LAB 1

Computer Vision 2018

P. Zanuttigh

What is OpenCV ?

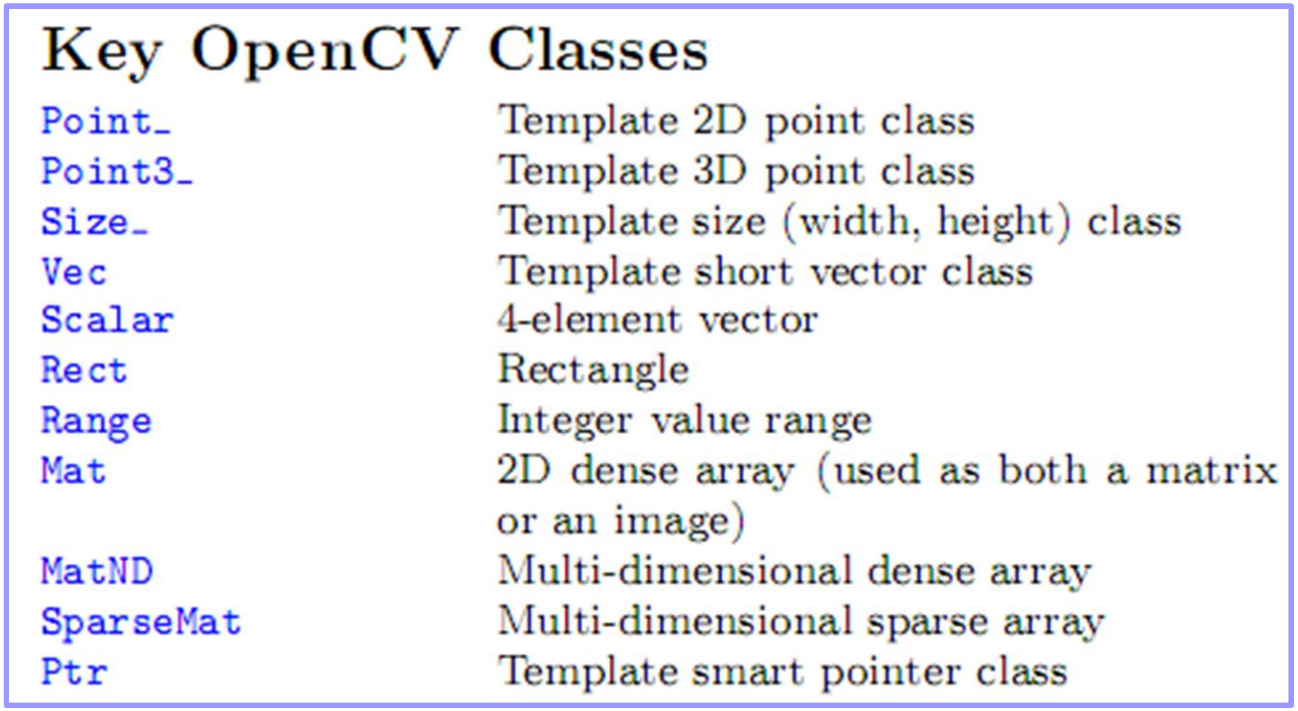
- OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision
- Originally developed by Intel, it was later supported by Willow Garage and is now maintained by Itseez
- The library is cross-platform and free for use under the open-source BSD

OpenCV Modules

- **core. Core functionality**
- **imgproc. Image processing**
- imgcodecs. Image file reading and writing
- videoio. Video I/O
- **highgui. High-level GUI**
- video. Video Analysis
- calib3d. Camera Calibration and 3D Reconstruction
- features2d. 2D Features Framework
- objdetect. Object Detection
- dnn. Deep Neural Network module
- ml. Machine Learning
- flann. Clustering and Search in Multi-Dimensional Spaces
- photo. Computational Photography
- stitching. Images stitching
- cudaarithm. Operations on Matrices
- cudabgsegm. Background Segmentation
- cudacodec. Video Encoding/Decoding
- cudafeatures2d. Feature Detection and Description
- cudafilters. Image Filtering
- cudaimgproc. Image Processing
- cudalegacy. Legacy support
- cudaobjdetect. Object Detection
- cudaoptflow. Optical Flow
- cudastereo. Stereo Correspondence
- cudawarping. Image Warping
- cudev. Device layer
- shape. Shape Distance and Matching
- superres. Super Resolution
- videostab. Video Stabilization
- viz. 3D Visualizer

