ITMO University

Faculty of Control Systems and Robotics

Report for laboratory work №1

course "Digital Image Processing"

Histograms, Profiles, Projections

Done by:

Ha The Long Vuong
Group: R42334c

Supervised by:

Shavetov Sergey Vasilievich

Saint Petersburg
2021

Contents

List of Figures

Listings

1 Purpose

Digital images basic brightness and geometric characteristics studying and using it for image analysis.

2 Assignment

2.1 Image Histogram

2.1.1 Create Histogram

The histogram element Hist[i] is the image pixels sum with the brightness i. Using the visual form of the histogram we can assess the need to change the image brightness and contrast, estimate the area occupied by light and dark elements, determine the location on the image plane of individual objects corresponding to brightness certain ranges. For a RGB color image it is need to create three histograms for each color.

OpenCV code

Listing 1: Image histogram creation using OpenCV and C++ programming language

```
void lab1::histogram(cv::Mat& orig_img, std::vector<cv::Mat>& vHist,
                     std::vector<cv::Mat>& vCumHist) {
  cv::Mat img = orig_img.clone();

  std::vector<cv::Mat> bgr_planes;
  split(orig_img, bgr_planes);

  float range[] = {0, 256};  // the upper boundary is exclusive
  const float* histRange[] = {range};

  cv::Mat b_hist, g_hist, r_hist;
  cv::Mat b_hist_norm, g_hist_norm, r_hist_norm;
  cv::calcHist(&bgr_planes[0], 1, nullptr, cv::Mat(), b_hist, 1, &histSize,
               histRange);
  cv::calcHist(&bgr_planes[1], 1, nullptr, cv::Mat(), g_hist, 1, &histSize,
               histRange);
  cv::calcHist(&bgr_planes[2], 1, nullptr, cv::Mat(), r_hist, 1, &histSize,
               histRange);

  vHist[0] = b_hist;
  vHist[1] = g_hist;
  vHist[2] = r_hist;

  // channel 0
  vCumHist[0] = b_hist.clone();
  for (int i = 1; i < b_hist.rows; ++i) {
    vCumHist[0].at<float>(i) += vCumHist[0].at<float>(i - 1);
  }
  vCumHist[0] /= orig_img.rows * orig_img.cols;

  // channel 1
  vCumHist[1] = g_hist.clone();
  for (int i = 1; i < g_hist.rows; ++i) {
    vCumHist[1].at<float>(i) += vCumHist[1].at<float>(i - 1);
  }
  vCumHist[1] /= orig_img.rows * orig_img.cols;

  // channel 2
  vCumHist[2] = r_hist.clone();
  for (int i = 1; i < r_hist.rows; ++i) {
```

```
    vCumHist[2].at<float>(i) += vCumHist[2].at<float>(i - 1);
  }
  vCumHist[2] /= orig_img.rows * orig_img.cols;
}
```

Listing 2: Plot histogram using OpenCV and C++ programming language

```
void lab1::plot_hist(cv::Mat& imHist, const std::vector<cv::Mat>& vHist)
    const {
  const cv::Mat b_hist = vHist[0];
  const cv::Mat g_hist = vHist[1];
  const cv::Mat r_hist = vHist[2];

  cv::Mat b_hist_norm, r_hist_norm, g_hist_norm;

  cv::normalize(b_hist, b_hist_norm, 0, imHist.rows, cv::NORM_MINMAX, -1,
                cv::Mat());
  cv::normalize(g_hist, g_hist_norm, 0, imHist.rows, cv::NORM_MINMAX, -1,
                cv::Mat());
  cv::normalize(r_hist, r_hist_norm, 0, imHist.rows, cv::NORM_MINMAX, -1,
                cv::Mat());

  for (int i = 1; i < histSize; i++) {
    cv::rectangle(imHist,
                  cv::Point(2 * i, imHist.rows - b_hist_norm.at<float>(i)),
                  cv::Point(2 * (i - 1), imHist.rows), cv::Scalar(255, 0, 0)
    );
    cv::rectangle(imHist,
                  cv::Point(2 * i, imHist.rows - g_hist_norm.at<float>(i)),
                  cv::Point(2 * (i - 1), imHist.rows), cv::Scalar(0, 255, 0)
    );
    cv::rectangle(imHist,
                  cv::Point(2 * i, imHist.rows - r_hist_norm.at<float>(i)),
                  cv::Point(2 * (i - 1), imHist.rows), cv::Scalar(0, 0, 255)
    );
  }
}
```

