Division of Engineering Science
UNIVERSITY OF TORONTO

ROB501: Computer Vision for Robotics

Assignment #3: Stereo Correspondence Algorithms

Fall 2020

The Story Evolves

SpaceY has continued to gather venture capital in support of their rover mission to the red planet (Alon Mosk has decided to sell advertising space on the top of the rover to raise extra cash, too). Since the coffers are full, Mosk thinks that it's probably worthwhile to outfit the rover with a fancy stereo camera system, rather than just a monocular camera. However, before he commits the extra money for stereo, he would like you to benchmark the performance of some stereo correspondence algorithms, to determine whether it's possible to obtain reliable depth estimates.

Assignment Overview and Learning Objectives

Stereo vision is useful for a wide variety of robotics tasks, since stereo is able to provide dense range (depth) and appearance information. In order to accurately recover depth from stereo, however, a correspondence or *matching* problem must be solved (for every pixel, ideally). This matching process generates a disparity map, from a pixel to an inverse depth (we use the terms disparity map and disparity image interchangeably below). In this assignment, you will experiment with dense stereo matching algorithms. The goals are to:

- introduce stereo block matching and demonstrate the complexities involved in block processing; and
- provide some experience with more sophisticated methods for stereo depth estimation.

The due date for assignment submission is **Friday, November 6, 2020, by 11:59 p.m. EDT**. All submissions will be in Python 3 via Autolab; you may submit as many times as you wish until the deadline. To complete the assignment, you will need to review some material that goes beyond that discussed in the lectures—more details are provided below.

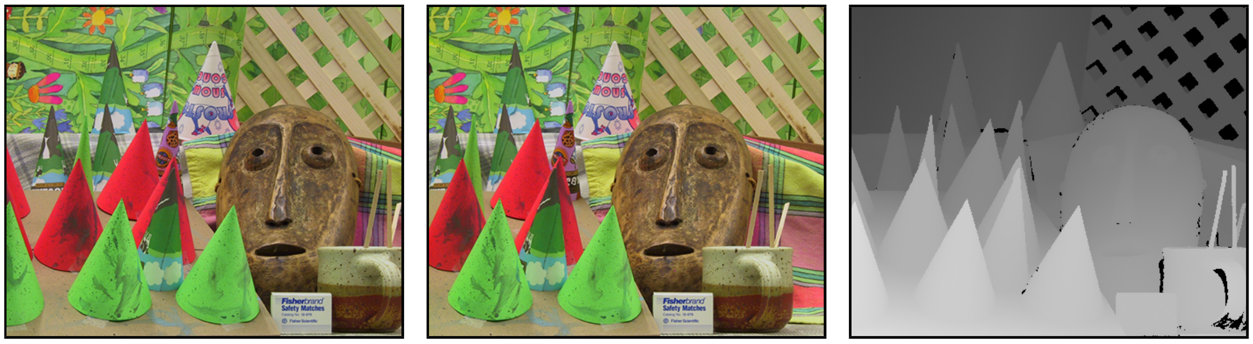

(a) (b) (c)

Figure 1: Stereo images from the Middlebury Cones dataset, acquired from the left (a) and right (b) cameras, and (c) the ground truth disparity image.

Stereo matching is easiest when the incoming stereo pairs are *rectified*, such that the epipolar lines coincide with the horizontal scanlines of each image. This fronto-parallel camera configuration reduces the matching problem to a 1D search. We will make use of the the Cones dataset (and others) from the Middlebury Stereo Vision repository and stereo images from the KITTI dataset.

Images from the Cones dataset (shown in Figure 1), which are already rectified, are provided for local testing. More details about the datasets are available on the respective dataset websites. We have also provided real stereo imagery gathered by the MER Spirit rover on Mars! You may optionally choose to try your algorithms on these images, although no ground truth is available, unfortunately (there were apparently no Martians around at the time of acquisition to make accurate measurements). The assignment has three parts, worth a total of **50 points**.

Please clearly comment your code and ensure that you only make use of the Python modules and functions listed at the top of the code templates. We will view and run your code.

Part 0: Your Secret Identifier

To make the assignment a bit more fun, we'll track the performance of each submission for Part 2 (see below). Once all submissions have been received, we'll generate a 'leaderboard' with the top scores (i.e., the lowest error)—bonus marks will be awarded to the top three leaderboard entries!

Each entry on the leaderboard will be associated with a 'secret ID,' which will be a simple string. You may pick your own secret ID (please be polite). For this portion of the assignment, you should submit:

- A single function in `secret_id.py` that returns your secret ID as a Python string. Please limit the length of the string to 32 characters.

