Lesson 2:
Pixel Values and Histograms

Jorge Beltrán, Arturo de la Escalera

Perception Systems

Course 2019-2020

Lesson 2: Pixel Values and Histograms

- OpenCV classes (C++):
  - The information is handled via classes
  - Classes definitions are found in OpenCV core
  - OpenCV classes ease the work with pixel, images, videos, etc.

The OpenCV C++ reference manual is here:
http://docs.opencv.org. Use ***Quick Search*** *to find descriptions of the particular functions and classes*

Key OpenCV Classes

Point_	Template 2D point class
Point3_	Template 3D point class
Size_	Template size (width, height) class
Vec	Template short vector class
Matx	Template small matrix class
Scalar	4-element vector
Rect	Rectangle
Range	Integer value range
Mat	2D or multi-dimensional dense array (can be used to store matrices, images, histograms, feature descriptors, voxel volumes etc.)
SparseMat	Multi-dimensional sparse array
Ptr	Template smart pointer class

Lesson 2: Pixel Values and Histograms

- Mat Class

OpenCV C++ n-dimensional dense array class

```
class CV_EXPORTS Mat
{
public:
    // ... a lot of methods ...
    ...

    /*! includes several bit-fields:
         - the magic signature
         - continuity flag
         - depth
         - number of channels
     */
    int flags;
    //! the array dimensionality, >= 2
    int dims;
    //! the number of rows and columns or (-1, -1) when the array has more than 2 dimensions
    int rows, cols;
    //! pointer to the data
    uchar* data;

    //! pointer to the reference counter;
    // when array points to user-allocated data, the pointer is NULL
    int* refcount;

    // other members
    ...
};
```

The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images.

Lesson 2: Pixel Values and Histograms

- Mat Class Constructors

Mat ()
Mat (int **rows**, int **cols**, int **type**)
Mat (**Size size**, int **type**)
Mat (int **rows**, int **cols**, int **type**, const **Scalar** &s)
Mat (**Size size**, int **type**, const **Scalar** &s)
Mat (int ndims, const int *sizes, int **type**)
Mat (int ndims, const int *sizes, int **type**, const **Scalar** &s)
Mat (const **Mat** &m)
Mat (int **rows**, int **cols**, int **type**, void ***data**, **size_t step=AUTO_STEP**)
Mat (**Size size**, int **type**, void ***data**, **size_t step=AUTO_STEP**)
Mat (int ndims, const int *sizes, int **type**, void ***data**, const **size_t** *steps=0)
Mat (const **Mat** &m, const **Range** &**rowRange**, const **Range** &**colRange=Range::all**())
Mat (const **Mat** &m, const **Rect** &roi)
Mat (const **Mat** &m, const **Range** *ranges)

Lesson 2: Pixel Values and Histograms

- Useful Mat constructors and methods

  - `Mat::~Mat()` → Destructor
  - `Mat::row` → Creates a matrix header for the specified row
  - `Mat::col` → Creates a matrix header for the specified column
  - `Mat::clone` → Creates a full copy of the matrix
  - `Mat::copyTo` → Copies the matrix to another one
  - `Mat::size` → Returns a matrix size
  - `Mat::empty` → Returns `true` if the array has no elements

  … and many more

Lesson 2: Pixel Values and Histograms

- Point *class* **Point_**

```
template<typename _Tp> class CV_EXPORTS Point_
{
public:
    typedef _Tp value_type;

    // various constructors
    Point_();
    Point_(_Tp _x, _Tp _y);
    Point_(const Point_& pt);
    Point_(const CvPoint& pt);
    Point_(const CvPoint2D32f& pt);
    Point_(const Size_<_Tp>& sz);
    Point_(const Vec<_Tp, 2>& v);

    Point_& operator = (const Point_& pt);
    //! conversion to another data type
    template<typename _Tp2> operator Point_<_Tp2>() const;

    //! conversion to the old-style C structures
    operator CvPoint() const;
    operator CvPoint2D32f() const;
    operator Vec<_Tp, 2>() const;

    //! dot product
    _Tp dot(const Point_& pt) const;
    //! dot product computed in double-precision arithmetics
    double ddot(const Point_& pt) const;
    //! cross-product
    double cross(const Point_& pt) const;
    //! checks whether the point is inside the specified rectangle
    bool inside(const Rect_<_Tp>& r) const;

    _Tp x, y; //< the point coordinates
};
```