Result

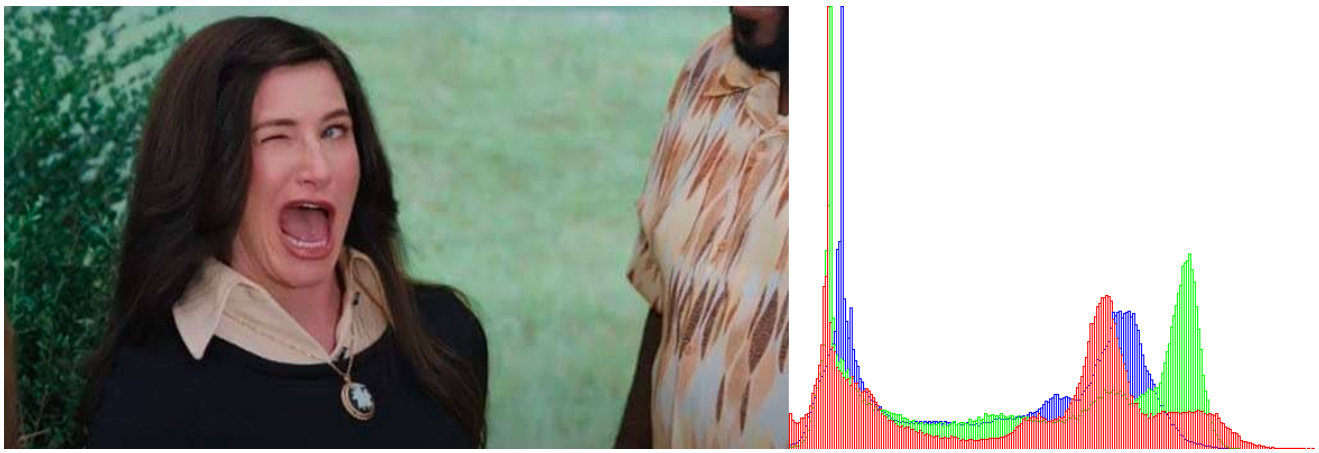

Figure 1: Image histogram creation using OpenCV and C++ programming

2.1.2 Arithmetic Operations

The simplest ways for histogram equalization are arithmetic operations on images. To increase the detail in dark areas, it is need to shift the histogram to the right lighter area, for example, by 50 gradations for each color.

OpenCV code

Listing 3: Arithmetic operations using OpenCV and C++ programming language

```
// Arithmetics Operation
cv::Mat new_img;
orig_img.convertTo(new_img, -1, 1, 50);
lab1->histogram(new_img, vHist, cum_hist);
```

Result

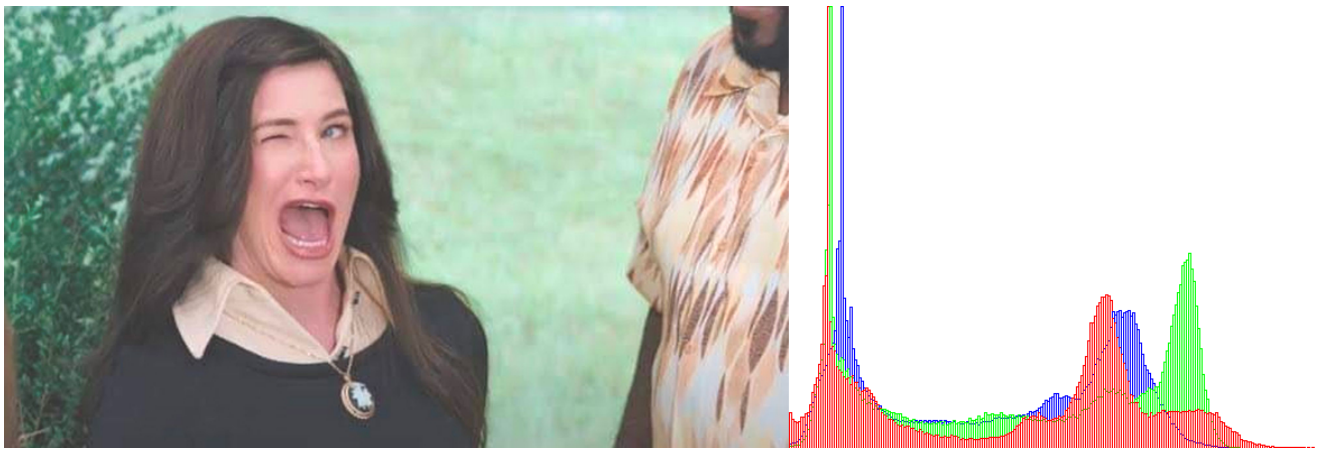

Figure 2: Arithmetic operations using OpenCV and C++ programming

2.1.3 Dynamic Range Stretching

If the ROI (regions of interests) pixel intensities are in a narrow dynamic range, we can stretch it. These transformations are performed according to the following expression

$$I_{new} = \left(\frac{I - I_{\min}}{I_{\max} - I_{\min}} \right)^{\alpha} \tag{1}$$

where I and I_{new} — intensity values arrays of the original and new images correspondingly; $I_{\min}$ and $I_{\min}$ — minimum and maximum intensity values arrays of the original image correspondingly; α — non-linearity coefficient.

OpenCV code

Listing 4: Dynamic range stretching using OpenCV and C++ programming language

```
void lab1::drs(cv::Mat& img, std::vector<cv::Mat>& vHist, const double alpha
    ) {
  std::vector<cv::Mat> BGR_Image;
  double min[3], max[3];
  int rows = img.rows;
  int cols = img.cols;

  // Convert Each Channel Pixel Values from 8U to 32F
  // Copy Image.data values to RGB_Image for Extracting All Three Frames [R,
    G,
  // B]
```

```
  img.convertTo(img, CV_32FC3);
  split(img, BGR_Image);

  // Store minimum and maximum values of [R, G, B] channels separately.
  for (int i = 0; i < 3; i++) {
    minMaxLoc(BGR_Image[i], &min[i], &max[i]);
    BGR_Image[i].convertTo(BGR_Image[i], CV_8U);
  }
  for (int row = 0; row < rows; row++) {
    for (int col = 0; col < cols; col++) {
      float B = BGR_Image[0].at<uchar>(row, col) - min[0];
      float G = BGR_Image[1].at<uchar>(row, col) - min[1];
      float R = BGR_Image[2].at<uchar>(row, col) - min[2];

      double contrast_B = max[0] - min[0];
      double contrast_G = max[1] - min[1];
      double contrast_R = max[2] - min[2];

      BGR_Image[0].at<uchar>(row, col) =
          std::pow(B, alpha) * 255 / std::pow(contrast_B, alpha);
      BGR_Image[1].at<uchar>(row, col) =
          std::pow(G, alpha) * 255 / std::pow(contrast_G, alpha);
      BGR_Image[2].at<uchar>(row, col) =
          std::pow(R, alpha) * 255 / std::pow(contrast_R, alpha);
    }
  }
  // Merge into output image
  merge(BGR_Image, img);
}
```

