IM490 Depth-Maps Recovery from an RGB-video
Project 1 Report

Giorgio Mariani

4th February, 2019

Abstract

This report details the partial implementation of the depth-maps estimation procedure described by Zhang et al. in [3], furthermore, an exhaustive explanation of said approach is given. More precisely, the addressed problem is the estimation of depth-maps from an RGB-video of a static scene. The original paper's main contribution is the expansion of a multi-view stereo approach with geometric coherence constraints, resulting in a method named *Bundle Optimization*. This addition allows the estimated depth-maps to be more stable when dealing with occlusion and disturbances in the source video. The implementation was mainly executed using the **Python** programming language, expanded with vectorized computation libraries such as **NumPy** and **OpenCV**; for the latter the official python bindings were used.

Contents

Chapter 1

Aims and Context

1.1 Introduction

The aim of this project is the implementation of the article *"Consistent Depth Maps 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 estimating the depth-maps[1] for all the frames contained in source RGB-video of a static scene. More precisely, given an image sequence, the aim of the paper is to estimate a sequence of depth-maps having temporally consistent depth values. This depth-maps can then be used in several different of tasks; scene reconstruction and layer separation are two such examples. Having the depth values for all images in a video could also allow the usage of more sophisticated image processing techniques, an example could be the addition of virtual shadows in the source video.

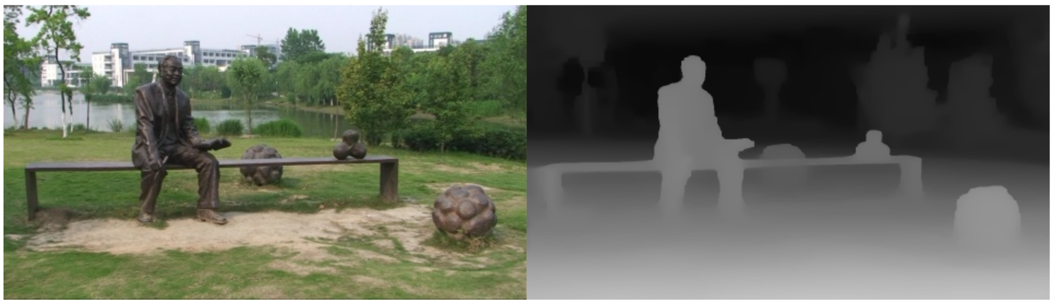

Figure 1.1: One of the estimated depth-maps obtained using the approach described in [3] over a sequence of 200 images.

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

Note:

While a free-of-charge implementation is available on the authors' website (www.zjucvg.net), 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. This report's implementation can be found in the **GitHub** repository www.github.com/giorgio-mariani/project1_IM_2018-2019.

1.2 Proposed Approach

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

Camera Parameters 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 also related to a previous publication (see [4]) done by Zhang and Bao in collaboration with another research group. It is worth to note that SFM algorithms can also produce as output a sparse cloud of 3D feature points (the ones used for the calibration). This is important since these points are also utilized during the depth-maps recovery.

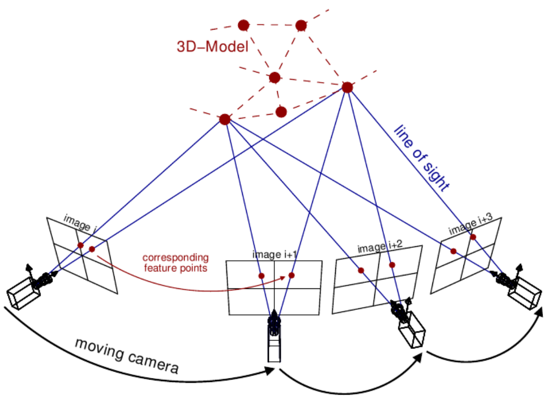

Figure 1.2: SFM photo-grammetric approach; it tracks features points over different frames in order to infer the camera parameters. Source: Theia-sfm.org

Disparity initialization. For each frame in the video, a raw estimation of the corresponding depth-map is computed; this is achieved by minimizing an 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], these segments are then considered as a set of planes 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.

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-consistency constraints.

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 iterative *Conjugate Gradient Method*, an optimization technique able to find approximate solution to energy-minimization problems. The designed loss function takes into consideration: *spatial continuity* between already computed depth-values, *temporal coherence* of the estimated depth-maps, and consistency with the sparse points obtained by the SFM algorithm used to compute the camera parameters.

Chapter 2

Complete Approach Description

The goal of the system is to estimate a sequence of disparity-maps $\hat{D} = D_0, \dots, D_n$, using a sequence of images $\hat{I} = I_0, \dots, I_n$ and camera parameters. Specifically, for each image $I_t \in \hat{I}$ the camera's *position*, *rotation*, and *intrinsic matrix* are assumed known, and are respectively noted as $\boldsymbol{T}_t$, $\boldsymbol{R}_t$, and $\boldsymbol{K}$ (note that the *intrinsic matrix* does not depend on the frame). The estimated pixel $\boldsymbol{x}$'s disparity[1] at time t is noted with $D_t(\boldsymbol{x})$ (sometimes also referred to as $d_{\boldsymbol{x}}$ for shortness sake); the admissible disparity values used during the disparity-maps recovery are taken from the set $[d_{min}, d_{max}]$.

2.1 Disparity Initialization

During the initialization phase, for each frame in the input video, an initial disparity-map is estimated; this estimation occurs within a two-step process: firstly, the depth-maps are initialized using a multi-view photo-consistency approach and *Loopy Belief Propagation* (LBP) [2]. Then, *mean-shift segmentation*[1] is employed over the sequence's images, dividing each picture into several segments similar in color. For each segment a "disparity plane" is thus fit to the image's disparity-map.

2.1.1 Energy Minimization

The initial disparity-maps estimation process works by minimizing the energy function

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

The variable t iterates over the video sequence frames, while the term $\widetilde{E}_{data}(\cdot)$ indicates how much photo-consistent is the input depth-map D_t. Finally, $E_{smooth}(\cdot)$ encodes how smooth[2] the D_t disparity-map is.

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

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

Important!

The disparity values used during this estimation are quantized into m uniformly spaced values $d_{min} = d_0, \dots, d_{m-1} = d_{max}$. This disparity quantization is necessary in order to execute the algorithms utilized by the system. This is especially true for Belief Propagation, since it makes use of discrete labels instead of real values.