Template class for 2D points specified by its coordinates x and y . An instance of the class is interchangeable with C structures, CvPoint and CvPoint2D32f . There is also a cast operator to convert point coordinates to the specified type. The conversion from floating-point coordinates to integer coordinates is done by rounding.

```
pt1 = pt2 + pt3;
pt1 = pt2 - pt3;
pt1 = pt2 * a;
pt1 = a * pt2;
pt1 += pt2;
pt1 -= pt2;
pt1 *= a;
double value = norm(pt); // L2 norm
pt1 == pt2;
pt1 != pt2;
```

For your convenience, the following type aliases are defined:

```
typedef Point_<int> Point2i;
typedef Point2i Point;
typedef Point_<float> Point2f;
typedef Point_<double> Point2d;
```

Example:

```
Point2f a(0.3f, 0.f), b(0.f, 0.4f);
Point pt = (a + b)*10.f;
cout << pt.x << ", " << pt.y << endl;
```

```
Point pt;
pt.x = 10;
pt.y = 8;

Point pt = Point(10, 8);
```

Lesson 2: Pixel Values and Histograms

- Drawings
  - Circle:

```
void circle(
        InputOutputArray img,
        Point center,
        int radius,
        const Scalar& color,
        int thickness=1,
        intlineType=LINE_8,
        int shift=0 )
```

Scalar(blue, green, red);

-1 for filled circle

Optional

```
Point center(img.rows / 2, img.cols / 2);
int radius = 25;
circle (img, center, radius, Scalar(0, 255, 0));
```

Lesson 2: Pixel Values and Histograms

- Drawings
  - Line:

```
void line(
        InputOutputArray img,
        Point pt1,
        Point pt2,
        const Scalar& color,
        int thickness = 1,
        int lineType = LINE_8,
        int shift=0 )
```

Scalar(blue, green, red);

Optional

```
Point start(img.rows / 2, img.cols / 2);
Point end (0, 0);
line (img, start, end, Scalar(0, 255, 0));
```

Lesson 2: Pixel Values and Histograms

- Drawings
  - Rectangle:

```
void rectangle(
        InputOutputArray img,
        Point pt1,
        Point pt2,
        const Scalar& color,
        int thickness = 1,
        int lineType = LINE_8,
        int shift=0 )
```

Scalar(blue, green, red);

-1 for filled rectangle

Optional

```
Point start(img.rows / 2, img.cols / 2);
Point end (img.rows , img.cols );
rectangle (img, start, end, Scalar(0, 255, 0));
```

Lesson 2: Pixel Values and Histograms

Example 1. Drawings

- Open an image
- Draw a red line
- Draw a filled yellow circle
- Draw a blue empty rectangle
- Show the final image
- Free memory

