Intelligent distributed systems

Daniele Fontanelli

Department of Industrial Engineering
University of Trento
E-mail address: *daniele.fontanelli@unitn.it*

2022/2023

Outline

1. Distributed WLS
2. Distributed Kalman Filter

Outline

1. Distributed WLS

2. Distributed Kalman Filter

Distributed Least Squares Algorithm

Let us define the same problem we have defined to introduce the LS problem, here reported for clarity:

- We have to determine the value of a *nonrandom* constant parameter;
- We collect the sensor readings from m *sensors* deployed in an environment by means of a communication system;
- The data are collected in *different time instants* $t = 0, \dots$ from a subset of $n(t) \geq m$ of the available sensors. We will refer generically to n for ease of notation;
- To ensure a correct *sensor fusion*, the data are *timestamped* at the source location and all the sensors are supposed to be *synchronised* at the *Coordinated Universal Time* (UTC), for example using GPS signals;

Distributed Least Squares Algorithm

- The set of sensor readings is not necessarily the same and, hence, *may change in time*;
- Each sensor provides the measures with a different (possibly time varying) *uncertainty*, i.e., the stochastic process affecting the sensors is *non-stationary*;
- The data are collected by a single entity fusing all the data, i.e., *centralised approach*.

The solution we provided previously is based on the existence of a *central node* that receives all the data and then performs, in a *centralised* way the estimates.

Now, we want to do exactly the same thing in a *distributed way*.

Distributed Least Squares Algorithm

To properly model this problem, let us define

$$z(i) = H(i)x + \varepsilon(i), \;\; i = 1, \dots, n$$

the time varying set of measurements collected at time t.
Then, let us define the following aggregating vectors

$$Z^n = \begin{bmatrix} z(1) \\ z(2) \\ \vdots \\ z(n) \end{bmatrix}, \;\; H^n = \begin{bmatrix} H(1) \\ H(2) \\ \vdots \\ H(n) \end{bmatrix}, \;\; \varepsilon^n = \begin{bmatrix} \varepsilon(1) \\ \varepsilon(2) \\ \vdots \\ \varepsilon(n) \end{bmatrix}.$$

Distributed Least Squares Algorithm

For the covariance matrix of the noises

$$C^n = \begin{bmatrix} \mathsf{C}\{\varepsilon(1)\} & 0 & \dots & 0 \\ 0 & \mathsf{C}\{\varepsilon(2)\} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathsf{C}\{\varepsilon(n)\} \end{bmatrix}$$

Notice how the noise is assumed uncorrelated in different time instants, while instead it can be correlated for each time instant ($\mathsf{C}\{\varepsilon(i)\}$ is not necessarily diagonal).

Distributed Least Squares Algorithm

We have then

$$J(x,n) = \sum_{i=1}^{n} (z(i) - H(i)x)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} (z(i) - H(i)x) =$$
$$= (Z^n - H^n x)^T C^{n-1} (Z^n - H^n x) \Rightarrow$$
$$\Rightarrow \hat{x}^{LS} = \arg\min_x J(x,n) = (H^{nT} C^{n-1} H^n)^{-1} H^{nT} C^{n-1} Z^n.$$

While the covariance matrix of the error is given by

$$P_n = (H^{nT} C^{n-1} H^n)^{-1}$$

Distributed Least Squares Algorithm

Using the definitions of the matrices given previously, the previous equations can be rewritten as

$$
\begin{aligned}
\hat{x}^{LS} &= \arg\min_x J(x,n) = (H^{nT} C^{n-1} H^n)^{-1} H^{nT} C^{n-1} Z^n = \\
&= \left(\sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} H(i) \right)^{-1} \sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} z(i).
\end{aligned}
$$

Hence, the covariance matrix of the error is given by

$$
P_n = \left(\sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} H(i) \right)^{-1}
$$

Distributed Least Squares Algorithm

One immediate solution for a *distributed algorithm* to work properly in this case is just using a *flooding algorithm*:

- Each node transmits to all the other nodes in its *communication range* all its data;
- Under mild connectivity properties, sooner or later, *all* the nodes in the network will have *all* the data in the network.
- Now, each node can solve the WLS and hence each node has the *same best LS solution*.

