Sparse Tracking and Dense Mapping

Liangzu Peng*, Xiaocheng Song*[1]
School of Information Science and Technology
ShanghaiTech University
penglz,songxch@shanghaitech.edu.cn

1 Introduction

The SLAM course has introduced to us several SLAM systems, including MonoSLAM [1], a SLAM system using the filtering approach, PTAM [2], which proposes to use two parallel threads for tracking and mapping respectively, ORB-SLAM [3], a recent success in SLAM community for its versatility and accuracy. Moreover, some (semi-)dense methods such as KinectFusion [4], DTAM [5], LSD-SLAM, Dynamic Fusion [6] are also covered.

In this course project, we aim at desktop modeling via *sparse tracking and dense mapping*, seemingly a mix of aforementioned approaches, where tracking part is done based on homework 2-3 and the mapping is achieved by the DTAM algorithm.

Our final products for this project are

- A working CPU[2] implementation of dense mapping that models the table dataset from the author's website [7][3].
- A complete report of what we have done, including 1) our understanding of the DTAM problem as described in §2, §3, where we point out the confusing part of the DTAM paper and thus it may be helpful for people with potential interest, 2) experimental results given by our implementation (§4). Finally, we give a short instruction on how to use our code in §5.

2 Problem Formulation

To begin with, we adopt the notations form [5]. The transformation $\boldsymbol{T}_{ff'} \in \mathbb{SE}(3)$ relates two frames f and f' so that $\boldsymbol{x} = \boldsymbol{T}_{ff'}\boldsymbol{x}'$ where $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^3$ are 3D points expressed in the frames f and f' respectively. We use $\pi(\boldsymbol{x}) = (x_1/x_3, x_2/x_3)^\top$ to denote the dehomogenization. For a keyframe r with world-camera frame transform $\boldsymbol{T}_{rw}$ is associated with an inverse depth map $\boldsymbol{\xi}_r : \Omega \to \mathbb{R}$ and an image $\boldsymbol{I}_r : \Omega \to \mathbb{R}^3$ where $\Omega \subset \mathbb{R}^2$ is the image domain. For a pixel $\boldsymbol{u} = (u_1, u_2)^\top \in \Omega$, we may back-project it to a 3D point via $\boldsymbol{x}_r = \pi^{-1}(\boldsymbol{u}, d) := (\boldsymbol{K}^{-1}\bar{\boldsymbol{u}})/d$, where $d = \boldsymbol{\xi}(\boldsymbol{u}), \bar{\boldsymbol{u}} = (u_1, u_2, 1)^\top$ and $\boldsymbol{K}$ is the intrinsic matrix.

For a given reference frame r, let $\mathcal{I}(r)$ be the set of frames nearby and overlapping the frame r. The dense mapping is achieved by iteratively estimating the depth $\boldsymbol{\xi}_r$ of the

[1] equal contribution, names listed in alphabetic order..

[2] although it is easy, if not trivial, to adopt our implementation on GPU with slight modification, we did not do with GPU since the only option is using GPUs on the server, which makes it inconvenient for us to debug (e.g., in that case we have to save all images on server and then download, observe them, instead of seeing the result immediately, on our laptop). Hence we avoid this complexity at a cost of running time.

[3] https://www.doc.ic.ac.uk/~ahanda/HighFrameRateTracking/traj_over_table.tgz

reference frame r, using the color information from $\boldsymbol{I}_r$ and images in $\mathcal{I}(r)$. To that end, one natural attempt may be to minimize the *photometric error* defined as

$$\boldsymbol{C}_r(\boldsymbol{u}, d) = \frac{1}{|\mathcal{I}(r)|} \sum_{m \in \mathcal{I}(r)} \left\| \boldsymbol{I}_r(u) - \boldsymbol{I}_m(\pi(\boldsymbol{K}\boldsymbol{T}_{mr}\pi^{-1}(\boldsymbol{u}, d))) \right\|_1, \tag{2.1}$$

where $\boldsymbol{T}_{mr}$ is given by the tracking algorithm[4]. Such an error 2.1 assumes the brightness constancy, and is only applicable to the collection of images within a relatively narrow region.