Lesson 2: Pixel Values and Histograms

- Example 1: Drawings

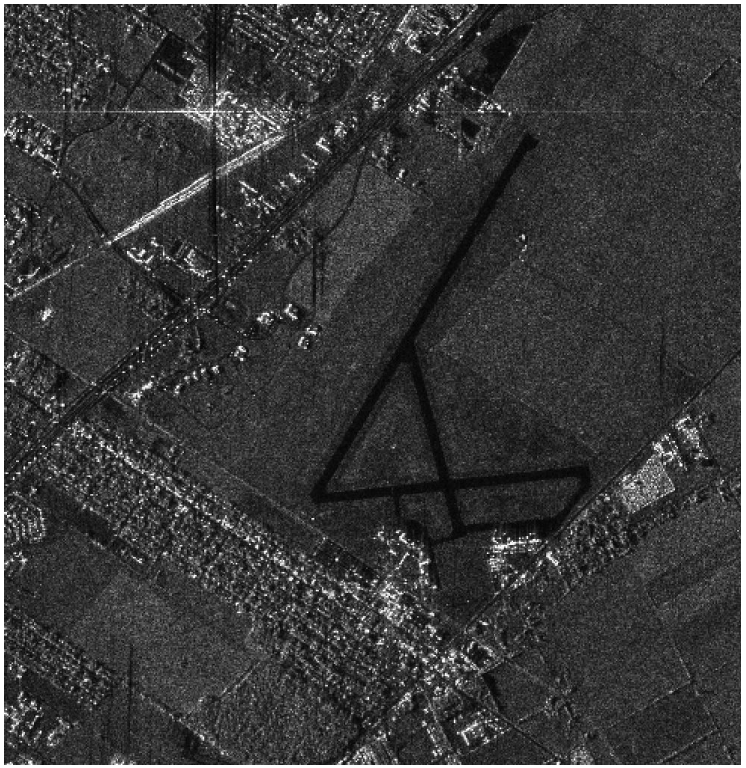
```cpp
#include "opencv\cv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
    // initialize object
    Mat original_image, modified_image;

    // load image from disk
    original_image = imread("lena.jpg");

    // check if the image is available
    if (!original_image.data)
    {
        cout << "Error in loading the image!" << endl;
    }
    else
    {
        // copy the image matrix to the modified image
        // method 1 clone()
        // modified_image = original_image.clone();

        // method 2 copyTo()
        original_image.copyTo(modified_image);

        // define points
        Point p1(modified_image.rows / 2, modified_image.cols / 2);
        Point p2((modified_image.rows / 2) + 150, (modified_image.cols / 2) + 100);

        // draw a line
        line(modified_image, p1, p2, Scalar(0, 0, 255), 3);

        // draw a circle
        Point p3(modified_image.rows / 4, modified_image.cols / 2);
        circle(modified_image, p3, 25, Scalar(32, 164, 255), -1);

        // draw a rectangle
        Point p4(3 * modified_image.rows / 4, modified_image.cols / 2);
        Point p5(3 * modified_image.rows / 4 + 50, modified_image.cols / 2 - 100);

        rectangle(modified_image, p4, p5, Scalar(255, 255, 0), 2);

        // create window canvas to show image
        namedWindow("L02_E01_Original", CV_WINDOW_AUTOSIZE);
        namedWindow("L02_E01_Modified", CV_WINDOW_AUTOSIZE);

        // add the image to the window
        imshow("L02_E01_Original", original_image);
        imshow("L02_E01_Modified", modified_image);

        // wait till a key is pressed
        waitKey(0);

        // free memory
        destroyWindow("L02_E01_Original");
        destroyWindow("L02_E01_Modified");
    }
}
```

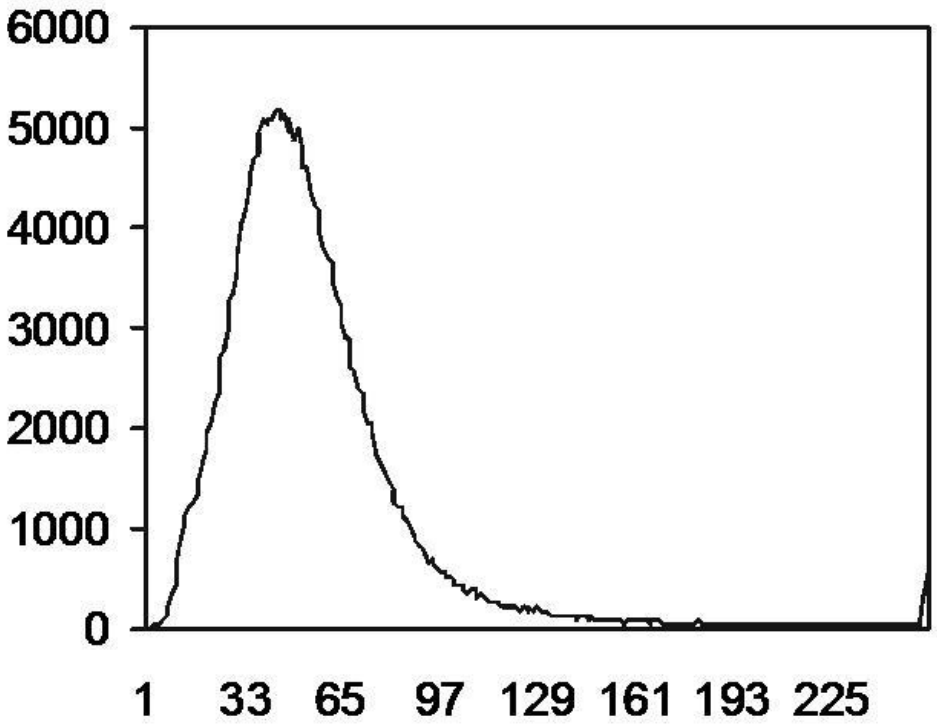
Lesson 2: Pixel Values and Histograms

- Example 1: Drawings

Lesson 2: Pixel Values and Histograms

- Histogram of the image
  - It is a graphical representation of the tonal distribution in a digital image, in other words it plots the number of pixels for each tonal value