Basic Structures and Classes

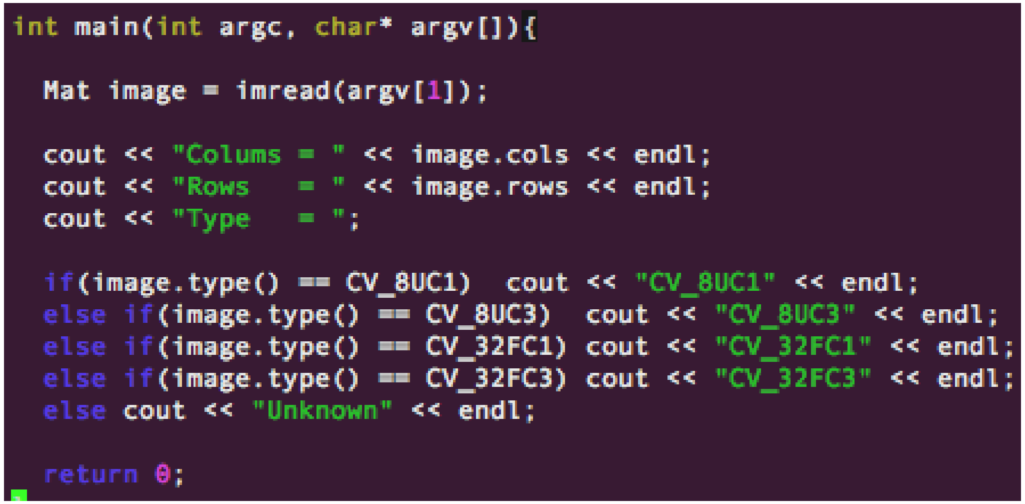
- **Mat** – matrix/image object
- **Point, Point2f** - 2D Point
- **Size** - 2D size structure
- **Rect** - 2D rectangle object
- **RotatedRect** - Rect object with angle



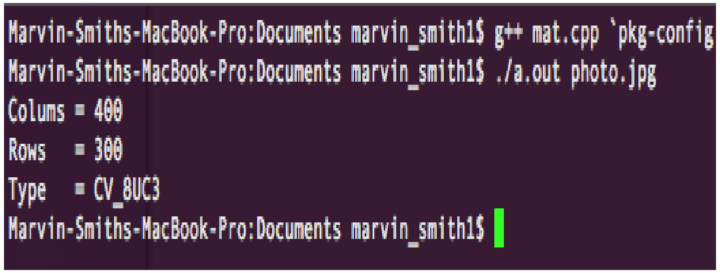
Key OpenCV Classes

Point_	Template 2D point class
Point3_	Template 3D point class
Size_	Template size (width, height) class
Vec	Template short vector class
Scalar	4-element vector
Rect	Rectangle
Range	Integer value range
Mat	2D dense array (used as both a matrix or an image)
MatND	Multi-dimensional dense array
SparseMat	Multi-dimensional sparse array
Ptr	Template smart pointer class

cv::Mat (1)



```
int main(int argc, char* argv[]){

  Mat image = imread(argv[1]);

  cout << "Colums = " << image.cols << endl;
  cout << "Rows   = " << image.rows << endl;
  cout << "Type   = ";

  if(image.type() == CV_8UC1)  cout << "CV_8UC1" << endl;
  else if(image.type() == CV_8UC3)  cout << "CV_8UC3" << endl;
  else if(image.type() == CV_32FC1) cout << "CV_32FC1" << endl;
  else if(image.type() == CV_32FC3) cout << "CV_32FC3" << endl;
  else cout << "Unknown" << endl;

  return 0;
```

