Division of Engineering Science
UNIVERSITY OF TORONTO

ROB501: Computer Vision for Robotics

Project #3: Stereo Correspondence Algorithms

Fall 2017

Overview

Stereo vision is useful for a very 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 in the stereo image pair (potentially). This matching process generates a disparity map, from pixel to 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 some of the complexities involved, and
- provide basic experience with point clouds.

The due date for project submission is **Friday, November 10, 2017, by 11:59 p.m. EDT**. Submission instructions will be posted on Portal prior to this date. To complete the project, you will need to review some material that goes beyond that discussed in the lectures—more details are provided below.

Part 1: Fast Local Correspondence Algorithms

(a) (b) (c)

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, from the Middlebury Stereo Vision repository. Images from the dataset (shown in the figure above), which are already rectified, have been provided in the project archive. Details about the dataset are provided on the Middlebury Stereo Vision page.

Your first task is to implement a fast local stereo correspondence technique. This should be a basic, fixed-support (i.e., fixed window size) matching algorithm, as discussed in Section 11.4 of the Szeliski text. You may choose from the sum-of-absolute-difference (SAD) or sum-of-squared-different (SSD) similarity measures.

'Correct' matches can be identified using a simple winner-take-all strategy. Use the same encoding and search range defined on the Middlebury dataset page (i.e., a maximum disparity of 63 pixels). This portion of the assignment is worth **40 points** (out of 100 points total).

For this portion of the assignment, you should submit:

1. a function `stereo_disparity_fast.m`, which accepts an image pair (grayscale or colour, e.g., the two example images in the `images` 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.

To determine how well your algorithm is doing, you may compare the disparity image your function generates with the ground truth disparity image for the Cones stereo example (also included). One possible performance metric is the RMS error between the images; a MATLAB function, `stereo_disparity_score.m`, is provided, which computes the RMS error between the estimated and true disparity images:

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

where N is the total number of valid pixels in the ground truth disparity image.

Please clearly comment your code, and ensure that you use the filenames/function stubs specified exactly. We will run your code.

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 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 choices. This portion of the assignment is worth **60 points** (out of 100 points total).

You may choose to implement a different local matching technique, or a method that makes use of 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 RMS error score you are able to achieve overall—you should aim to consistently exceed the performance of the local matcher from Part 1.

For this portion of the assignment, you should submit:

1. a function `stereo_disparity_best.m`, which accepts an image pair (grayscale or colour) 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. You should implement your algorithm inside this function,
2. a very short (1/2 page maximum) `readme.pdf` document (PDF format only, with this exact filename please) that explains the matching algorithm you implemented.

To make the project a bit more fun, you should also include a files, `codename.txt`, that contains a short, plain text (ASCII) codename for your submission. We will assign bonus points to the top 10 submissions (based on the RMS error a hold-out image pair), and post the ranking in class.

Please clearly comment your code, and ensure that you use the filenames/function stubs specified exactly. We will run your code.

A Helpful Utility for Point Cloud Rendering

You should be able to produce reasonable disparity maps from pairs of stereo images. The point of stereo processing (pun intended), ultimately, is not just to generate disparity data, but to use this data to compute a 3D representation of the world. Every valid pixel in the disparity image represents, through the simple transform described in Lecture 11, the depth of a 3D point in the stereo rig reference frame. Given known camera intrinsic parameters and the baseline of the stereo system, the 3D coordinates of the set of points can be determined. The resulting data structure is called a *point cloud*; many tools exist to manipulate point clouds (there is an entire library, in fact, called PCL).

To assist in visualizing the 3D point clouds, we have included an additional function, `plot_point_cloud.m`, that accepts a left stereo image, a disparity image, a camera intrinsic parameter matrix (which we assume is the same for both cameras), and a baseline value (distance between the camera optical centres), and produces a 3D rendering in the MATLAB figure window. Note this process may be fairly slow (since many small patches are being plotted). Feel free to try out the test script `point_cloud_example.m`.

Grading

Points for each portion of the project will be assigned as follows:

Part 1: Fast Local Correspondence Algorithms

- Basic block matching function – **35 points**
- Reasonable RMS error score – **5 points**

Total: **40 points**

Part 2: A Different Approach

- Improved stereo matching function (and `readme.pdf` document) – **55 points**
- Stereo matching performance – **5 points**
- Leader board entry – up to **10 bonus points!**

Total: **60 points** (+ **bonus**)

The total number of points for the assignment is **100 points** (plus 10 bonus points possible). Grading criteria include: correctness and succinctness of the implementation of support functions, proper overall program operation, and correct stereo results. Please also note that we will test your code *and it must run successfully*, so please do not forget to include all required program files in your submission.