Result

Figure 3: Dynamic range stretching using OpenCV and C++ programming

2.1.4 Uniform Transformation

It is carried out according to the following formula:

$$I_{new} = (I_{\max} - I_{\min}) \cdot P(I) + I_{\min} \tag{2}$$

where I_{new} — intensity values arrays of the new images correspondingly; $I_{\min}$ and $I_{\min}$ — minimum and maximum intensity values arrays of the original image correspondingly; $P(I)$ — original image probability distribution function which is approximated by cumulative histogram:

$$P(I) \approx \sum_{m=0}^{i} \text{Hist}(m) \tag{3}$$

OpenCV code

The OpenCV code for computing cumulative histogram is shown Listing 1.

Listing 5: Uniform transformation using OpenCV and C++ programming language

```
void lab1::uniform_trans(cv::Mat& img, std::vector<cv::Mat>& vHist) {
  std::vector<cv::Mat> vCumHist, BGR_Image;
  double min[3], max[3];

  vCumHist.resize(3);
  histogram(img, vHist, vCumHist);

  // Convert Each Channel Pixel Values from 8U to 32F
  // Copy Image.data values to RGB_Image for Extracting All Three Frames [R,
    G,
  // B]
  img.convertTo(img, CV_32F);
  split(img, BGR_Image);

  // Store minimum and maximum values of [R, G, B] channels separately.
  for (int i = 0; i < 3; i++) {
    minMaxLoc(BGR_Image[i], &min[i], &max[i]);
    BGR_Image[i].convertTo(BGR_Image[i], CV_8U);
  }
  for (int row = 0; row < img.rows; row++) {
    for (int col = 0; col < img.cols; col++) {
      double contrast_B = max[0] - min[0];
      double contrast_G = max[1] - min[1];
      double contrast_R = max[2] - min[2];

      BGR_Image[0].at<uchar>(row, col) =
          contrast_B * vCumHist[0].at<float>(BGR_Image[0].at<uchar>(row, col
   )) +
          min[0];
      BGR_Image[1].at<uchar>(row, col) =
          contrast_G * vCumHist[1].at<float>(BGR_Image[1].at<uchar>(row, col
   )) +
          min[1];
      BGR_Image[2].at<uchar>(row, col) =
          contrast_R * vCumHist[2].at<float>(BGR_Image[2].at<uchar>(row, col
   )) +
          min[2];
    }
  }
  // Merge into output image
  merge(BGR_Image, img);
}
```

Result

2.1.5 Exponential Transformation

It is carried out according to the following formula:

$$I_{\text{new}} = I_{\text{min}} - \frac{1}{\alpha} \cdot \ln(1 - P(I)), \tag{4}$$

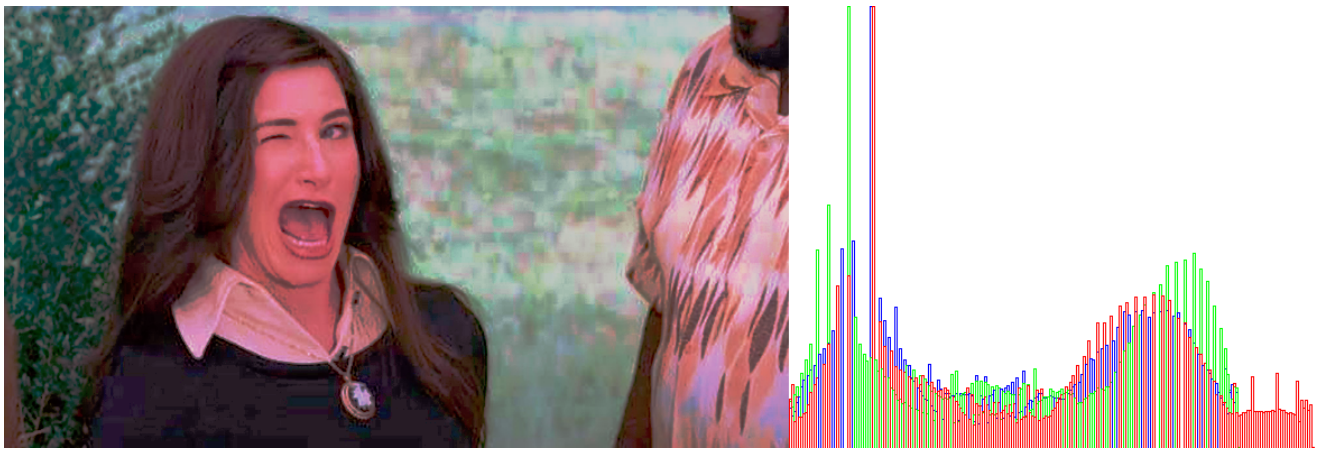

Figure 4: Uniform transformation using OpenCV and C++ programming

where α — constant characterizes transformation slope.