To enforce depth smoothness, a regularization term is added to (2.1), resulting the following energy

$$E_{\boldsymbol{\xi}_r} = \int_\Omega \{\lambda \boldsymbol{C}(\boldsymbol{u}, \boldsymbol{\xi}_r(\boldsymbol{u})) + g_r(\boldsymbol{u}) \left\| \nabla \boldsymbol{\xi}_r(\boldsymbol{u}) \right\|_\epsilon \} \mathrm{d}\boldsymbol{u}, \tag{2.2}$$

where $g_r(\boldsymbol{u}) = e^{-\alpha \|\nabla \boldsymbol{I}_r(\boldsymbol{u})\|_2^\beta}$ is the weight for each pixel, and $\|\cdot\|_\epsilon$ is the huber norm so that for a scalar $x \in \mathbb{R}$,

$$\|x\|_\epsilon = \begin{cases} x^2/(2\epsilon) & \text{if } |x| \le \epsilon, \\ |x| - \epsilon/2 & \text{otherwise.} \end{cases} \tag{2.3}$$

The huber norm of a vector is an elementwise application of (2.3).

The missing explanation for the symbol ∇ is given as follows. Let $\boldsymbol{I}$ be a $m \times n$ matrix, then $\nabla \boldsymbol{I} \in \mathbb{R}^{m \times n \times 2}$ is the differentiation operation such that [8]

$$\nabla \boldsymbol{I}(i, j) = (\nabla \boldsymbol{I})_{i,j} = ((\nabla \boldsymbol{I})^1_{i,j}, (\nabla \boldsymbol{I})^2_{i,j})^\top, \tag{2.4}$$

where

$$(\nabla \boldsymbol{I})^1_{i,j} = \begin{cases} \boldsymbol{I}_{i+1,j} - \boldsymbol{I}_{i,j} & \text{if } i < m, \\ 0 & \text{if } i = m, \end{cases} \tag{2.5}$$

$$(\nabla \boldsymbol{I})^2_{i,j} = \begin{cases} \boldsymbol{I}_{i,j+1} - \boldsymbol{I}_{i,j} & \text{if } j < n, \\ 0 & \text{if } j = n. \end{cases} \tag{2.6}$$

Observing that in (2.2) the data term is non-convex although the regularization is convex and smooth, it is difficult to directly optimize (2.2). As a workaround, a common practice [9] is to decouple the data term and the regularization such that the energy becomes

$$E_{\boldsymbol{\xi}_r, \boldsymbol{\alpha}} = \int_\Omega \{\lambda \boldsymbol{C}(\boldsymbol{u}, \boldsymbol{\alpha}(\boldsymbol{u})) + \frac{1}{2\theta}(\boldsymbol{\alpha}(\boldsymbol{u}) - \boldsymbol{\xi}_r(\boldsymbol{u}))^2 + g_r(\boldsymbol{u}) \left\| \nabla \boldsymbol{\xi}_r(\boldsymbol{u}) \right\|_\epsilon \} \mathrm{d}\boldsymbol{u}. \tag{2.7}$$

[4] we found that the motion between consecutive frames are very small and our sparse tracking begins to fail after about 10 frames, and the tracking results differ very much from the groundtruth. Hence we use the groundtruth pose directly and mainly focus on the dense mapping part. Nevertheless, we provide the code for sparse tracking; see §5.

3 Algorithms

It is then quite natural to employ alternating minimization to compute the minimizers of (2.7), i.e., to alternatively update $\boldsymbol{\xi}_r$ (with $\boldsymbol{\alpha}$ fixed) and $\boldsymbol{\alpha}$ (with $\boldsymbol{\xi}_r$ fixed) for decreasing the energy. Given $\boldsymbol{\xi}_r$, (2.7) is non-convex in $\boldsymbol{\alpha}$, but it has been shown that the brute-force search over the (discrete) space of $\boldsymbol{\alpha}$ is a possible real-time solution on GPU. On the other hand, with $\boldsymbol{\alpha}$ known, (2.7) is convex in $\boldsymbol{\xi}_r$, which allows for convex optimization algorithms to efficiently compute an optimal $\boldsymbol{\xi}_r$. To have a suitable initialization for such an alternating minimization strategy, [5] takes

$$\boldsymbol{\alpha}^0(\boldsymbol{u}) = \boldsymbol{\xi}_r^0(\boldsymbol{u}) = \operatorname*{argmin}_{\boldsymbol{\alpha}} \boldsymbol{C}_r(\boldsymbol{u}, \boldsymbol{\alpha}(\boldsymbol{u})), \tag{3.1}$$

and we follow this initialization scheme. Note that when updating $\boldsymbol{\alpha}$ in each iteration, the brute-force search will find the minimum of the function $\int_\Omega \{\lambda \boldsymbol{C}(\boldsymbol{u}, \boldsymbol{\alpha}(\boldsymbol{u})) + \frac{1}{2\theta}(\boldsymbol{\alpha}(\boldsymbol{u}) - \boldsymbol{\xi}_r(\boldsymbol{u}))^2 \mathrm{d}\boldsymbol{u}$, but it will not alter the cost volume, a data structure entirely depending on the inputs. Hence the value of $\boldsymbol{\xi}_r(\boldsymbol{u})$ has main control on the minimum of the above function since only[5] $\boldsymbol{\xi}_r$ changes in each iteration. It seems a drawback because the information given by the regularization is not fully used. Could it be used to inform how the cost volume should improve itself? We leave this problem for future research.

