Distributed Computer Vision Algorithms Through Distributed Averaging

Roberto Tron René Vidal
Center for Imaging Science, Johns Hopkins University, Baltimore MD 21218, USA
{tron,rvidal}@cis.jhu.edu http://www.vision.jhu.edu

Abstract

Traditional computer vision and machine learning algorithms have been largely studied in a centralized setting, where all the processing is performed at a single central location. However, a distributed approach might be more appropriate when a network with a large number of cameras is used to analyze a scene. In this paper we show how centralized algorithms based on linear algebraic operations can be made distributed by using simple distributed averages. We cover algorithms such as SVD, least squares, PCA, GPCA, 3-D point triangulation, pose estimation and affine SfM.

1. Introduction

Imagine we have a large network of N cameras for analyzing a scene. A traditional solution would be to send all the images to a central node, where a large problem is set up and solved. Unfortunately, this approach is not scalable. As the number of sensors increases, the load on the central node increases, possibly exceeding the available resources. In addition, if the central node fails, the entire network becomes useless. Therefore, it might be better to use a distributed approach, where each camera in the network processes its own data, collaborates with neighboring cameras and finds a globally optimal solution to the problem using simple local computations. In this way, the computation load is shared by all the nodes (which require only a fraction of the resources) and the architecture is robust to individual node failures.

We show that, for specific linear algebra operations, such as least squares and SVD (§3), the problem reduces to finding the average or the minimum of a set of values in a distributed fashion, for which well-studied algorithms are available (§2). We also show that this is the case for a few standard machine learning (§4) and computer vision (§5) algorithms.

Prior work. Over the past few years, many distributed algorithms for camera networks have been developed (see [8] and references therein for a review). However, different distributed approaches have been used for different problems. In this paper we propose a more general approach to directly translate specific centralized algorithms to their distributed counterparts. Our basic building blocks are distributed algorithms for least squares estimation, which were first presented in [12]. We show that the same principles can be applied to all the algorithms covered in this paper. The first one, Distributed PCA, has been the subject of numerous papers (see [11] and references therein). In particular, our method can be related to [1], with the difference that [1] assumes fixed tree topology and uses local QR decompositions instead of arbitrary topologies and local weighted averages. We are not aware of any previous distributed version of GPCA [10]. Triangulation of 3-D points in a camera network has been studied in the context of camera localization [4, 8]. In those works, however, the triangulation is performed independently at each camera, while our approach uses all the images simultaneously. Prior work on pose estimation [9] averages pose estimates computed independently at each node, while we work directly with the image points. Finally, while SfM is very much related to the problem of network localization (see [8] and references therein), we are not aware of any work that uses orthographic projection model and factorization methods in a distributed setting. Overall, our main contribution is to address all the aforementioned problems under a common distributed optimization framework.

2. Basic Distributed Algorithms

In this section, we review some basic distributed algorithms. We represent the network with a connected undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, N\}$ represents the nodes and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ represents the pairs $(i, j) \in \mathcal{E}$ that can communicate directly. The set of neighbors of node i is denoted as $\mathcal{N}_i = \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$ and its *degree* as $d_i = |\mathcal{N}_i|$. A *path* on the graph from node i to node j is a sequence of nodes $(w_0, w_1, \dots, w_m)$ such that $w_k \in \mathcal{V}$, $(w_k, w_{k+1}) \in \mathcal{E}$ for all k and m is the *length* of the path. The *diameter* of $\mathcal{G}$ is defined as the maximum length of the shortest path among any pair of nodes.

2.1. Distributed Averaging

Assume each node has a measurement $u_i \in \mathbb{R}$ and that we need to compute the average $\bar{u} = \frac{1}{N} \sum_{i=1}^{N} u_i$.

A first trivial approach is to aggregate averages along a spanning tree up to a designated central node. This approach is the most efficient in some situations. However, it requires selecting a central node, which could fail, and maintaining a spanning tree, which could be difficult in the case of packet losses. In addition, this kind of algorithm does not take advantage of wireless communications, where messages can be easily broadcasted from one node to its neighbors.

A second alternative approach is to use an *average consensus algorithm* [5]. In its simplest form, each node maintains a state x_i, which is initialized with a measurement $x_i(0) = u_i$. The nodes then iterate the difference equations

$$x_i(t+1) = x_i(t) + \varepsilon \sum_{j \in \mathcal{N}_i} (x_j(t) - x_i(t)), \qquad (1)$$

where $\varepsilon < (\max_i\{d_i\})^{-1}$. It can be shown that each state converges to the average of the initial values, i.e., $\lim_{t\to\infty} x_i = \bar{u}$. The main advantages are that each node has an estimate of the mean at each iteration, that each node needs to communicate only with its neighbors and that no central coordination is needed. The price to pay is a larger number of iterations to converge. Consensus algorithm can also be adapted to situations where the topology of the network is changing or packet losses are present [13]. In all our experiments we will employ average consensus.

Extensions. Both averaging algorithms mentioned above can be extended to the case of multivariate data $\boldsymbol{u}_i \in \mathbb{R}^m$ by applying them to each component. They can also be extended to the case where each node has multiple data points and we are interested in averaging the entire dataset. Specifically, assume each node i has n_i measurements $\{u_{ij}\}_{j=1}^{n_i}$ and we want to compute $\bar{u} = \frac{1}{\sum_{i=1}^N n_i} \sum_{i=1}^N \sum_{j=1}^{n_i} u_{ij}$. By defining $u_i = \sum_{j=1}^{n_i} u_{ij}$ we can write $\bar{u} = \frac{N}{\sum_{i=1}^N n_i} \frac{\sum_{i=1}^N u_i}{N}$. Therefore, $\bar{u}$ can be obtained by dividing the average of the u_i by the average of the n_i (i.e., by averaging twice).