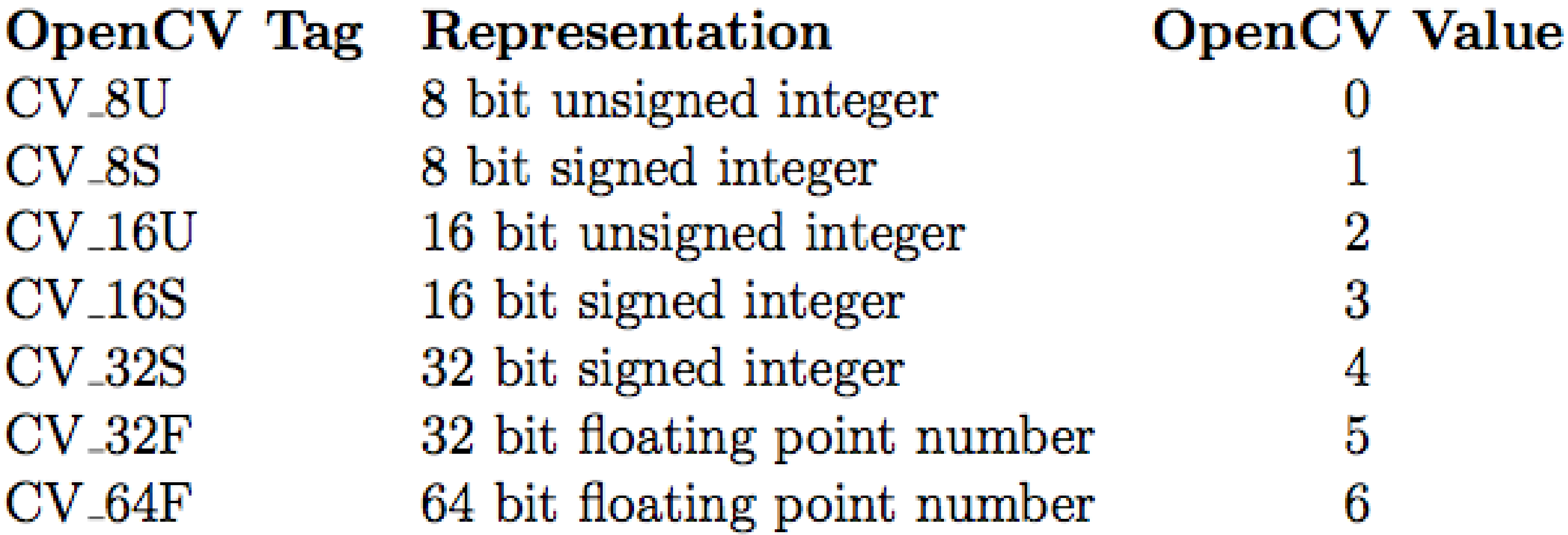


```
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$ g++ mat.cpp `pkg-config
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$ ./a.out photo.jpg
Colums = 400
Rows   = 300
Type   = CV_8UC3
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$
```

- The primary data structure in OpenCV is the Mat object
- It stores images and their components.
- Main items
  - rows, cols - length and width(int)
  - bit depth: 8, 16, 32, 64 bits per value
  - channels - 1: grayscale, 3: BGR, 4: BGR+Alpha
  - **U**nsigned, **S**igned or **F**loating points values
  - data type: CV_<bit depth><U/S/F>C<num channels>

