IM490 Depth-Map Recovery from RGB-video

Project 1 Report

Giorgio Mariani

9th 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"* [4], 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 [4] 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* (SFM) algorithm; unsurprisingly, the one used in the article is the same as the one described in [5], with Zhang and Bao two of the article's authors.

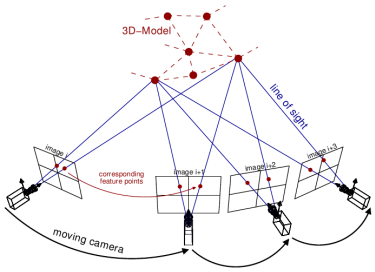

Figure 1.2: SFM approach visualization

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 coherence 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.

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 *Conjugate Gradient Method*, an optimization technique similar to *Gradient Descent*. The designed loss function takes into consideration: *spatial continuity* between already computed depth-values, depth-maps temporal coherence, and consistency with the sparse points 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.

1.3.1 Previous Work

[TODO]

1.3.2 Future Work

[TODO]

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**, which 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 a 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 interpreters, forcing the author to use a compatible interpreter.

2.2 Approach in Detail

The goal of the system is to estimate a sequence of depth-maps $\hat{D} = D_0, \dots D_n$, using a sequence of pictures $\hat{I} = I_0, \dots I_n$ and camera paramet,ers: $\{R_i\}_{i=0..n}$, $\{T_i\}_{i=0..n}$, and K (which are respectively *Rotation*, *Position*, and *intrinsic matrix* of the camera.)

The estimated pixel x's depth/disparity[1] at time t is noted with $D_t(x)$ (d_x for short); the admissible disparity values are taken from the set $[d_{min}, d_{max}]$, consequently to its quantization into m uniformly spaced values $d_{min} = d_0, \dots, d_{m-1} = d_{max}$. This quantization is necessary in order to execute the algorithms utilized by the system (especially true for *Belief Propagation*, since it works over labels, not values in $\mathbb{R}$).

[1] The disparity of a certain pixel x correspond to the reciprocal of the pixel's depth ($\frac{1}{z_x}$), however, the terms *depth* and *disparity* are sometimes used interchangeably.

2.2.1 Initialization Phase

During the initialization phase, for each frame in the input video, an initial depth-map is estimated; this estimation occurs with a two-step process: firstly, the depth-maps are initialized using a multi-view photo-consistency method and *Loopy Belief Propagation* (LBP) [3]. Then, *mean-shift segmentation* and a plane fitting algorithm are used to refine the obtained results.

The initial depth-maps estimation process works by minimizing the energy function E_{init}, which is defined as

$$E_{init}(\hat{D}) = \sum_t \left(E^t_{data}(D_t) + E_{smooth}(D_t) \right). \tag{2.1}$$

The variable t iterates over the video sequence frames, while the term E^t_{data} indicates how much photo-consistent is the input depth-map D_t. Finally, E^t_{smooth} indicates how smooth[2] the D_t depth-map is.

Energy Data Term. The term $E^t_{data}(D_t)$ is defined in terms of *disparity likelihood*, noted as L_{init} and defined as

$$L_{init}(x, d) = \sum_{t'} p_c(x, d, t, t').$$

This likelihood is used to the describe the photo-consistency of a certain disparity value d; furthermore, the function $p_c(x, d, t, t')$ describes how much the pixel x, using depth $1/d$, is photo-consistent. p_c is expressed as

$$p_c(x, d, t, t') = \frac{\sigma_c}{\sigma_c + \|I_t(x) - I_{t'}(I_{t,t'}(x, d))\|},$$

with σ_c a constant value and $I_{t,t'}(x, d)$ the projection of pixel x (taken from I_t) at time t', using disparity d. Finally,

$$E^t_{data}(D_t) = \sum_x 1 - u(x) \cdot L_{init}(x, D_t(x)), \tag{2.2}$$

with $u(x)$ a normalization factor, such that the maximum value of the likelihood is 1; more precisely, it is true that $\max_d \{u(x) \cdot L_x(d)\} = 1$ (i.e. the normalization is applied only with respect of the disparity value d and not the pixel x).

Energy Smoothness Term. The smoothness term $E_{smooth}(D_t)$ is used to impose a smoother gradient during estimation. This smoothness imposing strategy assigns an higher cost if two adjacent pixels have starkly different disparity-labels. This label difference is measured using the function

$$\rho(d_x, d_y) = \min\{|d_x - d_y|, \eta\}.$$

The term η is a real constant positive value, and it represents the upper limit of this smoothness imposing approach: after η the distance between two labels does not matter during the energy minimization.