Energy Data Term. $\widetilde{E}_{data}(\cdot)$ is defined in terms of *disparity likelihood*, which in turn is defined as

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

This likelihood is used to the describe the photo-consistency of a certain disparity value d. Indeed, the function $p_c(\boldsymbol{x}, d, t, t')$ describes how much the pixel $\boldsymbol{x}$, using disparity d, is photo-consistent, which is expressed as

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

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

$$\widetilde{E}_{data}(D_t) = \sum_{\boldsymbol{x}} 1 - u(\boldsymbol{x}) \cdot L_{init}(\boldsymbol{x}, D_t(\boldsymbol{x})), \tag{2.2}$$

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

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

$$\rho(d_{\boldsymbol{x}}, d_{\boldsymbol{y}}) = \min\{|d_{\boldsymbol{x}} - d_{\boldsymbol{y}}|, \eta\}.$$

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.

The value $\rho(\cdot)$ is then weighted using an adaptive smoothness weight $\lambda(\boldsymbol{x}, \boldsymbol{y})$, which encodes changes of color between the adjacent pixels $\boldsymbol{x}$, $\boldsymbol{y}$: if $\boldsymbol{x}$ and $\boldsymbol{y}$ 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

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

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

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

The smoothness term definition is then

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

2.1.2 Disparity Planes Fitting

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 Bundle Optimization

Bundle optimization is similar to the first step of the initialization phase, since it also makes use of consistency constraints and LBP for disparity-maps polishing. However, the energy function 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 $E_{smooth}(\cdot)$ is the same as in eq. (2.1), in contrast with $E_{data}(\cdot,\cdot)$, which substitutes $\widetilde{E}_{data}(\cdot)$, and requires the sequence of initialized disparity-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 $E_{data}(\cdot,\cdot)$ terms is defined similarly to $\widetilde{E}_{data}(\cdot)$, with only one major difference: the likelihood L_{init} term is replaced by L, which is defined as

$$L(\boldsymbol{x}, d) = \sum_{t'} p_c(\boldsymbol{x}, d, t, t') \cdot p_v(\boldsymbol{x}, d, D_{t'}), \tag{2.5}$$

with p_v the function used to encode geometric coherence:

$$p_v(\boldsymbol{x}, d, D_{t'}) = \exp\left(-\frac{\|\boldsymbol{x} - l_{t',t}(\boldsymbol{x}', D_{t'}(\boldsymbol{x}'))\|^2}{2\sigma_d^2} \right), \tag{2.6}$$

with $\boldsymbol{x}' = l_{t,t'}(\boldsymbol{x}, d)$. According to this definition, p_v encodes the distance between the coordinates $\boldsymbol{x}$ and $l_{t',t}(\boldsymbol{x}', D_{t'}(\boldsymbol{x}'))$ 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.

2.3 Space-Time Fusion

While bundle optimization is able to improve the recovered disparity-maps obtained after disparity initialization, it is still not able to remove all the noise caused by the required disparity quantization and possible estimation errors; to solve this, a *space-time fusion* algorithm is introduced in the recovery procedure. This further phase will polish the obtained results and provide smoother and real valued disparity-maps. The space-time fusion will also exploit the sparse feature points generally obtained by SFM algorithms in order to polish and enhance the disparity-maps. This space-time fusion works by defining a number of constraints using the estimated disparity values $\{D_t\}_{t=0..n}$ in order to setup a linear system which is consequently solved using an iterative *conjugate gradient solver.* The constraints used by the linear system can be divided into three categories: *spatial continuity*, *temporal coherence*, and *sparse feature correspondences.* The solver resulting disparity-values are noted as $\{D_t^*\}_{t=0..n}$.

Note:

In order to avoid severe memory consumption, space-time fusion is performed in batches of only 5-10 frames, instead of using the whole sequence.

2.3.1 Spatial Continuity

The depth structure of the recovered disparity-maps is generally correct, so it should be preserved in the fused disparity-maps. To do so, it is imposed that the difference between two neighboring pixels' disparities should mirror the one in the bundle disparity-maps. That is, for each frame t and pixel (x, y), the spatial constraints

$$\begin{aligned} D_t^*(x+1,y) - D_t^*(x,y) &= D_t(x+1,y) - D_t(x,y), \\ D_t^*(x,y+1) - D_t^*(x,y) &= D_t(x,y+1) - D_t(x,y) \end{aligned}$$

are taken into consideration during the space-time fusion.

2.3.2 Temporal Coherence

It is important for the estimated disparity-maps to be temporally coherent, that is, if a pixel x has a certain disparity at frame t, then the same pixel projected at frame t' (noted as x') must have a coherent disparity values. This can be mathematically expressed, using epipolar geometry, through the formula

$$\boldsymbol{x'} = (x_{\boldsymbol{x}'}, y_{\boldsymbol{x}'}, z_{\boldsymbol{x}'})^\top = z_{\boldsymbol{x}} \boldsymbol{R}_{t'}^\top \boldsymbol{R}_t \boldsymbol{K}_{t'}^{1} \boldsymbol{x}^h + \boldsymbol{R}_{t'}^\top (\boldsymbol{T}_t - \boldsymbol{T}_{t'}), \tag{2.7}$$

where $\boldsymbol{R}$ is the rotation matrix, $\boldsymbol{T}$ the translation vector, and $\boldsymbol{K}$ is the intrinsic matrix. It should also be noted, in order to avoid confusion, that $z_{\boldsymbol{x}}$ and $z_{\boldsymbol{x}'}$ are depth values, not disparities.

