Division of Engineering Science
UNIVERSITY OF TORONTO

ROB501: Computer Vision for Robotics

Project #3: Stereo Correspondence Algorithms

Fall 2019

Overview

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 project, 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 basic experience with more sophisticated stereo depth estimation.

The due date for project submission is **Friday, November 8, 2019, by 11:59 p.m. EDT**. All submissions will be in Python 3 via Autolab (more details will be provided in class and on Quercus); you may submit as many times as you wish until the deadline. To complete the project, 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. Images from the Cones dataset (shown in the figure above), which are already rectified, are provided. More details about the dataset are available on the Middlebury Stereo Vision webpage. The project 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 project 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, `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. Use the encoding and search range defined on the Middlebury dataset page (i.e., there is a maximum possible disparity of **63 pixels**). For this portion of the assignment, you should submit:

- A function, `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 indicating the valid overlap region (which we will supply), to avoid attempting to match points that are not visible in both images.

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).

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, `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), to avoid attempting to match points that are not visible in both images.

There will be a runtime limit of **300 seconds** placed on your function—it must successfully return the disparity image within this time.

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 project will be assigned as follows:

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

  The local correspondence algorithm should achieve $E_{\text{RMS}} < 10$ and $p_{\text{bad}} < 0.20$ (incorrect disparities divided by total number of valid disparities) for both the Cones dataset and Teddy dataset (the first and second tests). For the Books dataset (the third test), the algorithm should achieve $E_{\text{RMS}} < 15$; we will not score the bad disparities in this case (some datasets are easier than others)

- 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 $E_{\text{RMS}} < 6$ and $p_{\text{bad}} < 0.12$ for both the Cones dataset and Teddy datasets. For the Books dataset, aim for $E_{\text{RMS}} < 12$.

Total: **50 points**

Bonus: Additional bonus points will be awarded for the top three submissions that produce the 'best' disparity maps for a holdout image pair. For the holdout test, we will weight E_{RMS} and p_{bad} equally. There will be 5 bonus points for first place, 3 points for second place, and 2 points for third place.

Grading criteria include: correctness and succinctness of the implementation of support functions, proper overall program operation, and code commenting. 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 10%.