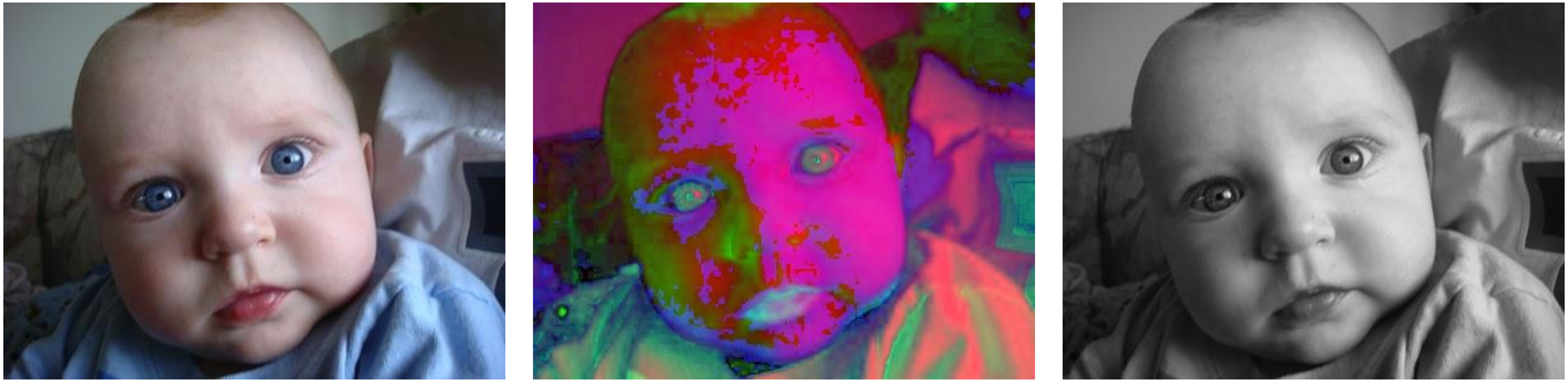
Image Types

- The TYPE is a very important aspect of OpenCV
- Represented as CV_<Datatype>C<# Channels>
- OpenCV uses templates!!
- Example Datatypes/ Depths



OpenCV Tag	Representation	OpenCV Value
CV_8U	8 bit unsigned integer	0
CV_8S	8 bit signed integer	1
CV_16U	16 bit unsigned integer	2
CV_16S	16 bit signed integer	3
CV_32S	32 bit signed integer	4
CV_32F	32 bit floating point number	5
CV_64F	64 bit floating point number	6

Pixel Types



- How the image is represented
  - BGR - The default color of imread(). Normal 3 channel color
    - Different ordering of the 3 components than standard RGB representation
  - GRAYSCALE - Gray values, Single channel

OpenCV requires that images be in BGR or Grayscale in order to be shown or saved. Otherwise, undesirable effects may appear.

cv::Mat Constructor

```
// basic constructor
cv::Mat(nrows, ncols, type [,fillValue])

// example (grayscale image 640x480  8-bit)
cv::Mat(480, 640, CV_8UC1)
// example (grayscale image 640x480  8-bit), init to white
cv::Mat(480, 640, CV_8UC1, 255)

// vectors 3 dim, yellow  color (BGR space)
Vec3b yellow(0, 255, 255);
// example (color image 640x480 BGR 8-bit), init to yellow
cv::Mat(480, 640, CV_8UC3, yellow)
```

Inspectors:

```
int Mat::channels() const // # of channels
int Mat::depth() const // element depth
int Mat::type() const  // type ID (ex. CV_8UC3)
```