It is then possible to simplify eq. (2.7) to $z_{\boldsymbol{x}'} = \boldsymbol{A} \cdot z_{\boldsymbol{x}} + \boldsymbol{B}$, with $\boldsymbol{A}$ and $\boldsymbol{B}$ dependent on the pixel $\boldsymbol{x}$ and the camera parameters. This is useful, since it is able to give a direct correlation between the two depth values. Indeed, if it is assumed that

$$D^*_{t'}(\boldsymbol{x}') = \frac{1}{z_{\boldsymbol{x}'}} \text{ and } D^*_t(\boldsymbol{x}) = \frac{1}{z_{\boldsymbol{x}}},$$

then it is possible to assert

$$\frac{1}{z_{\boldsymbol{x}'}} = \frac{1}{\boldsymbol{A} \cdot z_{\boldsymbol{x}} + \boldsymbol{B}} \iff D^*_{t'}(\boldsymbol{x}') = \frac{D^*_t(\boldsymbol{x})}{\boldsymbol{A} + \boldsymbol{B} \cdot D^*_t(\boldsymbol{x})},$$

which in turn can be used to define a temporal coherence constraint over adjacent frames (t and $t+1$), i.e.

$$\alpha \cdot \left(D^*_{t+1}(\boldsymbol{x}^{t \to t+1}) - \frac{D^*_t(\boldsymbol{x})}{\boldsymbol{A} + \boldsymbol{B} \cdot D^*_t(\boldsymbol{x})} \right) = 0. \tag{2.8}$$

The vector $\boldsymbol{x}^{t \leftarrow t+1}$ indicates the projection of pixel x at time $t+1$ using disparity $D^*_t(\boldsymbol{x})$, while the constant α is used to adjust the constraint's influence during estimation. In the experiments described in [3], $\alpha = 2$.

Important!

Since the used conjugate gradient solver requires linear constraints in order to work efficiently, eq. (2.8) is substituted by

$$\alpha \cdot \left(D^*_{t+1}(\boldsymbol{x}^{t \to t+1}) - \frac{D^*_t(\boldsymbol{x})}{\boldsymbol{A} + \boldsymbol{B} \cdot \widetilde{D}^*_t(\boldsymbol{x})} \right) = 0,$$

with $\widetilde{D}^*_t(\boldsymbol{x})$ the value of $D^*_t(\boldsymbol{x})$ in the previous iteration (or $D_t(\boldsymbol{x})$ during the first iteration).

2.3.3 Sparse Feature Correspondences

During the camera parameters estimation (SFM phase), it is possible to extract a set of sparse 3D feature points from the video. Hence, it is possible to exploit such feature points in order to guide the space-time fusion estimation process to more precise results. This is accomplished by projecting such points to a specific frame t coordinate system, and then imposing the feature point's depth to the corresponding pixel for frame t.

Having a 3D feature point $\boldsymbol{X}$ and a frame t, it is possible to compute its projection $\boldsymbol{u}^t_{\boldsymbol{X}}$ using the equation

$$\boldsymbol{u}^t_{\boldsymbol{X}} = \boldsymbol{K}\boldsymbol{R}^\top_t \left(\boldsymbol{X} - \boldsymbol{T}_t \right),$$

with $\boldsymbol{K}$, $\boldsymbol{R}_t$, and $\boldsymbol{T}_t$ the camera parameters. The depth coordinate of $\boldsymbol{u}^t_{\boldsymbol{X}}$ (noted as $d^{\boldsymbol{X}}_t$) is then used to add the constraint

$$\beta \cdot \left(D^*_t(\boldsymbol{u}^t_{\boldsymbol{X}}) - d^{\boldsymbol{X}}_t \right) = 0. \tag{2.9}$$

with $\beta = 100$ used to adjust the constraint's influence during estimation. It is also important to note that constraints like eq. (2.9) are used only for feature points $\boldsymbol{X}$ considered *reliable*, that is, such that $\| D^*_t(\boldsymbol{u}^t_{\boldsymbol{X}}) - d^{\boldsymbol{X}}_t \| < \kappa$, where κ is a threshold.

Chapter 3

Project Details and Implementation

As already introduced, this project is a partial implementation of the article described in [3]. In this chapter it will be explained more in detail *what* and *how* was everything implemented. The decision to not do a complete implementation of the system was adopted due to the following reasons:

- Reduced time on the author's part, which rendered the completion of some section of the paper not plausible.
- Inexperience in some field of study necessary to the implementation of some part of the system, namely *Conjugate Gradient Method* in the space-time fusion phase (section 2.3).
- Difficult compatibility of used technology for some of the faced problems, i.e. mean shift segmentation (section 2.1.2) using the python's **OpenCV** binding was more burdensome to implement than expected.

3.1 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.

3.1.1 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.

3.2 Implementation

In this section the various algorithms employed by the system and implemented for the project are explained through the use of pseudo-code. To get a more in-depth look at how the system works, the reader should refer to the project's **GitHub** repository at www.github.com/giorgio-mariani/project1_IM_2018-2019.

3.2.1 Implemented Components

The parts of the system described in [3] that were de-facto implemented are the following:

- Disparity Initialization (section 2.1) without the color segmentation/disparity plane fitting component, i.e. only the LBP optimization part is used for initializing the disparity-maps.
- Bundle Optimization (section 2.2), which was entirely implemented.

In other words, the omitted parts are both the second step of the initialization phase and the space-time fusion of the bundle results.

Furthermore, the camera parameters are assumed known; this should not be a restricting assumption since there is rich literature in terms of camera calibration and parameters estimation, and a variety of algorithms, such as the one described in [4], can be employed in order to solve this known problem.

3.2.2 Notation

In this section, some guidelines on how to read the pseudo-code are defined:

- Matrices and vectors are noted using bold serif font: $\boldsymbol{M}$, $\boldsymbol{x}$, e.t.c.
- Sequences of objects are noted using calligraphic font: $\mathcal{I}$, $\mathcal{D}$, e.t.c.
- The notation $3 \times 4 \times 5$ indicates the shape of array with three elements in the first dimension, four in the second one, and five in the last one.
- Functions of type `reduce_`*`func`*`(`*`array`*`,`*`axis`*`)` apply the reduce function *`func`* (min for example) over the dimension at position *`axis`*.
- Functions of type `elementwise_`*`func`*`(`*`array`*`)` apply the function *`func`* to each element in *`array`*.
- The function `reshape(`*`array`*`, `*`shape`*`)` reshapes *`array`* to the shape *`shape`*.

3.2.3 Loopy Belief Propagation

As already mentioned, LBP is an optimization technique that can be used in order to approximate minimal graph labelings. For a more in-depth description of the algorithm see appendix A.1. In this section, the pseudo-code of the implementation used in this project is shown. It should be mentioned that the given code (for simplicity sake) does only partially reflect the actual implementation, since some optimizations and complications (like the smoothness factor $\lambda(\cdot,\cdot)$ and the Multiscale variant of LBP) are not present.

Note:

In order to understand this section, the reader is strongly suggested to first read appendix A.1.

