Division of Engineering Science
UNIVERSITY OF TORONTO

ROB501: Computer Vision for Robotics

Project #3: Stereo Correspondence Algorithms

Fall 2016

Overview

Stereo vision is useful for a very wide variety of robotics tasks, as 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.

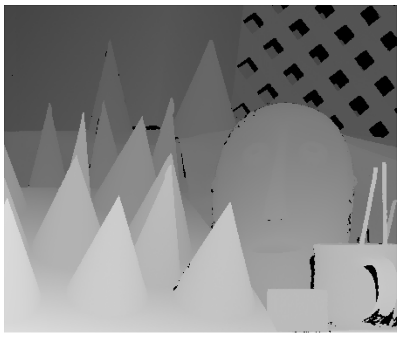
The due date for project submission is **Friday, November 25, 2016, by 11:59 p.m. EST**. Submission instructions will be posted on Portal prior to this date. To complete the project, you will need to review some material that goes beyond that discussed in the lectures—more details are provided below.

We have provided a series of (empty) MATLAB files that specify the inputs and outputs of the functions you need to write. **Please clearly comment your code and ensure that the filenames/function stubs specified are used, as this will help to speed up marking. We will run your code.**

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), sum-of-squared-different (SSD), or normalized cross-correlation similarity measures. 'Correct' matches can be identified using a simple winner-take-all strategy. Code should be written in the file `stereo_disparity_a.m`. Your MATLAB function should take in a stereo image pair (e.g., the two example images in the `cones` directory), and produce a disparity image (map). 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).

To determine how well your algorithm is doing, you can 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:

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

In your project submission, please include a `readme.txt` file (plain text format), which identifies which algorithm you implemented, and the RMS error score you obtained for the Cones image pair.

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. Code should be written in the file `stereo_disparity_b.m`. This portion of the assignment is worth **40 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—you should aim to exceed the performance of the local matcher from Part 1. In your `readme.txt` file, please provide (brief) details on your algorithm choice, and the best RMS error score you were able to obtain.

Part 3: Point Cloud Rendering

You should now 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, PCL).

Your final task is implement a function, `plot_point_cloud.m`, that accepts left and right stereo images, 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 point cloud. In your submission, you should also include a plot of the cloud generated using the Cones images, from a suitable angle. Please name the plot file `rendered_cloud.png` (PNG format). This portion of the assignment is worth **20 points** (out of 100 points total).

Grading

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

Part 1: Calibrating a Camera for Robotics Applications

- Basic block matching function (and `readme.txt` documentation) – **35 points**
- Reasonable RMS error score – **5 points**

Total: **40 points**

Part 2: A Different Approach

- Advanced stereo matching function (and `readme.txt` documentation) – **35 points**
- Stereo matching performance – **5 points**

Total: **40 points**

Part 3: Part 3: Point Cloud Rendering

- Point cloud rendering function – **15 points**
- Image of rendered point cloud data – **5 points**

Total: **20 points**

The total number of points for the assignment is **100 points**. 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.