Online Algorithms for Factorization-Based Structure from Motion

Ryan Kennedy
University of Pennsylvania

Laura Balzano
University of Michigan

Stephen J. Wright
University of Wisconsin

Camillo J. Taylor
University of Pennsylvania

Abstract

We present a family of online algorithms for real-time factorization-based structure from motion, leveraging a relationship between the incremental singular value decomposition and recent work in online matrix completion. Our methods are orders of magnitude faster than previous state of the art, can handle missing data and a variable number of feature points, and are robust to noise and sparse outliers. Experiments show that they perform well in both online and batch settings. We also provide an implementation which is able to produce 3D models in real time using a laptop with a webcam.

1. Introduction

The problem of structure from motion — recovering the 3D structure of an object and locations of the camera from a monocular video stream — has been studied extensively in computer vision. For rigid scenes, many algorithms are based on the seminal work of Tomasi and Kanade [32], in which it was shown that a noise-free measurement matrix of point tracks has rank at most 3 for an affine camera when the data are centered at the origin. The 3D locations of all tracked points and camera positions can be easily obtained from a factorization of this matrix. Due to occlusion, however, many matrix entries are typically missing and standard matrix factorization techniques can no longer be applied.

Recent work in *low-rank matrix completion* has explored conditions under which the missing entries in a low-rank matrix can be determined, even when the matrix is corrupted with noise or sparse outliers [12, 13, 29]. Most algorithms for matrix completion are static in nature, working with a "batch" of matrix columns. In this paper, by contrast, we focus on *online* structure from motion, in which the 3D model of point locations must be updated in real time as the video is taken. The algorithm must be efficient enough to run in real time, yet still must deal effectively with missing data and noisy observations. The online problem has received little attention in comparison to batch algorithms, although online algorithms have been developed for matrix completion [3] that have shown promise in other real-time computer vision applications [21]. In this paper, we extend these online algorithms to the problem of rigid structure from motion. Our algorithms naturally address difficulties that have been observed in online rigid structure from motion: (1) our method is inherently online, (2) we naturally deal with data that are offset from the origin, (3) we directly deal with missing data, (4) we are able to handle a dynamically changing number of features, (5) our algorithms can be made robust to outliers, and (6) our method is extremely fast.

In addition to testing with online data, we compare our methods to batch algorithms, showing that they are competitive with – and often orders of magnitude faster than – state-of-the-art batch algorithms. To demonstrate the utility of our approach, we have also implemented our algorithm using a laptop, and we show that it is able to create 3D models of objects in real-time using video from an attached webcam.

2. Related Work

2.1. Structure From Motion

Much research on the rigid structure-from-motion problem is based on Tomasi and Kanade's factorization algorithm [32] for orthographic cameras, and subsequent work [28] that extended its applicability to other camera models [1, 8, 17, 18, 26, 30]. Comparatively little work has been done on *online* structure from motion, apart from algorithms that employ batch methods or local bundle adjustment in an online framework [23, 27].

In [26], Mortia and Kanade proposed a sequential version of the factorization algorithm but cannot deal with missing data, nor can they handle outliers or a dynamically changing set of features. The approach of McLauchlan and Murray [25] can handle missing data but uses simplifying heuristics to achieve a low computational complexity. The related algorithm of Trajković and Hedley [33] dispenses with the heuristics; they focus on tracking moving objects within a scene. More recently, Cabral et al. [10] proposed a method that performs matrix completion iteratively, in an online manner. However, it is difficult to dynamically add new features, and their approach can be numerically unstable. The most similar algorithm to our own is the incremental SVD (ISVD) approach of Bunch and Nielsen [9], which has been previously used for sequential structure from motion by Brand [6]. In Section 4.2 we show how ISVD is related to our algorithms.

2.2. Matrix Completion and Subspace Tracking

Low-rank matrix completion is the problem of recovering a low-rank matrix from an incomplete sample of the entries. It was shown in [11, 29] that under assumptions on the number of observed entries and on incoherence of the singular vectors of this matrix, the convex nuclear norm minimization problem solves the NP-hard rank minimization problem exactly. Since this breakthrough, a flurry of research activity has centered around developing faster algorithms for this convex optimization problem, both exact and approximate; see [22, 31] for just a couple. The online algorithm Grouse [3] (Grassmannian Rank-One Update Subspace Estimation) outperforms all non-parallel algorithms in computational efficiency, often by an order of magnitude, while remaining competitive in terms of estimation error.

The Grouse algorithm was also developed for low-dimensional subspace tracking, which has been an active area of research. Comprehensive reference lists for complete-data adaptive methods for tracking subspaces and extreme singular values and vectors of covariance matrices can be found in [15]; the authors discuss methods from the matrix computation literature as well as gradient-based methods from the signal processing literature.