OpenCV code

Listing 6: Uniform transformation using OpenCV and C++ programming language

```
std::vector<cv::Mat> vCumHist, BGR_Image;
  double min[3], max[3];

  vCumHist.resize(3);
  histogram(img, vHist, vCumHist);

  // Convert Each Channel Pixel Values from 8U to 32F
  // Copy Image.data values to RGB_Image for Extracting All Three Frames [R,
    G,
  // B]
  img.convertTo(img, CV_32FC3);
  split(img, BGR_Image);

  // Store minimum and maximum values of [R, G, B] channels separately.
  for (int i = 0; i < 3; i++) {
    minMaxLoc(BGR_Image[i], &min[i], &max[i]);
    BGR_Image[i].convertTo(BGR_Image[i], CV_8U);
  }

  for (int row = 0; row < img.rows; row++) {
    for (int col = 0; col < img.cols; col++) {
      float B = vCumHist[0].at<float>(BGR_Image[0].at<uchar>(row, col));
      float G = vCumHist[1].at<float>(BGR_Image[1].at<uchar>(row, col));
      float R = vCumHist[2].at<float>(BGR_Image[2].at<uchar>(row, col));

      BGR_Image[0].at<uchar>(row, col) = cv::saturate_cast<uchar>(
          255 * (min[0] - 1 / alpha * std::log(1 - B)));
      BGR_Image[1].at<uchar>(row, col) = cv::saturate_cast<uchar>(
          255 * (min[1] - 1 / alpha * std::log(1 - G)));
      BGR_Image[2].at<uchar>(row, col) = cv::saturate_cast<uchar>(
          255 * (min[2] - 1 / alpha * std::log(1 - R)));
    }
  }
  // Merge into output image
  merge(BGR_Image, img);
}
```

Result

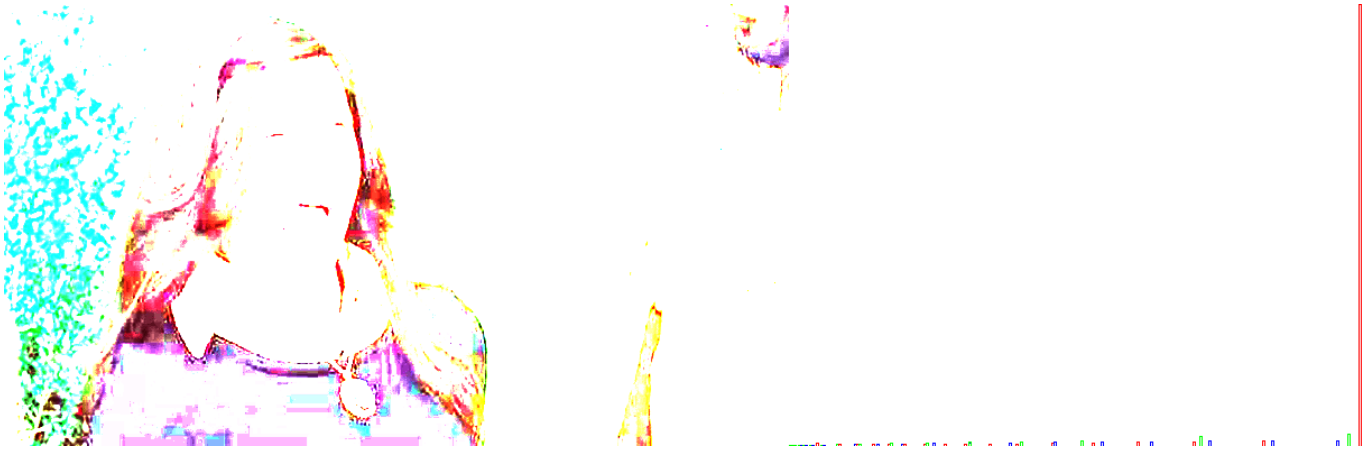

Figure 5: Exponential transformation using OpenCV and C++ programming

2.1.6 Rayleigh Transformation

It is carried out according to the following formula:

$$I_{new} = I_{\min} + \left(2\alpha^2 \ln\left(\frac{1}{1 - P(I)}\right)\right)^{1/2} \tag{5}$$

OpenCV code

Listing 7: Rayleigh transformation using OpenCV and C++ programming language

```
void lab1::ray_trans(cv::Mat& img, std::vector<cv::Mat>& vHist,
                     const double alpha) {
  std::vector<cv::Mat> vCumHist, BGR_Image;
  double min[3], max[3];
  double alpha2 = 2 * alpha * alpha;
  vCumHist.resize(3);
  histogram(img, vHist, vCumHist);

  // Convert Each Channel Pixel Values from 8UC3 to 32FC3
  // Copy Image.data values to RGB_Image for Extracting All Three Frames [R,
    G,
  // B]
  img.convertTo(img, CV_32FC3);
  split(img, BGR_Image);

  // Store minimum and maximum values of [R, G, B] channels separately.
  for (int i = 0; i < 3; i++) {
    minMaxLoc(BGR_Image[i], &min[i], &max[i]);
    BGR_Image[i].convertTo(BGR_Image[i], CV_8U);
  }
  for (int row = 0; row < img.rows; row++) {
    for (int col = 0; col < img.cols; col++) {
      float B = vCumHist[0].at<float>(BGR_Image[0].at<uchar>(row, col));
      float G = vCumHist[1].at<float>(BGR_Image[1].at<uchar>(row, col));
      float R = vCumHist[2].at<float>(BGR_Image[2].at<uchar>(row, col));

      BGR_Image[0].at<uchar>(row, col) = cv::saturate_cast<uchar>(
          255 * (min[0] + std::sqrt(alpha2 * std::log(1 / (1 - B)))));
      BGR_Image[1].at<uchar>(row, col) = cv::saturate_cast<uchar>(
          255 * (min[1] + std::sqrt(alpha2 * std::log(1 / (1 - G)))));
      BGR_Image[2].at<uchar>(row, col) = cv::saturate_cast<uchar>(
          255 * (min[2] + std::sqrt(alpha2 * std::log(1 / (1 - R)))));
    }
```

```
  }
  // Merge into output image
  merge(BGR_Image, img);
}
```