cv::Mat Data Access

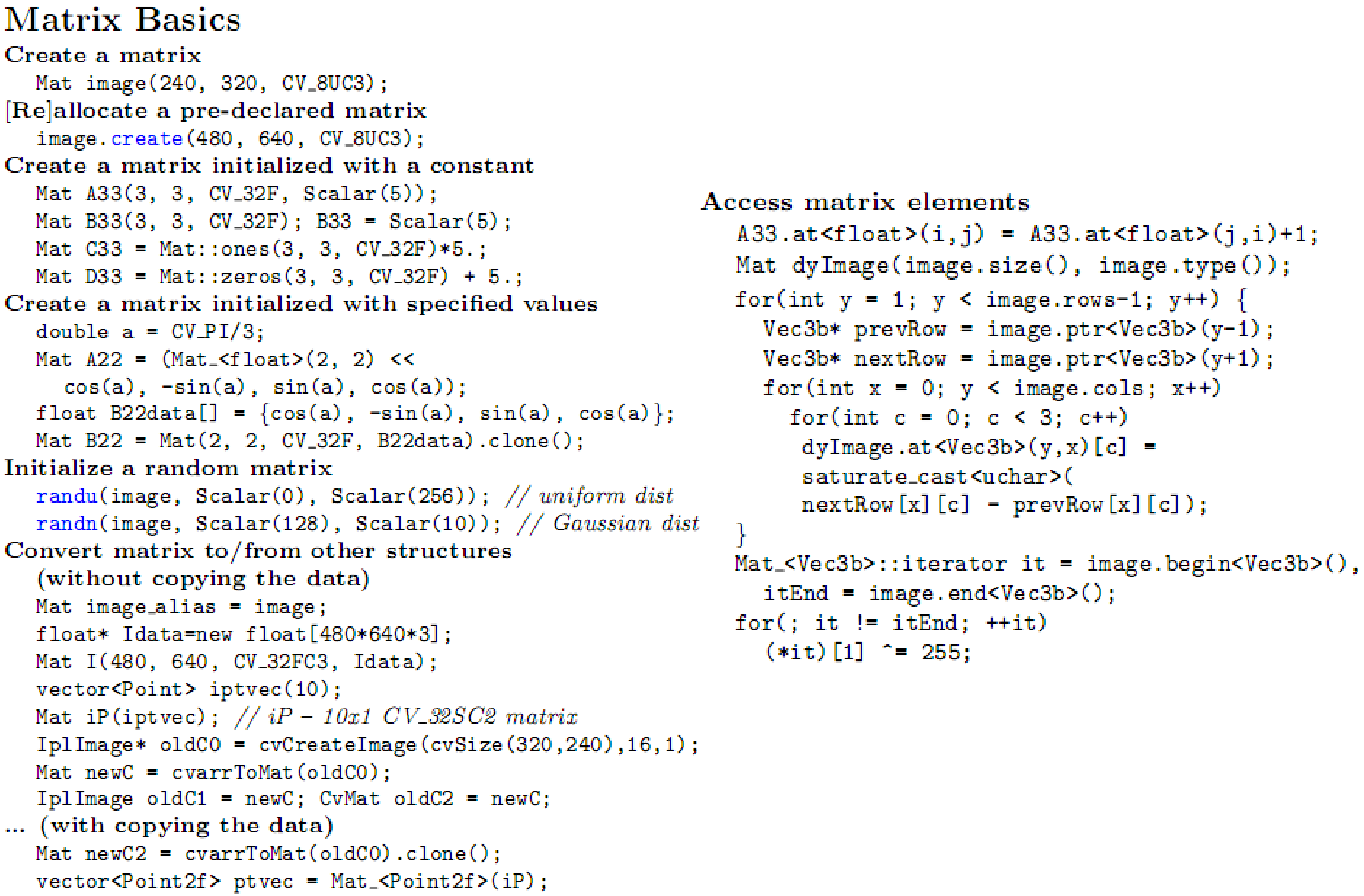
Two possibilities:

```
// 1) at function: template<typename T> T& Mat::at(int i, int j)
Mat.at<datatype>(row, col)[channel]  // returns reference to image location

//  example
M.at<unsigned char>(i, j) = 23; // set grayscale value
M.at<Vec3b>(i, j)[0] = 23; // set blue component in BGR image


// 2) with pointers
unsigned char* cur_row = M.ptr(i);
cur_row[j] = 23;
```

cv::Mat Functions

- Mat.**channels**() - returns the number of channels
- Mat.**clone**() - returns a deep copy of the image
- Mat.**create**(rows, cols, TYPE) - re-allocates new memory to matrix
- Mat.**cross**(<Mat>) - computes cross product of two matrices (need to be 3-elements vectors)
- Mat.**depth**() - returns data type of matrix
- Mat.**dot**(<Mat>) - computes the dot product of two matrices (vectors or read element by element)
- Mat(**Range**(row_min, row_max), **Range**(col_min,col_max)) - returns sub image
- Mat.**type**() - returns the TYPE of a matrix (e.g., CV_8UC3)
- Mat.**begin**() - moves Mat iterator to beginning of image/matrix
- Mat.**end**() - moves Mat iterator to end of image/matrix

Manipulate an Image (Mat)



Matrix Basics

Create a matrix

```
Mat image(240, 320, CV_8UC3);
```

[Re]allocate a pre-declared matrix

```
image.create(480, 640, CV_8UC3);
```

Create a matrix initialized with a constant

```
Mat A33(3, 3, CV_32F, Scalar(5));
Mat B33(3, 3, CV_32F); B33 = Scalar(5);
Mat C33 = Mat::ones(3, 3, CV_32F)*5.;
Mat D33 = Mat::zeros(3, 3, CV_32F) + 5.;
```

Create a matrix initialized with specified values

```
double a = CV_PI/3;
Mat A22 = (Mat_<float>(2, 2) <<
  cos(a), -sin(a), sin(a), cos(a));
float B22data[] = {cos(a), -sin(a), sin(a), cos(a)};
Mat B22 = Mat(2, 2, CV_32F, B22data).clone();
```