A useful extension of these algorithms is "Robust PCA," in which we seek to recover a low-rank matrix in the presence of outliers. This work has had application in computer vision, decomposing a scene from several video frames as the sum of a low-rank matrix of background (which represents the global appearance and illumination of the scene) and a sparse matrix of moving foreground objects [13, 24]. The algorithm Grasta [20, 21] (Grassmannian Robust Adaptive Subspace Tracking Algorithm) is a robust extension of Grouse that performs online estimation of a changing low-rank subspace while subtracting outliers.

3. Problem Description

Given a set of points tracked through a video, the *measurement matrix* W is defined as

$$W = \begin{bmatrix} x_{1,1} & \dots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \dots & x_{n,m} \end{bmatrix}, \tag{1}$$

where $x_{i,j} \in \mathbb{R}^{1\times 2}$ is the projection of point i in frame j, giving W a dimension of $n \times 2m$, where n is the number of points and m is the number of frames. If we assume that the data are centered at the origin, then in the absence of noise this matrix has rank at most 3 and can be factored as

$$W = \tilde{S}\tilde{M}^T, \tag{2}$$

where $\tilde{S}$ is the $n \times 3$ *structure* matrix containing 3D point locations, while $\tilde{M}$ is the $2m \times 3$ *motion* matrix, which is the set of affine camera matrices corresponding to each frame [32]. Data that are offset from the origin have an additional translation vector τ; we can write

$$W = \begin{bmatrix} \tilde{S} & 1 \end{bmatrix} \begin{bmatrix} \tilde{M} & \tau \end{bmatrix}^T = SM^T, \tag{3}$$

where W now has rank at most 4 and the constant ones vector is necessarily part of its column space. We use this fact to naturally deal with offset data in our algorithms (Section 5.1).

In the presence of noise, one can use the singular value decomposition (SVD) [32] in order to find S and M such that SM^T most closely approximates W in either the Frobenius or operator norm. In this paper we address the problem of online structure from motion in which we have an estimated factorization at time t, $\hat{W}_t = \hat{S}_t\hat{M}_t^T$, and wish to update our estimate at time $t+1$ as we track points into the next video frame. The new information takes the form of two additional columns for W_t, leading to successive updates of the form $W_{t+1} = \begin{bmatrix} W_t & v_t \end{bmatrix}$. The algorithm for updating the approximate factorization must be efficient enough to be used in real time, able to handle missing data as points become occluded, and able to incorporate new points.

4. Matrix Completion Algorithms for SFM

4.1. The Grouse Algorithm [3]

Here we review the Grouse algorithm in the context of structure from motion in order to set the stage for our proposed algorithms, which are given in Section 4.2.

In the following we introduce the subscript t to denote values at time t. Let n_t be the number of tracks started up to time t and $U_t \in \mathbb{R}^{n_t \times 4}$ be an orthogonal matrix whose columns span the rank-4 column space of the measurement matrix W_t, and let v_t be a new column such that $W_{t+1} = \begin{bmatrix} W_t & v_t \end{bmatrix}$. (We consider the two new columns provided by each video frame one at a time.) The indices of v_t that are observed are given by $\Omega_t \subseteq \{1, \dots, n_t\}$ such that v_{Ω_t} and U_{Ω_t} are row submatrices of v_t and U_t corresponding to Ω_t.

The Grouse algorithm [3] measures the error between the current subspace-spanning matrix U_t and the new vector v_t using the ℓ_2 distance

$$\mathcal{E}(U_{\Omega_t}, v_{\Omega_t}) = \|U_{\Omega_t}w_t - v_{\Omega_t}\|_2^2, \tag{4}$$

where[1]

$$w_t = U_{\Omega_t}^{+} v_{\Omega_t} \tag{5}$$

is the set of weights that project v_{Ω_t} orthogonally onto the subspace given by U_{Ω_t}. We would like to replace U_t by an updated orthogonal matrix of the same dimensions, but which reduces the error $\mathcal{E}$ in accordance with the new observation v_t. Denoting by r_t the residual vector for the latest observation, we define the subvector corresponding to the indices in Ω_t by

$$r_{\Omega_t} = v_{\Omega_t} - U_{\Omega_t} w_t, \tag{6}$$

and set the components of r_t whose indices are not in Ω_t to zero. We can express the sensitivity of the error $\mathcal{E}$ to U_t as

$$\frac{\partial \mathcal{E}}{\partial U_t} = -2 r_t w_t^T. \tag{7}$$

Grouse essentially performs projected gradient descent on the Grassmann manifold, taking a step in the negative gradient direction and then projecting onto this manifold to maintain orthonormality. For details, see [3].

Because it is known *a priori* that $\text{rank}(W) \leq 4$, Grouse can be applied directly to the factorization problem, but there are several issues that prevent Grouse from being easily used for online structure from motion. First, Grouse only maintains an estimate of the column space of W_t, so the corresponding matrix $R_t \in \mathbb{R}^{2m_t \times 4}$ for which $\hat{W}_t = U_t R_t^T$ must be computed whenever a final reconstruction is needed. This can be a problem for online applications since it requires keeping all data until the algorithm is complete. Additionally, the choice of a critical parameter in Grouse — the step size for gradient descent — can affect the rate of convergence strongly. In the following section, we take advantage of a recent result on the relationship between Grouse and incremental SVD to resolve these difficulties.