2.2. Distributed computation of the minimum

Assume now that we are interested in computing $\underline{u} = \min_i u_i$. A simple distributed algorithm can be defined as

$$x(0) = u_i, \qquad x_i(t+1) = \min_{j \in \{\mathcal{N}_i \cup i\}} x_j(t). \qquad (2)$$

It is easy to show that, after at most $T = \mathrm{diam}(\mathcal{G})$ iterations, this algorithm converges to $x_i(T) = \underline{u}\ \forall i \in \mathcal{V}$.

3. Distributed Linear Algebra Algorithms

In this section, we introduce the key insight for reducing distributed linear algebra operations to averages. Assume that each node i has a matrix $\mathbf{A}_i \in \mathbb{R}^{n_i \times m}$ and let $n = \sum_{i=1}^N n_i$. Define the matrix

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1^\top & \mathbf{A}_2^\top & \dots & \mathbf{A}_N^\top \end{bmatrix}^\top \in \mathbb{R}^{n \times m}. \qquad (3)$$

For the sake of brevity, from now on we will denote (3) and other similar quantities as $\mathbf{A} = \mathrm{stack}(\{\mathbf{A}_i\}_{i=1}^N)$.

Assume that we want to compute $\mathbf{A}^\top\mathbf{A}$ (or a scaled version of it) at each node. The problem is distributed, because it involves data from all the nodes. Let us define

$$\mathbf{C} = \frac{1}{N}\mathbf{A}^\top\mathbf{A} = \frac{1}{N}\sum_{i=1}^N \mathbf{A}_i^\top\mathbf{A}_i = \frac{1}{N}\sum_{i=1}^N \mathbf{C}_i, \qquad (4)$$

where $\mathbf{C}_i = \mathbf{A}_i^\top\mathbf{A}_i$ is the local correlation matrix at node i.

We first remark that $\mathbf{C}$ is a matrix with dimension $m \times m$, independently of the number of nodes N and of measurements n. Therefore, as the dimension of the problem grows (for instance, if we add more nodes or data at each node), the requirements for storing $\mathbf{C}_i$ and $\mathbf{C}$ at each node do not change. Our second and most important remark is that $\mathbf{C}$ can be computed as an average of the local $\mathbf{C}_i$ by using the methods of §2.1. If N is also known at each node, then the matrix $\mathbf{A}^\top\mathbf{A}$ can be computed exactly. Moreover, since $\mathbf{C}_i$ is symmetric, the nodes can save transmission bandwidth by computing the Cholesky decomposition $\mathbf{C}_i = \mathbf{Q}_i^\top\mathbf{Q}_i$ and transmitting only the factor $\mathbf{Q}_i$, which can be represented as a lower triangular $m \times r$ matrix, where r is the rank of $\mathbf{C}_i$. However, computing $\mathbf{C}$ in this way is advantageous only if $r \ll n$ (otherwise we can just use the original data).

Assume now that each node i has also a vector $\boldsymbol{b}_i \in \mathbb{R}^{n_i}$ and define the vector $\boldsymbol{b} = \mathrm{stack}(\{\boldsymbol{b}_i\}_{i=1}^N)$. Assume we want to compute $\mathbf{A}^\top\boldsymbol{b}$ at each node or a scaled version of it. Similarly to before, define the cross-correlation vector

$$\boldsymbol{d} = \frac{1}{N}\mathbf{A}^\top\boldsymbol{b} = \frac{1}{N}\sum_{i=1}^N \mathbf{A}_i^\top\boldsymbol{b}_i = \frac{1}{N}\sum_{i=1}^N \boldsymbol{d}_i, \qquad (5)$$

where $\boldsymbol{d}_i = \mathbf{A}_i^\top\boldsymbol{b}_i \in \mathbb{R}^m$ is the local cross-correlation vector at node i. As before, $\boldsymbol{d}$ can be computed using distributed averaging and its dimension depends only on m.

In the following, we use the distributed computation of $\mathbf{C}$ and $\boldsymbol{d}$ to build distributed versions of various algorithms.

3.1. Singular Value Decomposition (SVD)

Assume we want to compute the compact SVD of the matrix $\mathbf{A}$ defined in (3), namely, $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{n\times r}$ and $\mathbf{V} \in \mathbb{R}^{m\times r}$ have orthonormal columns, $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ is diagonal with non-negative entries and r is the rank of the matrix. From (4) and the SVD of $\mathbf{A}$ we have

$$\mathbf{C} = \mathbf{V}(\frac{1}{N}\mathbf{\Sigma}^2)\mathbf{V}^\top, \qquad (6)$$

which is the SVD of $\mathbf{C}$. Therefore, the nodes can recover $\mathbf{V}$ (even without knowing N) after a distributed average. If N is known, then the nodes can also recover $\mathbf{\Sigma}$. While the matrices $\mathbf{V}$ and $\mathbf{\Sigma}$ are common to all the nodes, the matrix $\mathbf{U}$ must be stored similarly to $\mathbf{A}$, i.e., each node computes the part of the factorization corresponding to its data as

$$\mathbf{U}_i = \mathbf{A}_i\mathbf{V}\mathbf{\Sigma}^{-1}. \qquad (7)$$

3.2. Distributed Nullspace Estimation

Assume that, given the matrix $\mathbf{A}$, we want to find the vector

$$\boldsymbol{x}_0 = \underset{\boldsymbol{x}:\|\boldsymbol{x}\|=1}{\operatorname{argmin}} \|\mathbf{A}\boldsymbol{x}\|^2. \tag{8}$$

It is well known that the solution is given by $\boldsymbol{x}_0 = \boldsymbol{v}_m$ where $\boldsymbol{v}_m$ is the last column of $\mathbf{V}$ (completed to be a square orthonormal matrix, if necessary). Therefore, by computing the SVD of $\mathbf{C}$, the desired result can be found at each node.