To describe the more involved algorithm for minimizing

$$E^1_{\boldsymbol{\xi}_r} = \int_\Omega \{\frac{1}{2\theta}(\boldsymbol{\alpha}(\boldsymbol{u}) - \boldsymbol{\xi}_r(\boldsymbol{u}))^2 + g_r(\boldsymbol{u}) \left\|\nabla \boldsymbol{\xi}_r(\boldsymbol{u})\right\|_\epsilon\} \mathrm{d}\boldsymbol{u}, \tag{3.2}$$

we need to first change our notations. Assuming $\boldsymbol{\xi}_r$ is of size $m \times n$, we rewrite (3.2) into the discrete matrix form (instead of integral):

- The first term $\int_\Omega \{(\boldsymbol{\alpha}(\boldsymbol{u}) - \boldsymbol{\xi}_r(\boldsymbol{u}))^2/(2\theta)\} \mathrm{d}\boldsymbol{u}$ is simply $\left\|\boldsymbol{a} - \boldsymbol{d}_r\right\|_2^2/(2\theta)$, where $\boldsymbol{a}, \boldsymbol{d}_r \in \mathbb{R}^{mn}$ are the vector version of $\boldsymbol{\alpha}$ and $\boldsymbol{\xi}_r$ respectively.
- As indicated in [5], the second term can be written as $\left\|\boldsymbol{A}\boldsymbol{G}_r\boldsymbol{d}_r\right\|_\epsilon$[6], where $\boldsymbol{G}_r \in \mathbb{R}^{mn \times mn}$ is the diagonal weight matrix that corresponds to g_r and weights $\boldsymbol{d}_r$, and $\boldsymbol{A}$ models the linear transformation in (2.4) so that $\boldsymbol{A}\boldsymbol{G}_r\boldsymbol{d}_r \in \mathbb{R}^{2mn}$ is the gradient vector. However, what mentioned above is a little bit confusing because 1) some terms of $\boldsymbol{G}_r$ will be squared due to the ℓ_2 norm part of the huber norm, and 2) ϵ will not remain the same and indeed the new threshold for the huber norm becomes a function of $\boldsymbol{g}_r$ and the old ϵ. Anyway, we will follow the paper and rewrite this term as $\left\|\boldsymbol{A}\boldsymbol{G}_r\boldsymbol{d}_r\right\|_\epsilon$. We note that this will not influence the result too much from engineering's perspective, since we only need to take care of choosing ϵ.

To conclude, (3.2) may be written as

$$E^1_{\boldsymbol{d}_r} = \frac{1}{2\theta}\left\|\boldsymbol{a} - \boldsymbol{d}_r\right\|_2^2 + \left\|\boldsymbol{B}_r\boldsymbol{d}_r\right\|_\epsilon, \tag{3.3}$$

[5] λ may also be dynamically changed in each iteration.

[6] this rewriting is trivial in view of (2.4) if $\boldsymbol{G}_r$ is the identity matrix.

where $\boldsymbol{B}_r = \boldsymbol{A}\boldsymbol{G}_r$. The energy $E^1_{\boldsymbol{d}_r}$ may be considered as an instance of regularized least-squares problem. Indeed, (3.3) can be thought of as a special case of a more general convex optimization problem, where the energy is given by

$$G(\boldsymbol{x}) + F(\boldsymbol{K}\boldsymbol{x}), \tag{3.4}$$

with respect to $\boldsymbol{x}$, where $\boldsymbol{K}$ is a linear transformation, and G, F are *simple* functions in the sense that their *proximal operator* [10] has a closed-form representation (or can be computed efficiently up to a high precision) [8]. In the following we shortly review the algorithm given by [8], and then go back to solve (3.3) by employing that algorithm.

3.1 Algorithm in [8]: A Review

(3.4) is called the *prime* problem. To apply the algorithm in [5, 8] we derive its prime-dual formulation based on lagrange multipliers.