4.2. The Incremental SVD (ISVD) Formulation

The incremental SVD algorithm [9] is a simple method for computing the SVD of a collection of data by updating an initial decomposition one column at a time. Given a matrix W_t whose SVD is $W_t = U_t \Sigma_t V_t^T$, we wish to compute the SVD of a new matrix with a single column added: $W_{t+1} = \begin{bmatrix} W_t & v_t \end{bmatrix}$. Defining $w_t = U_t^T v_t$ and $r_t = v_t - U_t w_t$, we have

$$W_{t+1} = \begin{bmatrix} U_t & \frac{r_t}{\|r_t\|} \end{bmatrix} \begin{bmatrix} \Sigma_t & w_t \\ 0 & \|r_t\| \end{bmatrix} \begin{bmatrix} V_t^T & 0 \\ 0 & 1 \end{bmatrix}, \tag{8}$$

where one can verify that the left and right matrices are still orthogonal. We compute an SVD of the center matrix:

$$\begin{bmatrix} \Sigma_t & w_t \\ 0 & \|r_t\| \end{bmatrix} = \hat{U} \hat{\Sigma} \hat{V}^T, \tag{9}$$

[1] $A^{+}b$ denotes the least-residual-norm solution to the least-squares problem $\min_x \|Ax - b\|_2^2$, obtainable from a singular value decomposition of A or a factorization of $A^T A$.

which yields the new SVD, $W_{t+1} = U_{t+1} \Sigma_{t+1} V_{t+1}^T$, where

$$U_{t+1} = \begin{bmatrix} U_t & \frac{r_t}{\|r_t\|} \end{bmatrix} \hat{U}; \Sigma_{t+1} = \hat{\Sigma}; V_{t+1} = \begin{bmatrix} V_t & 0 \\ 0 & 1 \end{bmatrix} \hat{V}. \tag{10}$$

If only the top k singular vectors are needed, then as a heuristic we can drop the smallest singular value and the associated singular vector after each such update. In the case of missing data when only entries $\Omega_t \subseteq \{1, \ldots, n_t\}$ are observed, we can define the weights and residual vector in these update formulae as in Equations (5) and (6), respectively.

We now examine the relationship of this algorithm to Grouse, in the context of SFM. Let $\hat{W}_t = U_t R_t^T$ be an estimated rank-4 factorization of W_t such that U_t has orthonormal columns. Given a new column v_t with observed entries Ω_t, if w_t and r_t are the least-squares weight and residual vector, respectively, defined with respect to the set of observed indices Ω_t as in Equations (5) and (6), then we can write

$$\begin{bmatrix} U_t R_t^T & \tilde{v}_t \end{bmatrix} = \begin{bmatrix} U_t & \frac{r_t}{\|r_t\|} \end{bmatrix} \begin{bmatrix} I & w_t \\ 0 & \|r_t\| \end{bmatrix} \begin{bmatrix} R_t & 0 \\ 0 & 1 \end{bmatrix}^T. \tag{11}$$

where the subvector of $\tilde{v}_t$ corresponding to Ω_t is set to v_{Ω_t}, while the remaining entries in v_t are imputed as the inner product of w_t and the rows $i \notin \Omega_t$ of U_t. Define the SVD of the center matrix to be

$$\begin{bmatrix} I & w_t \\ 0 & \|r_t\| \end{bmatrix} = \hat{U} \hat{\Sigma} \hat{V}^T. \tag{12}$$

In [4], it was shown that updating U_t to

$$U_{t+1} = \begin{bmatrix} U_t & \frac{r_t}{\|r_t\|} \end{bmatrix} \hat{U} \tag{13}$$

and subsequently dropping the last column is equivalent to Grouse for a particular choice of step size. Additionally, combining Equations (11) and (12), updating R_t becomes

$$R_{t+1} = \begin{bmatrix} R_t & 0 \\ 0 & 1 \end{bmatrix} \hat{V} \hat{\Sigma}, \tag{14}$$

and dropping the last column provides a corresponding update for the matrix R. The result is a new rank-4 factorization $\hat{W}_{t+1} = U_{t+1} R_{t+1}^T$.

The advantages of this method are two-fold. First, by updating both U and R simultaneously, there is no need to calculate R when a reconstruction is needed. Instead, we keep a running estimate of both U and R; estimates of the motion and structure matrices can be easily found at any point in time. We thus have a truly online SFM approach because we no longer need to store the entire observation matrix W in order to solve for R. Keeping a subset of the data is still useful so that old data can be revisited, but this can be limited to a fixed amount if memory becomes an issue. Second, this formulation uses an implicit step size; we no longer are required to specify a step size as in the original Grouse formulation, although the residual vector can still be scaled to affect the step size if needed. In Section 6, we show that the parameter-free algorithm outperforms other online SFM algorithms on real data.

