Image Processing and Pattern Recognition

Assignment 4 - TGV Fusion

January 27, 2020

Deadline: February 24, 2020 at 23:55h.
Submission: Upload your report as a .pdf file and your implementation as a .py file to the Teach-Center. Please do not zip your files and use the provided framework-file for your implementation.

1 Multi-View Stereo and Fusion

The goal of this assignment is to implement a fusion algorithm for Multi-View Stereo (MVS). In MVS we have given multiple images of the same scene from different viewpoints (see Fig. 1). We can use neighboring images, rectify them and compute a disparity/depth map for all different viewpoints. This could e.g. be done with the SGM algorithm which we implemented in the last assignment. Since we precisely know all camera positions, we can use this information to map all disparity maps into a common coordinate system. Thus, as shown in Fig. 2a we get multiple

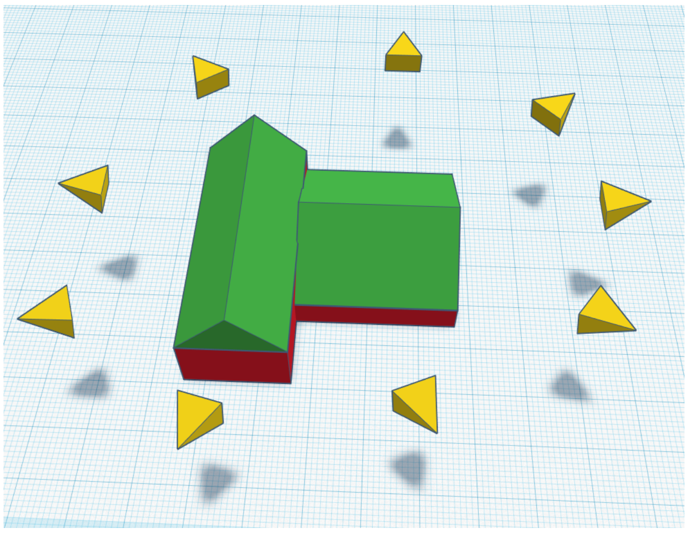
Figure 1: Multi-View Stereo Setting. The scene is captured by multiple cameras (visualized as yellow pyramids). We can use pairs of images to compute a 3D reconstruction of the scene.

observations for every pixel for a specific reference viewpoint. Note that the disparity maps usually contain outliers and wrong estimates which will be different for different observations. In this assignment we assume that the (noisy) disparity maps are already computed and aligned. The noise is here simple salt and pepper noise, where 25% of the pixels are affected. The task in this assignment is now to compute a fused result including the information contained in all observations. A fused result of the given scene is shown in Fig. 2b. To this end, we are going to use the Total Generalized Variation (TGV) Fusion algorithm which is described in the next section.