Lesson 2: Pixel Values and Histograms

Example 2. Compute and plot the histogram of an image

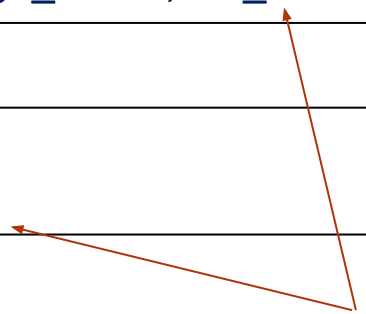
- Load image in grayscale
- Manually compute the histogram
- Create an empty image and draw the histogram
- Show the original image and the corresponding histogram
- Free memory

Lesson 2: Pixel Values and Histograms

Example 2. Compute and plot the histogram of an image

```
// Create a Mat called image_histogram of size 512x400, one channel (8UC1) and black (0)
int image_width = 512;
int image_height = 400;
Mat image_histogram(image_height, image_width, CV_8UC1, Scalar(0));
```

```
// Access pixel i of the loaded image
uchar bin_value = image.at<uchar>(i);
```



One channel [0, 255]

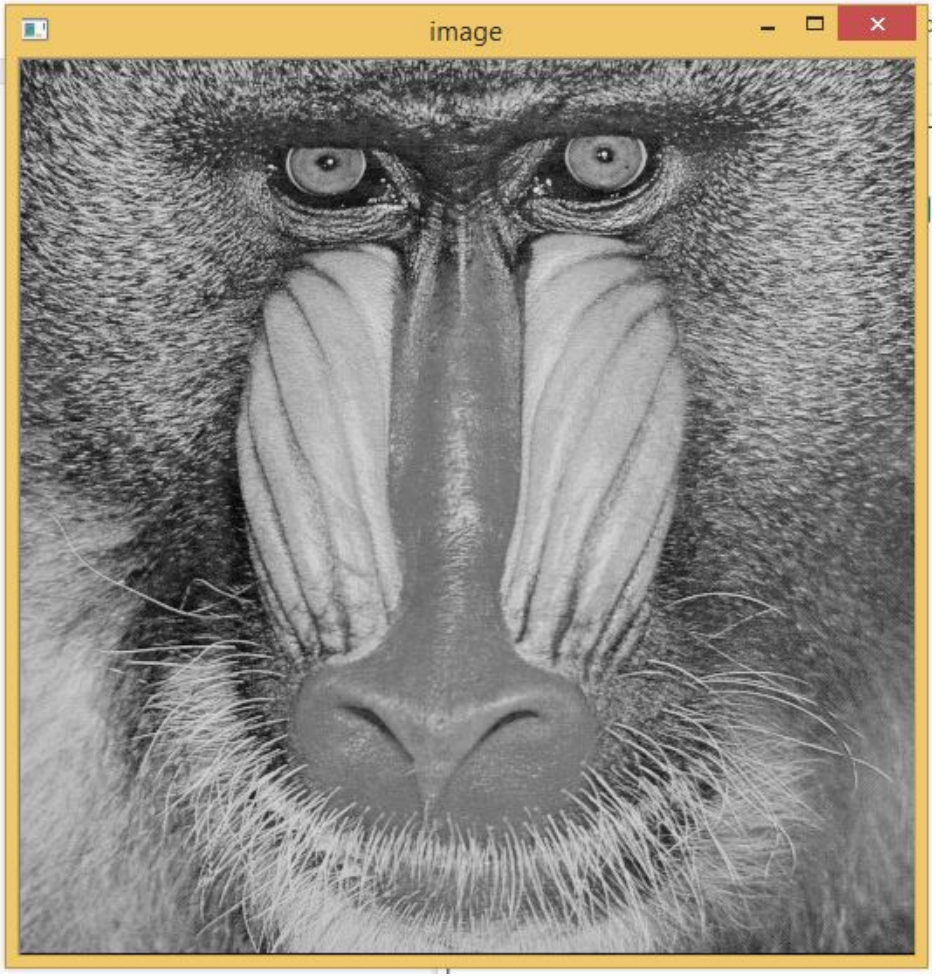
Lesson 2: Pixel Values and Histograms

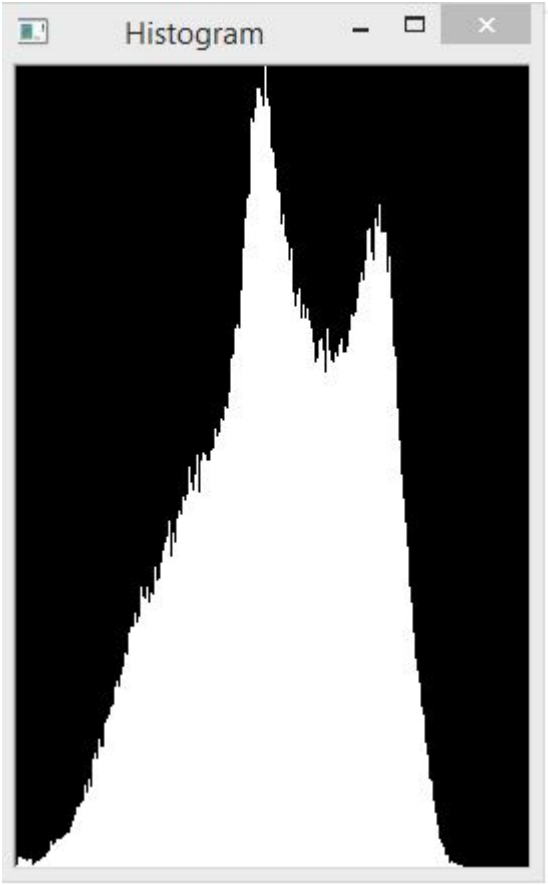
- Example 2

```
#include <opencv/cv.hpp>
#include <iostream>
#include <vector>
using namespace cv;
using namespace std;

int main(){
    Mat a = imread("mandril.jpg", CV_LOAD_IMAGE_GRAYSCALE);

    if (!a.data){
        std::cout << "Error" << std::endl;
        return 1;
    }

    int hist_size = 256;
    vector<int> bins (hist_size, 0);
    for (size_t row = 0; row < a.rows; row++){
        for (size_t col = 0; col < a.cols; col++){
            uchar graylevel = a.at<uchar>(row, col);
            bins[graylevel]++;
        }
    }
    // Normalization
    int max = *std::max_element(bins.begin(), bins.end());
    for (int graylevel = 0; graylevel<hist_size; ++graylevel){