3.3. Distributed Linear Least Squares

Assume we want to solve a distributed system of equations $\mathbf{A}x = b$ in a least squares sense, i.e., we want to find

$$\boldsymbol{x}_0 = \underset{\boldsymbol{x}}{\operatorname{argmin}} \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|^2. \tag{9}$$

The solution can be obtained by solving $\mathbf{A}^\top\mathbf{A}\boldsymbol{x}_0 = \mathbf{A}^\top\boldsymbol{b}$, or equivalently, $\mathbf{C}\boldsymbol{x}_0 = \boldsymbol{d}$, which is a $m \times m$ linear system of equations that each node can solve locally after computing the two distributed averages $\mathbf{C}$ and $\boldsymbol{d}$.

4. Applications in Subspace Learning

In this section we use the basic algorithms from the previous section to build distributed versions of algorithms for dimensionality reduction and subspace clustering.

4.1. Principal Component Analysis (PCA)

Imagine that each node i has a collection of n_i vectors $\mathbf{Y}_i = \left[\boldsymbol{y}_{i1}, \ldots, \boldsymbol{y}_{in_i}\right] \in \mathbb{R}^{m\times n_i}$. Considering all the nodes together, we have a distributed dataset $\mathbf{Y} = \left[\mathbf{Y}_1, \ldots, \mathbf{Y}_N\right] \in \mathbb{R}^{m\times n}$. We can reduce the dimensionality of this dataset by employing PCA, which approximates the vectors using an affine subspace model given by

$$\boldsymbol{y}_{ij} \simeq \hat{\boldsymbol{y}}_{ij} = \hat{\mathbf{V}}\boldsymbol{c}_{ij} + \hat{\boldsymbol{\mu}}, \tag{10}$$

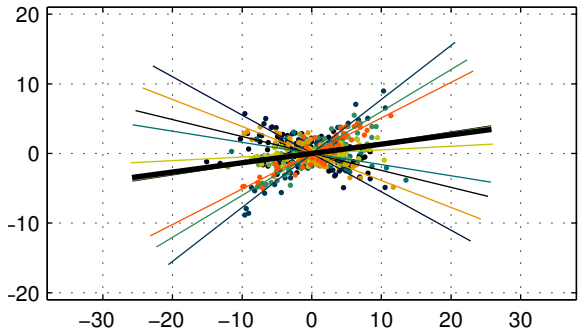
where $\hat{\boldsymbol{y}}_{ij} \in \mathbb{R}^m$ is the approximation of $\boldsymbol{y}_{ij}$, $\hat{\boldsymbol{\mu}} \in \mathbb{R}^m$ is an offset vector, $\hat{\mathbf{V}} \in \mathbb{R}^{m\times\hat{r}}$ represents an orthonormal basis for an $\hat{r}$-dimensional space in $\mathbb{R}^m$ and $\boldsymbol{c}_{ij}$ are the coefficients for $\hat{\boldsymbol{y}}_{ij}$ in this basis. PCA finds $\hat{\mathbf{V}}$, $\{\boldsymbol{c}_{ij}\}$ and $\hat{\boldsymbol{\mu}}$ that minimize the total reconstruction error in three steps:

1. Compute $\hat{\boldsymbol{\mu}}$ as the average of the entire dataset, i.e., $\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{n_i}\boldsymbol{y}_{ij}$ and compute a centered version of the data $\tilde{\boldsymbol{y}}_{ij} = \boldsymbol{y}_{ij} - \hat{\boldsymbol{\mu}}$ (to which we collectively refer as the matrix $\tilde{\mathbf{Y}}$).
2. Compute the SVD of the centered data matrix, say $\tilde{\mathbf{Y}} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^\top$ and set $\hat{\mathbf{V}}$ as the first $\hat{r}$ columns of $\mathbf{V}$.
3. Compute the coefficients as $\boldsymbol{c}_{ij} = \hat{\mathbf{V}}^\top\tilde{\boldsymbol{y}}_{ij}$.

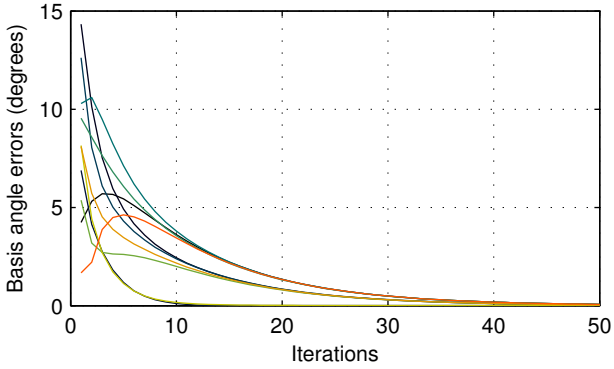
Given this centralized solution and the algorithms presented in §3, it is easy to give a distributed version of PCA. The average $\hat{\boldsymbol{\mu}}$ can be computed using the algorithms in §2.1. The matrix $\mathbf{V}$ can be obtained from the global correlation matrix $\mathbf{C}$ as described in §3 by setting $\mathbf{A} = \tilde{\mathbf{Y}}^\top$. The remaining operations can all be performed locally.

Conceptually, the proposed algorithm is just a distributed average of vectors followed by a distributed SVD. Notice that the local storage requirements do not depend on the number of nodes N. However, this approach gives a real advantage only if $\operatorname{rank}(\mathbf{Y}) \ll n$.

We tested our distributed PCA algorithm on a synthetic dataset distributed across a network with ten nodes in a ring topology. Each node has $n_i = 50$ vectors in $\mathbb{R}^2$ clustered around a 1-D subspace that is slightly different for each node. Figure 1(a) shows the distribution of the vectors and the 1-D subspaces estimated using either only the local data at each node or the entire dataset. Note that each single node cannot estimate the global subspace fit alone. We used consensus to compute the distributed average and we report in Figure 1(b) the maximum subspace angles between the estimate at each node and the centralized solution after each consensus iteration. The errors converge to zero for all the nodes in the network.



