Homework 4: Binocular Stereo

November 6, 2025

Due Date: November 27, by 23:59

Introduction

In this project, you will implement a stereo matching algorithm for rectified stereo pairs. For simplicity, you will work under the assumption that the image planes of the two cameras are parallel to each other and to the baseline. The project requires implementing algorithms to compute disparity maps from stereo image pairs and visualizing depth maps.

To see examples of disparity maps, run `python main.py --tasks 0` to visualize the comparison of disparity map generated by `cv2.StereoBM` and the ground truth.

1 Basic Stereo Matching Algorithm (60 pts.)

1.1 Disparity Map Computation (30 pts.)

Implement the function `task1_compute_disparity_map_simple()` to return the disparity map of a given stereo pair. The function takes the reference image and the second image as inputs, along with the following hyperparameters:

- `window_size`: the size of the window used for matching.
- `disparity_range`: the minimum and maximum disparity value to search.
- `matching_function`: the function used for computing the matching cost.

The function should implement a simple window-based stereo matching algorithm, as outlined in the **Basic Stereo Matching Algorithm** section in lecture slides 08:

For each pixel in the first (reference) image, examine the corresponding scanline (in our case, the same row) in the second image to search for a best-matching window. The output should be a disparity map with respect to **the first (reference) image**.

Note that you should also manage to record the **running time** of your code, which should be included in the report.

1.2 Hyperparameter Settings and Report (30 pts.)

Set hyperparameters in function `task1_simple_disparity()` to get the best performance. You can try different window sizes, disparity ranges, and matching functions. The comparison of your generated disparity maps and the ground truth maps can be visualized (or saved) by calling function `visualize_disparity_map()`.

After finishing the implementation, you can run `python main.py --tasks 1` to generate disparity maps with different settings and save them in the **`output`** folder.

According to the comparison of your disparity maps and ground truth maps under different settings, report and discuss

- How does the running time depend on window size, disparity range, and matching function?
- Which window size works the best for different matching functions?
- What is the maximum disparity range that makes sense for the given stereo pair?
- Which matching function may work better for the given stereo pair?

With the results above

- Discuss the trade-offs between different hyperparameters on quality and time.
- Choose the best hyperparameters and show the corresponding disparity map.
- Compare the best disparity map with the ground truth map, discuss the differences and limitations of basic stereo matching.

2 Depth from Disparity (25 pts.)

2.1 Pointcloud Visualization (20 pts.)

Implement `task2_compute_depth_map()` to convert a disparity map to a depth map, and `task2_visualize_pointcloud()` to save the depth map as pointcloud in *ply* format for visualization (recommended using MeshLab).

For depth map computation, follow the **Depth from Disparity** part in slides 08. You should try to estimate proper depth scaling constants **`baseline`** and **`focal_length`** to get a better performance. The depth of a pixel $\mathbf{p}$ can be formulated as:

$$\text{depth}(\mathbf{p}) = \frac{\text{focal\_length} \times \text{baseline}}{\text{disparity}(\mathbf{p})} \tag{1}$$

For pointcloud conversion, the x and y coordinates of a point should match pixel coordinates in the reference image, and the z coordinate shoule be set to the depth value. You should also set the color of the points to the color of the corresponding pixels the reference image. For better performance, you may need to exclude some outliers in the pointcloud.

After finishing the implementation, you can run `python main.py --tasks 02` to generate a *ply* file using the disparity map generated with **`cv2.StereoBM`**, saved in the **`output`** folder.

By modifying the settings of the hyperparameters in `task1_simple_disparity()` and running `python main.py --tasks 12`, you can generate pointclouds with your implemented stereo matching algorithm under different settings and they will be saved in the **`output`** folder.

2.2 Report (5 pts.)

Include in your report and compare the results of the pointclouds generated with

- disparity map computed using `cv2.StereoBM`
- disparity map computed using your implemented algorithm under **optimal** settings you found in task 1.

3 Stereo Matching with Dynamic Programming (15 pts.)

3.1 Algorithm Implementation (10 pts.)

Incorporate non-local constraints into your algorithm to improve the quality of the disparity map. Specifically, you are to implement the function `task3_compute_disparity_map_dp()` with dynamic programming algorithms. You may refer to the **Stereo Matching with Dynamic Programming** section in lecture slides 08.

Note that you should also manage to record the **running time** of your code, which should be included in the report.

After finishing the implementation, you can run `python main.py --tasks 3` to generate the disparity map and save it in the `output` folder.

You can also run `python main.py --tasks 23` to simultaneously generate pointclouds.

3.2 Report (5 pts.)

Report the running time, the disparity map, and the pointcloud generated with dynamic programming algorithm. Compare the results with basic stereo matching algorithm.

Submission Requirements

- Due date of this homework is **November 27, by 23:59**. Late submission is acceptable but with a penalty of 10% per day.
- Zip your code, report, and all the visualization results (including disparity maps and the pointclouds) into a single file named `StuID_YourName_HW4.zip`. A wrong naming format may lead to a penalty of 10%. Make sure that the zip file can be unzipped under Windows.
- For the code, it should run without errors and can reproduce the results in your report. If you use artificial intelligence tools to help generate codes, explain in your report of (1) how you use them, and (2) the details of implementation in your own words. If your code simultaneously (1) is suspected to be generated by AI tools, and (2) cannot run properly, you may get a penalty of 100%.
- For the report, either Chinese or English is acceptable. Please submit a single PDF file, which can be exported from LaTeX, Word, MarkDown, or any other text editor. You may get a penalty of 10% if the file format is not correct.

Hints

Here are some supplemental materials:

- cv2.StereoBM: `https://docs.opencv.org/4.x/d9/dba/classcv_1_1StereoBM.html`
- cv2.StereoBeliefPropagation: `https://docs.opencv.org/4.x/de/d7a/classcv_1_1cuda_1_1StereoBeliefPropagation.html`