Algorithm 3.1: Loopy Belief Propagation

```
Require: $\boldsymbol{D}$ has shape $h \times w \times k$.    ▷ Data Cost
Require: $s > 0$ and $\eta \geq 0$.

 1: procedure LBP($\boldsymbol{D}$, $s$, $\eta$)
 2:     Create $\boldsymbol{m_x}$ with shape $h \times w \times k$ (init. to zero).    ▷ $\forall x \in \{\boldsymbol{up}, \boldsymbol{down}, \boldsymbol{left}, \boldsymbol{right}\}$

 3:     for $t \leftarrow 0, \ldots, T$ do
 4:         $\boldsymbol{h_{tot}} \leftarrow \boldsymbol{D} + \boldsymbol{m_{up}} + \boldsymbol{m_{down}} + \boldsymbol{m_{left}} + \boldsymbol{m_{right}}$
 5:         $\boldsymbol{h_{up}} \leftarrow$ moveDown($\boldsymbol{h_{tot}} - \boldsymbol{m_{down}}$)    ▷ Start computing $h(f)$
 6:         $\boldsymbol{h_{down}} \leftarrow$ moveUp($\boldsymbol{h_{tot}} - \boldsymbol{m_{up}}$)
 7:         $\boldsymbol{h_{left}} \leftarrow$ moveRight($\boldsymbol{h_{tot}} - \boldsymbol{m_{right}}$)
 8:         $\boldsymbol{h_{right}} \leftarrow$ moveLeft($\boldsymbol{h_{tot}} - \boldsymbol{m_{left}}$)    ▷ Finish computing $h(f)$

 9:         for $\boldsymbol{x} \in \{\boldsymbol{up}, \boldsymbol{down}, \boldsymbol{left}, \boldsymbol{right}\}$ do    ▷ Start computing $m(f)$
10:             $\boldsymbol{m_x} \leftarrow \boldsymbol{h_x}$
11:             for $i = 0, \ldots, k$ do
12:                 $\boldsymbol{m_x} \leftarrow \min(\boldsymbol{m_x}[:, :, i], \boldsymbol{m_x}[:, :, i-1] + s)$
13:             for $i = k, \ldots, 0$ do
14:                 $\boldsymbol{m_x} \leftarrow \min(\boldsymbol{m_x}[:, :, i], \boldsymbol{m_x}[:, :, i+1] + s)$    ▷ Finish computing $m(f)$

15:         for $\boldsymbol{x} \in \{\boldsymbol{up}, \boldsymbol{down}, \boldsymbol{left}, \boldsymbol{right}\}$ do
16:             $\boldsymbol{tmp} \leftarrow$reduce_min($\boldsymbol{h_x}$, $\textit{last-axis}$)$+\eta$    ▷ $\boldsymbol{tmp}$ has shape $h \times w$
17:             $\boldsymbol{m_x} \leftarrow$ elmentwise_min($\boldsymbol{m_x}$, $\boldsymbol{tmp}$)    ▷ $\boldsymbol{tmp}$ is broadcast to shape $h \times w \times k$

18:     $\boldsymbol{B} \leftarrow$ Copy of $\boldsymbol{D}$
19:     for $\boldsymbol{x} \in \{\boldsymbol{up}, \boldsymbol{down}, \boldsymbol{left}, \boldsymbol{right}\}$ do    ▷ Compute belief vector
20:         $\boldsymbol{B} \leftarrow \boldsymbol{B} + \boldsymbol{m_x}$
21:     return reduce_argmin($\boldsymbol{B}$, $\textit{last-axis}$)    ▷ output labels, shape: $h \times w$
22: end procedure
```

The LBP's pseudo-code can be observed in algorithm 3.1. The input parameter $\boldsymbol{D}$ represents the *data cost* term in eq. (A.1), while the values s, η represent the homonymous constants in eq. (A.2). $\boldsymbol{D}$ is a multi-dimensional array having shape $h \times w \times k$, with k the number of utilized labels. The value v at position $\boldsymbol{D}[x, y, f]$ indicates the data cost for label f at pixel (x, y).

Function:

The class of functions `move`*`Direction`*`(`$\boldsymbol{M}$`)` move (by one position) all elements in the input matrix with respect to the direction expressed by *`Direction`*. In practical terms, an affine transformation is applied over $\boldsymbol{M}$.

The sent messages $m^t_{(\boldsymbol{p},\boldsymbol{q})}$ are implemented through the matrices $\boldsymbol{m_{up}}$, $\boldsymbol{m_{down}}$, $\boldsymbol{m_{left}}$, and $\boldsymbol{m_{right}}$. Each of these has shape $h \times w \times k$ and represents $m^t_{(\boldsymbol{p},\boldsymbol{q})}$ with respect to edges in one of the possible four directions. For example,

$$\boldsymbol{m_{up}}[y, x] = m^t_{((x,y+1),(x,y))}.$$

The code between line 9 and line 17 implements the eq. (A.4), while computation of the *belief vectors* is done at lines 19 and 20 by summing $\boldsymbol{m_{up}}$, $\boldsymbol{m_{down}}$, $\boldsymbol{m_{left}}$, and $\boldsymbol{m_{right}}$ to $\boldsymbol{D}$. Finally, the output disparity-labels are calculated in the last line of the procedure by extracting from $\boldsymbol{B}$'s last dimension (which contains label indeces) the labels having minimal weights.

3.2.4 Disparity Initialization

In order to minimize the energy function described in eq. (2.1), the LBP algorithm (see above) is used. For the disparity-map recovery problem, the disparity-map 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 $d_0, \ldots, d_{m-1}$. Therefore, an assignment of these labels over the graph's nodes is a possible disparity-map.

In order to work, the LBP algorithm requires as input the *data cost*, i.e. the cost of assigning a disparity value to a pixel $\boldsymbol{x} = (x, y)$, for all pixels. This cost is specified by eq. (2.2), particularly for every pixel $\boldsymbol{x}$ and disparity label d, the value

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

should be computed. The value $u(\boldsymbol{x}) \cdot L_{init}$ is essentially the likeliness (normalized through $u(\boldsymbol{x})$) of a pixel to have disparity d.

Computation of the disparity data cost is done through algorithm 3.2. The algorithm takes as input a frame t, the camera parameters (denoted by $\boldsymbol{K}$, $\mathcal{R}$, and $\mathcal{T}$), and the image sequence $\mathcal{I}$ and outputs a multi-dimensional array with shape $h \times w \times m$ representing the data cost for the disparity-map estimation problem.

