IM490 Depth-Map Recovery from RGB-video

Project 1 Report

Giorgio Mariani

4th January, 2019

Abstract

Contents

Chapter 1

Aims and Context

1.1 Introduction

The aim of this project is the implementation of the article *"Consistent Depth Map Recovery from a Video Sequence"* [3], by Zhang, Jia, Wong, and Bao and published in *Transaction on Pattern Analysis and Machine Intelligence* (TPAMI). The problem faced by Zhang et al. consists in the estimation of a sequence of *depth-maps*[1] starting from an RGB-video. More precisely, given an image sequence representing a video of a static scene, the aim of the paper is to estimate a sequence of images with consistent depth values for each pixel and frame. This depth values can then be used for a variety of tasks, such as scene reconstruction and layer separation.

Figure 1.1: One of the estimated depth-map obtained using [3] over a sequence of 200 images.

While a free implementation is available on the authors' website [2], they do not offer its source code; hence, the reasons behind this project are to offer an open-source implementation of said article, and to offer didactic improvement for the project's author.

1.2 Proposed Approach

The approach proposed in the original article consists of four steps:

[1] A *depth-map* is an image in which each pixel maintains depth information.

1. Camera Parameter Estimation
2. Depth-maps Initialization
3. Depth-maps Bundle Optimization
4. Depth-maps Space-Time Fusion

Camera parameter estimation: During this phase, the camera parameters (i.e. *position*, *rotation*, and *intrinsic matrix*) are estimated, using a *Structure From Motion* (STM) algorithm; unsurprisingly, the one used in the article is the same as the one described in [4], with Zhang and Bao two of the article's authors.

Depth-maps initialization: For each frame in the video a raw estimation of the corresponding depth-map is computed; this is achieved by minimizing a (per-frame) energy function whose parameters are the depth values to estimate.

These raw depth-maps are successively processed using a plane fitting algorithm; each image is divided into segments using *mean-shift color segmentation* [1]. Each segment is then considered as a plane and fitted to the existing depth-maps raw values. The resulting images are the initialized depth-maps, ready to be used in the next step.

Depth-maps bundle optimization: The depth-maps obtained during the initialization process are then refined by applying geometric constraints. This refinement process iteratively elaborates the given raw data by minimizing an energy function similar to the one used in the previous step, but enriched with geometric constraints over the initialized depth-maps values (that is, the depth values between frames should be consistent).

Depth-maps space-time fusion: This is the final step of the process, and it is used to polish the results obtained from the previous steps and removing eventual remaining noise.

The estimated depth-map values are used to define a loss function that can subsequently be optimized through the use of *Coniugate Gradient Method*, an optimization technique similar to *Gradient Descent*. The designed loss function takes into consideration *spatial continuity* between already computed depth-values, as well as temporal coherence and the sparse points that can be obtained by the SFM algorithm used to compute the camera parameters.

1.3 Related Work

Since the proposed approach makes use of a variety of techniques, and it is relatively old (2009), the related work section is organized as follows:

1. Previous work related to individual algorithms used in the proposed method.
2. Work that makes use or improves the proposed solution.
3. More recent work that uses a neural network approach.

Chapter 2

Project Details

2.1 The Architecture

The systems is written using the *Python Programming Language*; the reason behind this decision is the relative good performance that vectorized computation libraries can achieve on python, the flexibility and simplicity of the language, and the author's past experience with python libraries such as *NumPy* and **TensorFlow** makes the development less laborious.

2.1.1 Dependencies and Third Party Libraries

NumPy: The main library used for computations in the project is the **NumPy** module: it is set of function designed for intensive computing in vectorized fashion, similarly to the **MATLAB** programming language.

OpenCV: The **OpenCV** library (version 4.0.0) is also utilized in the project, using the official python bindings. **OpenCV** offers real-time computer vision by exploiting (whenever possible) accelerated hardware, such as GPUs. Unfortunately, the official bindings are available only for python $2.7.x$ interpreter, forcing the author to use a compatible interpreter.

2.2 Approach in Detail

2.2.1 Initialization Phase

During the initialization phase, an initial depth-map is estimated for each frame in the input video; this estimation occurs in a two steps process: firstly, the depth-maps are initialized using Loopy Belief Propagation (LBP) [0], thereafter, *mean-shift segmentation* and plain fitting are used to refined the obtained results.

The video's image sequence is denoted by the sequence $I_0, I_1 ... I_m$, with I_t representing the image at time t. The camera parameters are denoted with K, R, and T, which are respectively *intrinsic matrix*, *Rotation*, and *Position*.

LBP is a dynamic programming algorithm that can be used to find good approximations for energy minimization problems defined over labeled graphs (see appendix A.1 for more information).

The LBP algorithm requires the following inputs: a graph with vertices $\mathcal{V}$ and edges $\mathcal{N}$, a set D of possible vertex labels, a function V that maps label pairs to real values and an array U_x (for each vertex x) mapping labels to real values. The algorithms then finds an label assignment d_x, for each vertex x in the graph, such that the energy function A.1 is locally minimized.

Suppose to want to estimate the depth-map for an image I in the sequence $I_0...I_m$, using LBP; in such case, the graph nodes (the set $\mathcal{V}$) are the image's pixels, furthermore, each node has as neighborhood the (at most) four adjacent pixels. The labels (set D) represents the depth values that should be estimated. the edge function V can be defined as

$$V(d_x, d_y) = \lambda(x, y) \cdot \min\{|d_x - d_y|, \eta\}$$

with η a constant value and λ a smoothness weight proportional to the disparity in color of an edge (i.e. the more the $I(x)$ is distant from $I(y)$ the smaller $\lambda(x, y)$ is). This definition for V entice the difference between adjacent to not be more smooth, especially if said adjacent vertices are similar in color, and it is compliant with the LBP requirements[1].