Result

Figure 6: Rayleigh transformation using OpenCV and C++ programming

2.1.7 Transformation of 2/3-degree

It is carried out according to the following formula:

$$I_{new} = P(I)^{2/3} \tag{6}$$

OpenCV code

Listing 8: Transformation of 2/3-degree using OpenCV and C++ programming language

```
void lab1::trans23(cv::Mat& img, std::vector<cv::Mat>& vHist) {
  std::vector<cv::Mat> vCumHist, BGR_Image;
  vCumHist.resize(3);
  histogram(img, vHist, vCumHist);

  // Convert Each Channel Pixel Values from 8U to 32F
  // Copy Image.data values to RGB_Image for Extracting All Three Frames [R,
    G,
  // B]
  img.convertTo(img, CV_32FC3);
  split(img, BGR_Image);

  // Store minimum and maximum values of [R, G, B] channels separately.
  for (int i = 0; i < 3; i++) {
    BGR_Image[i].convertTo(BGR_Image[i], CV_8U);
  }

  for (int row = 0; row < img.rows; row++) {
    for (int col = 0; col < img.cols; col++) {
      float B = vCumHist[0].at<float>(BGR_Image[0].at<uchar>(row, col));
      float G = vCumHist[1].at<float>(BGR_Image[1].at<uchar>(row, col));
      float R = vCumHist[2].at<float>(BGR_Image[2].at<uchar>(row, col));

      BGR_Image[0].at<uchar>(row, col) =
          cv::saturate_cast<uchar>(255 * powf(B, (float)2 / 3));
      BGR_Image[1].at<uchar>(row, col) =
```

```
            cv::saturate_cast<uchar>(255 * powf(G, (float)2 / 3));
        BGR_Image[2].at<uchar>(row, col) =
            cv::saturate_cast<uchar>(255 * powf(R, (float)2 / 3));
      }
    }
    // Merge into output image
    merge(BGR_Image, img);
}
```

Result

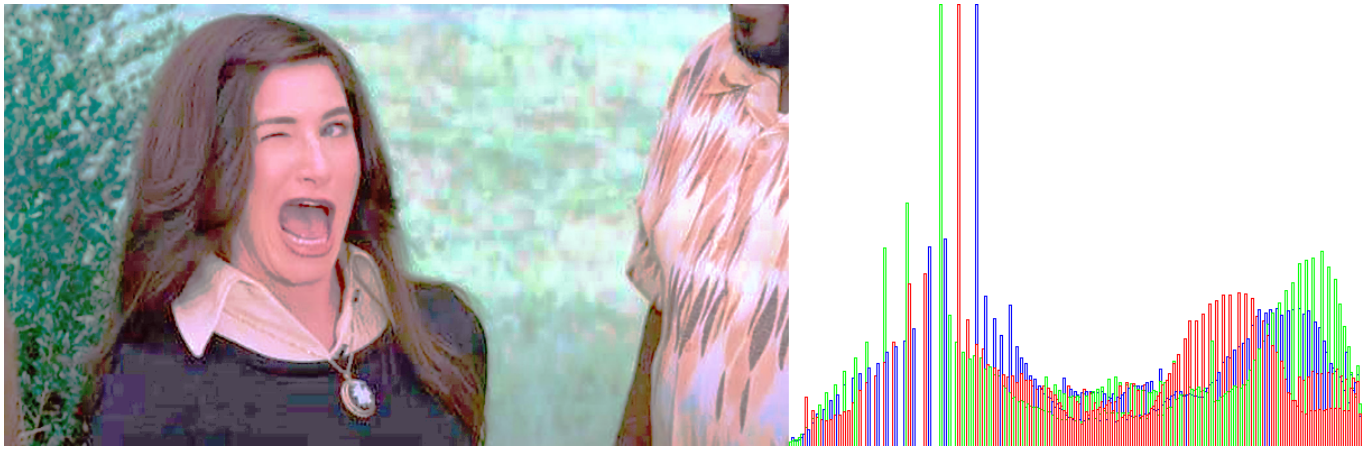

Figure 7: Transformation of 2/3-degree using OpenCV and C++ programming

2.1.8 Hyperbolic Transformation

It is carried out according to the following formula:

$$I_{new} = \alpha^{P(I)} \tag{7}$$

OpenCV code

Listing 9: Hyperbolic transformation using OpenCV and C++ programming language

```
void lab1::hyper_trans(cv::Mat& img, std::vector<cv::Mat>& vHist,
                       const double alpha) {
  std::vector<cv::Mat> vCumHist, BGR_Image;
  vCumHist.resize(3);
  histogram(img, vHist, vCumHist);

  // Convert Each Channel Pixel Values from 8U to 32F
  // Copy Image.data values to RGB_Image for Extracting All Three Frames [R,
    G,
  // B]
  img.convertTo(img, CV_32FC3);
  split(img, BGR_Image);

  // Store minimum and maximum values of [R, G, B] channels separately.
  for (int i = 0; i < 3; i++) {
    BGR_Image[i].convertTo(BGR_Image[i], CV_8U);
  }

  for (int row = 0; row < img.rows; row++) {
    for (int col = 0; col < img.cols; col++) {
      float B = vCumHist[0].at<float>(BGR_Image[0].at<uchar>(row, col));
      float G = vCumHist[1].at<float>(BGR_Image[1].at<uchar>(row, col));
```

```
22        float R = vCumHist[2].at<float>(BGR_Image[2].at<uchar>(row, col));
23
24        BGR_Image[0].at<uchar>(row, col) =
25            cv::saturate_cast<uchar>(255 * std::pow(alpha, B));
26        BGR_Image[1].at<uchar>(row, col) =
27            cv::saturate_cast<uchar>(255 * std::pow(alpha, G));
28        BGR_Image[2].at<uchar>(row, col) =
29            cv::saturate_cast<uchar>(255 * std::pow(alpha, R));
30      }
31    }
32    merge(BGR_Image, img);
33
34    for (auto& i : vHist) i.setTo(cv::Scalar::all(0));
35    for (auto& i : vCumHist) i.setTo(cv::Scalar::all(0));
36
37    histogram(img, vHist, vCumHist);
38 }
```

