A photographic negative imaging inspired method for low illumination night-time image enhancement

Student : Rogoz Bogdan
Group : 30433

Introduction

Low illumination night-time image enhancement is critical for a wide range of image-related outdoor applications, such as surveillance systems, intelligent vehicles, satellite imaging, and outdoor object recognition systems. Images captured in night-time under low illumination conditions are usually seriously degraded due to insufficient light, such as low contrast, distorted colors and unclear details etc., and seriously affect the performance of image-related outdoor systems. It is obvious that low illumination night-time image enhancement is highly desirable to ensure the reliable of outdoor vision systems.

The novel contributions of the method are summarized as follows:

(1) Negative night time images can be conserved as hazed images

(2) Low illumination image enhancement can be realized by rectification its corresponding negative images using a image dehazing method.

Algorithm steps

1. Compute the negative of the night-time image
2. Rectify the negative image using dark-channel prior
    1. Compute the dark channel of the negative image
    2. Compute the gray histogram of the dark channel
    3. Estimate the value of A (atmospheric light)
    4. Compute the transmission factor
    5. Compute the rectified image
3. Negate the rectified image

Code snippet – Negate image

```
Mat negateImage(Mat src)
{
    Mat res = Mat::zeros(src.rows, src.cols, CV_8UC3);
    for(int i=0; i<src.rows; i++)
    {
        for(int j=0; j<src.cols; j++)
        {
            Vec3b pix = src.at<Vec3b>(i, j);
            pix.val[0] = 255 - pix.val[0];
            pix.val[1] = 255 - pix.val[1];
            pix.val[2] = 255 - pix.val[2];
            res.at<Vec3b>(i, j) = pix;
        }
    }

    return res;
}
```

Code snippet – Dark channel

In order to compute the dark channel of an RGB Image, we take each pixel and store the channel With the lowest intensity value.

For example, for a pixel with RGB = (100, 90, 110), The dark channel image will store the value 90 on That position.

```
uchar minPixel(Mat src, int y, int x)
{
    int offset = patchSize / 2;
    uchar res = 255;

    for(int i=y-offset; i<=y+offset; i++)
    {
        if(i < 0 || i >= src.rows) continue;
        for(int j=x-offset; j<=x+offset; j++)
        {
            if(j < 0 || j >= src.cols) continue;
            Vec3b pix = src.at<Vec3b>(i, j);
            uchar minPix = min(pix.val[0], min(pix.val[1], pix.val[2]));
            res = min(res, minPix);
        }
    }

    return res;
}

Mat computeDarkChannel(Mat src)
{
    Mat res = Mat::zeros(src.rows, src.cols, CV_8UC1);

    for(int i=0; i<src.rows; i++)
    {
        for(int j=0; j<src.cols; j++)
        {
            res.at<uchar>(i, j) = minPixel(src, i, j);
        }
    }

    return res;
}
```

Code snippet – Atmospheric light

In order to compute the atmospheric light,

We take the top 0.1% brightest pixels in the

Dark channel image and compute their sum.

After computing the sum of the brightest pixels

In the image, we return their average value.

```
uchar computeA(Mat dark)
{
    std::priority_queue<uchar, std::vector<uchar>, std::greater<uchar> > topVals;
    int total = 0;
    int number = dark.rows * dark.cols / 1000;

    for(int i=0; i<dark.rows; i++)
    {
        for(int j=0; j<dark.cols; j++)
        {
            uchar pix = dark.at<uchar>(i, j);
            if(topVals.size() < number)
            {
                topVals.push(pix);
                total += pix;
            }
            else
            {
                if(pix > topVals.top())
                {
                    total -= topVals.top();
                    topVals.pop();
                    topVals.push(pix);
                    total += pix;
                }
                else continue;
            }
        }
    }

    // topVals.clear();
    return (uchar)(total / number);
}
```

Code snippet – Rectified image

In order to compute the rectified Image, we need to compute each pixel's transmission coefficient. The transmission coefficient Is computed with the formula:

$$\tilde{t}(\mathbf{x}) = 1 - \omega \min_{c}\left(\min_{\mathbf{y}\in\Omega(\mathbf{x})} \left(\frac{I^c(\mathbf{y})}{A^c}\right)\right).$$

If the transmission coefficient is Too low, a default value *t0* is Assigned. The final rectified image Is computed using the formula:

$$\mathbf{J}(\mathbf{x}) = \frac{\mathbf{I}(\mathbf{x}) - \mathbf{A}}{\max(t(\mathbf{x}), t_0)} + \mathbf{A}.$$

```
Mat computeRectified(Mat neg, Mat dark, uchar A)
{
    Mat res = Mat::zeros(neg.rows, neg.cols, CV_8UC3);
    for(int i=0; i<res.rows; i++)
    {
        for(int j=0; j<res.cols; j++)
        {
            float chosenT = max(1.0f - omega * dark.at<uchar>(i, j) / A, t0);
            Vec3b pix = neg.at<Vec3b>(i, j);
            Vec3b sol = pix;
            sol.val[0] = (uchar) min(((float)pix.val[0] - A) / chosenT + A, 255.0f);
            sol.val[1] = (uchar) min(((float)pix.val[1] - A) / chosenT + A, 255.0f);
            sol.val[2] = (uchar) min(((float)pix.val[2] - A) / chosenT + A, 255.0f);
            res.at<Vec3b>(i, j) = sol;
        }
    }

    return res;
}
```