Note that minimizing (3.4) is equivalent to the following problem,

$$\min_{\boldsymbol{x},\boldsymbol{z}}\{G(\boldsymbol{x}) + F(\boldsymbol{z})\} \tag{3.5}$$

$$\text{subject to } \boldsymbol{z} = \boldsymbol{K}\boldsymbol{x}. \tag{3.6}$$

Then its lagrangian is given by

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{y}) = G(\boldsymbol{x}) + F(\boldsymbol{z}) + \boldsymbol{y}^\top(\boldsymbol{K}\boldsymbol{x} - \boldsymbol{z}). \tag{3.7}$$

An observation here is that the original problem (3.4) is equivalent to the following optimization problem, namely

$$\begin{aligned}
&\max_{\boldsymbol{y}}\{\min_{\boldsymbol{x},\boldsymbol{z}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{y})\} \\
\iff &\max_{\boldsymbol{y}}\{\min_{\boldsymbol{x},\boldsymbol{z}}\{G(\boldsymbol{x}) + F(\boldsymbol{z}) + \boldsymbol{y}^\top(\boldsymbol{K}\boldsymbol{x} - \boldsymbol{z})\}\} \\
\iff &\max_{\boldsymbol{y}}\{\min_{\boldsymbol{x}}\{G(\boldsymbol{x}) + \boldsymbol{y}^\top\boldsymbol{K}\boldsymbol{x}\} + \min_{\boldsymbol{z}}\{-(\boldsymbol{y}^\top\boldsymbol{z} - F(\boldsymbol{z}))\}\} \\
\iff &\max_{\boldsymbol{y}}\{\min_{\boldsymbol{x}}\{G(\boldsymbol{x}) + \boldsymbol{y}^\top\boldsymbol{K}\boldsymbol{x}\} - F^*(\boldsymbol{y})\} \\
\iff &\max_{\boldsymbol{y}}\min_{\boldsymbol{x}}\{G(\boldsymbol{x}) + \boldsymbol{y}^\top\boldsymbol{K}\boldsymbol{x} - F^*(\boldsymbol{y})\} \\
\iff &\min_{\boldsymbol{x}}\max_{\boldsymbol{y}}\{G(\boldsymbol{x}) + \boldsymbol{y}^\top\boldsymbol{K}\boldsymbol{x} - F^*(\boldsymbol{y})\} \text{ (strong duality)}
\end{aligned} \tag{3.8}$$

where F^* is the *Fenchel conjugate* of F (such that $F^*(\boldsymbol{y}) = \max_{\boldsymbol{z}}\{\boldsymbol{y}^\top\boldsymbol{z} - F(\boldsymbol{z})\}$). Such an observation holds because maximizing $\boldsymbol{y}$ enforces $\boldsymbol{K}\boldsymbol{x} - \boldsymbol{z} = 0$, otherwise the energy will go to positive infinity. (3.8) is the prime-dual representation of the original problem, also known as the *saddle-point* problem [11].

To solve (3.8), [8] proposed the following algorithm

1. initialize $\boldsymbol{x}^0, \boldsymbol{y}^0, \tau > 0, \sigma > 0$.
2. $\boldsymbol{y}^{n+1} = \text{prox}_{\text{F}^*}(\boldsymbol{y}^n + \sigma\boldsymbol{K}\boldsymbol{x}^n)$.

3. $\boldsymbol{x}^{n+1} = \text{prox}_{\text{G}}(\boldsymbol{x}^n - \tau \boldsymbol{K}^\top \boldsymbol{y}^{n+1})$.

4. repeat steps 2 and 3 until convergence.

In the above algorithm $\text{prox}_h(x)$ represents the proximal operator of the function h at a point x. The convergence of this algorithm is guaranteed when $\tau\sigma \left\|\boldsymbol{K}\right\|_2^2 < 1$.

3.2 Solving (3.3) (as in [5])

Let $G(\boldsymbol{d}_r) = \frac{1}{2\theta} \left\|\boldsymbol{a} - \boldsymbol{d}_r\right\|_2^2$ and $F(\boldsymbol{d}_r) = \left\|\boldsymbol{B}_r \boldsymbol{d}_r\right\|_\epsilon$. We need to compute the Fenchel conjugate of $F(\boldsymbol{d}_r)$. Following [8], we have $F^*(\boldsymbol{d}_r) = \delta_q(\boldsymbol{q}) + \frac{\epsilon}{2} \left\|\boldsymbol{q}\right\|_2^2$[7], where $q = \{\boldsymbol{q} : \left\|\boldsymbol{q}\right\|_\infty \leq 1\}$ and δ is the indicator function on q such that

$$\delta_q(\boldsymbol{q}) = \begin{cases} \epsilon/2, & \text{if } \boldsymbol{q} \in q, \\ \infty, & \text{otherwise.} \end{cases} \tag{3.9}$$