Initialize a random matrix

```
randu(image, Scalar(0), Scalar(256)); // uniform dist
randn(image, Scalar(128), Scalar(10)); // Gaussian dist
```

Convert matrix to/from other structures

(without copying the data)

```
Mat image_alias = image;
float* Idata=new float[480*640*3];
Mat I(480, 640, CV_32FC3, Idata);
vector<Point> iptvec(10);
Mat iP(iptvec); // iP – 10x1 CV_32SC2 matrix
IplImage* oldC0 = cvCreateImage(cvSize(320,240),16,1);
Mat newC = cvarrToMat(oldC0);
IplImage oldC1 = newC; CvMat oldC2 = newC;
```

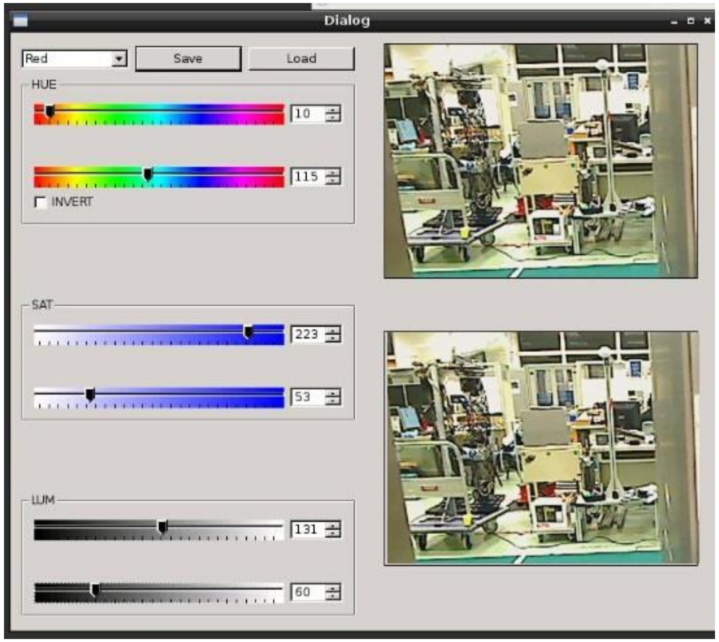
... (with copying the data)

```
Mat newC2 = cvarrToMat(oldC0).clone();
vector<Point2f> ptvec = Mat_<Point2f>(iP);
```

Access matrix elements

```
A33.at<float>(i,j) = A33.at<float>(j,i)+1;
Mat dyImage(image.size(), image.type());
for(int y = 1; y < image.rows-1; y++) {
  Vec3b* prevRow = image.ptr<Vec3b>(y-1);
  Vec3b* nextRow = image.ptr<Vec3b>(y+1);
  for(int x = 0; y < image.cols; x++)
    for(int c = 0; c < 3; c++)
     dyImage.at<Vec3b>(y,x)[c] =
     saturate_cast<uchar>(
     nextRow[x][c] - prevRow[x][c]);
}
Mat_<Vec3b>::iterator it = image.begin<Vec3b>(),
  itEnd = image.end<Vec3b>();
for(; it != itEnd; ++it)
  (*it)[1] ^= 255;
```

HighGUI Module



HUE 10
HUE 115
SAT 223
SAT 53
LUM 131
LUM 60

- Image I/O, rendering
- Processing keyboard and other events, timeouts
- Trackbars
- Mouse callbacks
- Video I/O

HighGUI: OpenCV Functions

- void cv::namedWindow(const string& winname, int flags=WINDOW_AUTOSIZE);
  - Creates window accessed by its name. Window handles repaint, resize events
- void cv::destroyWindow(const string& winname);
- void cv::imshow(const string& winname, cv::Mat& mat);
  - Copies the image to window buffer, then repaints it when necessary. {8u|16s|32s|32f}{C1|3|4} are supported.
  - Only the whole window contents can be modified. Dynamic updates of parts of the window are done using operations on images, drawing functions etc.