(a) Data points (different colors indicate data from different nodes), subspaces estimated locally at each node (thin colored lines) and subspace estimated from the entire dataset (thick black line).



(b) Maximum subspace angle between the subspace estimated locally at each node and the centralized solution after each iteration of consensus.

Figure 1. Results for distributed PCA.

4.2. Generalized PCA (GPCA)

Imagine now that the measurements at each node $\mathbf{Y}_i = \begin{bmatrix}\boldsymbol{y}_{i1} \dots \boldsymbol{y}_{in_i}\end{bmatrix}$ are drawn from a union of K hyperplanes with normals $\{\boldsymbol{b}_k\}_{k=1}^N \in \mathbb{R}^m$. In other words, for any point $\boldsymbol{y}_{ij}$ there exist a $k \in \{1, \dots, K\}$ such that $\boldsymbol{b}_k^\top \boldsymbol{y}_{ij} = 0$. This is the case, for instance, when the data $\mathbf{Y}_i$ represent images of faces belonging to different individuals under varying lighting conditions. Our goal is to obtain a distributed algorithm that recovers the normals $\{\boldsymbol{b}_k\}$ and clusters the data accordingly. Generalized PCA (GPCA) [10] is a non-iterative algorithm for solving this clustering problem. The idea of GPCA is to fit a polynomial to the data and then recover the normals from this polynomial. The main insight is to notice that any point $\boldsymbol{y}_{ij}$ in any one of the hyperplanes satisfies

$$0 = \prod_{k=1}^{K} \boldsymbol{b}_k^\top \boldsymbol{y}_{ij} = p_K(\boldsymbol{y}_{ij}) = \nu_K(\boldsymbol{y}_{ij})^\top \boldsymbol{c}, \qquad (11)$$

where p_K is a homogeneous polynomial in m variables with $M_K(m) = \binom{K+m-1}{m-1}$ coefficients, which we denote with the vector $\boldsymbol{c} \in \mathbb{R}^{M_K(m)}$. Also, $\nu_K(\boldsymbol{y}_{ij})$ denotes the Veronese map of the vector $\boldsymbol{y}_{ij}$, which contains all the monomials of degree K in the entries in $\boldsymbol{y}_{ij}$. For the exact definition of this map, we refer the reader to [10].

The vector of coefficients $\boldsymbol{c}$ can be linearly estimated from the data. Specifically, define the matrix $\mathbf{A}_i = \text{stack}(\{\nu_K(\boldsymbol{y}_{ij})^\top\}_{j=1}^{n_i})$ at each node i and the matrix $\mathbf{A} = \text{stack}(\{\mathbf{A}_i\}_{i=1}^N)$. Then, according to (11), the coefficient vector $\boldsymbol{c}$ must satisfy $\mathbf{A}\boldsymbol{c} = 0$, i.e., $\boldsymbol{c}$ is in the nullspace of $\mathbf{A}$. In the case of noisy data points that do not exactly correspond to the subspaces, the coefficient vector can be still estimated in a least square sense as $\boldsymbol{c} = \text{argmin}_{\boldsymbol{x}:\|\boldsymbol{x}\|=1} \|\mathbf{A}\boldsymbol{x}\|^2$. In both cases, each node can estimate $\boldsymbol{c}$ using the algorithm of §3.2. Once we have obtained the coefficients $\boldsymbol{c}$ of the polynomial p_K, we use polynomial differentiation to recover the normals. In the case of data lying perfectly on the hyperplanes, the gradient of p_K computed at one of the points $\boldsymbol{y}_{ij}$ gives a vector which is parallel to the normal $\boldsymbol{b}_k$ of the k-th hyperplane to which that point belongs, i.e.,

$$\nabla_{\boldsymbol{x}} p_K(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{y}_{ij}} \sim \boldsymbol{b}_k, \text{ where } \boldsymbol{b}_k^\top \boldsymbol{y}_{ij} = 0. \qquad (12)$$

When the data is noisy, we can still use this relation to get an estimate (although not perfect) of the normal at each point. Hence, all we need is to get one point in each cluster such that we can obtain the normal for each subspace. We will now explain how this can be done in a distributed way. In the following, we use $\hat{\boldsymbol{b}}_{ij}$ (with two indexes) to indicate the normal estimated at point $\boldsymbol{y}_{ij}$ using (12) and $\hat{\boldsymbol{b}}_k$ (with a single index) to indicate the normal estimated by the algorithm for the k-th subspace. The algorithm proceeds as follows:

1. Estimate $\hat{\boldsymbol{b}}_{ij} = \left.\frac{\nabla_{\boldsymbol{x}} p_K(\boldsymbol{x})}{\|\nabla_{\boldsymbol{x}} p_K(\boldsymbol{x})\|}\right|_{\boldsymbol{x}=\boldsymbol{y}_{ij}}$ for each point.
2. Estimate the distance of each point from its subspace as $d_{ij} = |\hat{\boldsymbol{b}}_{ij}^\top \boldsymbol{y}_{ij}|$.
3. For the first subspace $k = 1$, pick the normal at the point with minimum d_{ij}, i.e., $\hat{\boldsymbol{b}}_1 = \hat{\boldsymbol{b}}_{\hat{i}\hat{j}}$ where $(\hat{i}, \hat{j}) = \text{argmin}_{(i,j)} d_{ij}$.
4. For the other subspaces $k = 2, \dots, K$, repeat
   (a) Update the distances using $d_{ij} \leftarrow \frac{d_{ij}}{|\hat{\boldsymbol{b}}_{k-1}^\top \boldsymbol{y}_{ij}| + \delta}$, with δ being a small regularization constant.
   (b) Pick the normal at the point for which d_{ij} is minimal, i.e., $\hat{\boldsymbol{b}}_k = \hat{\boldsymbol{b}}_{\hat{i}\hat{j}}$ where $(\hat{i}, \hat{j}) = \text{argmin}_{(i,j)} d_{ij}$.