A typo[8] made in [5] is that they wrote the set q as $q = \{\boldsymbol{q} : \left\|\boldsymbol{q}\right\|_1 \leq 1\}$, not aware of the correct norm here should be the *dual* norm of the ℓ_1 norm, which is the ℓ_∞ norm. The computation of Fenchel conjugate results the following form of our energy:

$$\begin{aligned} &\min_{\boldsymbol{d}_r} \max_{\boldsymbol{q}} \{E_{\boldsymbol{q},\boldsymbol{d}_r}\}, \text{ where} \\ &E_{\boldsymbol{q},\boldsymbol{d}_r} = \frac{1}{2\theta} \left\|\boldsymbol{a} - \boldsymbol{d}_r\right\|_2^2 + (\boldsymbol{B}_r \boldsymbol{d}_r)^\top \boldsymbol{q} - \delta_q(\boldsymbol{q}) - \frac{\epsilon}{2} \left\|\boldsymbol{q}\right\|_2^2. \end{aligned} \tag{3.10}$$

After having the closed form for the Fenchel conjugate, what remains is to compute the proximal operator. More conveniently, since the gradient information is available, [5] instead uses a gradient ascent/descent approach, described below.

0. Notice that $\frac{\partial E_{\boldsymbol{q},\boldsymbol{d}_r}}{\partial \boldsymbol{q}} = \boldsymbol{B}_r \boldsymbol{d}_r - \epsilon \boldsymbol{q}$ and $\frac{\partial E_{\boldsymbol{q},\boldsymbol{d}_r}}{\partial \boldsymbol{d}_r} = \boldsymbol{B}_r^\top \boldsymbol{q} + \frac{1}{\theta}(\boldsymbol{d}_r - \boldsymbol{a})$.

1. initialize $\boldsymbol{d}_r^0 = \boldsymbol{a}, \boldsymbol{q}^0 = \boldsymbol{0}, \sigma_{\boldsymbol{d}_r} > 0, \sigma_{\boldsymbol{q}} > 0$.

2. compute $\boldsymbol{q}^{n+1}$ such that

$$\frac{\boldsymbol{q}^{n+1} - \boldsymbol{q}^n}{\sigma_{\boldsymbol{q}}} = \boldsymbol{B}_r \boldsymbol{d}_r^n - \epsilon \boldsymbol{q}^{n+1} \text{ (gradient ascend).} \tag{3.11}$$

3. project $\boldsymbol{q}^{n+1}$ onto the set q via $\Pi_{\boldsymbol{q}}(x) = x / \max(1, \left\|x\right\|_2)$. This projection step accounts for the existence of the indicator function δ_q in (3.10).

4. compute $\boldsymbol{d}^{n+1}$ such that

$$\frac{\boldsymbol{d}_r^{n+1} - \boldsymbol{d}_r^n}{\sigma_{\boldsymbol{d}_r}} = -\boldsymbol{B}_r^\top \boldsymbol{q}^{n+1} - \frac{1}{\theta}(\boldsymbol{d}_r^{n+1} - \boldsymbol{a}) \text{ (gradient descent).} \tag{3.12}$$

[7] we are not convinced by [8] that the Fenchel conjugate is that one (they did not explain); we feel that it is not true: the Fenchel conjugate of ℓ_1 norm is that indicator function, and the Fechel conjugate of ℓ_2 norm is $\frac{\epsilon}{2} \left\|\boldsymbol{q}\right\|_2^2$ as above. Hence the Fechel conjugate of the huber norm just can not be that one, i.e., the sum of the conjugate of ℓ_1 and ℓ_2 norm.

[8] several other typos were made in terms of mathematical writing.

5. repeat steps 2-4 until convergence.