HighGUI: Read and Save

- Mat imread(const string& filename, int flags=1);
  - loads image from file, converts to color or grayscale, if need, and returns it (or returns empty cv::Mat())
  - image format is determined by the file contents
- bool imwrite(const string& filename, Mat& image);
  - saves image to file, image format is determined from extension

- Example: convert JPEG to PNG
  - cv::Mat img = cv::imread(“picture.jpeg”);
  - if(!img.empty()) cv::imwrite(“picture.png”, img);

Image I/O Example

- OpenCV provides simple and useful ways to read and write images
- Note that there are many extra options to these commands which are available on the documentation
- waitKey(int x) has two main features
  - if x > 0, then waitKey will wait x milliseconds
  - if x = 0, then waitKey will not move until key is pressed

- **Examples**

```
//Read an image
Mat image = imread( <string>, <0 -gray, 1 -BGR>)
     //Note 1 is default

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

//Create window for output
namedWindow( <window name> );

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

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

Mouse Callback

```
// Set the callback function for any mouse event
// The function MouseFunc will be called when some mouse event happens
// You can pass data to the function (e.g., the image), use cast to recover the data
setMouseCallback("My Window", MouseFunc, void *userdata);

// This function is automatically called when a mouse event happens
// x,y: coordinates of mouse position, event: type of event, flags: get buttons status
void MouseFunc(int event, int x, int y, int flags, void* userdata)
{
        if  ( event == EVENT_LBUTTONDOWN )
        {
                cout << "Left button clicked - position (" << x << ", " << y << ")" << endl;
        }
}
```

HighGUI Hello World

- **Example code: load an image from disk and display it on the screen**