However, such a solution requests a *lot of messages* to travel along the network.
Is there any possibility to reduce this problem?

Distributed Least Squares Algorithm

Let us start with a *local estimate*.
The sensor considers a local *composite information matrix* $F_i(k) \in \mathbb{R}^{m \times m}$ and a local *composite information state* $a_i(k) \in \mathbb{R}^m$, *one for each node*.
The i-th sensor makes a measurement and initialises its composite elements as:

$$F_i(0) = H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} H(i), \text{ and } a_i(0) = H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} z(i).$$

Therefore, it is able to provide a (first) estimate at time $k = 0$ using the local batch solution of the WLS, i.e.

$$\hat{x}^{LS}(0) = F_i(0)^{-1} a_i(0).$$

Distributed Least Squares Algorithm

Instead, the solution of the *centralised* (i.e. *global*) WLS would be given by

$$\hat{x}^{LS} = \left(\sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} H(i)\right)^{-1} \sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} z(i) =$$

$$= \left(\sum_{i=1}^{n} F_i(0)\right)^{-1} \sum_{i=1}^{n} a_i(0),$$

$$P_n = \left(\sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} H(i)\right)^{-1} = \left(\sum_{i=1}^{n} F_i(0)\right)^{-1}.$$

The issue is: can we compute the *same* global solution starting from the *local* values of $F_i(0)$ and $a_i(0)$, $\forall i$?

Distributed Least Squares Algorithm

Since each sensor has its own

$$F_i(0) = H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} H(i), \text{ and } a_i(0) = H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} z(i),$$

it is immediate to notice that

$$\sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} z(i) = \sum_{i=1}^{n} a_i(0) = n\left(\frac{1}{n}\sum_{i=1}^{n} a_i(0)\right),$$

$$\sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} H(i) = \sum_{i=1}^{n} F_i(0) = n\left(\frac{1}{n}\sum_{i=1}^{n} aF_i(0)\right).$$

Hence, if each node is able to compute the average among $F_i(0)$ and $a_i(0)$, $\forall i$, it can compute the *global* WLS!
This is indeed an *average consensus* problem!

Distributed Least Squares Algorithm

To account for the design of a protocol ensuring the average consensus, we can make use of the time-varying weighting scheme based on the *maximum-degree weight*

$$q_{ij}(k) = \begin{cases} \frac{1}{n} & \text{if } (j,i) \in \mathcal{E} \text{ and } i \neq j, \\ 1 - \frac{d_i(k)}{n} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

or the *Metropolis weights*

$$q_{ij}(k) = \begin{cases} \frac{1}{\max(d_i(k), d_j(k)) + 1} & \text{if } (j,i) \in \mathcal{E}(k) \text{ and } i \neq j, \\ 1 - \sum_{j=1, i\neq j}^{n} q_{ij}(k) & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Distributed Least Squares Algorithm

The update rule is hence based on the *average consensus*

$$F_i(k+1) = F_i(k) + \sum_{j=1}^{n} q_{ij}(k)(F_j(k) - F_i(k)) =$$

$$= q_{ii}(k)F_i(k) + \sum_{j=1, j\neq i}^{n} q_{ij}(k)F_j(k)$$

$$a_i(k+1) = a_i(k) + \sum_{j=1}^{n} q_{ij}(k)(a_j(k) - a_i(k)) =$$

$$= q_{ii}(k)a_i(k) + \sum_{j=1, j\neq i}^{n} q_{ij}(k)a_j(k)$$

Distributed Least Squares Algorithm

By the previous theorem we have then for j-th node

$$\lim_{k\to\infty} F_j(k) = \frac{1}{n}\sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} H(i)$$

$$\lim_{k\to\infty} a_j(k) = \frac{1}{n}\sum_{i=1}^{n} H(i)^T \mathsf{C}\left\{\varepsilon(i)\right\}^{-1} z(i)$$

