2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Asynchronous Implementation of a Distributed Average Consensus Algorithm

Maximilian Kriegleder, Raymond Oung, and Raffaello D'Andrea

***Abstract*— This paper discusses distributed average consensus in the context of a distributed embedded system with multiple agents connected through a communication network. Adversities such as switching of network topologies, agents joining or leaving the network, and communication link creation or failure may arise in these systems. To address these difficulties, we propose an asynchronous implementation of a distributed average consensus algorithm that has the following properties: (1) unbiased average, (2) homogeneous implementation, (3) robustness to network adversities, (4) dynamic consensus, and (5) well-defined tuning parameters. We demonstrate an application of the implementation on a specific distributed embedded system, the Distributed Flight Array, where we solve two average consensus problems to estimate altitude and tilt of the vehicle from multiple distance measurements.**

I. Introduction

The coordination of distributed systems with multiple agents connected through a communication network is crucial to their success and is mainly enabled by a family of protocols known as consensus algorithms [1]. These protocols allow a set of agents $\mathcal{N} = \{1, \ldots, N\}$ to reach an agreement on a quantity of interest by exchanging information over the communication network. Specifically, this quantity can be the average of values z_i,

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} z_i, \tag{1}$$

where z_i is stored at agent i. The problem can be solved in a distributed fashion through a family of algorithms known as *distributed average consensus*, where each agent exchanges information only with a subset of its peers, see [2]–[10] and references therein. This is useful in applications such as sensor fusion [11] and clock synchronisation [12].

In a previous paper [13] we presented a generalized method of estimating altitude and tilt of a rigid body using a network of distance measurement sensors. The solution to this problem in a centralized system (where all measurements are available) is obtained by solving a linear least squares problem. In a distributed system, however, it is usually the case that only limited information is available to each agent. To account for this, the least squares problem can be reformulated as two distributed average consensus problems.

We continue and extend the work of [13] to demonstrate the application of average consensus on a distributed embedded system, the Distributed Flight Array (DFA) [14], see Figure 1.

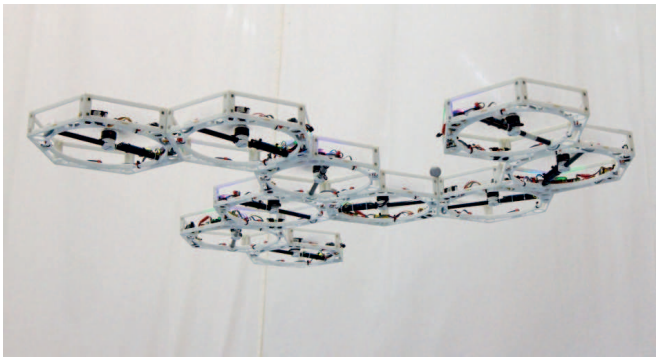

Fig. 1. The Distributed Flight Array is a modular robotics test bed, consisting of multiple autonomous agents that are able to coordinate with one another in order to drive and fly in a limitless number of configurations.

The main contribution of this paper is that we modify a synchronous distributed average consensus algorithm and provide an asynchronous implementation, which together satisfy the following key properties:

1) *Unbiased Average*. The consensus value reached is the precise average of the values z_i.
2) *Homogeneous Implementation*. The agents exchange information with one another employing a common set of rules.
3) *Robust to Network Adversities.* The implementation functions reliably in the case of asynchronous communication, loss of communication, and switching network topologies.
4) *Dynamic Consensus.* The consensus value eventually tracks the new average if the values z_i change.
5) *Well-Defined Tuning Parameters*. The convergence properties are well-defined and can be adjusted using a single tuning parameter.

The outline of this paper is as follows: We begin Section II by describing a synchronous distributed average consensus algorithm, which we modify to handle switching network topologies and dynamic consensus. In Section III we propose an asynchronous implementation of this algorithm to deal with network adversities. Section IV provides experimental results of this method subject to network adversities such as switching network topologies. We then demonstrate in Section V an application of this method in solving the described distributed estimation problem. Concluding remarks and an outlook on future work are presented in Section VI.