A distributed implementation of this procedure is straightforward. All the steps use only local information, except for steps (3) and (4b), which require the computation of a global minimum. For these steps, we can use the algorithm of §2.2. Note that, the node must exchange the normal that attains the minimum in addition to the value of the minimum distance. Once the normals $\{\hat{\boldsymbol{b}}_k\}$ have been estimated, each node can assign each one of its points to the closest subspace. This gives a complete, distributed version of GPCA.

We tested our algorithm on a synthetic dataset distributed in a network of ten nodes connected with a ring topology. Each node generates $n_i = 60$ data points in one of three 1-D subspaces in $\mathbb{R}^2$ and adds isotropic Gaussian noise whose variance is about 15% the variance of the data. Note that each node has only samples from one of the subspaces. Figure 2(a) shows an instance of the generated data together with the estimated subspaces. We report in Figure 2(b) the squared Euclidean distances between the coefficient vector c computed with a centralized algorithm and the vectors computed after each iteration of the consensus algorithm in the distributed version. All the local estimates converge to the centralized solution. In addition, the maximum angle between any normal estimated at any node and the corresponding normal from the centralized solution is $1.48 \cdot 10^{-6}$.

5. Applications in Computer Vision

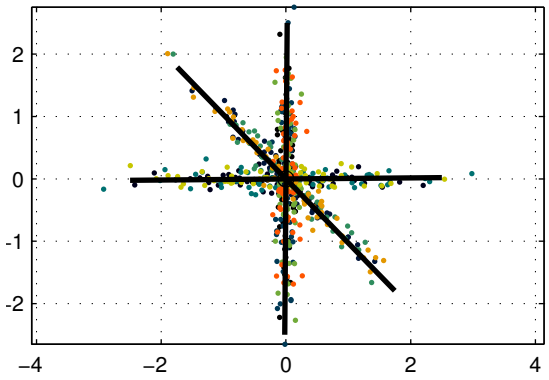
In this section we give distributed versions of three basic but important computer vision algorithms: point triangulation, linear pose estimation and affine Structure from Motion.

5.1. Point triangulation

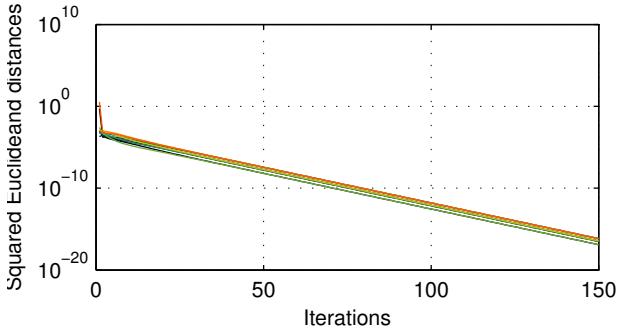
Assume that a 3-D point $\boldsymbol{P} = \begin{bmatrix}P_x & P_y & P_z\end{bmatrix}^\top \in \mathbb{R}^3$ is visible to all the cameras and let $\boldsymbol{p}_i = \begin{bmatrix}p_{ix} & p_{iy}\end{bmatrix}^\top \in \mathbb{R}^2$ denote its image in the i-th camera. Denote as $\mathbf{R}_i \in SO(3)$ and $\boldsymbol{T}_i \in \mathbb{R}^3$ the rotation and translation of the i-th camera. Using the standard projective camera model [2] we have

$$\lambda_i \boldsymbol{p}_i^h = \begin{bmatrix}\mathbf{R}_i & \boldsymbol{T}_i\end{bmatrix} \boldsymbol{P}^h, \qquad (13)$$

where $\lambda_i \in \mathbb{R}$ is the depth of the point $\boldsymbol{P}$ for camera i and the notation $\boldsymbol{v}^h = \text{stack}(\boldsymbol{v}, 1)$ indicates the vector $\boldsymbol{v}$ in



(a) Data points (different colors indicate data from different nodes) and estimated 1-D subspaces (black lines).



(b) Squared Euclidean distances between the coefficient vectors c estimated with the centralized and distributed GPCA algorithms after each iteration of consensus.

Figure 2. Results for distributed GPCA.

homogeneous coordinates. Given the images $\{\boldsymbol{p}_i\}_{i=1}^N$, one can triangulate the 3-D point $\boldsymbol{P}$ using a well known linear algorithm. First, each node i constructs

$$\mathbf{A}_i = [\boldsymbol{p}_i^h]_\times \begin{bmatrix} \mathbf{R}_i & \boldsymbol{T}_i \end{bmatrix}, \tag{14}$$

where $[\boldsymbol{p}_i]_\times$ is the matrix representing the cross product[1]. Then, the coordinates of $\boldsymbol{P}$ can be recovered using the relation $\mathbf{A}\boldsymbol{P}^h = 0$, where $\mathbf{A} = \text{stack}(\{\mathbf{A}_i\}_{i=1}^N)$. In practice, in our distributed setting, each node can recover $\boldsymbol{P}^h$ from the nullspace of $\mathbf{A}$ by applying the algorithm of §3.2 and then normalize the vector so that the last entry is equal to one. The 3-D coordinates of $\boldsymbol{P}$ are then easily extracted from $\boldsymbol{P}^h$.

We tested our algorithm on a synthetic camera setup with five calibrated cameras looking at an object (Figure 3(a)). For simplicity, we assumed that all the cameras can see all the 3-D points. The images of the points in each camera have been corrupted by adding Gaussian noise with standard deviation equivalent to 15 pixels on a 1000×1000 image. In Figure 3(b) we show the final reconstruction together with the trajectories of the points as they are estimated during successive iterations and converge to the centralized solution.