Distributed Least Squares Algorithm

Hence, for all the $j = 1, \dots, n$ we have

$$\hat{x}^{LS} = \lim_{k \to \infty} F_j(k)^{-1} a_j(k).$$

Moreover, at each step (i.e., before reaching the steady state), every node can still compute a *locally unbiased weighted least-square estimate* providing that $F_i(t)$ is invertible.
As a consequence, *at each iteration* of the consensus protocol, a *new set of measurements* may arrive and fused coherently by means of the WLS.
At steady state, all the estimates converge to the *global WLS solution*!

Distributed Least Squares Algorithm

Dealing with time varying visibility

We now analyse the effect of time-varying graphs for the distributed WLS, introducing a couple of definitions.

Definition

Let $\mathcal{G}_i = (\mathcal{N}, \mathcal{E}_i)$, $i = 1, \dots, r$ denote a finite set of graphs sharing *the same set of nodes*. Their *union* is a graph $\mathcal{G} = \cup_{i=1}^{r} \mathcal{G}_i = (\mathcal{N}, \cup_{i=1}^{r} \mathcal{E}_i)$.

This is obvious by simply computing the overall *t-step transition matrix*.

Definition

Let $\mathcal{G}_i = (\mathcal{N}, \mathcal{E}_i)$, $i = 1, \dots, r$ denote a finite set of graphs sharing *the same set of nodes*. The set of graphs $\{\mathcal{G}_1, \dots, \mathcal{G}_r\}$ is called *jointly connected* if their *union* is *strongly connected*.

Distributed Least Squares Algorithm

Dealing with time varying visibility

We can now state that the graph is *connected in the long run* if the *infinitely occurring* communication graphs are *jointly connected*.
More precisely, for any graph with *switching topology*, there is a finite set of r graphs describing its configuration (at most, they are all the possible combination of communication links).
Therefore there is a finite set of associated r weight matrices Q_i, determined either with the *maximum degree* or the *Metropolis* algorithms.
Then the dynamic becomes

$$x(k+1) = Q_{i(k)}x(k), \text{ with } 1 \leq i(k) \leq r \text{ for all } k.$$

Distributed Least Squares Algorithm

Dealing with time varying visibility: an example

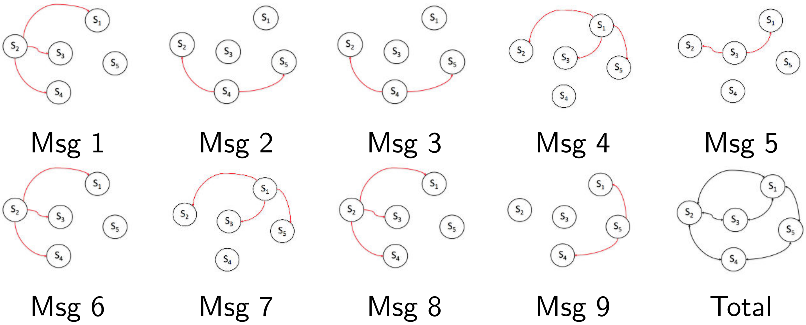

Figure: A possible sequence of a *jointly connected* graph.

Distributed Least Squares Algorithm

Dealing with time varying visibility

By the assumption derived previously, the sequence $\{i(k)\}_{k=0}^{\infty}$ is such that a subset of all the graphs occurs *infinitely often*.

