Stereo Vision

1 Disparity Maps

In this task, you will implement a simple algorithm to compute the disparity map from RGB images. The program runs the code on two images from the Middlebury Stereo Dataset. The algorithm to implement is a simple version of computing disparity maps followed by post-processing (filtering steps) to generate smooth disparity maps. The algorithm is implemented in the function `src/main.py` and consists of the following steps:

1. Loading a pair of stereo images; `src` and `dst`.
   - `src (left)`: image taken from initial camera position.
   - `dst (right)`: image taken after shifting the camera horizontally.
2. Computing the maximum horizontal `translation` (in pixels) between the `src` and `dst` images.
   - This establishes the upper bound for possible disparities.
3. Constructing a stack of all reasonable `disparity` hypotheses. From the maximum horizontal translation we can obtain the range of disparity values.
   - Each disparity value corresponds to a different disparity hypotheses. Each disparity hypothesis is associated with a depth plane.
   - Note: a disparity of zero assumes that the points are at infinity (or very far away; indicating no depth change).
4. We apply a 3D Gaussian filter on the stack of disparities from the above step to enforce a local coherence between location (x-axis) and disparity hypotheses (z-axis)
5. For each pixel, we choose the disparity that gives the minimum cost. Meaninng how well this pixel in `src` correponds to the pixel in the `dst`.
6. We then apply a median filter to enhance local consensus (similar to removing salt-pepper noise) from the disparity map.
7. Check consistency: given the `src` image and `disparity map`, we should be able to estimate `dst` image.

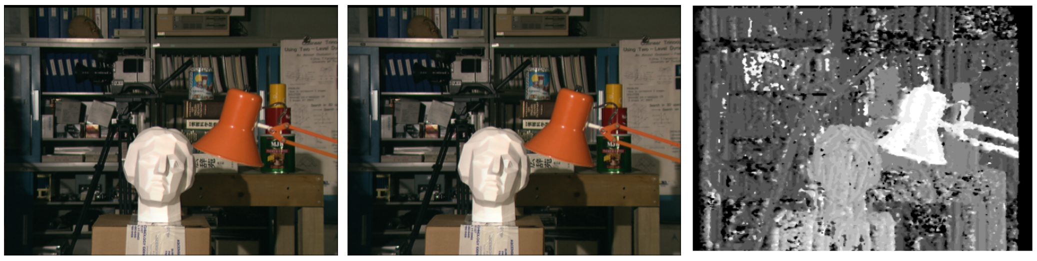

Figure 1: The two input images of the *Tsukuba* dataset and the disparity map computed in this task.

1.1 Estimating maximum translation

To start this exercise, we need to determine the maximum horizontal translation of the pixels from `src` to `dst`. This maximum translation will help us define the range of desparity values to consider when constructing the disparity map.

Utilize the `extract_features` function to extract feature points and descriptors from both `src` and `dst` images. Use `filter_and_align_descriptors` function to filter out irrelevant descriptors and align the remaining ones. Both fucnctions are in `utils/functions.py`. This will give us well-aligned feature points between the two images.

With the aligned feature points from the previous step, calculate the horizontal translation fro each pair of aligned points. Identify the furthest translation (in pixels) by determining the maximum absolute value among translations.

A disparity of zero indicates points at infinity (very far away with no depth change). Other disparity values should lie between 0 and the maximum translation value obtained in previous step. If the `src image` is to the right of the `dst image`, then all the translations should be positive and otherwise negative. Ensure the resulting translation value is rounded to the nearest integer to get the greatest absolute value.

Implement the function `get_max_translation` in `utils/functions.py` to perform the above steps. Make sure to validate your implementationby running the provided test cases before proceeding to the next task.

1.2 Render disparity hypothesis

To render a disparity hypotheses between a pair of images, you need to calculate how well the shifted version of the `src` image matches the `dst` image. The agreement is measured using Euclidean norm of the RGB values between the shifted `src` image and the `dst` image. The resulting map indicates the degree to which a given disparity hypothesis satisfies each pixel.

Implement the function `render_disparity_hypothesis` in `utils/functions.py` to complete this task.

1.3 Find the best/minimum disparity map

Several methods exist for estimating a disparity map from a pair of images. Steps 3-6 from the Disparity Maps algorithm describe a simple way of implementing it.

Implement `disparity_map` in `utils/functions.py` to realize this part of the exercise.

Additional Resources: If you wish to delve deeper into this topic and aim to generate improved disparity maps using advanced methods, here are additional resources that may be of interest. The `PatchMatch` paper and related resources provide valuable insights. These readings will enhance your understanding of disparity maps, noting that they differ from depth maps.

- Patch Match [Wikipedia], Patch Match [Paper], Patch Match [Video]
- Patch Match Stereo [Paper], Patch Match Stereo [Video]

Note: If you choose to implement an alternative algorithm (such as above), and diverge from the given exercise structure, the given test cases might be less helpful.

1.4 Bilinear interpolation

This part is identical to the Task 2.1 of the Exercise 4: `Optical Flow`. For more detailed description please refer to the Exercise 4.

To implement the function `bilinear_sample_grid` in `utils/functions.py`, you can follow these steps:

1. Estimate the left, top, right, bottom integer parts (l, r, t, b) and the corresponding coefficients (a, b, 1-a, 1-b) of each pixel
2. Take care of out-of-image coordinates
3. Produce a weighted sum of each rounded corner of the pixel
4. Accumulate and return all the weighted four corners

This function applies a sampling field of continuous (float) values on a `src` image to produce a `dst` image with the same size as the sampling field. The sampling field consists of a pair of x-axis and y-axis coordinates (each one with the same shape as `dst` image).