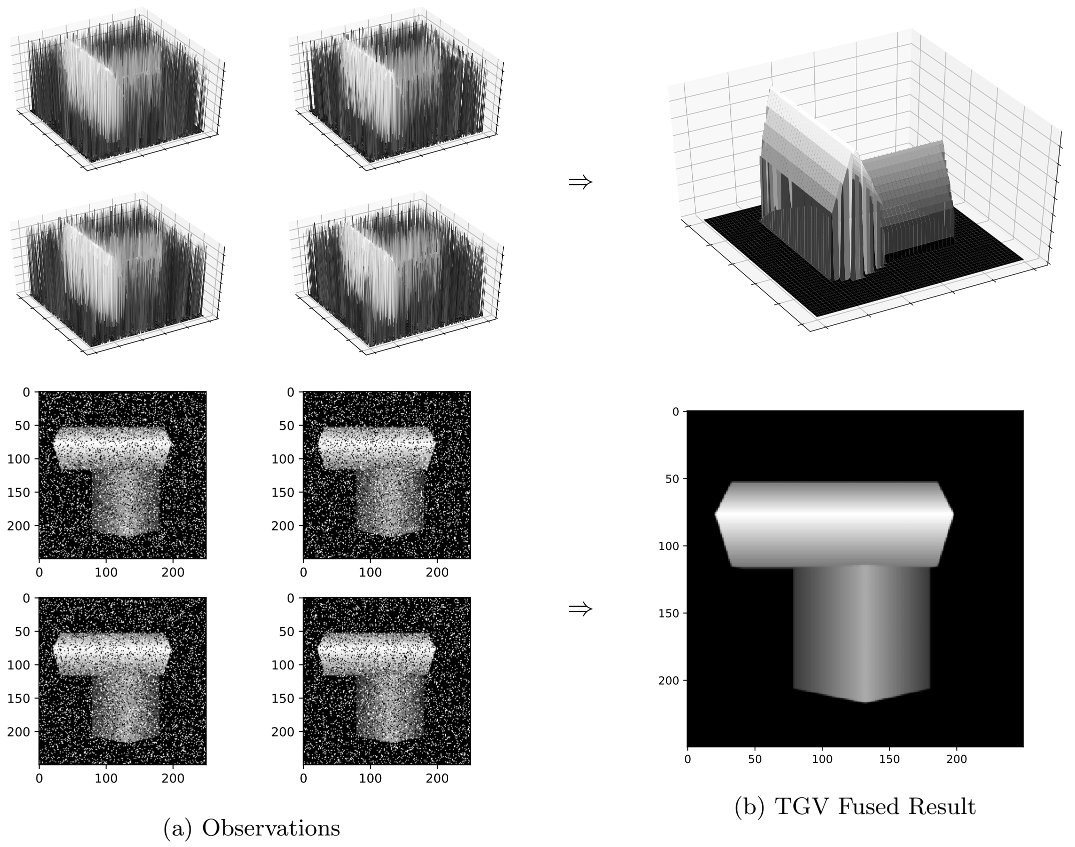

Figure 2: Algorithm overview. Multiple aligned observations of the same scene in (a) are fused together to get a high quality result in (b). Top shows a 3D visualization, bottom shows the corresponding 2D disparity maps.

2 TGV-Fusion

In the following we assume that the observations are given on a regular cartesian grid with size $M \times N$ with indices (i, j). We use lower-case symbols u for vectorized images and upper-case letters U for matrices. The functions $\text{vec}(\cdot)$ and $\text{mat}(\cdot)$ allow to convert between vector representation and matrix representation:

$$x \in \mathbb{R}^{MN} = \text{vec}(X) = x(k), \quad k = 1, \dots, MN \tag{1}$$

$$X \in \mathbb{R}^{M \times N} = \underset{M \times N}{\text{mat}}(x) = X(i,j), \quad i = 1, \dots, M, \quad j = 1, \dots, N \tag{2}$$

The functions $\text{vec}(\cdot)$ and $\text{mat}(\cdot)$ can be performed in Python using `ravel` and `reshape`, respectively.

Next we, precisely specify our model. Given multiple observations $\{f_k\}_{k=1}^K$ of the same scene, we can write the TGV regularized fusion task as

$$\min_{u,v} \Bigg\{ \underbrace{\alpha_1 \| \underset{2 \times MN}{\text{mat}}(\nabla u - v)\|_{2,1} + \alpha_2 \| \underset{4 \times MN}{\text{mat}}(\tilde{\nabla} v)\|_{2,1}}_{\text{TGV Regularizer}} + \underbrace{\sum_{k=1}^{K} \|u - f_k\|_1}_{\text{Dataterm}} \Bigg\}, \tag{3}$$

where $u \in \mathbb{R}^{MN}$ is the fused disparity map, $v = (v_x, v_y) \in \mathbb{R}^{2MN}$ is the final gradient map in x- and y-direction and $\alpha_0, \alpha_1 \in \mathbb{R}_{++}$ are strictly positive hyper-parameters of the model. $\|\cdot\|_{2,1}$ denotes the 2, 1 matrix norm, which is computed by taking the ℓ_2 norm across rows and the ℓ_1 norm across columns, i.e. $X \in \mathbb{R}^{M \times N}$, $\|X\|_{2,1} = \sum_{j=1}^{N} \left| \sqrt{\sum_{i=1}^{M} X_{ij}^2} \right|$. The operator $\nabla \in \mathbb{R}^{2MN \times MN}$ and $\tilde{\nabla} = \text{diag}(\nabla, \nabla) \in \mathbb{R}^{4MN \times 2MN}$ denote the discrete derivative operators for u and v, respectively.