```
#include "opencv2/opencv.hpp"
int main( int argc, char* argv[] ) {
    cv::Mat image = cv::imread( argv[1] );
    cv::namedWindow( “Example1”, CV_WINDOW_AUTOSIZE );
    cv::imshow( “Example1”, image );
    cv::waitKey(0);
    cv::destroyWindow( “Example1” );
    return 0;
}
```

Assignment

Goal: Change the soccer shirt color of the players in the image

Write a program that:

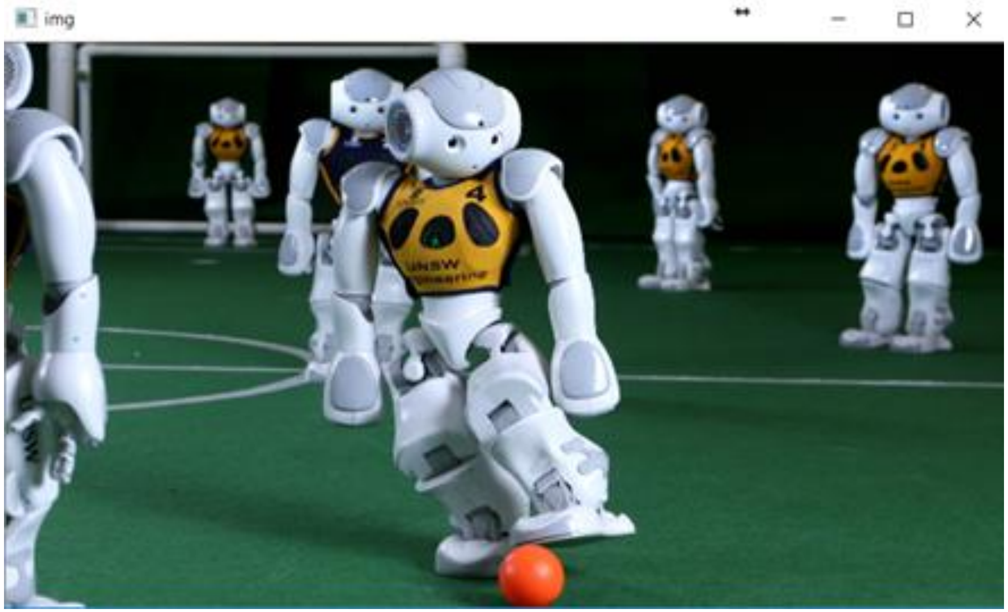
Loads the image stored inside the data folder (you can use the “robocup.jpg” or the “roma.jpg” images)

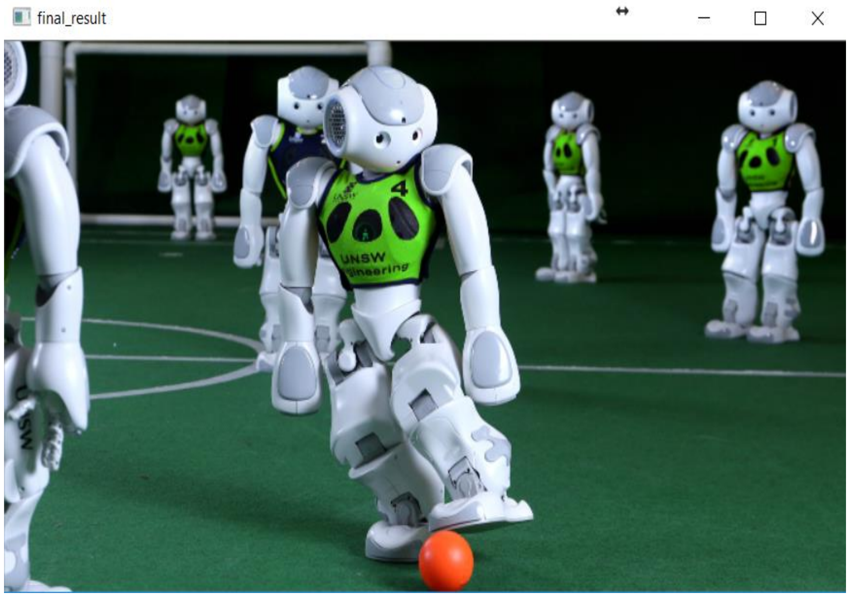
1. Shows the image on a window
2. Captures the left click of the mouse and computes the mean RGB color over a 9x9 neighborhood of the clicked point
3. Segment the soccer shirts by applying a static threshold to the three channels R, G and B (e.g., ΔR < 50, ΔG < 50, ΔB < 50, but try to change the value)
4. Apply a new color to the selected regions (let’s use BGR = (37,201,92))

Write a program that:

- Does the same as before, but uses the HSV space. (SUGGESTION: you should apply the threshold only on the H channel, once you segmented the shirts, you can change only the H component, for example to 45 to have green shirts)
- For color space conversion use *cv::cvtColor(image, image_hsv, CV_BGR2HSV);*

Example of the Results





Compile & Link

- **Compilation** refers to the processing of source code files (.cpp) and the creation of an 'object' file. The compiler produces the machine language instructions that correspond to the source code
  - If you compile (but don't link) three separate files, you will have three object files created
  - You can't run them yet!

- **Linking** refers to the creation of a single executable file from multiple object files
  - The linker may look at multiple files and try to find references for the used functions

Setup a new project in Visual Studio (Windows)

- Create a new project
- Set "console application" for the type of project
- Remove automatically created source file and the stdafx.cpp and stdafx.h files from the project
- Add your source cpp (you can use the provided template source)
- Set as platform type "*Release x64*" (or "*Debug x64*" if you need to debug)
- Set the project options (see next slide)
- Compile & Run !

Project Options (Visual Studio)

- Set working directory (by default the one where the project file is)
  - dll and image or other data are searched in this folder
- Compilation: additional include directories: add opencv
  - Add \\nas2\datilab\opencv-3.4.1\build\include
- Compilation: Precompiled headers : set to "not using"
- Linker: input/additional dependencies: add opencv
  - add \\nas2\datilab\opencv-3.4.1\build\x64\vc15\lib\opencv_world341.lib
  - or for debug \\nas2\datilab\opencv-3.4.1\build\x64\vc15\lib\opencv_world341**d**.lib
- Copy opencv dlls and images to your working directory
  - copy \\nas2\datilab\opencv-3.4.1\build\x64\vc15\bin\opencv_world341.dll in your working directory
  - for debug \\nas2\datilab\opencv-3.4.1\build\x64\vc15\bin\opencv_world341d.dll

- Compile and run!

Compile and Run in Linux

- Edit the source with a text editor
- Compile with g++
  - *g++ -o test.o -I/nfsd/opt/opencv-3.3.1/include/ -L/nfsd/opt/opencv-3.3.1/lib/ -lopencv_core -lopencv_highgui -lopencv_imgcodecs -lopencv_imgproc lab1_rgb.cpp*
  - need to specify each single opencv module
  - path to include and libs
- Run
  - *LD_LIBRARY_PATH=/nfsd/opt/opencv-3.3.1/lib/ ./test.o*
  - Need to specify library path (corresponds to dll in Windows)