Finally, each $U_x(d)$ value is computed by projecting the pixel position for a particular time instant t, using the depth-label d and the camera parameters. The difference between the original pixel position color and the projected position color at time t is then used to describe the likeness that d is an acceptable depth-value for the pixel x. Summing and normalizing these differences for all time instants t gives the actual value $U_x(d)$.

$$L_x(d) = \sum_t \frac{\sigma_c}{\sigma_c + ||I(x) - I_t(\overrightarrow{x}^t(d))||} \tag{2.1}$$

$$U_x(d) = 1 - u(x) \cdot L_x(d) \tag{2.2}$$

where σ_c is a constant value, $\overrightarrow{x}^t(d)$ is the projection of pixel x using the camera parameter at time t and depth d, and $u(x)$ is a normalization factor such that

$$\max_{d \in D}\{u(x) \cdot L_x(d)\} = 1$$

2.2.2 Bundle Optimization Phase

[TODO]

2.2.3 Space-Time Fusion

[TODO]

2.3 Lessons Learned

[1] Only a specific class of edge functions can be used for LBP.

Chapter 3

System Documentation

3.1 Dependencies Installation

3.2 Running the System

The system can be executed by executing the following command from the operating system's command line terminal:

```
python estimate.py [-i][-b] <config.txt>
```

with `<config.txt>` a configuration file containing parameter values and paths to the folder storing the data necessary for the estimation process (i.e. either pictures or depth-maps). The options `-i` and `-b` are used to communicate the system which steps should be executed:

3.2.1 Special Folders and Files

Different folders and files are used by the system during depth estimation; these are divided into three types: *configuration files*, *picture-folders*, and finally *depth-folders*.

Configuration File. The configuration file is used to set the parameter values to be used while running the system, it contains key-value pairs (with syntax <*key*>=<*value*>). Each of these pairs must be interleaved by a newline character to the next one.

These pairs specify the parameters values used in the various algorithms, as well as more practical information, like the directories storing the input image sequence or the output depth-maps. Generally speaking, this file contains all the information necessary for the system in order to work properly. Most of these parameters have default values,so they are not required to be present in the configuration file, that is, with the exception of the first few lines: indeed, the first three lines in the file must set the values for the keys `picture_folder`, `depth_folder_input` (whose value is not used if the initialization step is being executed), and `depth_folder_output`, which will contain the estimated depth-maps.

3.3 Python Modules

A variety of custom modules were created for the project, an exhaustive list is the following:

`compute_energy`: This module contains all functions necessary for the energy computation of a single frame.

`estimate`: This module is used to estimate the depth-maps for a sequence of images. To do so, it makes use of the module **compute_energy**.

`lbp`: It contains an implementation of the *Loopy Belief Propagation* algorithm.

`utils`: This module contains a variety of class that are used by the other python modules, as well as debugging and visualization procedures.

`params`: It contains parameters to be used by the system and the various algorithms.

3.3.1 `compute_energy`

3.3.2 `estimate`

3.3.3 `lbp`

3.3.4 `utils`

3.3.5 `params`

Chapter 4

Summary

Appendix A

Supplementary Materials

A.1 Loopy Belief Propagation Algorithm

Loopy Belief Propagation (LBP) [0] is a dynamic programming algorithm, which can be used to calculate approximate solution for energy minimization problems defined over labeled graphs.

LBP is a specialization of the Belief Propagation (BP) algorithm used for *marginal distribution* approximation of *Markov Random Fields*. LBP is able to achieve better performance than regular BP by making assumptions on the structure of the input graph and energy function. Inputs of the LBP algorithm are:

- A grid graph $\mathcal{G}$ with vertices $\mathcal{V}$ and edges $\mathcal{N}$. This graph is usually used to represent an image; each node is a pixel, and each vertex is connected to the (at most four) adjacent pixels.
- An arbitrary finite set of labels D. This label set is usually $\{0, ..., n\}$.
- A function V that associates pairs of labels to values in $\mathbb{R}$.
- Two-dimensional table U that associates nodes and labels to values in $\mathbb{R}$. The notation $U_x(d) = \alpha$ with $x \in \mathcal{V}$, $d \in D$, and $\alpha \in \mathbb{R}$ is used to indicate table elements.

The algorithm can then be used to compute a mapping (denoted as d_x) from vertices to labels, such that the following energy function is locally minimized:

$$E(D) = \sum_{(x,y)\in\mathcal{N}} V(d_x, d_y) + \sum_{x\in\mathcal{V}} U_x(d_x) \tag{A.1}$$

References

[1] Dorin Comaniciu and Peter Meer. "Mean Shift: A Robust Approach Toward Feature Space Analysis". *IEEE Trans. Pattern Anal. Mach. Intell.* 24.5 (2002), pp. 603–619. URL: https://doi.org/10.1109/34.1000236 (cit. on p. 4).

[2] *Computer Vision Group (Zhejiang University) website*. URL: http://www.zjucvg.net (cit. on p. 3).

[0] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. "Efficient Belief Propagation for Early Vision". *International Journal of Computer Vision* 70.1 (2006), pp. 41–54 (cit. on pp. 6, 11).

[3] Guofeng Zhang et al. "Consistent Depth Maps Recovery from a Video Sequence". *IEEE Trans. Pattern Anal. Mach. Intell.* 31.6 (2009), pp. 974–988. URL: https://doi.org/10.1109/TPAMI.2009.52 (cit. on p. 3).

[4] Guofeng Zhang et al. "Robust Metric Reconstruction from Challenging Video Sequences". In: *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. 2007. URL: https://doi.org/10.1109/CVPR.2007.383118 (cit. on p. 4).