[2] That is, how much difference there is between adjacent disparities.

The value $\rho(\cdot)$ is then weighted using an adaptive smoothness weight $\lambda(x, y)$, which encodes changes of color between the adjacent pixels x, y: if x and y have strongly different colors in I_t then the disparity smoothness requirement should be less strict, since they are less likely to be in a contiguous three-dimensional area. This is expressed as

$$\lambda(x, y) = w_s \cdot \frac{u_y(x)}{||I_t(x) - I_t(y)|| + \epsilon},$$

with w_s a constant real positive value and $u_\lambda(x)$ a normalization factor

$$u_\lambda(x) = |N(x)| / \sum_{y' \in N(x)} \frac{1}{||I_t(x) - I_t(y')|| + \epsilon}.$$

The smoothness term definition is then

$$E_{smooth}(D_t) = \sum_x \sum_{y \in N(x)} \lambda(x, y) \cdot \rho(D_t(x), D_t(y)). \tag{2.3}$$

Minimization. E_{init} is then minimized using LBP, process which requires a-priori the computation of an $h \times w \times m$ bi-dimensional table that stores the value

$$1 - u(x) \cdot L_{init}(x, d)$$

for each possible disparity-values d and pixel x. To see how this table is computed look at section 2.2.2.

Color Segmentation Refinement. Finally, using *mean-shift segmentation*, it is possible to divide an image I_t into different segments, which are then fitted to the initial estimation D_t:

1. First, the depth of the plane is selected by using the disparity that minimizes eq. (2.1). The slope is assumed to be 0 during the fitting process.
2. Then, by using *Levenberg-Marquardt algorithm*, the planes' slopes are estimated.

These output planes represent the new refined depth-map, and are ready to be processed by the next phase of the algorithm.

2.2.2 Initialization Phase - Algorithm

In order to minimize the energy function described in eq. (2.1), the LBP algorithm (see appendix A.1 for more information) is used. LBP is a dynamic programming algorithm which can be used to find good approximations for energy minimization problems defined over labeled graphs. For the depth-map recovery problem, the image is expressed as a grid graph (with each node representing a pixel and having at most four adjacent nodes), and the possible labels are the disparity values $d_0 \dots d_{m-1}$. Therefore, an assignment of these labels over the graph can be thought of as a depth-map.

In order to work, the LBP algorithm requires some data, specifically, it requires, for each pixel x and disparity value d, the cost of assigning to x the disparity d: this is

expressed as $1 - u(x) \cdot L_{init}(x, d)$ (see eq. (2.2)). Thus, it is necessary to compute a table able to store such information. The computation of this table is performed by the function `compute_energy_data`, which evaluates $1 - u(x) \cdot L(d)$ for each possible pixel x and disparity value d.

Algorithm 2.1: `compute_energy_data_init` (camera, t, $\hat{I}$)

```
1:  h ← camera.height
2:  w ← camera.width
3:  I_t ← Î[t]
4:
5:  x^h ← homogeneous_coordinate_grid(h,w)
6:  create table L with m × h × w elements, initialized with zero
7:
8:  for t' ← 0 ... n do
9:      I_t' ← Î[t']
10:     for level ← 0 ... m do
11:         create matrix D with h × w elements, initialized with value d_level
12:         x'^h ← conjugate_coordinates(camera, t, t', x^h, D)
13:         I^r_t' ← remap(I_t', x'^h)
14:
15:         p_c ← σ_c / (σ_c + L2_norm(I_t − I^r_t'))
16:         L[level] ← L[level] + p_c
17:     end for
18: end for
19:
20: u ← 1 / max_reduce(L, 0)
21: return 1 − u · L
```

The procedure `homogeneous_coordinate_grid`(h,w) produces a multi-dimensional array x^h which stores a grid of homogeneous coordinates, such that the first axis represent the coordinate itself, i.e.

$$x^h[0, i, j] = i$$
$$x^h[1, i, j] = j$$
$$x^h[2, i, j] = 1$$

The coordinates assume integer values between 0 to h (excluded) for the second axis (that is, value i in above equation) and between 0 and w (excluded) for the third axis (value j). Consequently, x^h's shape is $3 \times h \times w$.

variable	shape
x^h, x'^h	$3 \times h \times w$
D	$h \times w$
I_t, $I_{t'}$, $I^p_{t'}$	$h \times w$
L	$m \times h \times w$
p_c, u	$h \times w$

Figure 2.1: Shapes of the variables in algorithm 2.1

The procedure `conjugate_coordinates`($camera$, t, t', x^h, D) computes, for each coordinate in x^h, the respective conjugate coordinate at time t if the depth stored in D is used. The implementation of this procedure is explained in appendix A.2.