Result

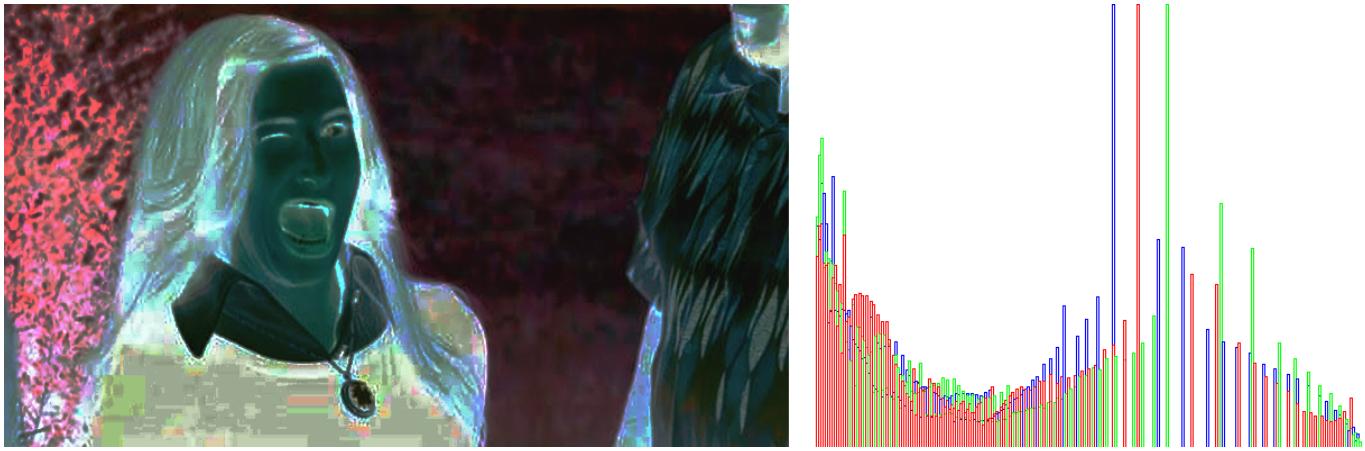

Figure 8: Hyperbolic transformation using OpenCV and C++ programming language

2.1.9 Histogram Equalization

Enhances the contrast of the grayscale image by transforming the values using contrast-limited adaptive histogram equalization (CLAHE).

OpenCV code

Listing 10: CLAHE - Enhances contrast using OpenCV and C++ programming language

```
cv::Mat orig_gray_img, new_gray_img;
  cv::cvtColor(orig_img, orig_gray_img, CV_BGR2GRAY);
  new_gray_img = orig_gray_img.clone();
  cv::Ptr<cv::CLAHE> clahe = cv::createCLAHE();
  clahe->setClipLimit(4);
  clahe->apply(orig_gray_img, new_gray_img);
  cv::hconcat(orig_gray_img, new_gray_img, show_im2);
  cv::imshow("Image", show_im2);
```

Result

Figure 9: CLAHE - Enhances contrast using OpenCV and C++ programming language

2.2 Image Profiles

Image profile along an arbitrary line is the intensity function image which is distributed along this line. The simplest case of image profile is a row profile:

$$\text{Profile } i(x) = I(x, i) \tag{8}$$

where i is a row number of image I.

Column profile is:

$$\text{Profile } j(x) = I(j, y) \tag{9}$$

OpenCV code

Listing 11: Image profiles using OpenCV and C++ programming language

```
void lab1::profile(const cv::Mat& img, cv::Mat& pro_x, cv::Mat& pro_y) {
  cv::Mat gray_img;
  cv::cvtColor(img, gray_img, CV_BGR2GRAY);
  pro_x = gray_img.row(gray_img.rows / 2);
  pro_y = gray_img.col(gray_img.cols / 2);
}
```

Result

2.3 Image Projections

Image projections is the sum of the image pixels intensities in the direction perpendicular to this axis. The vertical projection on the axis O_x, which is the pixel intensities sum by columns of the image:

$$\text{Proj}\, X(y) = \sum_{y=0}^{\dim Y - 1} I(x, y) \tag{10}$$

The horizontal projection on the axis O_y, which is the pixel intensities sum by rows of the image:

$$\text{Proj}\, Y(x) = \sum_{x=0}^{\dim X - 1} I(x, y) \tag{11}$$

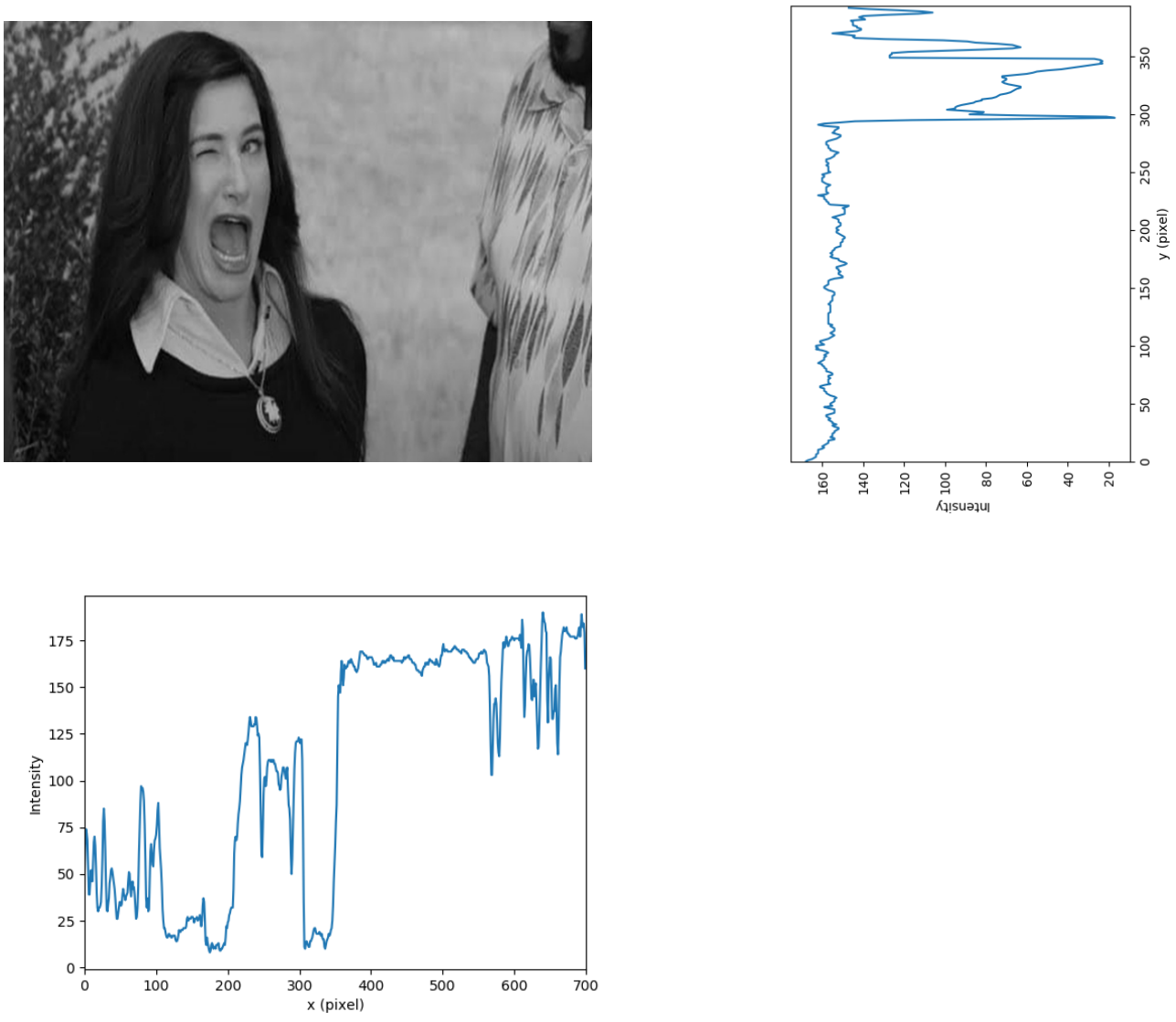

Figure 10: Image profiles using OpenCV and C++ programming language

OpenCV code

Listing 12: Image projections using OpenCV and C++ programming language

```
void lab1::projection(const cv::Mat& img, cv::Mat& proj_x, cv::Mat& proj_y)
    {
  cv::Mat gray_img;
  cv::cvtColor(img, gray_img, CV_BGR2GRAY);
  // Sum by Rows - ProjY
  cv::reduce(gray_img, proj_y, 1, cv::REDUCE_SUM, CV_32F);
  // Sum by Cols - ProjX
  cv::reduce(gray_img, proj_x, 0, cv::REDUCE_SUM, CV_32F);
  proj_x /= 255;
  proj_y /= 255;
}
```

