Disparity Map using Stereo Image Pairs

CS4186 Assignment 2

Tam Chin Pang
56226481
22 April 2022

Introduction

In this assignment, 3 disparity maps are computed using 3 corresponded pairs of stereo images. Peak Signal-to-Noise Ratio (PSNR) is implemented to do the comparison against the ground-truth disparity maps.

What-is-used

Software

Visual Studio 2019 (for computation of disparity maps)

Visual Studio Code (for PSNR)

Programming Language

C++

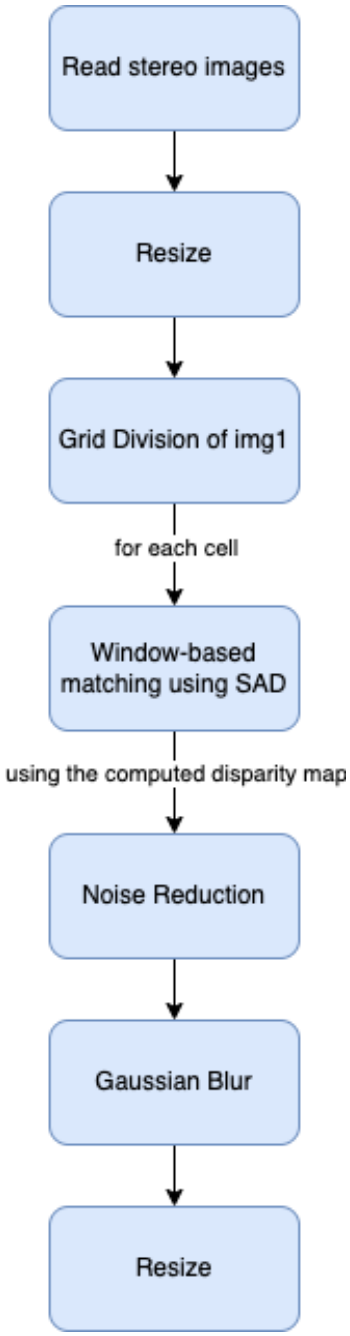
Python

Additional Library

OpenCV 4.5.5

Solution

Flow of Program



Resize

```
double img_w = img1.cols;
double img_h = img1.rows;
int num_window_w = img_w / (double)window_size;
int num_window_h = img_h / (double)window_size;
int new_w = num_window_w * window_size;
int new_h = num_window_h * window_size;
Mat new_img1;
Mat new_img2;
resize(img1, new_img1, Size(new_w, new_h), INTER_LINEAR);
resize(img2, new_img2, Size(new_w, new_h), INTER_LINEAR);
```

To obtain a better scale for dividing the image into cells, I resize the images. For example, given that the cell's size is 50, the dimension of the Art's image changes from 1390*1110 to 1350*1100.

Grid's Division of View_1 image

```
for (int y = 0; y < num_window_h; y++) {
    for (int x = 0; x < num_window_w; x++) {
        Mat crop_img1;
        crop_img1 = Grey_img1(Range(y * window_size, y * window_size + window_size - 1), Range(x * window_size, x * window_size + window_size - 1));
```

I divide the image into num_window_w * num_window_h grid. Each of the cells will be matched with a cell in View_5 and therefore represent a disparity's pixel during the window-based matching. Therefore, the size of the cell determines the final resolution of the disparity map. The smaller the cells' size, the higher the resolution.

Window-Based Matching

```
double min_score = 10000000;
int best_window = 0;

int min_slide = x * window_size -  (int)(num_window_w*0.3) * window_size;
if (min_slide < 0) {
    min_slide = 0;
}
int max_slide = x * window_size;
for (int slide = min_slide; slide < max_slide-1; slide ++) {
    Mat crop_img2;
    crop_img2 = Grey_img2(Range(y * window_size, y * window_size + window_size - 1), Range(slide, slide + window_size -1));

    double score = SAD(crop_img1,crop_img2);
    if (score < min_score) {
        min_score = score;
        best_window = slide;
    }
}
int dist_x = abs(x*window_size - best_window);
```

Window-based matching is exploited to find the best matching in View_5. As the provided stereo images are rectified, the corresponding epipolar scan-line in the view_5 is the horizontal line with the same height as the View_1. Additionally, to reduce the computation cost, I reduce the length of the scan-line to 30% of the image's width. Given the above information, windows can be assigned to slide along the scan-line in View_5. The best match for every disparity's pixel in View_1 will be stored and used to measure their disparity.