We note another tricky thing in the above algorithm. Step 4 in the algorithm will give the update rule

$$\boldsymbol{d}_r^{n+1} = (1 + \frac{\theta}{\sigma_{\boldsymbol{d}_r}})(\boldsymbol{d}_r^n + \sigma_{\boldsymbol{d}_r}(\frac{\boldsymbol{\alpha}}{\theta} - \boldsymbol{B}_r^\top \boldsymbol{q}^{n+1})). \tag{3.13}$$

When we did our implementation following (3.13), the algorithm did not converge. On the other hand, in [5] another typo was observed by us: they wrote

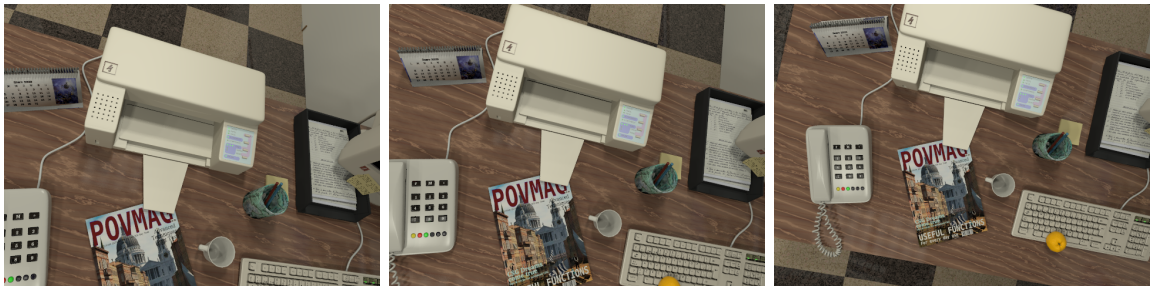
$$\boldsymbol{d}_r^{n+1} = (1 + \frac{\theta}{\sigma_{\boldsymbol{d}_r}})(\boldsymbol{d}_r^n + \sigma_{\boldsymbol{d}_r}(\frac{\boldsymbol{\alpha}}{\theta} + \boldsymbol{B}_r^\top \boldsymbol{q}^{n+1})). \tag{3.14}$$

The algorithm then converges with the update rule (3.14). We think the problem here might be that the derived gradient descent/ascend approach does not follow exactly [8], where the proximal operator is computed. Perhaps (3.14) is the desired result by computing the proximal operator (we are just too lazy to compute and compare the results. Anyway, the algorithm works).

4 Experiments

4.1 Dataset.

As mentioned before, the dataset is taken from this link. This synthetic dataset satisfies the assumptions of the DTAM algorithm: 1) brightness constancy and 2) narrow-baseline frames. Some example images are shown in Figure 1.



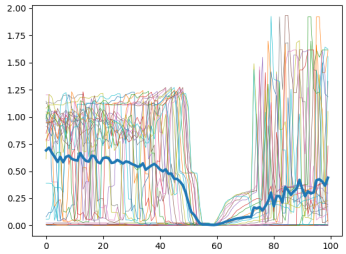
(a) The reference frame. (b) The frame 8. (c) The frame 16.

Figure 1

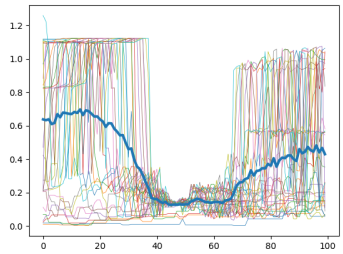
4.2 Cost Volume Computation

We plot the photometric error as a function of inverse depth at some points in the reference image. 6 points are colored as in Figure 2. Figure 3 depicts the photometric errors for the points on the phone 3a, table 3b, corner 3c, and the edge 3d. Such error curves are in agreement with the results in [5].

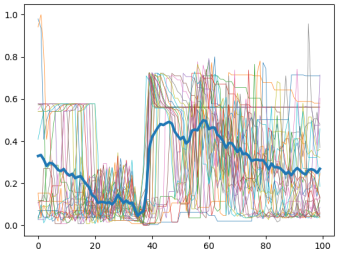
Figure 2: The reference image with 6 colored points on the phone (black), table (yellow), corner (purple), edge (green), printer (red), and wall (blue).



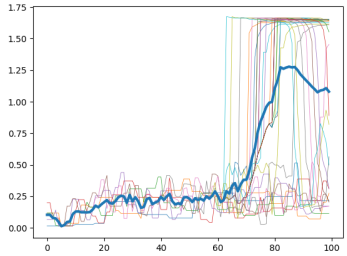
(a) Phone (black)



(b) Table (yellow)



(c) Corner (purple)

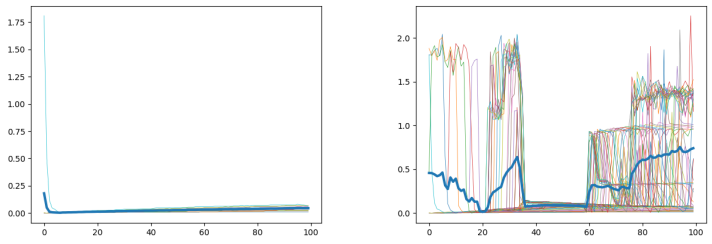


(d) Edge (green)

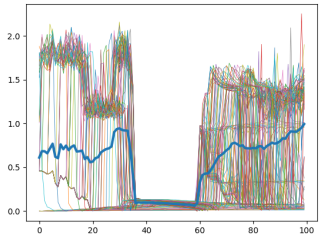
Figure 3: Photometric error as a function of the inverse depth.