4.3. Robust SFM

The Grasta algorithm [20, 21] is an extension of Grouse which is robust to outliers. Instead of minimizing the ℓ_2 cost function given in Equation (4), Grasta uses the robust ℓ_1 cost function $\mathcal{E}_{grasta}(U_{\Omega_t}, v_{\Omega_t}) = \|U_{\Omega_t} w_t - v_{\Omega_t}\|_1$. Grasta estimates the weights w_t of this ℓ_1 projection as well as the sparse additive component using ADMM [5] and then updates U_t using Grassmannian geodesics, replacing r_t with a variable Γ_t which is a function of the sparse component and the ℓ_1 weights [21]. Thus, we can use these w_t and Γ_t in place of w_t and r_t in the ISVD formulation of Grouse (Section 4.2), resulting in an ISVD formulation of Grasta.

4.4. The Resulting Family of Algorithms

The algorithms that we have presented so far make up a family of algorithms that can be applied to SFM in various circumstances. Both Grouse and Grasta are studied here, as well as the ISVD versions which carry forward an estimate for the singular values Σ.

To obtain a final variant within this family, we scale the residual norm in (12) by some value α_t before taking its SVD:

$$\begin{bmatrix} I & w_t \\ 0 & \alpha_t \|r_t\| \end{bmatrix} = \hat{U}\hat{\Sigma}\hat{V}^T . \tag{15}$$

This scaling provides us with more control over the contribution of the residual vector to the new subspace estimate. In our experiments, we took α_t to be decreasing with t. This scaling is useful in the batch setting, but with online data we found that no scaling performs best.

5. Implementation

5.1. The All-1s Vector

Two more issues need to be addressed for implementation of Grouse for SFM. The first issue is to exploit the fact that the constant vector of ones should always be in the column space of W_t and thus should be in the span of any estimate U_t of the column space of W_t, since the points may be offset from the origin. Without loss of generality, suppose that U_t has the ones-vector as its last column (appropriately scaled), so that

$$U_t = \begin{bmatrix} \tilde{U}_t & 1/\sqrt{n_t} \end{bmatrix}, \quad R_t = \begin{bmatrix} \tilde{R}_t & \tau_t \sqrt{n_t} \end{bmatrix}. \tag{16}$$

A Family of Matrix Completion SFM Algorithms		
	Grouse (ℓ_2 cost)	**Grasta (ℓ_1 cost)**
Track singular values Σ	Eqns (9),(10), "isvd"	(9),(10), using w_t from ADMM, and replacing r_t with Γ_t [21], "isvd-grasta"
Use $\Sigma = I$	Eqns (13),(14), "grouse"	Eqns (13),(14) with replacements, "grasta"
Use $\Sigma = I$, scale residual	Eqns (13),(14) using center matrix (15), "grouse (scaled)"	(13),(14) with replacements, using (15), "grasta (scaled)"

Table 1: Update steps for our family of matrix completion algorithms for structure from motion. The names in quotes correspond to names used in our experiments.

Here τ_t is the corresponding translation vector, which is the (scaled) last column of R_t. Similarly, let $w_t = \begin{bmatrix} \tilde{w}_t{}^T & \gamma_t \end{bmatrix}^T$. The derivative of the error $\mathcal{E}$ in Equation (4) with respect to just the first three columns is

$$\frac{\partial \mathcal{E}}{\partial \tilde{U}_t} = -2 r_t \tilde{w}_t{}^T . \tag{17}$$

By not considering the derivative of $\mathcal{E}$ with respect to the ones vector it will remain in the span of U, since the Grouse update will be applied only to $\tilde{U}$. Our Grouse update for structure from motion is obtained by first setting

$$\tilde{U}_{t+1} = \begin{bmatrix} \tilde{U}_t & \frac{r_t}{\|r_t\|} \end{bmatrix} \hat{U} \tag{18}$$

$$\tilde{R}_{t+1} = \begin{bmatrix} \tilde{R}_t & 0 \\ 0 & 1 \end{bmatrix} \hat{V}\hat{\Sigma}; \quad \tau_{t+1} = \begin{bmatrix} \tau_t \\ \gamma_t/\sqrt{n} \end{bmatrix}, \tag{19}$$

(where $\hat{U}$, $\hat{\Sigma}$, and $\hat{V}$ are defined as in Equation (12) using $\tilde{w}_t$), and then dropping the last column of $\tilde{U}_{t+1}$ and $\tilde{R}_{t+1}$. Because the residual vector r_t is still based on the full matrix U_t, including the ones-vector, r_t will necessarily be orthogonal to the ones-vector and U_{t+1} will retain the ones-vector in its span and will remain orthogonal.

5.2. Adding New Points

In online structure from motion, we are initially unaware of the total number of points to be tracked and need to account for newly added points as the video progresses. If $U_t \in \mathbb{R}^{n_t \times 4}$ is the current subspace estimate, then new points will manifest themselves as additional rows of U_t. However, when updating U_t, we have to make sure that the columns of U_t remain orthonormal and the last column continues to be the vector of ones. We thus perform the following update when each new point is added:

$$U_t \leftarrow \begin{bmatrix} \tilde{U}_t & 1/\sqrt{n_t+1} \\ 0 & 1/\sqrt{n_t+1} \end{bmatrix}. \tag{20}$$

By appending zeros to each column of $\tilde{U}_t$, they remain orthonormal. In the supplementary material we provide an alternative way to add new points, but do not use this method in our experiments.