The authors are with the Institute for Dynamic Systems and Control, ETH Zurich, Sonneggstr. 3, 8092 Zurich, Switzerland {krmax, rdandrea}@ethz.ch, raymond.oung@alumni.ethz.ch.

II. Synchronous Distributed Average Consensus

In the following we describe an iterative distributed average consensus algorithm, which has a single tuning parameter. The convergence properties of this algorithm and the optimal tuning parameter, which minimizes the convergence time, have been explored in [3]. We repeat some of the results for completeness and provide a modification inspired by [5], which will be the basis for the asynchronous implementation.

We model a multi-agent network as an undirected and connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ consisting of a set of agents $\mathcal{N}$ and a set of edges $\mathcal{E}$, where each edge $\{i, j\} \in \mathcal{E}$ represents a bidirectional communication link between two distinct agents. The set of neighbours belonging to agent i is denoted by $\mathcal{N}_i = \{j | \{i, j\} \in \mathcal{E}\}$. For simplification, we may write i and j to respectively denote agent i and agent j.

Each agent i stores a state $x_i(k) \in \mathbb{R}$, which is dependent on an iteration counter value k and is initialized to $x_i(0) = z_i$. Essentially, each iteration of the distributed average consensus algorithm can be described through the following two-step sequence: 1) each agent i communicates its state information to its immediate neighbours $j \in \mathcal{N}_i$, and 2) all agents in the network update their state information through a linear combination of their own state information and the state information of their neighbours, which was communicated in the first step. Consequently, the states of all agents converge to the desired average $\bar{x}$ of values z_i.

The states $x_i(k)$ of all agents can be compactly represented by a vector $\mathbf{x}(k) \in \mathbb{R}^N$, where $\mathbf{x}(0)$ contains the values z_i, such that the described iterative procedure can be expressed mathematically as the following:

$$\mathbf{x}(k+1) = (\mathbf{I} - \alpha \mathbf{L})\mathbf{x}(k), \tag{2}$$

where $\mathbf{I}$ is the identity matrix, α is a scalar tuning parameter, and $\mathbf{L}$ is the Laplacian matrix [14] of graph $\mathcal{G}$. This matrix captures the graph topology and its elements are defined as

$$L_{ij} = \begin{cases} d_i & i = j, \\ -1 & \{i, j\} \in \mathcal{E}, \\ 0 & \text{otherwise}, \end{cases} \tag{3}$$

where d_i denotes the degree of i (the number of neighbours). Convergence of the algorithm is guaranteed if the tuning parameter α satisfies the following constraint:

$$0 < \alpha < \frac{1}{d_{\max}}, \tag{4}$$

where

$$d_{\max} = \max_i d_i, \tag{5}$$

denotes the maximum degree of all agents in the network. Therefore, this choice of weights requires some prior knowledge about the graph. Assuming the possible number of neighbours is constrained, however, an agent can compute the upper bound on α from the maximum possible number of neighbours. This leads to the possibility that a conservative bound on α will compromise the convergence time, which is defined as the (asymptotic) number of iterations needed for the error $\|\mathbf{x}(0) - \bar{\mathbf{x}}\|_2$ to decrease by a factor of $1/e$, where $\bar{\mathbf{x}} = (\bar{x}, \ldots, \bar{x}) \in \mathbb{R}^N$ denotes the average vector.

The convergence time decreases in general with the algebraic connectivity of a network, which is defined as the second smallest eigenvalue $\lambda_{N-1}(\mathbf{L})$ of the Laplacian matrix [14]. In the case where the agents know the entire graph represented by the Laplacian matrix, the agents can compute the optimal tuning parameter α^*, which minimizes the convergence time for a given network:

$$\alpha^* = \frac{2}{\lambda_1(\mathbf{L}) + \lambda_{N-1}(\mathbf{L})}, \tag{6}$$

where $\lambda_1(\mathbf{L})$ represents the largest eigenvalue of $\mathbf{L}$.