        bins[graylevel] = bins[graylevel] * 400 / max;
        cout << graylevel << ": " << bins[graylevel] << endl;
    }
    Mat histImage(400, 256, CV_8UC1, Scalar(0));

    for (int graylevel = 0; graylevel<hist_size; ++graylevel){
        line(histImage, Point(graylevel, 400), Point(graylevel, 400 - bins[graylevel]), Scalar(255), 1, 8, 0);
    }

    imshow("image", a);
    imshow("Histogram", histImage);

    waitKey(0);

    return 0;
}
```

Lesson 2: Pixel Values and Histograms

- Example 2





Lesson 2: Pixel Values and Histograms

```
void calcHist(
        const Mat* images,
        int nimages,
        const int* channels,
        InputArray mask,
        OutputArray hist,
        int dims,
        const int* histSize,
        const float** ranges,
        bool uniform=true,
        bool accumulate=false )
```

→ image (or images) to be processed

→ number of images

→ list of channels to be used

→ Optional. Use Mat() if there is no mask

→ output Mat. It must be declared beforehand

→ number of channels to consider

→ number of bins (size) per dimension

→ array of arrays, containing min and max values per dimension. Lower bound is included and upper bound is not.

→ Opcional, true by default

→ Opcional, false by default

Lesson 2: Pixel Values and Histograms

Example 3. Compute and plot the histogram of an image

- Load image in grayscale
- Configure histogram for each channel
- Compute the histogram
- Create an empty image and draw the histogram
- Show the original image and the corresponding histogram
- Free memory