5.3. Updating Past Points

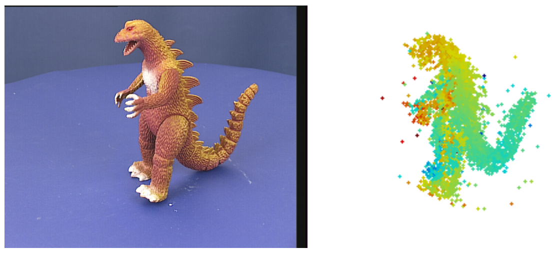
The Grouse update for structure from motion is fast enough that many updates can be done for each new video frame. It is therefore advantageous to be able to revisit old frames and reduce the error more than would be possible using a single pass over the frames. Using the original Grouse formulation described in Section 4.1, we simply run additional Grouse updates using past columns of W. This does not work in the new online formulation, where we also keep track of the matrix R_t, since running another Grouse update will add a new row to R_t. Instead, we simply drop the associated row of R_t before the update and replace it with the resulting new row. Because we do not impose any orthogonality restrictions on the matrix R_t, no correction is needed. In our experiments, however, we also compare to ISVD, which requires that the the right-side matrix be orthogonal. In this case, we first "downdate" our SVD using the algorithm given by Brand [7] before performing an update.

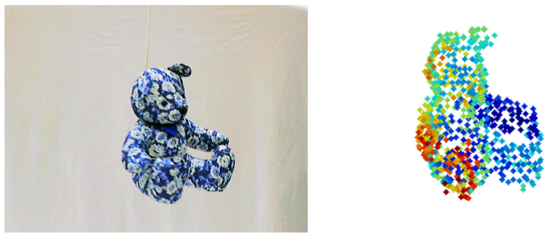
6. Experiments

We used three different datasets for comparison: the Dinosaur sequence from [16], the Teddy Bear sequence from [30], and a sequence of a Giraffe that we created ourselves. The Dinosaur sequence consists of 4983 point tracks over 36 frames. We use the full data matrix and do not remove low-quality tracks. The Teddy Bear sequence has 806 tracks over 200 frames and the Giraffe has 2634 tracks over 343 frames. The three sequences have measurement matrices that are missing 91%, 89% and 93% of their entries, respectively. Our reconstruction results for these datasets are illustrated in Figure 1. Since no groundtruth is available, we report 2D RMSE error. We found this error measure to be a good indicator of the quality of the reconstruction, and results were similar to a synthetic cylinder dataset where 3D RMSE errors could be calculated. All reconstructions were generated by applying standard scaled orthographic camera constraints.

6.1. Batch SFM

We first consider the batch or offline setting where the entire measurement matrix W is input. We compare the


(a) Dinosaur


(b) Teddy Bear

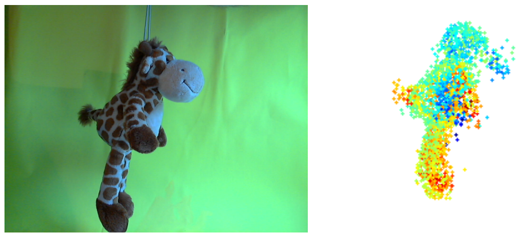
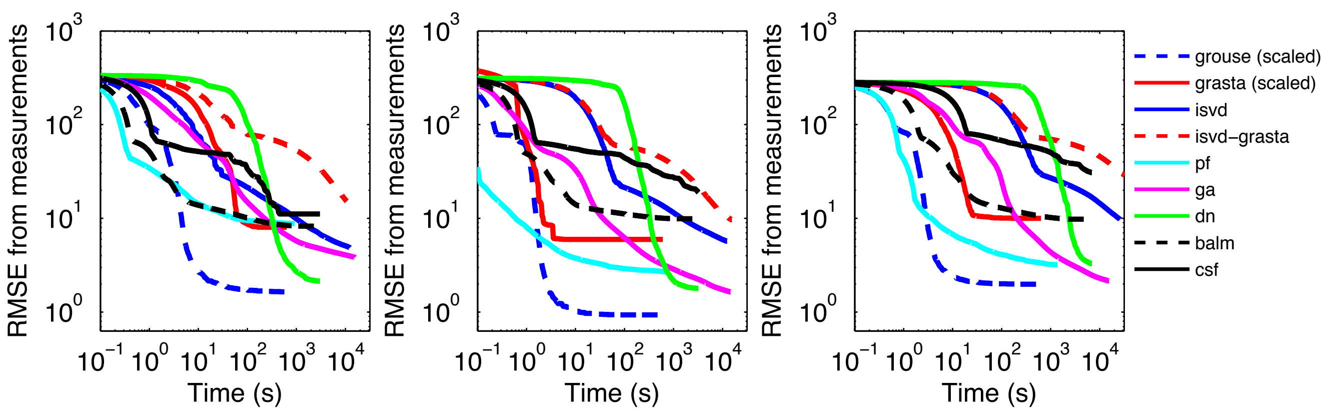

(c) Giraffe

Figure 1: Dinosaur, Teddy Bear and Giraffe datasets used in our experiments. We show one video frame and the reconstruction from our online algorithm as viewed from the estimated camera model of each frame. Color indicates depth.