Procedure `conjugate_coor`, described in algorithm 3.3, computes for each homogeneous coordinate in the array $\boldsymbol{x}^h$, the respective conjugate point w.r.t. the given camera parameters and depth values. This procedure is a vectorized implementation of the formula in appendix A.2.

Algorithm 3.2: `compute_energy_data_init`

```
Require: $\boldsymbol{K}$ to be the camera intrinsic matrix
Require: $\mathcal{R}$ to be a sequence of camera rotation matrices
Require: $\mathcal{T}$ to be a sequence of camera translation vectors
Require: $\mathcal{I}$ to be a sequence of images

 1: procedure compute_energy_data_init($t$, $\boldsymbol{K}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{I}$)
 2:     $\boldsymbol{I_t}$, $\boldsymbol{R_t}$, $\boldsymbol{T_t} \leftarrow \mathcal{I}[t]$, $\mathcal{R}[t]$, $\mathcal{T}[t]$        ▷ get image and camera param. for frame $t$
 3:     $\boldsymbol{x}^h \leftarrow$ homogeneous_coor_grid($h$,$w$)        ▷ create a grid of hom. indices
 4:     Create table $\boldsymbol{L}$ with $h \times w \times m$ elements (init. to zero).

 5:     for $t' \leftarrow 0 \dots n$ do
 6:         $\boldsymbol{I_{t'}}$, $\boldsymbol{R_{t'}}$, $\boldsymbol{T_{t'}} \leftarrow \mathcal{I}[t']$, $\mathcal{R}[t']$, $\mathcal{T}[t']$  ▷ get image and camera param. for frame $t'$

 7:         for $label \leftarrow 0 \dots m$ do
 8:             Create matrix $\boldsymbol{D}$ with $h \times w$ elements, initialized with value $d_{label}$
 9:             $\boldsymbol{x'}^h \leftarrow$ conjugate_coor($\boldsymbol{x}^h$, $\boldsymbol{K}$, $\boldsymbol{R_t}$, $\boldsymbol{R_{t'}}$, $\boldsymbol{T_t}$, $\boldsymbol{T_{t'}}$, $\boldsymbol{D}$)
10:             $\boldsymbol{I^r_{t'}} \leftarrow$ remap($\boldsymbol{I_{t'}}$, $\boldsymbol{x'}^h$)
11:             $p_c \leftarrow \sigma_c$ / ($\sigma_c$+ reduce_norm($\boldsymbol{I_t} - \boldsymbol{I^r_{t'}}$))
12:             $\boldsymbol{L}[:,:,label] \leftarrow \boldsymbol{L}[:,:label] + p_c$        ▷ likeliness for label $label$

13:     $\boldsymbol{u} \leftarrow 1/$ reduce_max($\boldsymbol{L}$, $last\text{-}axis$)        ▷ normalization factor w.r.t. to label axis
14:     return $1 - \boldsymbol{u} \cdot \boldsymbol{L}$
15: end procedure
```

Algorithm 3.3: `conjugate_coor`

```
1: procedure conjugate_coor($\boldsymbol{x}^h$, $\boldsymbol{K}$, $\boldsymbol{R_1}$, $\boldsymbol{R_2}$, $\boldsymbol{T_1}$, $\boldsymbol{T_2}$, $\boldsymbol{D}$)
2:     reshape($\boldsymbol{x}^h$, $3 \times (h \cdot w)$)        ▷ reshape for matrix mul.
3:     reshape($\boldsymbol{D}$, $1 \times (h \cdot w)$)        ▷ reshape for matrix mul.
4:     $\boldsymbol{x'}^h \leftarrow \boldsymbol{K}\boldsymbol{R_2}^\top \cdot \left(\boldsymbol{R_1}\boldsymbol{K}^{-1}\boldsymbol{x}^h + (\boldsymbol{T_1} - \boldsymbol{T_2})^\top \cdot \boldsymbol{D}\right)$
5:     reshape($\boldsymbol{x'}^h$, $3 \times h \times w$)        ▷ go back to normal shape
6: end procedure
```

Function:

The procedure `homogeneous_coordinate_grid(`h`, `w`)` produces a multi-dimensional array $\boldsymbol{x}^h$ filled with a grid of homogeneous coordinates, such that the first axis represent the coordinate itself, i.e.

$$\boldsymbol{x}^h[0,i,j] = i$$
$$\boldsymbol{x}^h[1,i,j] = j$$
$$\boldsymbol{x}^h[2,i,j] = 1$$

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, $\boldsymbol{x}^h$'s shape is $3 \times h \times w$.

Function:

The procedure `remap(`$\boldsymbol{I}$`, `$\boldsymbol{map}$`)` is used to transform the input image $\boldsymbol{I}$ using a certain mapping $\boldsymbol{map}$ such that the output image $\boldsymbol{I'}$ is defined as

$$\boldsymbol{I'}[\boldsymbol{x}] = \boldsymbol{I}[\boldsymbol{map}[\boldsymbol{x}]].$$

3.2.5 Bundle Optimization

As with Disparity Initialization, Bundle Optimization makes also use of LBP in order to estimate the disparity-maps. As a result, the algorithm used for Bundle Optimization is similar to the one used to compute the data cost during the disparity initialization (see algorithm 3.2), the only major difference is the requirement to compute the term p_v, found in eq. (2.6). This term is indeed used in order to define the Bundle Optimization likelihood (function L in eq. (2.5)) By adding the pseudo-code at algorithm 3.4 in the inner for loop and by changing how the array $\boldsymbol{L}$ is updated, it is possible to expand algorithm 3.2 with Bundle Optimization. The use of such component is managed by the boolean variable *use_bundle*. Thus, the final algorithm, able to exploit the term geometric coherence constraint p_v, is described in algorithm 3.5.