Result

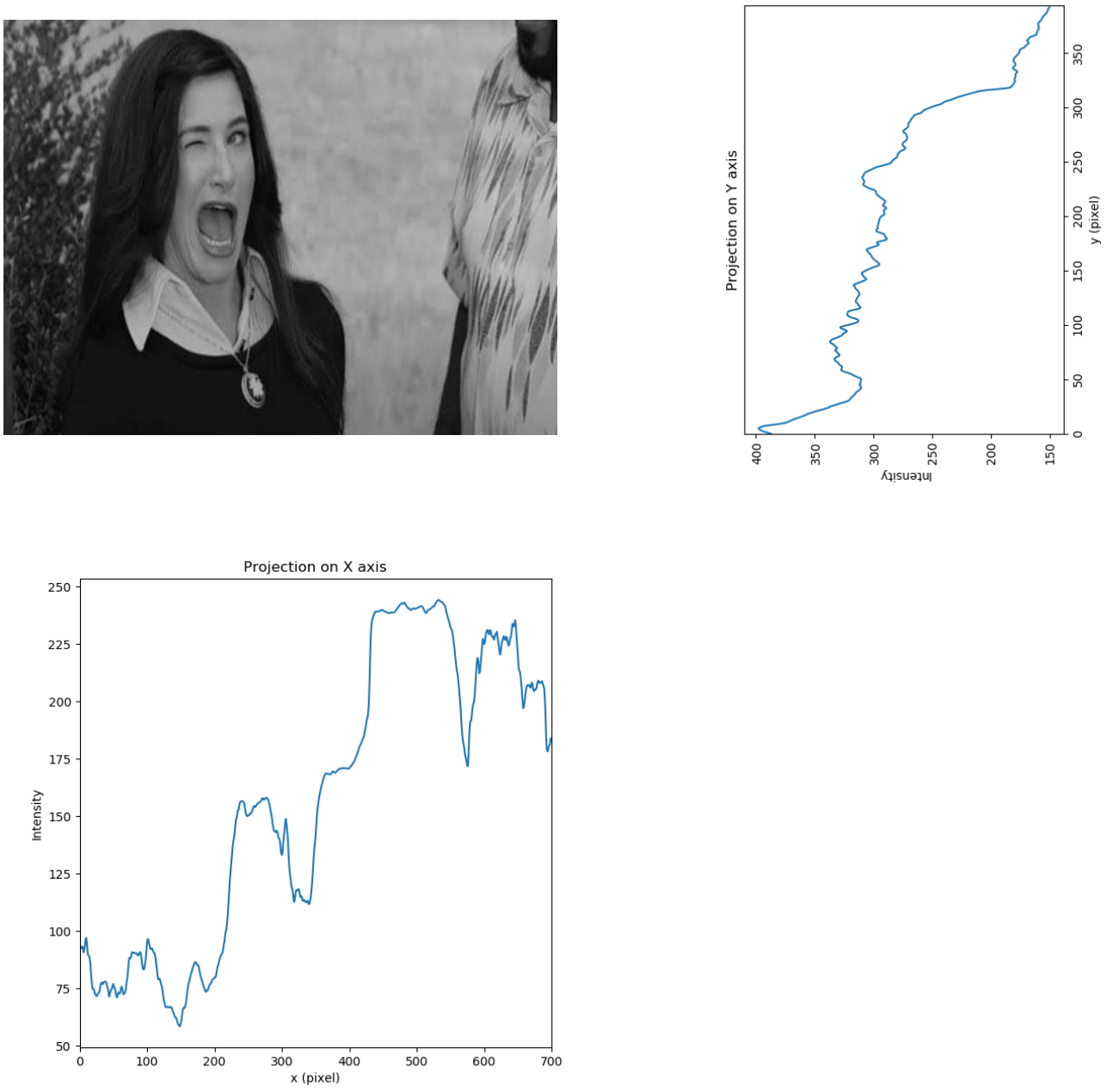

Figure 11: Image projections using OpenCV and C++ programming language

3 Conclusion

In this work, we implement the calculation of histogram image, several contrast transformations and histogram equalization technique. Image profile and image projection on two basic axes $O_x,\ O_y$ has been performed.

A Plot Image Profile by *matplotlibcpp* Library using C++ programming language

```
// Image Profile
cv::Mat profile_ox,profile_oy;
DIP::lab1::profile(orig_img, profile_ox, profile_oy);
std::vector<int> cols, pix_x,rows, pix_y;
for (int i = 0; i < profile_ox.cols; ++i) {
    cols.push_back(i);
    pix_x.push_back((int)profile_ox.at<uchar>(0, i));
}
for (int i = 0; i < profile_oy.rows; ++i) {
    rows.push_back(i);
    pix_y.push_back((int)profile_oy.at<uchar>(0, i));
}

matplotlibcpp::plot(cols, pix_x);
matplotlibcpp::xlabel("x (pixel)");
matplotlibcpp::ylabel("Intensity");
matplotlibcpp::xlim(0, profile_ox.cols);
matplotlibcpp::figure_size(512, 512);
matplotlibcpp::show(true);

matplotlibcpp::plot(rows, pix_y);
matplotlibcpp::xlabel("y (pixel)");
matplotlibcpp::ylabel("Intensity");
matplotlibcpp::xlim(0, profile_oy.rows);
matplotlibcpp::figure_size(512, 512);
matplotlibcpp::show(true);
```

B Plot Image Projection by *matplotlibcpp* Library using C++ programming language

```
// Image Projection
cv::Mat proj_oy;
cv::Mat proj_ox;

DIP::lab1::projection(orig_img, proj_ox, proj_oy);
std::vector<double> proj_x, proj_y, vCol, vRow;
for (int i = 0; i < proj_ox.cols; ++i) {
  vCol.push_back(i);
  proj_x.push_back(proj_ox.at<float>(0, i));
}
for (int i = 0; i < proj_oy.rows; ++i) {
  vRow.push_back(i);
  proj_y.push_back(proj_oy.at<float>(i, 0));
}
matplotlibcpp::figure_size((int)proj_x.size(), (int)proj_x.size());
matplotlibcpp::plot(vCol, proj_x);
matplotlibcpp::title("Projection on X axis");
matplotlibcpp::xlabel("x (pixel)");
matplotlibcpp::ylabel("Intensity");
matplotlibcpp::xlim(0, (int)proj_x.size());
matplotlibcpp::show(true);

matplotlibcpp::xlim(0, (int)proj_y.size());
matplotlibcpp::plot(vRow, proj_y);
matplotlibcpp::title("Projection on Y axis");
matplotlibcpp::xlabel("y (pixel)");
matplotlibcpp::ylabel("Intensity");
matplotlibcpp::figure_size((int)proj_y.size(), (int)proj_y.size());
matplotlibcpp::show(true);
```