[1] $[\boldsymbol{p}_i]_\times \boldsymbol{v} = \boldsymbol{p}_i \times \boldsymbol{v} \, \forall \boldsymbol{v} \in \mathbb{R}^3$ and $[\boldsymbol{p}_i]_\times \boldsymbol{p}_i = 0$

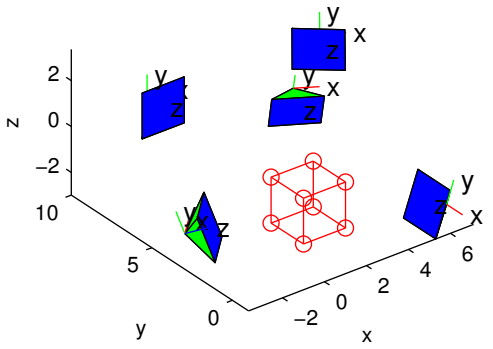
5.2. Linear pose estimation

Assume we have an object defined in a predetermined reference frame by N_P points $\{\boldsymbol{P}_l\}_{l=1}^{N_p} \in \mathbb{R}^3$ and that this object undergoes a rigid-body transformation defined by the rotation $\mathbf{R}_0 \in SO(3)$ and the translation $\boldsymbol{T}_0 \in \mathbb{R}^3$. Again, using the standard projective camera model, we have

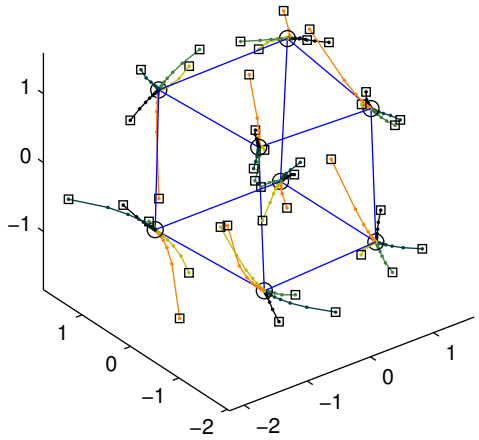
$$\lambda_{il}\boldsymbol{p}_{il}^h = \mathbf{R}_i \left(\mathbf{R}_0 \boldsymbol{P}_l + \boldsymbol{T}_0\right) + \boldsymbol{T}_i, \tag{15}$$

where $\boldsymbol{p}_{il} \in \mathbb{R}$ and λ_{il} are the image and the depth of the l-th point for the i-th camera. The camera poses $(\mathbf{R}_i, \boldsymbol{T}_i)$ are assumed to be known. Our goal is to design a distributed version of the linear algorithm for recovering the pose $(\mathbf{R}_0, \boldsymbol{T}_o)$ from the images [2]. Similar to the triangulation problem, if we multiply on the left by the cross product matrix $[\boldsymbol{p}_{il}^h]_\times$, we get the relationship

$$[\boldsymbol{p}_{il}^h]_\times \left(\mathbf{R}_i \left(\mathbf{R}_0 \boldsymbol{P}_l + \boldsymbol{T}_0\right) + \boldsymbol{T}_i\right) = 0. \tag{16}$$

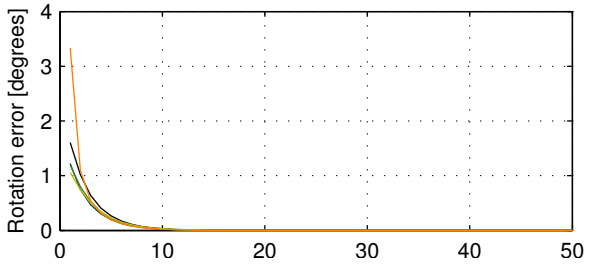


(a) Synthetic camera setup of five cameras looking at an object (a cube) composed of eight 3-D points.

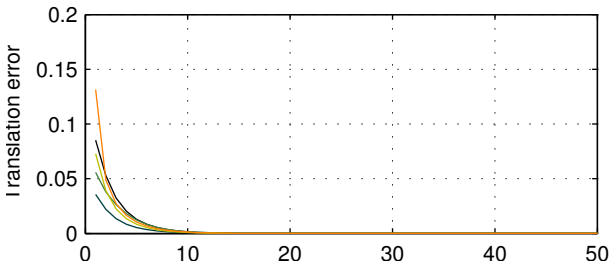


(b) The centralized reconstruction of the eight 3-D points (circles) and the trajectories of each point as they are estimated after each consensus iteration. Different colors indicate data from different nodes. The squares indicate the estimates of the nodes after one iteration. The errors are magnified five times for visualization purposes.

Figure 3. Results for distributed triangulation.



(a) Rotation errors (degrees).



(b) Translation errors (Euclidean distance).

Figure 4. Results for distributed pose estimation. The plots show the errors between the pose estimated at each node after each consensus iteration and the pose estimated by the centralized algorithm.

This relation gives equations that are linear in the unknowns $\mathbf{R}_0$ and $\boldsymbol{T}_0$ and can be rewritten as $\mathbf{D}_{il}x = \boldsymbol{d}_{il}$, where the vector $\boldsymbol{x} = \text{stack}(\text{vec}(\mathbf{R}_0), \boldsymbol{T}_0) \in \mathbb{R}^{12}$ contains all the entries of $\mathbf{R}_0$ and $\boldsymbol{T}_0$, while $\mathbf{D}_{il}$ and $\boldsymbol{d}_{il}$ contain the coefficients of the equation given by (16). To obtain a distributed solution, each node constructs $\mathbf{A}_i = \text{stack}\big(\{\mathbf{D}_{ij}\}_{j=1}^{N_p}\big)$ and $\boldsymbol{b}_i = \text{stack}\big(\{\boldsymbol{d}_{ij}\}_{j=1}^{N_p}\big)$ from all the constraints given by the points that are visible in camera i. It is then possible to use the distributed algorithm of §3.3 to estimate $\boldsymbol{x}$. Given this vector, one can finally extract $\mathbf{R}_0$ and $\boldsymbol{T}_0$. In the presence of noise, it might be necessary to project the estimated $\mathbf{R}_0$ back to the space of rotations $SO(3)$, see [2].