Code snippet – Constants and main function

```
const int patchSize = 15;
const float omega = 0.95f;
const float t0 = 0.1f;
```

```
int main(int argc, char **argv)
{
    // char fname[100];
    Mat src;
    Mat neg;
    Mat darkChannel;
    Mat rectified;
    Mat enhanced;

    //
    uchar A;

    // while(openFileDlg(fname))
    // {
    //     src = imread(fname, CV_LOAD_IMAGE_COLOR);
    //     neg = negateImage(src);
    // }


    src = imread(argv[1], CV_LOAD_IMAGE_COLOR);
    neg = negateImage(src);
    darkChannel = computeDarkChannel(neg);

    A = computeA(darkChannel);
    rectified = computeRectified(neg, darkChannel, A);
    // GaussianBlur(rectified, rectified, Size(5, 5), 7.0f);
    enhanced = negateImage(rectified);

    imshow("Source", src);
    // imshow("Negated", neg);
    // imshow("Dark channel", darkChannel);
    imshow("Rectified", rectified);
    imshow("Enhanced", enhanced);

    // imwrite("output.jpg", rectified);

    waitKey();

    return 0;
}
```

Examples

Original

Enhanced

Examples

Original

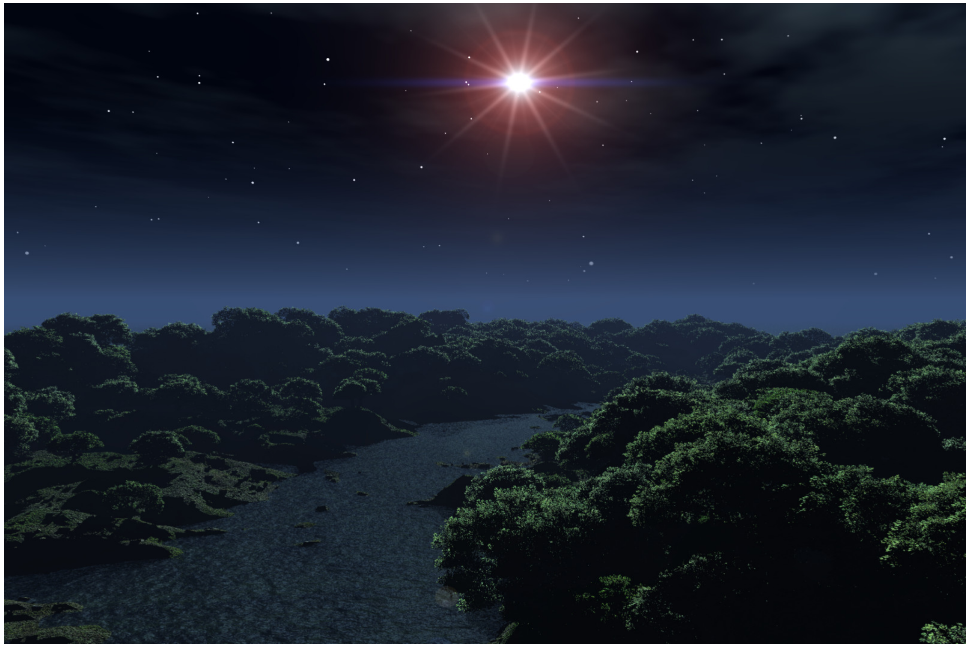
Enhanced

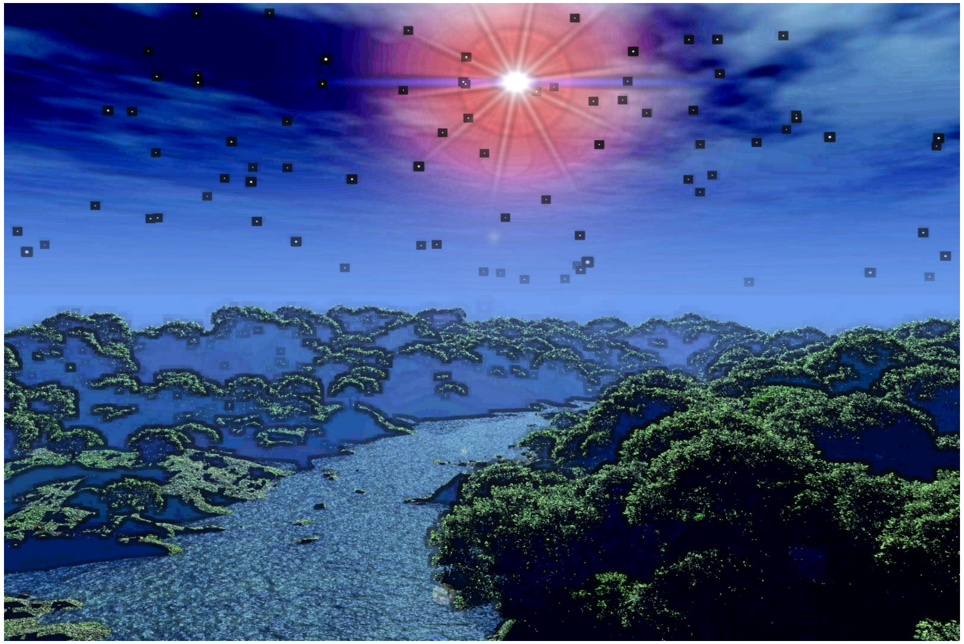
Examples

Original

Enhanced

Examples





Original

Enhanced

Examples

Original

Enhanced

Future improvements

Future improvements include:

- Soft matting for improved output image quality
- Mobile port
- Video (real-time image) rendering

References

https://link.springer.com/article/10.1007/s11042-017-4453-z

https://www.robots.ox.ac.uk/~vgg/rg/papers/hazeremoval.pdf

https://jivp-eurasipjournals.springeropen.com/articles/10.1186/s13640-016-0104-y