Algorithm 3.4: `compute_pv`

```
1: procedure conjugate_coor($\boldsymbol{x'}^h$, $\boldsymbol{K}$, $\boldsymbol{R_1}$, $\boldsymbol{R_2}$, $\boldsymbol{T_1}$, $\boldsymbol{T_2}$, $\boldsymbol{D}$)
2:     $\boldsymbol{D^r} \leftarrow$ remap($\boldsymbol{D}$, $\boldsymbol{x'}^h$)
3:     $\boldsymbol{l_{t',t}} \leftarrow$ conjugate_coordinates($\boldsymbol{x'}^h$, $\boldsymbol{K}$, $\boldsymbol{R_2}$, $\boldsymbol{R_1}$, $\boldsymbol{T_2}$, $\boldsymbol{T_1}$, $\boldsymbol{D}^r$)
4:     return $\exp\left(-\|\boldsymbol{x} - \boldsymbol{l_{t',t}}\|^2 \cdot \frac{1}{2\sigma_d^2}\right)$
5: end procedure
```

Algorithm 3.5: `compute_energy_data`

Require: $\boldsymbol{K}$ to be the camera intrinsic matrix
Require: $\mathcal{R}$ to be a sequence of camera rotation matrices
Require: $\mathcal{T}$ to be a sequence of camera translation vectors
Require: $\mathcal{I}$ to be a sequence of images
Require: $\mathcal{D}$ to be a sequence of disparity-maps

```
1:  procedure compute_energy_data_init($t$, $\boldsymbol{K}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{I}$, $\mathcal{D}$)
2:      $\boldsymbol{I_t}$, $\boldsymbol{R_t}$, $\boldsymbol{T_t} \leftarrow \mathcal{I}[t]$, $\mathcal{R}[t]$, $\mathcal{T}[t]$      ▷ get image and camera param. for frame $t$
3:      $\boldsymbol{x}^h \leftarrow$ homogeneous_coor_grid($h$,$w$)
4:      Create table $\boldsymbol{L}$ with $h \times w \times m$ elements, initialized with zero

5:      for $t' \leftarrow 0 \ldots n$ do
6:          $\boldsymbol{I_{t'}}$, $\boldsymbol{R_{t'}}$, $\boldsymbol{T_{t'}} \leftarrow \mathcal{I}[t']$, $\mathcal{R}[t']$, $\mathcal{T}[t']$      ▷ get image and camera param. for frame $t'$

7:          for $label \leftarrow 0 \ldots m$ do
8:              Create matrix $\boldsymbol{D}$ with $h \times w$ elements, initialized with value $d_{label}$
9:              $\boldsymbol{x'}^h \leftarrow$ conjugate_coor($\boldsymbol{x}^h$, $\boldsymbol{K}$, $\boldsymbol{R_t}$, $\boldsymbol{R_{t'}}$, $\boldsymbol{T_t}$, $\boldsymbol{T_{t'}}$, $\boldsymbol{D}$)
10:             $\boldsymbol{I^r_{t'}} \leftarrow$ remap($\boldsymbol{I_{t'}}$, $\boldsymbol{x'}^h$)
11:             $p_c \leftarrow \sigma_c$ / ($\sigma_c$+ reduce_norm($\boldsymbol{I_t} - \boldsymbol{I^r_{t'}}$))
12:             if use_bundle then      ▷ Check if Bundle Optimization
13:                 $p_v \leftarrow$compute_pv($\boldsymbol{x}^h$, $\boldsymbol{K}$, $\boldsymbol{R_t}$, $\boldsymbol{R_{t'}}$, $\boldsymbol{T_t}$, $\boldsymbol{T_{t'}}$, $\mathcal{D}[t']$)
14:                 $\boldsymbol{L}[:,:,label] \leftarrow \boldsymbol{L}[:,:\,label] + p_c \cdot p_v$      ▷ likeliness for label $label$
15:             else
16:                 $\boldsymbol{L}[:,:,label] \leftarrow \boldsymbol{L}[:,:\,label] + p_c$      ▷ likeliness for label $label$

17:     $\boldsymbol{u} \leftarrow 1/$ reduce_max($\boldsymbol{L}$, $last\text{-}axis$)      ▷ normalization factor w.r.t. to label axis
18:     return $1 - \boldsymbol{u} \cdot \boldsymbol{L}$
19: end procedure
```

Chapter 4

System Documentation

4.1 Dependencies Installation

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

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

Note:

Python 2.7 is required in order to run the system, consequently, the appropriate version of **pip** should be used.

4.2 Running the System

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

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

The argument `<config.txt>` is a configuration file containing parameter values and paths to the folder storing all data necessary for the disparity-maps recovering process (this data is generally either pictures or other disparity-maps). The options `-i` and `-b` are used to communicate the system which of the available phase should be executed:

`-i` it executes only Disparity Initialization; the resulting disparity-maps are then stored in an output folder.

`-b` it executes only Bundle Optimization. Note that ,because of how bundle optimization works, it also requires as input the previously estimated depth-maps. As before, the resulting disparity-maps are then stored in an output folder.

`-ib` it executes both Disparity Initialization and Bundle Optimization.

4.3 System's Input

4.3.1 Camera File

This file contains information about the camera parameters; in particular, it stores information about intrinsic matrix, rotations, and translation of the camera for each picture in the sequence. The camera file itself is quite simple; for each image the intrinsic matrix, rotation matrix, and position must be included in plain text, one after the other, without interleaving blank lines.

The code below shows an example of such format: the first three rows are the *intrinsic matrix*, the second three rows are the *rotation matrix*, and the final row is the *position vector*.

```
1139.7929     0.0000000     479.50000
0.0000000     1139.7929     269.50000
0.0000000     0.0000000     1.0000000
1.0000000     0.0000000     0.0000000
0.0000000     1.0000000     0.0000000
0.0000000     0.0000000     1.0000000
0.0000000     0.0000000     0.0000000
```

....

This pattern must be repeated for each image in the target sequence, and two blank lines must be used between data of different images. Also, the first line of the camera file must be a number, indicating the number of camera frames in the file itself. A blank line must follow. A complete example of a camera file which makes use of a sequence with three frames is the following:

```
3

1139.7929     0.0000000     479.50000
0.0000000     1139.7929     269.50000
0.0000000     0.0000000     1.0000000
1.0000000     0.0000000     0.0000000
0.0000000     1.0000000     0.0000000
0.0000000     0.0000000     1.0000000
0.0000000     0.0000000     0.0000000


1139.7929     0.0000000     479.50000
0.0000000     1139.7929     269.50000
0.0000000     0.0000000     1.0000000
0.9999972     -0.001567     0.0017456
0.0015681     0.9999985     -0.000634
-0.001744     0.0006373     0.9999982
-0.320146     0.0141622     -0.049300


```

```
1139.7929     0.0000000     479.50000
0.0000000     1139.7929     269.50000
0.0000000     0.0000000     1.0000000
0.9999884     -0.002738     0.0039581
0.0027459     0.9999945     -0.001806
-0.003953     0.0018177     0.9999905
-0.720470     -0.028543     -0.089562
```

Note:

The location of the camera file is defined by the configuration parameter `"camera_file"`.