We tested our algorithm on the same setup used for §5.1 (Figure 3(a)), but we now assumed that the geometry of the cube is known and that we want to recover its pose. In Figure 4 we report the rotation and translation errors of the distributed algorithm with respect to the centralized version. Both errors converge toward zero as the iterations proceed.

5.3. Distributed Affine Structure from Motion

Assume now that we have N_P unknown 3-D points $\{\boldsymbol{P}_l\}_{l=1}^{N_P}$ that are moving. Assume that each camera i collects a video and tracks all the 3-D points for K_i frames. Theoretically, each camera could solve a local Structure from Motion (SfM) problem and obtain the 3-D structure [6]. However, in this section we propose a distributed algorithm that uses all the frames from all the cameras to obtain better results.

Denote as $\{\boldsymbol{p}_{ilk}\}_{l=1,\dots,N_P}^{k=1,\dots,K_i}$ the images of the 3-D points projected in the i-th camera and let $K = \sum_{i=1}^{N} K_i$. Instead of the projective camera model used previously, in this section we will consider the orthographic projection model (which can be easily generalized to the affine model [6]). Under this assumption, the image points satisfy the equation

$$\boldsymbol{p}_{ilk} = \mathbf{M}_{ik}\boldsymbol{P}_l + \boldsymbol{t}_{ik}. \tag{17}$$

When the cameras are calibrated, $\mathbf{M}_{ik} \in \mathbb{R}^{2\times 3}$ and $\boldsymbol{t}_{ik}$ contain the first two rows of, respectively, the rotation and translation between the camera i and the 3-D object at frame k. For ease of explanation, we define the *motion matrices* $\mathbf{M}_i = \text{stack}(\{\mathbf{M}_{ik}\}_{k=1}^{K_i})$, $\mathbf{M} = \text{stack}(\{\mathbf{M}_i\}_{i=1}^{N})$, and the *structure matrix* $\mathbf{S} = \big[\boldsymbol{P}_1 \cdots \boldsymbol{P}_{N_P}\big]_{3\times N_P}$. The goal of SfM is to recover $\mathbf{M}$ and $\mathbf{S}$ from the image points $\{\boldsymbol{p}_{ilk}\}$.

Since the motion and structure of the scene can only be recovered up to a choice of the world's reference frame, it is customary to place the origin of the world's reference frame at the center of the 3-D points, so that $\sum_{l=1}^{N_P} \boldsymbol{P}_l = 0$. As a consequence, $\boldsymbol{t}_{ik}$ can be estimated by each camera as $\boldsymbol{t}_{ik} = \frac{1}{N_P}\sum_{l=1}^{N_P} \boldsymbol{p}_{ilk}$. Each camera i can then form the matrices $\mathbf{W}_{ik} \in \mathbb{R}^{2\times N_P}$, whose N_P columns are the mean-subtracted images $\{\boldsymbol{w}_{ilk} = \boldsymbol{p}_{ilk} - \boldsymbol{t}_{ik}\}_{l=1}^{N_P}$. Define $\mathbf{W}_i = \text{stack}(\{\mathbf{W}_{ik}\}_{k=1}^{K_i})$ and $\mathbf{W} = \text{stack}(\{\mathbf{W}_i\}_{i=1}^{N}$. It is easy to verify that $\mathbf{W}$ can be factorized as

$$\mathbf{W} = \mathbf{M}\mathbf{S} \tag{18}$$

and that $\mathbf{W}_i = \mathbf{M}_i\mathbf{S}$ for all i. From (18), the matrix $\mathbf{W}$ is of rank at most three and it can be factorized using the SVD of $\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. This can be done by using the distributed algorithm of §3.1, where we identify $\mathbf{A}_i = \mathbf{W}_i$, $\mathbf{S} = \mathbf{V}^\top$ and $\mathbf{M}_i = \mathbf{U}_i\mathbf{\Sigma} = \mathbf{W}_i\mathbf{V}$. We define $\{\mathbf{U}_{ik}\}_{k=1}^{K_i}$ by partitioning $\mathbf{U}_i$ in the same way as $\mathbf{W}_i$.

The estimates $\mathbf{M}$ and $\mathbf{S}$ obtained from the SVD are referred to as affine motion and structure, since they are valid only up to an affine transformation $\mathbf{Q} \in \mathbb{R}^{3\times 3}$. In fact,

$$\mathbf{M} = \mathbf{U}\mathbf{Q} \quad \text{and} \quad \mathbf{S} = \mathbf{Q}^{-1}\mathbf{\Sigma}\mathbf{V}^\top \tag{19}$$

give a valid factorization for any invertible $\mathbf{Q}$. However, with calibrated cameras, one can fix $\mathbf{Q}$ by knowing that the first two rows of each motion matrix $\mathbf{M}_{ik}$ should be rows of a rotation matrix. Since $\mathbf{M}_{ik} = \mathbf{U}_{ik}\mathbf{Q}$, we have that $\mathbf{M}_{ik}\mathbf{M}_{ik}^\top = \mathbf{U}_{ik}\mathbf{Q}\mathbf{Q}^\top\mathbf{U}_{ik}^\top = \mathbf{I}_{2\times 2}$, the two by two identity matrix. This gives the following three linear equations for finding the entries of the symmetric matrix $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^\top$

$$\begin{aligned} \boldsymbol{e}_1^\top\mathbf{U}_{ik}^\top\mathbf{Y}\mathbf{U}_{ik}\boldsymbol{e}_1 = \boldsymbol{e}_2^\top\mathbf{U}_{ik}^\top\mathbf{Y}\mathbf{U}_{ik}\boldsymbol{e}_2 = 1, \\ \boldsymbol{e}_1^\top\mathbf{U}_{ik}^\top\mathbf{Y}\mathbf{U}_{ik}\boldsymbol{e}_2 = 0, \end{aligned} \tag{20}$$

where $\boldsymbol{e}_1 = \begin{bmatrix}1 & 0\end{bmatrix}^\top$, $\boldsymbol{e}_2 = \begin{bmatrix}0 & 1\end{bmatrix}^\top$ and $i = 1, \dots, N$. These equations can be rewritten as $\mathbf{A}_i\boldsymbol{y} = 0$, where $\boldsymbol{y} = \text{vec}(\mathbf{Y})$ and the entries of $\mathbf{A}_i$ are functions of $\mathbf{U}_i$ as given by (20). To obtain $\boldsymbol{y}$ (and hence $\mathbf{Y}$) from all the equation