The procedure $I' \leftarrow \texttt{remap}(I, map)$ is used to transform the input image I using a certain mapping m (which should have shape $h \times w \times 2$) such that

$$I'[x, y] = I[map[x, y]].$$

The reader should note that in algorithm 2.1 there is an abuse of notation, since the variable x'^h (which assumes role of the argument *map* in the function call) has shape $3 \times h \times w$ instead of the required $h \times w \times 2$; however, this can be easily solved by transforming x'^h to the non-homogeneous coordinates x' and then transposing it.

2.2.3 Bundle Optimization Phase

The bundle optimization phase is similar to the first step of the initialization phase; indeed, the previously estimated depth-maps are again refined using LBP, however, the energy function E to minimize is slightly different:

$$E(\hat{D}) = \sum_t \left(E_{data}(D_t, \hat{D}) + E_{smooth}(D_t) \right). \tag{2.4}$$

The term E_{smooth} is the same as in eq. (2.1), in contrast with $E_{data}(D_t, \hat{D})$, which substitutes $E^t_{init}(D_t)$, and requires the sequence of initial depth-maps ($\hat{D}$). These are then used to define geometry coherence constraints, which in turn will allow the estimation of more coherent and realistic disparity values.

The $E_{data}(D_t, \hat{D})$ terms is defined similarly to $E^t_{data}(D_t)$, with only one major difference: the likelihood L_{init} term is replaced by L, which is defined as

$$L(x, d) = \sum_{t'} p_c(x, d, t, t') \cdot p_v(x, d, D_{t'})$$

with p_v the function used to impose geometric coherence:

$$p_v(x, d, D_{t'}) = \exp\left(-\frac{\|x - l_{t',t}(x', D_{t'}(x'))\|^2}{2\sigma_d^2} \right) \tag{2.5}$$

$$x' = l_{t,t'}(x, d)$$

Using this definition, p_v encodes the distance between the coordinates x and $l_{t',t}(x', D_{t'}(x'))$ using a gaussian distribution: the closer the two coordinates are, the higher p_v is going to be; if they are the same, then it means that the depth values d and $D_{t'}$ are geometrically coherent.

2.2.4 Bundle Optimization Phase - Algorithm

The implementation is similar to the one used for the first step of the initialization phase (section 2.2.1), one major difference consists in the computation of the term p_v, which is multiplied to p_c in before updating the likelihood L.

By adding the pseudo-code

$D^r_{t'} \leftarrow \texttt{remap}(D_{t'}, x'^h)$

$I_{t',t} \leftarrow$ `conjugate_coordinates`$(camera,\ t',\ t,\ x'^h,\ D^r_{t'})$
$p_v \leftarrow \exp\left(-\frac{\|x - I_{t',t}\|^2}{2\sigma_d^2}\right)$
in the inner for loop and by changing how L is updated, it is possible to add the bundle optimization phase to algorithm 2.1; the use of such optimization is managed by the boolean variable *use_bundle*. Thus, the final algorithm, able to exploit the term p_v, is described in algorithm 2.2. Note that the functions `homogeneous_coordinate_grid`, `conjugate_coordinates`, `remap`, `L2_norm`, and `max_reduce_dim0` work as described in section 2.2.2.

Algorithm 2.2: `compute_energy_data` $(camera,\ t,\ \hat{I},\ \hat{D})$

```
1:  h ← camera.height
2:  w ← camera.width
3:  I_t ← Î[t]
4:  use_bundle ← True if D̂ is not empty, False otherwise.
5:
6:  x^h ← homogeneous_coordinate_grid(h,w)
7:  create table L with m × h × w elements, initialized with zero
8:
9:  for t' ← 0 ... n do
10:     I_t' ← Î[t']
11:     for level ← 0 ... m do
12:         create matrix D with h × w elements, initialized with value d_level
13:         x'^h ← conjugate_coordinates(camera, t, t', x^h, D)
14:         I^r_t' ← remap(I_t', x'^h)
15:         p_c ← σ_c / (σ_c + L2_norm(I_t − I^r_t'))
16:
17:         if use_bundle then
18:             D^r_t' ← remap(D_t', x'^h)
19:             I_t',t ← conjugate_coordinates(camera, t', t, x'^h, D^r_t')
20:             p_v ← exp(−‖x − I_t',t‖² / (2σ_d²))
21:             L[level] ← L[level] + p_c · p_v
22:         else
23:             L[level] ← L[level] + p_c
24:         end if
25:     end for
26: end for
27:
28: u ← 1 / max_reduce_dim0(L)
29: return 1 − u · L
```

2.2.5 Space-Time Fusion

[TODO]

Chapter 3

System Documentation

3.1 Dependencies Installation

The system makes use of several external libraries, fortunately these can be easily installed by using `pip` and the `requirements.txt` file:

```
pip install -r requirements.txt
```

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:

`-i` it executes only the initialization phase and stores the output depth-maps in the output folder.