Theorem

If the collection of communication graphs that occur infinitely often are jointly connected, *then the system*

$$x(k+1) = Q_{i(k)}x(k), \quad \textit{with } 1 \leq i(k) \leq r \textit{ for all } k,$$

converges to

$$\lim_{k\to\infty} x(k) = \left(\frac{1}{n}\mathbf{1}^T x(0)\right)\mathbf{1},$$

or, equivalently

$$\lim_{k\to+\infty} \Phi(k) = \frac{1}{n}\mathbf{1}\mathbf{1}^T.$$

Distributed Least Squares Algorithm

Dealing with time varying visibility

This theorem is given without proof. The proof can be found in:
Lin Xiao, Stephen Boyd, Sanjay Lall, *A Scheme for Robust Distributed Sensor Fusion Based on Average Consensus*, in Fourth International Symposium on Information Processing in Sensor Networks, pp.63-70, 15 April 2005
The proof is based on the existence of *paracontracting matrices*, i.e.,

$$Mx \neq x \Leftrightarrow \|Mx\| < \|x\|$$

Therefore, even with switching topology and under mild assumptions on communications (e.g., mobile agents that move randomly in the environment with finite communication range), the *optimal estimation problem is solved*!

Outline

1 Distributed WLS

2 Distributed Kalman Filter

Distributed Kalman Filter

Let us consider the problem of obtaining a *distributed estimate* of a linear dynamic system.
An example can be the estimate of an agent (e.g. a person) moving in an environment equipped with a surveillance system equipped with a network of n sensors (e.g. surveillance cameras).
As in the standard Kalman filter notes, the model can then be given by

$$x(k+1) = Ax(k) + Bu(k).$$

Distributed Kalman Filter

For the distributed Kalman filter, the model turns to

$$\hat{x}(k+1) = A\hat{x}(k) + \nu(k),$$

where $\nu(k) \sim \mathcal{N}(0, Q)$ models the uncertainties in the motion of the target.
Notice that this is quite obvious if the target is a person, but it is the same if the target is a robot, since it is not possible to get access to the *proprioceptive sensor data*, e.g. odometry.
For the measurement we have then

$$z_i(k) = H_i x_i(k) + \varepsilon_i(k),$$

where $\varepsilon_i(k) \sim \mathcal{N}(0, R_i)$ models the measurement error.
As customary in the Kalman filter literature, we assume the noises white and uncorrelated.

Distributed Kalman Filter

For the *centralised Kalman filter* we would have

- *Prediction step*:

$$
\begin{aligned}
\hat{x}(k+1)^- &= A\hat{x}(k) \\
P(k+1)^- &= AP(k)A^T + Q
\end{aligned}
$$

- *Update step*:

$$
\begin{aligned}
S(k+1) &= HP(k+1)^- H^T + R \\
W(k+1) &= P(k+1)^- H^T S(k+1)^{-1} \\
\hat{x}(k+1) &= \hat{x}(k+1)^- + W(k+1)\left(z(k+1) - H\hat{x}(k+1)^-\right) \\
P(k+1) &= (I - W(k+1)H)P(k+1)^-
\end{aligned}
$$

where $R = \text{diag}(R_1, R_2, \dots, R_n)$.

Distributed Kalman Filter

The equations can be rewritten more compactly as follows:

- *Prediction step* (the same as before):

$$\hat{x}(k+1)^- = A\hat{x}(k)$$
$$P(k+1)^- = AP(k)A^T + Q$$

- *Update step*:

$$\begin{aligned}\hat{x}(k+1) &= \hat{x}(k+1)^- + P(k+1)^- H^T \left(HP(k+1)^- H^T + R\right)^{-1} \cdot \\ &\quad \cdot \left(z(k+1) - H\hat{x}(k+1)^-\right) \\ P(k+1) &= P(k+1)^- - P(k+1)^- H^T \left(HP(k+1)^- H^T + R\right)^{-1} \cdot \\ &\quad \cdot HP(k+1)^-\end{aligned}$$

Distributed Kalman Filter

Using the *matrix inversion lemma*:

$$(A + BCB^T)^{-1} = A^{-1} - A^{-1}B(B^TA^{-1}B + C^{-1})^{-1}B^TA^{-1},$$

The update equations can be rewritten as follows.