3 Optimization

In order to compute the optimal values for u and v we need to perform an optimization. This can be done with the primal-dual algorithm. Given an with zeros initialized pair of primal points $u \in \mathbb{R}^{MN}, v \in \mathbb{R}^{2MN}$ and also with zeros initialized dual points $p \in \mathbb{R}^{2MN}, q \in \mathbb{R}^{4MN}$ and stepsizes $\tau, \sigma > 0$. Using the shortcuts $\mathbf{u} = (u, v)^T$ and $\mathbf{p} = (p, q)^T$, we need to perform the following iterates to optimize (3):

$$\begin{cases} \mathbf{u}^{n+\frac{1}{2}} = \mathbf{u}^n - \tau K^T \mathbf{p}^n \\ u^{n+1} = \operatorname{proj}_{\triangle}\left(u^{n+\frac{1}{2}}\right) \\ v^{n+1} = v^{n+\frac{1}{2}} \\ \mathbf{p}^{n+\frac{1}{2}} = \mathbf{p}^n + \sigma K(2\mathbf{u}^{n+1} - \mathbf{u}^n) \\ p^{n+1} = \operatorname{proj}_{\alpha_1 \circ}(p^{n+\frac{1}{2}}) \\ q^{n+1} = \operatorname{proj}_{\alpha_2 \circ}(q^{n+\frac{1}{2}}) \end{cases} \tag{4}$$

where $\operatorname{proj}_{\lambda\circ}$ denotes the projection to a ball, $\operatorname{proj}_{\triangle}$ is the projection for the dataterm and the operators $K \in \mathbb{R}^{6MN \times 3MN}$ is given by

$$K = \begin{pmatrix} \nabla_x & -I & \\ \nabla_y & & -I \\ & \nabla_x & \\ & \nabla_y & \\ & & \nabla_x \\ & & \nabla_y \end{pmatrix}, \tag{5}$$

where I is the identity matrix with size $MN \times MN$.

Furthermore the projection to the ball is given by

$$\operatorname{proj}_{\lambda\circ}(\hat{Y}) \Leftrightarrow Y(i) = \frac{\hat{Y}(i)}{\max\left(1, \frac{1}{\lambda} \cdot \|\hat{Y}(i)\|_2\right)}, \tag{6}$$

where $\|Y(i)\|_2$ is the pixel-wise ℓ_2 norm and $Y = \underset{2\times MN}{\operatorname{mat}}(p)$ or $Y = \underset{4\times MN}{\operatorname{mat}}(q)$, respectively. The projection for the dataterm [1] is slightly more involved and thus we provide this implementation for you.

Convergence of the algorithm In order for algorithm (4) to converge, the condition

$$\tau \sigma L^2 \leq 1 \tag{7}$$

has to be fulfilled. τ and σ are the stepsize for the primal and dual steps respectively. L is the operator norm of the discrete derivative operator, which is in our case given by

$$L^2 = 12. \tag{8}$$

4 Framework and Tasks

Your task is to implement TGV-Fusion as described in the previous sections. You can load the data with `np.load("observation?.npy")`, where "?" should be replaced with the ID of the observation. Additionally, we provide a framework where the following functions are already for you implemented: $\operatorname{prox}_{\triangle}$ and `make_nabla` to compute the sparse ∇ operator. Hint: You can use `scipy.sparse.bmat` to compute K.

- Implement algorithm (4)
- Implement a function to compute the energy (3)

- Compute the largest possible stepsizes for τ and σ (assuming $\tau = \sigma$) which fulfill the inequality (7)
- Implement the accuracy in order to evaluate your solution.

$$accX(u, u^*) = \frac{1}{Z} \sum_{i,j} m(i,j) \cdot \begin{cases} 1 & \text{if } |u(i,j) - u^*(i,j)| \le X \\ 0 & \text{else,} \end{cases} \tag{9}$$

  where u is the fused result after optimization, u^* is the ground-truth disparity map, m is the mask containing zeros for invalid pixels and ones for valid pixels and Z is the number of valid pixels.
- Choose three different combinations for α_1 and α_2. Run your implementation for 2000 iterations and plot the energy over the iterations.
- Report the accuracy for the different α.
- What is the effect of α_1? What is the effect of α_2?
- Plot the fused result in 2D as disparity map and in 3D similar to Fig. 2.
- Reshape the optimized variable v into two images and plot the results. How can you interpret the results?

References

[1] Yingying Li and Stanley Osher. A new median formula with applications to pde based denoising. *Communications in Mathematical Sciences*, 7(3):741–753, 2009.