`-b` it executes the bundle optimization phase; it requires previously estimated depth-maps as input.

`-ib` it executes both the initialization and the bundle optimization phases.

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 few lines in the file must set the values for the keys: `picture_folder`, containing the input pictures, and `depth_folder_output`, which will store the estimated depth-maps.

Picture Folder. This folder contains the picture sequence to be used during the estimation process. The images must be named using the syntax `img_<num>`, with `<num>` numeric string which assumes values from 000 to 999. The images filenames must be organized in a contiguous fashion: no gaps should be present between the smallest and the largest filename numbers. All the image files should have format `.png` or `.jpg`, and same resolution.

Depth Folder. This folder contains the output depth-maps, stored as `.npy` files. This is standard binary format used by **NumPy** to store a single **NumPy** array. These files are named `depth_<num>`, as before, `<num>` can assume values between 000 and 999; said number indicates the image used to extract the depth-map.

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`

[TODO]

3.3.2 `estimate`

[TODO]

3.3.3 `lbp`

[TODO]

3.3.4 `utils`

[TODO]

3.3.5 `params`

[TODO]

Chapter 4

Summary

[TODO]

Appendix A

Supplementary Materials

A.1 Loopy Belief Propagation Algorithm

Loopy Belief Propagation (LBP) [3] is a dynamic programming algorithm, which can be used to calculate approximate solutions 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 can be 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:

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

A.2 Conjugate Pixel Computation

Epipolar geometry is branch of mathematics which focus on stereo vision; it can be used to compute the position of a pixel x with respect of different views using said pixel's disparity. This new pixel is called "the *conjugate* of x". Computation of the conjugate of x requires knowledge about x's disparity d_x (or depth) and use of both the *old* and *new* views. More precisely, their camera parameters[1] are necessary: K, T, R, and K', T', R' respectively. Thus, the conjugate of x, denoted as x', is computed as

$$x'^h = K'R'^T \cdot \left(RK^{-1}x^h + d_x \cdot (T - T')\right)$$

x^h denotes the homogeneous coordinates of x, while x'^h denotes the homogeneous coordinates of x'.

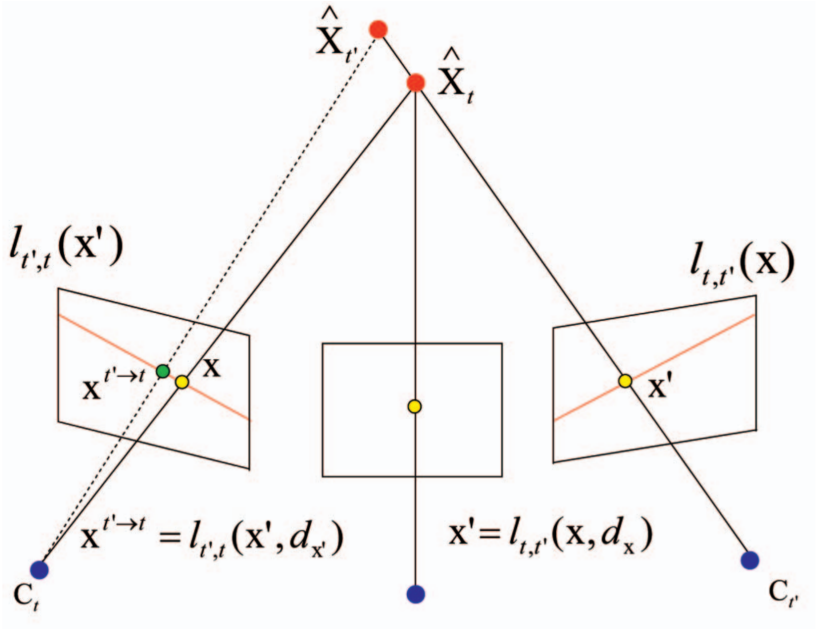

Figure A.1

[1] The camera parameters are respectively K for intrinsic matrix, T for position, and R for rotation.

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).

[3] 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. 7, 16).

[4] 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).

[5] 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).