Distributed Kalman Filter

State Update

State update:

$$
\begin{aligned}
\hat{x}(k+1) &= \hat{x}(k+1)^{-} + P(k+1)^{-} H^{T} \left(H P(k+1)^{-} H^{T} + R \right)^{-1} \cdot \\
&\cdot \left(z(k+1) - H \hat{x}(k+1)^{-} \right) = \\
&= P(k+1) \left(P(k+1)^{-} \hat{x}(k+1)^{-} + H^{T} R^{-1} z(k+1) \right) = \\
&= P(k+1) \left(P(k+1)^{-} \hat{x}(k+1)^{-} + \sum_{i=1}^{n} H_i^{T} R_i^{-1} z_i(k+1) \right) = \\
&= P(k+1) \left(P(k+1)^{-} \hat{x}(k+1)^{-} + a(k+1) \right)
\end{aligned}
$$

where

$$
a(k+1) = \sum_{i=1}^{n} H_i^{T} R_i^{-1} z_i(k+1).
$$

Distributed Kalman Filter

Covariance Update

Covariance update:

$$
\begin{aligned}
P(k+1) &= P(k+1)^- - P(k+1)^- H^T \left(HP(k+1)^- H^T + R\right)^{-1} \cdot \\
&\cdot HP(k+1)^- = \\
&= \left(P(k+1)^- + H^T R^{-1} H\right)^{-1} = \\
&= \left(P(k+1)^- + \sum_{i=1}^{n} H_i^T R_i^{-1} H_i\right)^{-1} = \\
&= \left(P(k+1)^- + F(k+1)\right)^{-1}
\end{aligned}
$$

where

$$
F(k+1) = \sum_{i=1}^{n} H_i^T R_i^{-1} H_i = F.
$$

i.e. it is independent from k if the number of sensors does not change.

Distributed Kalman Filter

To summarise:

$$\hat{x}(k+1) = P(k+1)\left(P(k+1)^{-}\hat{x}(k+1)^{-} + a(k+1)\right)$$
$$P(k+1) = \left(P(k+1)^{-} + F\right)^{-1}$$

Notice that the problem is quite similar to the WLS case!
In particular, to correctly execute the Kalman filter, only $a(k+1)$ and F involves *message exchanges*, while the other quantities can be computed locally.
The positive aspect is that $a(k+1)$ and F can be simply computed using *average consensus*.

Distributed Kalman Filter

So, as in the WLS case, we start by initialising for the i-th node:

$$a_i(k+1, 0) = H_i^T R_i^{-1} z_i(k+1) \text{ and } F_i(0) = H_i^T R_i^{-1} H_i,$$

where the 0 stands for the initialisation of the *average consensus* distributed state.

Hence, using the *average linear consensus*, we have

$$\lim_{q\to\infty} F_i(q) = \frac{1}{n}\sum_{i=1}^{n} H_i^T R_i^{-1} H_i = \frac{F}{n}$$

$$\lim_{q\to\infty} a_i(k+1, q) = \frac{1}{n}\sum_{i=1}^{n} H_i^T R_i^{-1} z(k+1) = \frac{a(k+1)}{n}$$

where q is the number of messages exchanged for the consensus protocol.

Distributed Kalman Filter

For the correct application of the Kalman Filter we have to assume that *each sensor node knows the number of nodes n* in order to update the filter coherently, i.e. for the i-th sensor:

$$
\begin{aligned}
\hat{x}_i(k+1) &= P_i(k+1)\left(P_i(k+1)^{-}\hat{x}_i(k+1)^{-} + n a_i(k+1,q)\right)\\
P_i(k+1) &= \left(P_i(k+1)^{-} + nF_i(q)\right)^{-1}.
\end{aligned}
$$

Therefore, before computing the *Update step*, the nodes run a consensus algorithm to reach an agreement for $a(k+1)$ and F.
Of course, an agreement is reached after a sufficiently large number of consensus iterations q, theoretically infinite. If this is not the case, a specific algorithm needs to be designed, such as *finite step convergence* or alternative Kalman approaches.