Next, Figure 4 shows how the cost volume improves as more frames are added for the red point on the pointer in Figure 2. When there are only 10 frames (4a), the photometric errors for the chosen depth range are very close to each other. Because there are many points on the epipolar line (of another frame) that have nearly the same color as the chosen point on the printer, hence the algorithm seems to have difficulty to deal with large texture-less areas for example this printer. However, as we show in Figures 4b-4c, if the number of frames are adequate, the error curve becomes more distinguishable and the minimum tends to be unique.

Finally, we present a failure case in Figure 5, which shows the photometric errors of the blue point in Figure 2. This point has large depth and hence small inverse depth. However, the photometric errors of that point for small inverse depths are constantly 3, a default value meaning that the reprojected point is out of the image. In that case the algorithm, which



(a) Printer (red), 10 frames (b) Printer (red), 50 frames



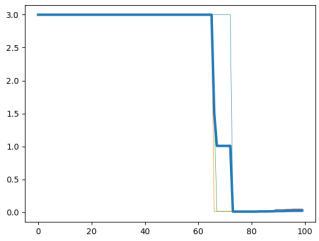
(c) Printer (red), 100 frames

Figure 4: Photometric error as a function of the inverse depth.

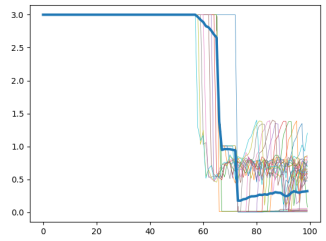
takes the inverse depth that causes the minimal photometric error, gives a wrong prediction of the inverse depth. This happens because the dataset does not contain sufficient observations of the blue point, or more generally, some boundary on the image; see Figure 1 for having a sense of the motion for this dataset and see also Figure 6a. Interestingly, to overcome this issue, the motion shown in the DTAM paper ensures every point in the reference image has been observed sufficiently; see their demo video on Youtube (this link). We tried some hacks to fix this problem, quickly feeling that things were becoming less interesting.

4.3 Regularization Result

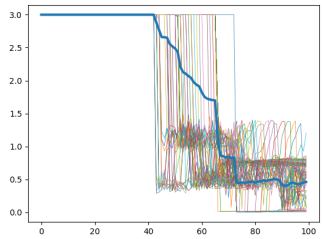
We present our regularization result in Figure 6. The regularization makes the prediction much smoother, and is able to reduce the influence of some poor local minima introduced by the cost volume computation. However, it seems incapable to regularize the edges where the gradient is large (Figure 6b) because the weight matrix is simply defined this way: give less weight for larger gradient.



(a) Wall (blue), 10 frames



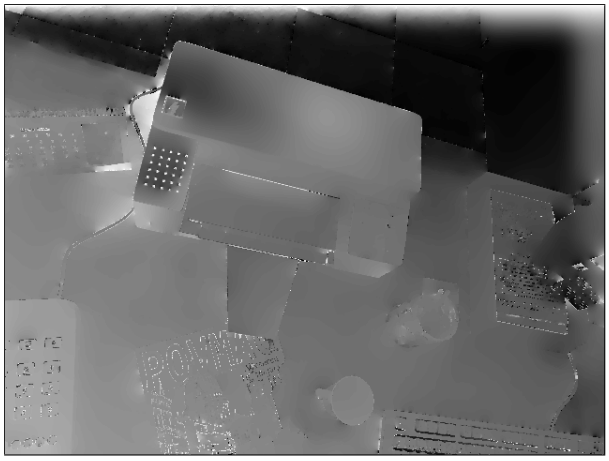
(b) Wall (blue), 50 frames



(c) Wall (blue), 100 frames

Figure 5: Photometric error as a function of the inverse depth.

(a) Before regularization



(b) After Regularization

Figure 6: Evaluation of the regularization algorithm.

5 Code

In the final section we describe our code structure and basic usage.