Part 1: Fast Local Correspondence Algorithms

Your first task is to implement a fast local stereo correspondence algorithm. This should be a basic, fixed-support (i.e., fixed window size) matching routine, as discussed in Section 11.4 of the Szeliski text. You should use the sum-of-absolute-difference (SAD) similarity measure. 'Correct' matches can be identified using a simple winner-take-all strategy. For this portion of the assignment, you should submit:

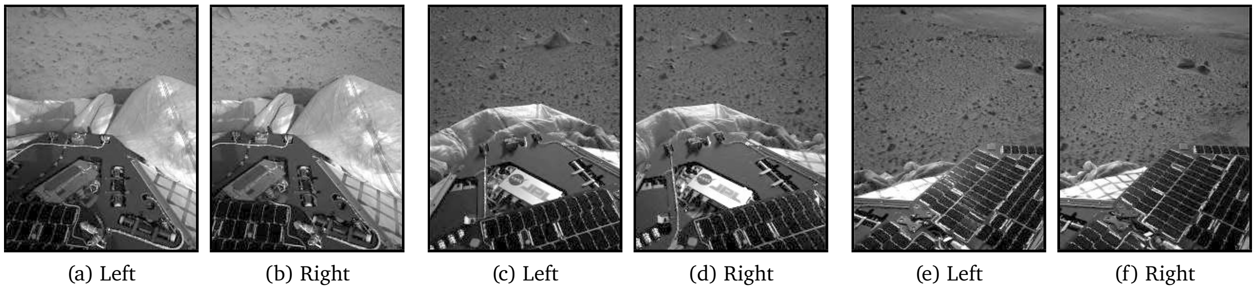
- A function in `stereo_disparity_fast.py`, which accepts an image pair (greyscale, e.g., two example images in the `stereo` directory) and produces a disparity image (map). The function should also accept a bounding box that indicates the valid overlap region (which we will supply), to avoid attempting to match points that are not visible in both images. Finally, the function should make use of a *maximum disparity* parameter (which we will again supply) to bound the 1D search.[1]

It will be important to adjust the window size used by the SAD algorithm (via some experimentation) to achieve the best performance. To determine how well your algorithm is doing, you may compare the disparity image your function generates with the ground truth disparity image. One possible performance metric is the RMS error between the disparity images; a Python function, `stereo_disparity_score.py`, is provided, which computes the RMS error between the estimated and true disparity images:

$$E_{\text{RMS}} = \left(\frac{1}{N} \sum_{u,v} \left(I_d(u,v) - I_t(u,v) \right)^2 \right)^{1/2}.$$

where N is the total number of valid (nonzero) pixels in the ground truth disparity image (inside the bounding box). We will score your submissions using the function above, and also by computing the number of correct disparity values (as a percentage—the scoring function also supplies this value).

[1]That is, your code should never search over a disparity range larger than the parameter called `maxd`—this will avoid wasted computation time.



(a) Left (b) Right (c) Left (d) Right (e) Left (f) Right

Figure 2: Actual stereo images acquired by the Mars Exploration Rover 'Spirit'.

Part 2: A Different Approach

Simple block algorithms offer reasonable performance and run quickly. A wide range of more sophisticated algorithms for stereo exist, however, and most offer better performance. After testing your simple block matching function, your second task is to select and implement an alternative matching algorithm. You may wish to refer to the paper by Scharstein (2002) from the course reading list for possible algorithm choices.

You may choose to implement a different local matching technique or a method that makes use of some global information. We will not grade based on the complexity of your source code (i.e., you need *not* choose a exceedingly sophisticated algorithm). However, a portion of your assigned mark will depend on the error scores you are able to achieve overall (see Part 1). You should aim to consistently exceed the performance of the local matcher from Part 1. For this portion of the assignment, you should submit:

- A function in `stereo_disparity_best.py`, which accepts an image pair (greyscale) and produces a disparity image (map). The function should also accept a bounding box indicating the valid overlap region (which we will supply) and a maximum disparity parameter (which we will also supply).

Note that there will be a **runtime limit in terms of CPU time on AWS** placed on your function for each test—it *must* successfully return the disparity image within this time. The runtime limit is printed at the start of each test.

Please include a short comment section (10-12 short lines) at the top of the function that describes the matching algorithm you implemented. If you referenced a paper or another source, please identify the source in the comment section.

Grading

Points for each portion of the assignment will be determined as follows:

- Fast local stereo correspondence function – **20 points** (3 tests; 6 points, 7 points, and 7 points)

  The local correspondence algorithm should achieve $E_{\text{RMS}} < 8$ and $p_{\text{bad}} < 0.17$ (incorrect disparities divided by total number of valid disparities) for both the Cones dataset and Teddy dataset (the first and second tests). For the KITTI images (the third test), the algorithm should achieve $E_{\text{RMS}} < 23$ (the disparity values in KITTI are larger) and $p_{\text{bad}} < 0.25$.

- Improved stereo correspondence function – **30 points** (3 tests $\times$ 10 points per test)

  The improved algorithm should achieve better performance than the algorithm from Part 1. Aim for $E_{\text{RMS}} < 5$ and $p_{\text{bad}} < 0.11$ for both the Cones dataset and Teddy datasets. For the KITTI images, aim for $E_{\text{RMS}} < 15$ and $p_{\text{bad}} < 0.15$ (this last goal is a stretch goal and might be harder to achieve).

Total: **50 points**

Bonus: Additional bonus points will be awarded for the top three submissions that produce the 'best' disparity maps for a holdout stereo pair. For the holdout test, we will weight E_{RMS} and p_{bad} equally. There will be 5 bonus points (out of 50, so a maximum score of 55/50) for first place, 3 bonus points for second place, and 2 bonus points for third place. The leaderboard will appear on Quercus.

Grading criteria include: correctness and succinctness of the implementation of support functions, proper overall program operation and code commenting and details, and a correct composite image output (subject to some variation). Please note that we will test your code *and it must run successfully*. Code that is not properly commented or that looks like 'spaghetti' may result in an overall deduction of up to **15%** in case (so be sure to include your algorithm description).