4.3.2 Pictures Directory

Directory which 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 with value in the range between `0000` ang `9999`. The images filenames must be organized in a contiguous fashion: no gaps should be present between the smallest and largest filename numbers. All such image files should have same format and resolution.

Note:

The image format, as well as the location of the directory are indicated by the configuration parameters `"pictures_file_extension"` and`"pictures_directory"` respectively.

4.3.3 Depth-maps Directory

Directory containing a sequence of disparity-maps. These can then be used during execution of algorithm 3.5 in order to perform the Bundle Optimization. It should be noted that these disparity-maps are generally the result of a previous invocation of the system.

The disparity-maps are stored as **.npy** files. Disparity-maps stored inside this directory shoulb be named `depth_<num>`, with `<num>` numeric string which assumes values inside the range `0000` to `9999`. As with the images, the depth-maps filenames must be organized in a contiguous fashion: no gaps should be present between the smallest and largest filename numbers.

Note:

The location of the directory is indicated by the configuration parameter `"depthmaps_directory"`.

4.4 System's Output

The output of the system is a disparity-map directory, as such, names and format follows the same convention as described in section 4.3.3. The location of the output directory is indicated by the configuration parameter `"output_directory"`.

4.5 Configuration File

File containing information used by the system in order to estimate and store depth-maps from a sequence of images. The file must contain a single **JSON** objects; the fields of said objects are then used by the system during execution. An example of a configuration file is as follows:

```json
{
  "camera_file" : "camera.txt",
  "depthmaps_directory" : "depth",
  "pictures_directory" : "source",
  "output_directory" : "output",
  "pictures_file_extension" : ".png",
  "height" : 270,
  "width" : 480,
  "start_frame" : 9,
  "end_frame" : 30
  "depth_levels" : 100,
  "depth_min" : 0.0,
  "depth_max" : 0.008,
  "epsilon" : 50.0,
  "sigma_c" : 10.0
}
```

4.5.1 Parameters

A number of parameters are required to be in the configuration file if the correct behavior of the system is desired. A descriptive list of all the required parameters can be found at table 4.1. The most important between these are `"camera_file"`, `"pictures_directory"`, and `"output_directory"`. Indeed, they are used to manage the files and directories that are used by the system.

Other important parameters are `"depth_min"`, `"depth_max"`, and `"depth_levels"` which are used to respectively define the value of d_{min}, d_{max}, and the quantization precision.

Important!

The optional parameter "depthmaps_directory" is required in order to execute Bundle Optimization during the estimation process. The parameter's value determines the directory containing the initialized disparities used by the geometry constraint.

Required Parameters	**Description**	**Type**
"camera_file"	File containing the the camera parameters.	**String**
"pictures_directory"	Directory containing the images used during depth-maps recovery.	**String**
"output_directory"	Directory in which the output disparity-maps are stored.	**String**
"pictures_file_extension"	File extension used by the images in the pictures directory.	**String**
"height"	Height of the output disparity-maps. It can be different from the height of the original images.	**Int**
"width"	Width of the output disparity-maps. It can be different from the width of the original images.	**Int**
"start_frame"	Frame from which the disparity-maps recovery process will start. For example if it has value 10, then the first picture whose disparity-map is computed will be `img_0010`.	**Int**
"end_frame"	Ending frame used for disparity-map recovery.	**Int**
"depth_min"	Minimum admissible disparity-value used during recovery.	**Float**
"depth_max"	Maximum admissible disparity-value used during recovery.	**Float**
"depth_levels"	Number of discrete disparity labels used during estimation.	**Int**

Table 4.1: Table indicating all the parameters necessary in order to run the system without errors.

4.6 Python Modules

The implementation make use of a number of custom python modules:

`compute_energy`	This module contains functions necessary for the computation of the data cost in eq. (2.1) and eq. (2.2), it also contains the function that computes the smoothness weights $\lambda(\cdot, \cdot)$.
`estimate`	This module is used to estimate the disparity-maps and subsequently store them in the output directory. To do so, it makes use of the module `compute_energy` and `lbp`. This module can be seen as the main entry-point of the system.
`lbp`	This module contains an implementation of the *Loopy Belief Propagation* algorithm used by the `estimate` module during the depth-maps estimation process.
`utils`	This module contains a variety of class used by the other custom python modules in order to pass data between function calls, as well as debugging and visualization procedures.

To see the actual procedures and classes used by these modules, the reader should refer to the project's public repository www.github.com/giorgio-mariani/project1_IM_2018-2019 and documentation.

Chapter 5

Summary

This project is a partial implementation of the article described in [3], using the **NumPy** module and the **OpenCV** python bindings. The reader should look at section 3.2.1 if a more in-depth understanding of which components of the original article were implemented. A comparison between a depth-map obtained by Zhang (left) and one estimated using the provided implementation(right) can be seen in fig. 5.1.

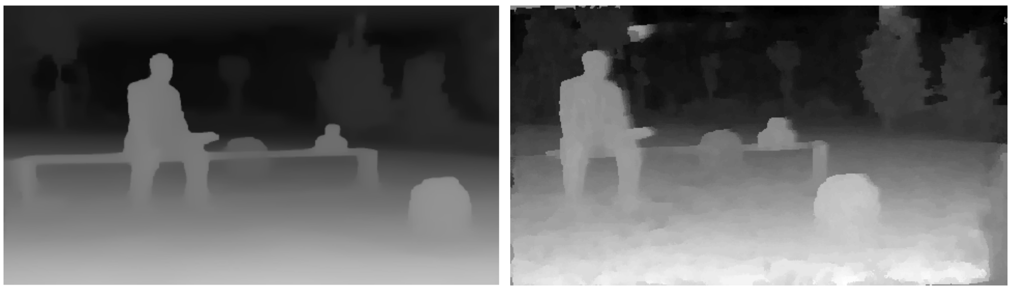

Figure 5.1: Comparison between depth-maps estimated using the complete original implementation (left) and depth-maps estimated using the partial implementation provided.