5.1 Physical Structure of the Program

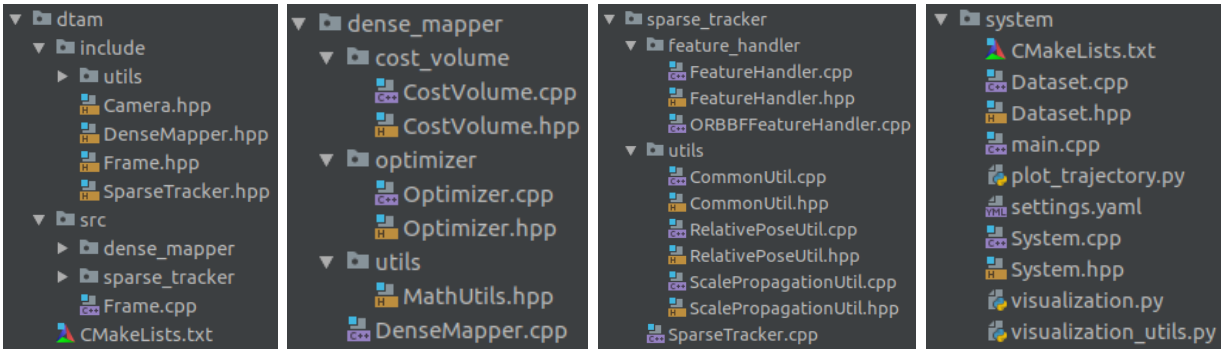


Figure 7: Code Structure

The project consists of the core part in `dtam` folder which contains the implementation of dense mapping and sparse tracking algorithms, and an example in `system` folder which runs the whole procedure on a given dataset. The code organization is shown in Figure 7.

In the `dtam` folder, the class `Camera` and `Frame` are encapsulations of the cameras and image frames. The class `DenseMapper` and `SparseTracker` implement respectively the dense mapping algorithm as in [5] and sparse tracking similar to our homework. The `DenseMapper` uses 1) the `CostVolume` to store the average cost data with different inverse depth hypothesis and 2) the `Optimizer` for the regularization algorithm. We implemented the entire 7-points RANSAC pipeline with the metric being the reprojection error in `RelativePoseUtil`, and also did a sparse tracking implementation based on OpenCV. The scale propagation is implemented in `ScalePropagationUtil`. The ORB feature and brute force matching are implemented using OpenCV functionality in `FeatureHandler` and `ORBBFFeatureHandler`.

In the `system` folder, the class `Dataset` reads the images and ground truth poses from the dataset, and the class `System` performs a whole procedure which consecutively read images and add them into `DenseMapper` and/or `SparseTracker`. The mapping and tracking may be running separately with minor modification of the code. The `main.cpp` is the launcher of this program, which receives the dataset path as its argument and starts a `System` instance. We have three python scripts `plot_trajectory.py`, `visualization.py` and `visualization_util.py` is used for plotting and debugging purpose.

5.2 Instructions on usage

Our code relies on the following third-party libraries.

- `Eigen 3.7+` for matrix operation.
- `OpenCV 3.2+` for feature extraction, matching and relative pose.
- `cnpy` for serializing the cost volume to .npy file with C++.

- `Zlib` is a dependence library of cnpy.

The `Cmake 3.14+` and compilers which support `c++17` including std::filesystem is required to make and compile this project. The command line way is

```
cd <path/to/project_root>
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
cmake --build . --target example
```

The dataset is taken from the homepage of A. Handa and can be download in https://www.doc.ic.ac.uk/~ahanda/HighFrameRateTracking/downloads.html (The one named *Camera moving above table*). The data folder must include images and ground truths named `scene_XXX.png`, `scene_XXX.txt`. For other datasets, only the file `Dataset.cpp` needs to be changed to make it work. Run the compiled executable program with the following usage.

```
cd system
example <path/to/data_folder>
```

References

[1] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, June 2007.

[2] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, Nov 2007.

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015.

[4] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, Oct 2011.

[5] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327, Nov 2011.

[6] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, June 2015.

[7] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison. Real-time camera tracking: When is high frame-rate best? In *Proc. of the European Conference on Computer Vision(ECCV)*, Oct. 2012.

[8] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, May 2011.

[9] F. Steinbrücker, T. Pock, and D. Cremers. Large displacement optical flow computation withoutwarping. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1609–1614, Sep. 2009.

[10] A. Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017.

[11] Michele Benzi, Gene H Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta numerica*, 14:1–137, 2005.