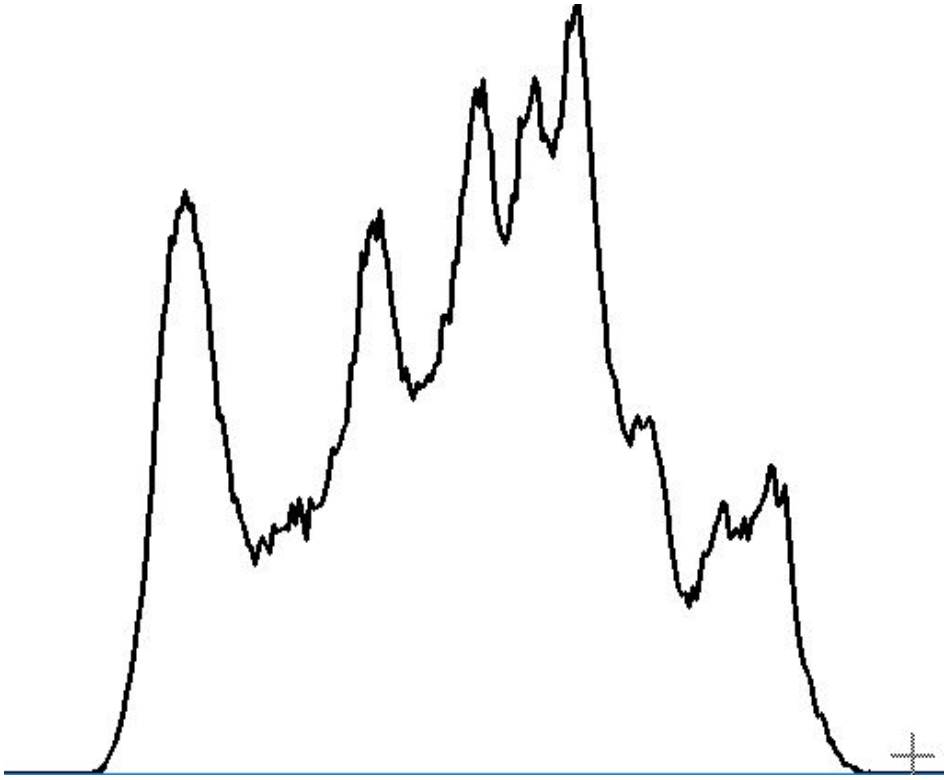
Lesson 2: Pixel Values and Histograms

```
2   #include "opencv\cv.hpp"
3   #include <iostream>
4
5   using namespace cv;
6   using namespace std;
7
8   int main(int argc, char* argv[])
9   {
10      // initialize object
11      Mat original_image;
12
13      // load image from disk
14      original_image = imread("lena.jpg", IMREAD_GRAYSCALE);
15
16      // check if the image is available
17      if (!original_image.data)
18      {
19          cout << "Error in loading the image!" << endl;
20      }
21      else
22      {
23          // initialize histogram calculating parameters
24          int histogram_size = 256;
25          float histogram_range[] = { 0, 256 };
26          const float* histogram_ranges[] = { histogram_range };
27          Mat histogram;
28
29          // calculate image histogram
30          calcHist(&original_image, 1, 0, Mat(), histogram, 1, &histogram_size, histogram_ranges);
31
32          // print calculated histogram in the command window
33          for (int i = 0; i < histogram_size; i++)
34          {
35              float bin_value = histogram.at<float>(i);
36              //cout << " " << bin_value;
37          }
```

Lesson 2: Pixel Values and Histograms

```
39          // initialize histogram plotting parameters
40          int bin_width = 2;
41          int histogram_width = 512;
42          int histogram_height = 400;
43          Mat normalized_histogram;
44
45          // empty image for the histogram plot
46          Mat image_histogram(histogram_height, histogram_width, CV_8UC1, Scalar(255, 0, 0));
47          // normalize histogram to fit the window
48          normalize(histogram, normalized_histogram, 0, histogram_height, NORM_MINMAX, -1, Mat());
49
50          for (int i = 1; i < histogram_size; i++)
51          {
52              Point p1(bin_width*(i - 1), histogram_height - cvRound(normalized_histogram.at<float>(i - 1)));
53              Point p2(bin_width*(i), histogram_height - cvRound(normalized_histogram.at<float>(i)));
54              line(image_histogram, p1, p2, Scalar(0, 0, 0), 2);
55          }
56
57          // create window canvas to show image
58          namedWindow("L02_E02_Original", CV_WINDOW_AUTOSIZE);
59          namedWindow("L02_E02_Histogram", CV_WINDOW_AUTOSIZE);
60
61          // add the image to the window
62          imshow("L02_E02_Original", original_image);
63          imshow("L02_E02_Histogram", image_histogram);
64
65          // wait till a key is pressed
66          waitKey(0);
67
68          // free memory
69          destroyAllWindows();
70      }
71  }
```

Lesson 2: Pixel Values and Histograms



Lesson 2:
Pixel Values and Histograms

Jorge Beltrán, Arturo de la Escalera

Perception Systems

Course 2019-2020