`scaled` versions of our algorithms to `pf` [19], `dn` [8], `ga` [18], `csf` [17] and `balm` [14]. For `balm`, we used the scaled-orthographic projector. Each algorithm was run until convergence. All of the methods we compared to are in some way iterative and require initialization, so in order to fairly compare across different methods, we initialized each algorithm to the same random subspace and averaged over 5 random initializations.

Results are shown in Figure 2, where we plot the root-mean-squared error of each algorithm with respect to the measurement matrix over time. Notice that our algorithm `grouse (scaled)` converges significantly faster than most other batch algorithms, with `pf` being the only competitor in terms of convergence rate. However, we found that `pf` was sensitive to initial conditions and did not always converge to the same accuracy as `grouse`, especially for the Dinosaur sequence (Figure 2a). `grouse` also outperforms the more robust `grasta`. For real datasets, then, rather than using a robust algorithm such as `grasta`, it may be preferable to manually remove any extreme outliers and use the faster `grouse` algorithm, especially if the data



(a) Dinosaur (b) Teddy Bear (c) Giraffe (d) Legend

Figure 2: Comparison of batch SFM algorithms. We show how the root-mean-squared pixel error (RMSE) varies over time for each algorithm, averaged over 5 random initializations. Note that the data are shown on a log-log scale. Power-factorization (`pf`) has lower error in the first second of execution, but `grouse (scaled)` converges in 5-10 seconds to lower error than the other algorithms reach after 2+ hours.

is of relatively high quality to begin with.

6.1.1 Large Dataset Experiment

To test our algorithm on a large dataset and directly compare to [14], we tested our algorithm on the Venice dataset from Agarwal et. al [2] as modified in [14]. The 3D model was projected onto 100 random orthographic cameras, producing a measurement matrix of size 939551×200 from which 90% of the entries were randomly removed. We measure the error as $\|S - S_{gt}\|_F / \|S_{gt}\|_F$, where S is the resulting 3D reconstruction and S_{gt} is the groundtruth 3D model. The algorithm of [14] reported a reconstruction error of 6.67%, which `grouse` achieved within 50 seconds. After 2 minutes, `grouse` had reduced the error of 2.48% and it was down to 1.37% after 10 minutes.

6.2. Online SFM

For online structure-from-motion, we evaluate our algorithm `grouse` along with its robust counterpart `grasta`, and also evaluate the variants of both algorithms in which residual vectors for past frames are scaled (`grouse (scaled)` and `grasta (scaled)`). We also compare to incremental SVD (`isvd`) and a robust version of this approach, which we denote `isvd-grasta`. Our `isvd` algorithm is the same as in [6].

In [3], it was shown that for a static subspace, the step size for `grouse` should diminish over time. With our formulation, we still have control of the step size by scaling the residual; here, at iteration t we scale by the function C/t for a fixed C. We find that this is important for convergence when `grouse` is run in batch mode (Section 6.1).

For tests of the online methods, we show results both with and without scaling the residual for past frames, and do not scale the residual for new frames.

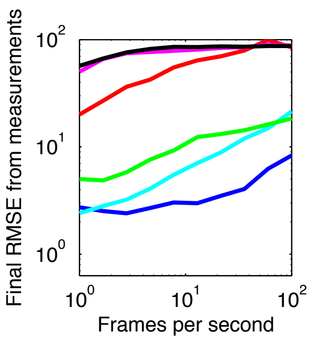
Our algorithms were implemented in Matlab and were initialized by finding all points that were fully tracked for the first 5 frames and running SVD on this subset of the measurement matrix.

Results for online SFM are shown in Figure 3. Each algorithm was run for enough iterations so that it maintained a fixed frame rate, which we varied from 1-100 fps. The plots show the root-mean-squared pixel error (RMSE) from the measurements after all frames have been processed for a given frame rate. For all datasets, `grouse` performed better than either its robust `grasta` counterpart or the `isvd` algorithms. The other methods do eventually catch up to `grouse` as the frame rate is slowed, but `grouse` is by far the best-performing algorithm for frame rates that would correspond to videos streams at 15-30 fps.