Sum of Absolute Differences (SAD)

```
double SAD(Mat img1,Mat img2) {
    Mat img_1 = img1;
    Mat img_2 = img2;
    if (img_1.type() != CV_32F) {
        img_1.convertTo(img_1, CV_32F);
    }
    if (img_2.type() != CV_32F) {
        img_2.convertTo(img_2, CV_32F);
    }

    double sum = 0;
    for (int y = 0; y < img_1.rows; y++) {
        for (int x = 0; x < img_1.cols; x++) {
            float diff = abs(img_1.at<float>(y, x) - img_2.at<float>(y, x));
            sum += (double)diff;
        }
    }

    return sum;
}
```

SAD is exploited to obtain the matching cost in this algorithm. The higher the cost, the smaller the similarity between the windows in View_1 and View_5. Moreover, in this program, since only the window itself is used to compute the SAD, its size also determines the efficiency of the matching method. Therefore, if the window is larger, more features can be obtained and thus enhances the higher matching accuracy.

Noise Reduction

```
for (int y = disparityMap.rows - 2; y > 0; y--) {
    for (int x = disparityMap.cols - 2; x > 0; x--) {

        int confidence = 8;
        int temp = disparityMap.at<uchar>(y, x + 1);

        if (abs(disparityMap.at<uchar>(y, x) - disparityMap.at<uchar>(y - 1, x - 1)) > 60) {
            confidence--;
            temp = disparityMap.at<uchar>(y - 1, x - 1);
        }
        if (abs(disparityMap.at<uchar>(y, x) - disparityMap.at<uchar>(y - 1, x)) > 60) {
            confidence--;
            temp = disparityMap.at<uchar>(y - 1, x);
        }
        if (abs(disparityMap.at<uchar>(y, x) - disparityMap.at<uchar>(y - 1, x + 1)) > 60) {
            confidence--;
            temp = disparityMap.at<uchar>(y - 1, x + 1);
        }
        if (abs(disparityMap.at<uchar>(y, x) - disparityMap.at<uchar>(y , x - 1)) > 60) {
            confidence--;
            temp = disparityMap.at<uchar>(y, x - 1);
        }
        if (abs(disparityMap.at<uchar>(y, x) - disparityMap.at<uchar>(y, x + 1)) > 60) {
            confidence--;
            temp = disparityMap.at<uchar>(y, x + 1);
        }
        if (abs(disparityMap.at<uchar>(y, x) - disparityMap.at<uchar>(y + 1, x - 1)) > 60) {
            confidence--;
            temp = disparityMap.at<uchar>(y + 1, x - 1);
        }
        if (abs(disparityMap.at<uchar>(y, x) - disparityMap.at<uchar>(y + 1, x)) > 60) {
            confidence--;
            temp = disparityMap.at<uchar>(y + 1, x);
        }
        if (abs(disparityMap.at<uchar>(y, x) - disparityMap.at<uchar>(y + 1, x + 1)) > 60) {
            confidence--;
            temp = disparityMap.at<uchar>(y + 1, x + 1);
        }
```

```
    if (confidence<4) {
        disparityMap.at<uchar>(y, x) = temp;
    }

    if (disparityMap.at<uchar>(y, x) == 255 || disparityMap.at<uchar>(y, x) == 0) {
        disparityMap.at<uchar>(y, x) = temp;
    }
}
```

Two methods are implemented to reduce noise. For the first one, the objective is to find the local extrema in the image. If the confidence is low, the program identifies the corresponded pixel as the noise. Then, the pixel value of all noise will be changed to that of their neighbors. In the second method, the objective is to eliminate the pixels with unknown disparity (likes 0, or 255).

Filter

```
GaussianBlur(disparityMap, disparityMap,Size(5,5),0);
```

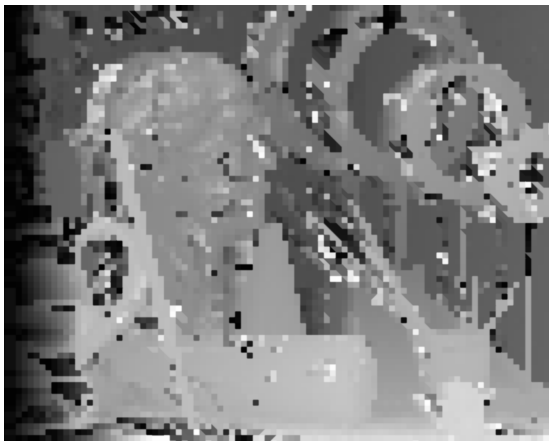
Eventually, Gaussian filter is applied to blur the computed disparity map.

Result

window size = 5

window size = 7

window size = 10



window size = 15

To reduce noise, I select the one with window size = 15.