5.0.1 Future Work

A simple improvement over the project could be the implementation of the remaining components. A further expansion to the project could also be the introduction of hardware specific accelerated libraries, such as **CuPy** or **PyTorch**.

Appendix A

Supplementary Algorithms

A.1 Loopy Belief Propagation Algorithm

Loopy Belief Propagation (LBP) [2] 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.

A.1.1 Problem Statement

Given a graph with vertices (pixels) P and edges N, and a set of labels L (with cardinality k), the goal of LBP is to find a labeling of the vertices $\{f_{\boldsymbol{p}}\}_{\boldsymbol{p}\in V}$ such that the energy function

$$E\left(\{f_{\boldsymbol{p}}\}_{\boldsymbol{p}\in V}\right) = \sum_{(\boldsymbol{p},\boldsymbol{q})\in N} V(f_{\boldsymbol{p}}, f_{\boldsymbol{q}}) + \sum_{\boldsymbol{p}\in P} D(\boldsymbol{p}, f_{\boldsymbol{p}}) \tag{A.1}$$

is minimized. The terms $V(\cdot,\cdot)$ and $D(\cdot,\cdot)$ are respectively named *discontinuity cost* and *data cost*. The data cost can be any arbitrary mapping between pixel-label pairs over real values. On the other hand, the discontinuity cost between two pixels $\boldsymbol{p}$ and $\boldsymbol{q}$ is defined as

$$V(\boldsymbol{p},\boldsymbol{q}) = \cdot \min(s \cdot \|f_{\boldsymbol{p}} - f_{\boldsymbol{q}}\|, \eta), \tag{A.2}$$

with η and s positive constants. This restriction allows the discontinuity cost to be computed in a more efficient way.

A.1.2 Implementation

LBP works by iteratively computing a table of messages exchanged between nodes, until convergence is reached or an iteration threshold is exceeded. Each message is a vector of length equal to the number of labels. Let $m^t_{(\boldsymbol{p},\boldsymbol{q})}$ be the message sent by the node

$\boldsymbol{p}$ to the adjacent node $\boldsymbol{q}$ at iteration t, the content of this message can be recursively described using the messages received by $\boldsymbol{p}$ at time $t-1$, i.e.

$$m^t_{(\boldsymbol{p},\boldsymbol{q})}(f) = \min_{f'} \left(V(f,f') + D(\boldsymbol{p},f') + \sum_{s \in N(\boldsymbol{p})-\boldsymbol{q}} m^t_{(\boldsymbol{q},\boldsymbol{s})}(f') \right), \tag{A.3}$$

for each possible label $f \in L$. The set $N(\boldsymbol{p}) - \boldsymbol{q}$ is the set of neighbors of $\boldsymbol{p}$ excluding $\boldsymbol{q}$. The base cases $m^0_{(\boldsymbol{p},\boldsymbol{q})}$ are initialized with zero values. After the desired amount of iterations T, it is possible to compute the *belief vector* for ech node $\boldsymbol{p}$ as

$$b_{\boldsymbol{p}}(f) = D_{\boldsymbol{p}}(f) + \sum_{\boldsymbol{q} \in N(\boldsymbol{p})} m^T_{(\boldsymbol{q},\boldsymbol{p})}(f).$$

For each node, the final label is the one that minimize the respective belief vector, i.e.

$$f_{\boldsymbol{p}} = \operatorname*{argmin}_{f \in L} \left(b_{\boldsymbol{p}}(f) \right).$$

It is trivial that by alternating two tables of size $|N| \times k$ it is possible to compute the belief vector for each node in the graph, obtaining thus an approximation of the desired graph labeling.

A.1.3 Optimizations

In [2] are described a number of possible optimization that allow the algorithms to execute less computations and reach convergence in less iterations. Particularly, it is shown an algorithm which allows to compute eq. (A.3) linearly to the number of labels by exploiting eq. (A.2), resulting in the formula

$$m^t_{(\boldsymbol{p},\boldsymbol{q})}(f) = \min \left(m(f), \min_{f'} \left(h(f' \in L) + \eta \right) \right), \tag{A.4}$$

with the function $h(f')$ defined for every $f' \in L$ as

$$h(f') = D_{\boldsymbol{p}}(f') + \sum_{\boldsymbol{s} \in N(\boldsymbol{p})-\boldsymbol{q}} m^{t-1}_{(\boldsymbol{s},\boldsymbol{p})}(f') \tag{A.5}$$

and $m(f)$ initialized to $h(f)$ and obtained as result from the procedure

```
for $f = 1, \dots, k-1$ do
    $m(f) \leftarrow \min(m(f), m(f-1) + s)$
for $f = k-2, \dots, 0$ do
    $m(f) \leftarrow \min(m(f), m(f+1) + s)$
```

executed "in-place", so that changes are propagated.

A.2 Conjugate Pixel Computation

Epipolar geometry is a branch of mathematics that focuses on stereo vision; it can be used to compute the position of a pixel $\boldsymbol{x}$ with respect to different frames. This new pixel is called "the *conjugate* of $\boldsymbol{x}$", and it is noted as $\boldsymbol{x'}$. Computation of $\boldsymbol{x'}$ requires knowledge about the disparity of $\boldsymbol{x}$, noted with $d_{\boldsymbol{x}}$ and use of the camera parameters associated to the respective frames.

More precisely, given two frames t and t', the corresponding camera parameters, noted as[1] $\{\boldsymbol{K}, \boldsymbol{R}, \boldsymbol{T}\}$, and $\{\boldsymbol{K'}, \boldsymbol{R'}, \boldsymbol{T'}\}$ respectively, are used to compute $\boldsymbol{x'}$, using the formula

$$\boldsymbol{x'}^h = \boldsymbol{K'}\boldsymbol{R'}^\top \cdot \left(\boldsymbol{R}\boldsymbol{K}^{-1}\boldsymbol{x}^h + d_{\boldsymbol{x}} \cdot \left(\boldsymbol{T} - \boldsymbol{T'}\right)\right).$$

Note that $\boldsymbol{x}^h$ denotes the homogeneous coordinates of $\boldsymbol{x}$.

[1]The camera parameters are noted as $\boldsymbol{K}$ for the intrinsic matrix, $\boldsymbol{T}$ for the translation vector, and $\boldsymbol{R}$ for the rotation matrix.

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 pp. 5, 7).

[2] 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, 26, 27).

[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 pp. 1, 4, 11–13, 25).

[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 pp. 5, 13).