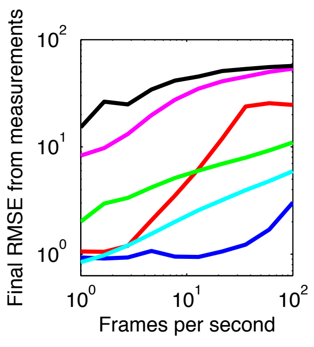
6.2.1 Real-time implementation

To demonstrate the utility of our method, we implemented `grouse` in C++ using OpenCV. Our implementation uses two threads running on separate cores: one thread reads frames from an attached webcam and tracks point using the KLT tracker in OpenCV, while the other thread continually runs `grouse` and incorporates new data as it becomes available. We used a MacBook Pro with a 2.66 GHz Intel Core 2 Duo processor and 8 GB of memory and a Logitech C270 webcam.

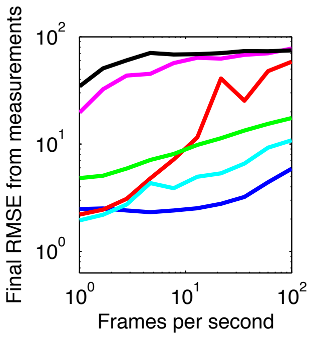
Our implementation was run at 15 fps and we used it to capture and reconstruct a 3D model of a toy giraffe in real time (Figure 1c). The bottleneck in this process is tracking



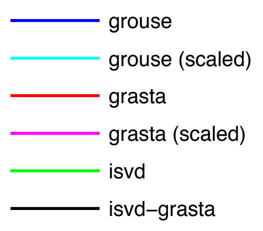
(a) Dinosaur



(b) Teddy Bear



(c) Giraffe



(d) Legend

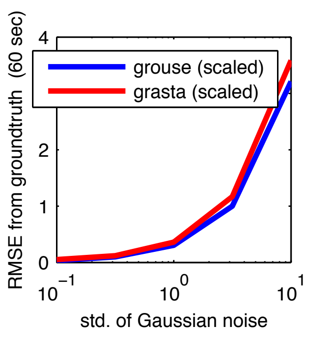
Figure 3: Comparison of online SFM algorithms. Each algorithm was run with enough iterations to ensure it ran at a fixed frame rate. We plot the final root-mean-squared pixel error (RMSE) between the estimated and actual measurement matrices as the number of frames per second is varied. `grouse` outperforms all other versions of our online SFM algorithm.

points between frames, and during this time `grouse` completed an average of 205 iterations per frame where each iteration of `grouse` was run on a randomly-selected past column of the measurement matrix. We envision this algorithm being useful in applications where it is desirable to build a 3D model with low-cost hardware, such as at-home 3D printing; using a simple webcam and a standard computer, we can obtain a 3D model of an object in real-time.

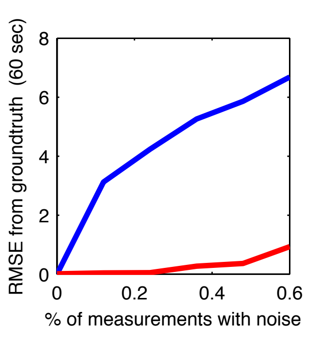
6.3. Effect of Noise

Using synthetic data allows us to systematically investigate the effect of noise on our method. We do so by generating a noise-free cylinder dataset, and then adding variable amounts of noise to it. The synthetically-generated cylinder has a radius of 10 pixels and a height of 50 pixels, centered at $(50, 50)$. Two hundred random points were tracked over 100 frames while the cylinder underwent one full rotation. The measurement matrix for this cylinder has 66% of its entries missing.

The results in Figure 4a show how our algorithm performs when run in batch when Gaussian noise with different standard deviations is added. Each algorithm was run for 60 seconds. We plot the resulting mean-squared pixel error (MSE) from the ground-truth, noise-free, measurement matrix. Similarly, we added sparse noise uniformly distributed in the interval $[-50, 50]$ to a variable proportion of observations, and show the results in Figure 4b. When only Gaussian noise is present, we find that the non-robust `grouse` gives the best results, although both `grouse` and `grasta` perform well. With sparse noise, `grasta` is preferable.



(a) Gaussian noise



(b) Sparse noise

Figure 4: Effect of noise on our algorithms. In (a), we generate a synthetic cylinder and add a variable level of Gaussian noise, while in (b) we add uniform random noise in the interval $[-50, 50]$ to a variable fraction of the measurements. For Gaussian noise, `grouse (scaled)` slightly outperforms the more robust `grasta (scaled)`, while `grasta (scaled)` has a clear advantage when there is sparse noise.

Because `grouse` is much faster than `grasta`, it is better to use `grouse` if the point tracks are of high quality. Even when bad tracks are present, due to the speed advantage it may be preferable to improve the tracking performance or throw out bad tracks rather than use `grasta`.

7. Discussion

We have presented a family of online algorithms for factorization-based structure from motion, opening the door to real-time recovery of 3D structure from a single camera video stream on low-power computers such as cell phones. Our algorithms have state-of-the-art speed and error performance for structure from motion. There are many remaining questions for exploration, but one of immediate importance for the online matrix completion approach to SFM is a better understanding of the algorithm's convergence properties and, in particular, their dependence on the sampled observations. Existing analyses assume that the observed subvector is chosen randomly and independently at each iteration. In SFM, however, there is a great deal of structure to the visible features over time and the observations are not independent at each iteration.