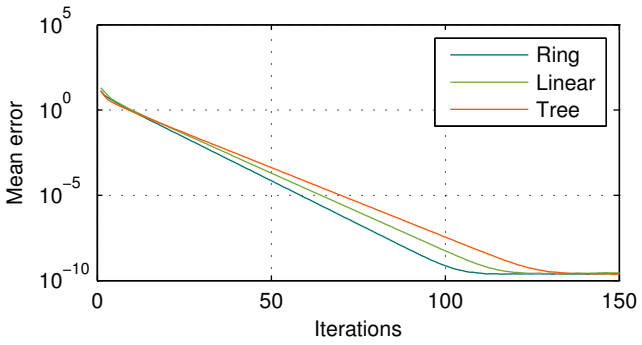


Figure 5. Results for distributed structure from motion. Subspace angle errors between the distributed and the centralized solutions averaged over all the nodes and objects different kinds of topology.

from all the cameras, one can apply the algorithm of §3.2. After this step, each node can independently compute the common $\mathbf{Q}$ from the Cholesky decomposition of $\mathbf{Y}$. Given $\mathbf{Q}$, from (19) we obtain $\mathbf{S}$ and $\mathbf{M}_{ik}$, i.e., the structure of the scene and the pose of each camera.

We use the Hopkins 155 dataset [7] to test our algorithm. This dataset contains features extracted from videos with multiple moving objects. We consider separately the tracks for each distinct object appearing in any of the videos, obtaining 135 single-object sequences. We simulated a network setup with $N = 5$ cameras connected with a ring topology. For each sequence, we divided the available frames roughly equally among the cameras (to simulate the fact that each camera can track the object from different angles). We used 150 iteration of consensus for each step of our algorithm.

We compared the final distributed and centralized solutions by computing the maximum subspace angle [3] between the subspaces spanned by the rows of the structure matrices at each node, which are invariant to the unknown change of basis for the affine structure. The maximum error is $1.12 \cdot 10^{-8}$ and about 95% of the time the error is below $7 \cdot 10^{-10}$. In all cases, the centralized and distributed solutions are numerically equivalent.

Using the same experimental setting, we have also compared different choices of topology. In Figure 5 we show the subspace angle errors averaged over all the nodes for all the sequences for three choices of topology: linear (each node has at most two neighbors), ring and a binary tree. Intuitively, the presence of loops (ring topology) leads to faster convergence for consensus.

6. Conclusions

In this paper we have shown how simple distributed linear problems, such as least squares and SVD, can be easily implemented by reducing them to distributed averages, which can be computed using the spanning tree based method or consensus. This insight provides a straightforward way to implement a number of machine learning (PCA, GPCA) and computer vision (triangulation, pose estimation, affine SfM) algorithms. A distinctive feature of our algorithms is that the amount of storage required at each node depends only on the local data and remains constant as the number of cameras increases. On the downside, these linear algorithms are usually far from optimal and only used as an initialization for other non-linear procedures (e.g., bundle adjustment). Also, from the perspective of communication cost, they might not give substantial advantages: for instance, the distributed SVD illustrated in this paper is not efficient when the rank of the data r is in the order of the number of vectors n. In our future work we plan to address these challenges.

Acknowledgments. This work has been supported by the National Science Foundation grant 0834470.

References

[1] Z.-J. Bai, R. Chan, and F. Luk. Principal component analysis for distributed data sets with updating. *Advanced Parallel Processing Technologies*, pages 471–483, 2005.

[2] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge, 2nd edition, 2004.

[3] H. Hotelling. Relations between two sets of variates. *Biometrika*, 28:321–372, 1936.

[4] W. Mantzel, H. Choi, and R. Baraniuk. Distributed camera network localization. In *Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1381–1386, 2004.

[5] R. Olfati-Saber, J. Fax, and R. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[6] C. J. Poelman and T. Kanade. A paraperspective factorization method for shape and motion recovery. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(3):206–18, 1997.

[7] R. Tron and R. Vidal. A benchmark for the comparison of 3-D motion segmentation algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[8] R. Tron and R. Vidal. Distributed algorithms for camera sensor networks. *Signal Processing Magazine*, 2011.

[9] R. Tron, R. Vidal, and A. Terzis. Distributed pose averaging in camera networks via consensus on $SE(3)$. In *International Conference on Distributed Smart Cameras*, 2008.

[10] R. Vidal, Y. Ma, and S. Sastry. Generalized Principal Component Analysis (GPCA). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1–15, 2005.

[11] A. Wiesel and A. Hero. Decomposable principal component analysis. *IEEE Transactions on Signal Processing*, 57(11):4369 –4377, 2009.

[12] L. Xiao, S. Boyd, and S. Lall. A scheme for robust distributed sensor fusion based on average consensus. In *Symposium on Information Processing of Sensor Networks (IPSN)*, pages 63–70, 2005.

[13] A. T. Y. Chen, R. Tron and R. Vidal. Corrective consensus: Converging to the exact average. In *Conference on Decision and Control*, 2010.