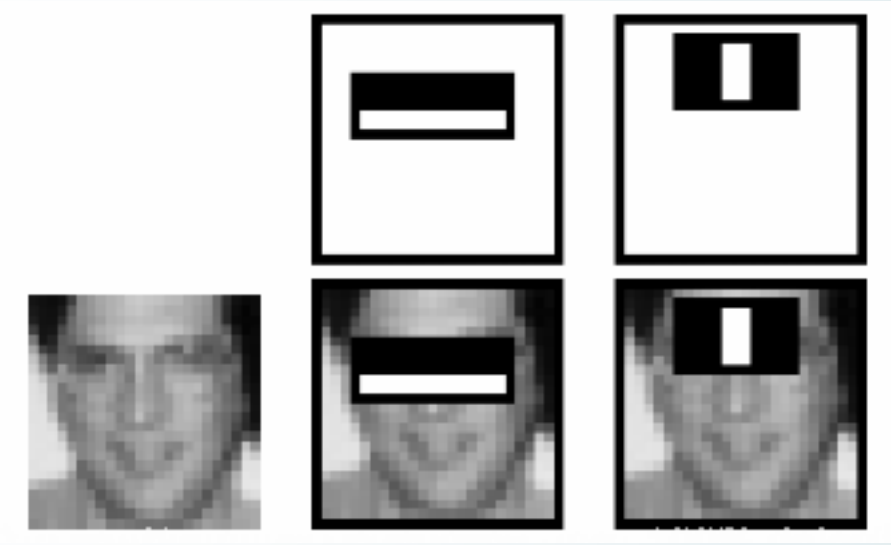
# Face Detection using OpenCV



# Viola–Jones Object Detection

- Haar Features



# Face Detection using OpenCV

- CascadeClassifier
  - CascadeClassifier face_cascade;
  - face_cascade.load(name of the pre-trained model);
    - to load a .xml classifier file
  - face_cascade.detectMultiScale(img, objects, ScaleFactor, minSize, maxSize);