References

[1] H. Aanæs, R. Fisker, K. Astrom, and J. Carstensen. Robust factorization. *Pattern Analysis and Machine Intelligence*, 24(9):1215–1225, 2002. 1

[2] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *European Conference on Computer Vision*, pages 29–42. Springer, 2010. 6

[3] L. Balzano, R. Nowak, and B. Recht. Online identification and tracking of subspaces from highly incomplete information. In *Communication, Control, and Computing (Allerton)*, pages 704–711. IEEE, 2010. 1, 2, 3, 6

[4] L. Balzano and S. J. Wright. On grouse and incremental svd. *arXiv preprint arXiv:1307.5494*, 2013. 3

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–123, 2011. 4

[6] M. Brand. Incremental singular value decomposition of uncertain data with missing values. *European Conference on Computer Vision*, pages 707–720, 2002. 2, 6

[7] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006. 5

[8] A. Buchanan and A. Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. In *Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 316–322. IEEE, 2005. 1, 5

[9] J. R. Bunch and C. P. Nielsen. Updating the singular value decomposition. *Numerische Mathematik*, 31:111–129, 1978. 10.1007/BF01397471. 2, 3

[10] R. Cabral, J. Costeira, F. De la Torre, and A. Bernardino. Fast incremental method for matrix completion: an application to trajectory correction. In *International Conference on Image Processing*, pages 1417–1420. IEEE, 2011. 2

[11] E. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, December 2009. 2

[12] E. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053 –2080, May 2010. 1

[13] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM*, 58(1):1–37, 2009. 1, 2

[14] A. Del Bue, J. Xavier, L. Agapito, and M. Paladini. Bilinear modeling via augmented lagrange multipliers (balm). *Pattern Analysis and Machine Intelligence*, 34(8):1496–1508, 2012. 5, 6

[15] A. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1998. 2

[16] A. Fitzgibbon, G. Cross, and A. Zisserman. Automatic 3d model construction for turn-table sequences. *3D Structure from Multiple Images of Large-Scale Environments*, pages 155–170, 1998. 5

[17] P. Gotardo and A. Martinez. Computing smooth time trajectories for camera and deformable shape in structure from motion with occlusion. *Pattern Analysis and Machine Intelligence*, 33(10):2051–2065, 2011. 1, 5

[18] R. Guerreiro and P. Aguiar. 3d structure from video streams with partially overlapping images. In *International Conference on Image Processing*, volume 3, pages 897–900. IEEE, 2002. 1, 5

[19] R. Hartley and F. Schaffalitzky. Powerfactorization: 3d reconstruction with missing or uncertain data. In *Australia-Japan Advanced Workshop on Computer Vision*, volume 74, pages 76–85, 2003. 5

[20] J. He, L. Balzano, and J. Lui. Online robust subspace tracking from partial information. *Arxiv preprint arXiv:1109.3827*, 2011. 2, 4

[21] J. He, L. Balzano, and A. Szlam. Incremental gradient on the grassmannian for online foreground and background separation in subsampled video. In *Computer Vision and Pattern Recognition*, June 2012. 1, 2, 4

[22] R. Keshavan, A. Montanari, and S. Oh. Matrix completion from noisy entries. *Journal of Machine Learning Research*, 11:2057–2078, July 2010. 2

[23] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *International Symposium on Mixed and Augmented Reality*, pages 225–234. IEEE, 2007. 1

[24] G. Mateos and G. B. Giannakis. Sparsity control for robust principal component analysis. In *Asilomar Conference on Signals, Systems, and Computers*, 2010. 2

[25] P. McLauchlan and D. Murray. A unifying framework for structure and motion recovery from image sequences. In *International Conference on Computer Vision*, pages 314–320. IEEE, 1995. 1

[26] T. Morita and T. Kanade. A sequential factorization method for recovering shape and motion from image streams. *Pattern Analysis and Machine Intelligence*, 19(8):858–867, 1997. 1

[27] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Generic and real-time structure from motion using local bundle adjustment. *Image and Vision Computing*, 27(8):1178–1193, 2009. 1

[28] C. Poelman and T. Kanade. A paraperspective factorization method for shape and motion recovery. *Pattern Analysis and Machine Intelligence*, 19(3):206–218, 1997. 1

[29] B. Recht. A simpler approach to matrix completion. *Journal of Machine Learning Research*, 12:3413–3430, 2011. 1, 2

[30] J. Tardif, A. Bartoli, M. Trudeau, N. Guilbert, and S. Roy. Algorithms for batch matrix factorization with application to structure-from-motion. In *Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007. 1, 5

[31] K.-C. Toh and S. Yun. An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems. *Pacific Journal of Optimization*, 6:615–640, 2010. 2

[32] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, 1992. 1, 2

[33] M. Trajković and M. Hedley. A practical algorithm for structure and motion recovery from long sequence of images. In *Image Analysis and Processing*, pages 470–477. Springer, 1997. 1