# Face Detection using OpenCV

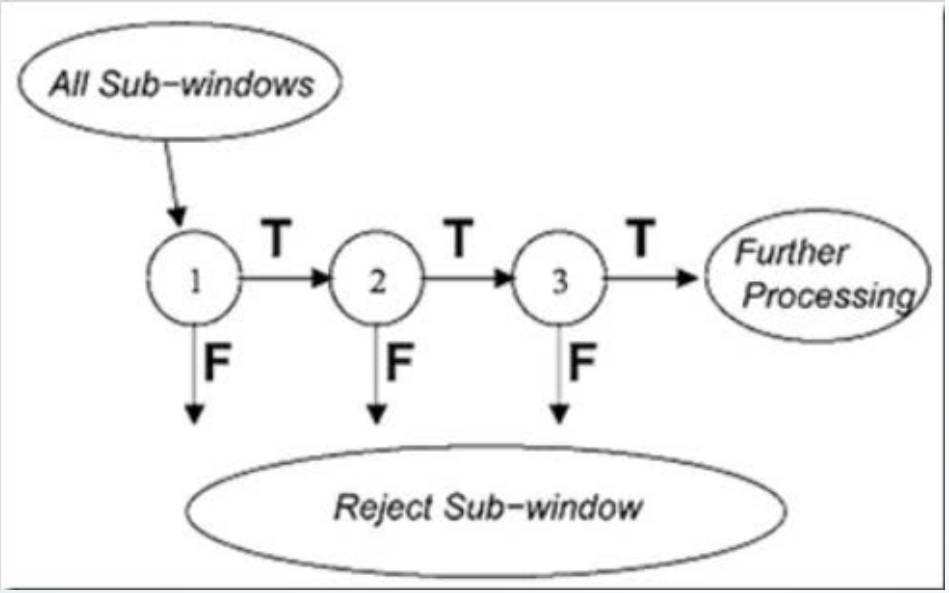
# Face Detection using OpenCV

# Face Detection using OpenCV

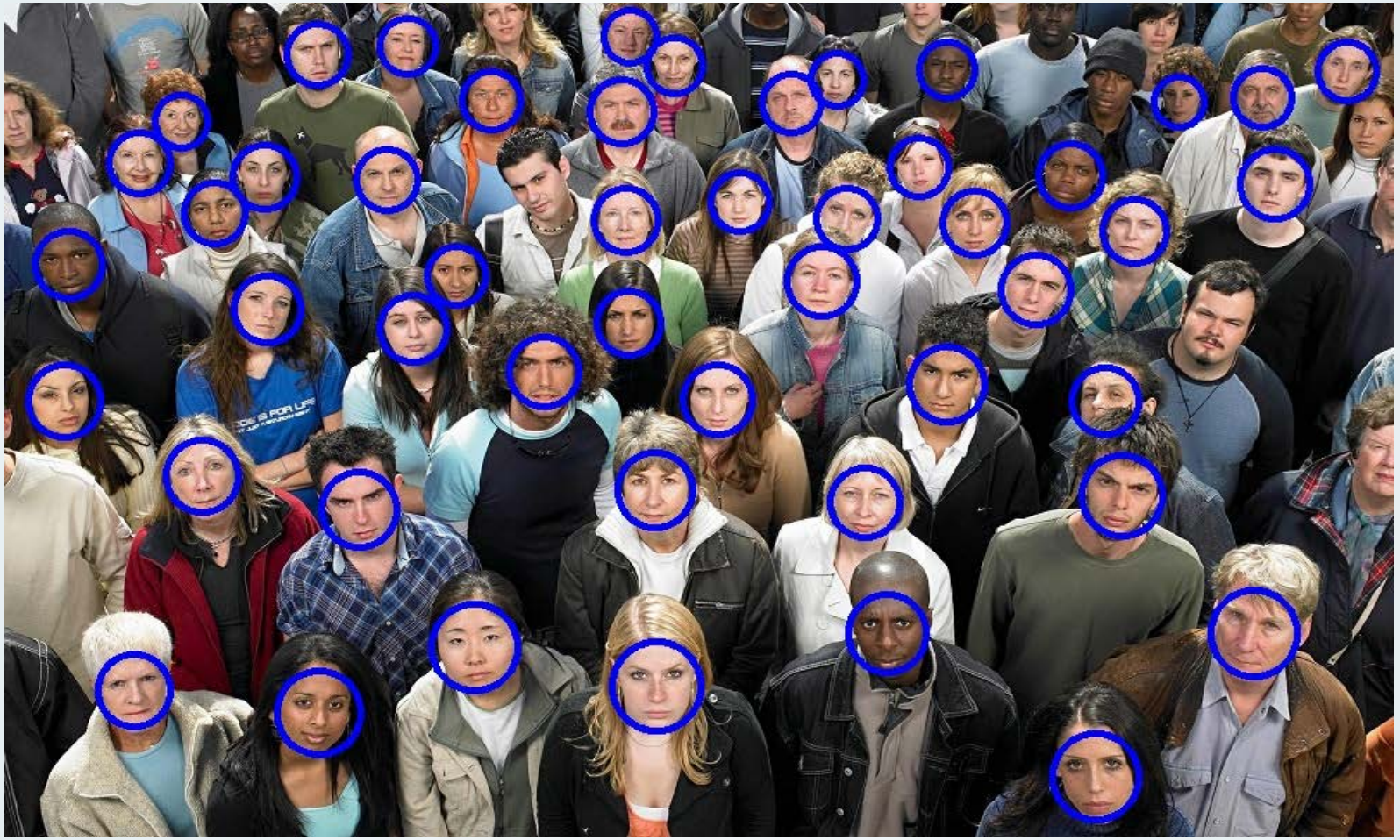
# Face Detection using OpenCV

# Face Detection using OpenCV

- CascadeClassifier



# Face Detection using OpenCV



# Other Modules & Functions

- Tutorials and examples of other modules and their functions can be found in OpenCV documentation, See:
  - https://docs.opencv.org/4.0.1/
  - https://docs.opencv.org/4.0.1/d9/df8/tutorial_root.html

# Thanks!

Please feel free to email as if you have questions?!

chihyun@mail.usf.edu