The iterative procedure (2) can be written element-wise as

$$x_i(k+1) = x_i(k) + \alpha \sum_{j \in \mathcal{N}_i} (x_j(k) - x_i(k)), \tag{7}$$

which suggests that each agent i computes its new state $x_i(k+1)$ using its previous state $x_i(k)$ and the sum of current disagreements $x_j(k) - x_i(k)$ scaled by a factor α. Note that the sum of the states $\sum_{i=1}^N x_i(k)$ at any iteration k is always equal to the sum of the values $\sum_{i=1}^N z_i$ stored at the agents. Intuitively, an agent i changes its state by the amount $\alpha(x_j(k) - x_i(k))$ on behalf of neighbour j and the neighbour changes its state by $\alpha(x_i(k) - x_j(k))$ in a single iteration. In total, the sum of all states remains constant at each iteration, which is one necessary condition for all states to converge to the desired average $\bar{x}$ [3].

The paper [5] proposes a modification for a continuous time consensus algorithm to handle switching network topologies. We modify the discrete time algorithm (7) accordingly by introducing an additional variable $\delta_{ij}(k)$ for each neighbour j of agent i, which stores the cumulative disagreement between two agents. With this modification, the algorithm (7) can be written as

$$x_i(k+1) = z_i + \alpha \sum_{j \in \mathcal{N}_i} \delta_{ij}(k+1) \tag{8}$$

$$\delta_{ij}(k+1) = \delta_{ij}(k) + x_j(k) - x_i(k), \tag{9}$$

where $\delta_{ij}(0)$ is zero for all i and j.

Writing the algorithm in this manner has two benefits:

1) The algorithm is explicitly dependent on the values z_i. Therefore, regardless of any change made to z_i, the algorithm will converge to the appropriate average. This is referred to as *dynamic consensus* [5] and is useful in a sensor network where the target being sensed changes with time, for example.
2) In the event that an agent leaves the network, all of its neighbours simply set the appropriate δ_{ij} to zero, such that its contribution is neutralized and the states converge to the appropriate average of values z_i in the new network. This assumes that an agent can detect when another agent leaves the network. As a result, the algorithm will converge to the appropriate average quantity over the network even after the topology has changed.

The major drawback concerning the approach (7), and consequently the modification (8) – (9), is that it assumes the state information from each agent in the network to be successfully transmitted to its corresponding neighbour(s) before the agents update their state information. As such, agents are prevented from stepping through to the next iteration until all agents in the network have received all state information from their neighbours. This approach therefore imposes a form of synchronization of information among all agents, which requires a non-trivial implementation of communication acknowledgements between agents. Consequently, this method is impeded by the slowest communication link.

III. Asynchronous Distributed Average Consensus

In this section, we propose an implementation of the distributed average consensus algorithm (8) – (9) that does not rely on synchronization of information. As mentioned previously one necessary condition for all states to converge to the desired average is that their sum remains constant. We propose an implementation that guarantees the preservation of this sum despite asynchronous exchange of information. In the following the iteration counter k is dropped to simplify notation.

The key idea of the asynchronous implementation is the following: if any change $\Delta_{ij} := x_j - x_i$ made to δ_{ij} is compensated by the opposite change $\Delta_{ji} = -\Delta_{ij}$ to δ_{ji}, the sum of the states x_i and x_j will remain constant, which can be verified by employing (8) – (9). We denote Δ_{ij} as the disagreement between i and j. Let an agent i initiate an interaction by sending its state to one of its neighbours j. The neighbour uses this information along with its own state to first compute its disagreement with i (which it adds to the cumulative disagreement with i), and then to update its state. Then j responds with the disagreement, which i subtracts from its cumulative disagreement with j and then also updates its state.

To guarantee that neighbouring agents do not initiate an interaction simultaneously (which could cause undesired effects) we impose a directed communication network based on unique numerical identifiers (ID) of the agents. In the following, i and j will be used to represent an agent's ID, where in this case i is not equal j. Between two neighbouring agents, only the agent with the lower ID can initiate an interaction and only the one with the higher ID can respond. To avoid situations where an agent uses a given neighbour's information multiple times, we append a unique sequence number SEQ_{ij} to each message sent. Each time an agent updates its state on behalf of a neighbour, it increments the corresponding sequence number to indicate that the next message to this neighbour will contain new information. The neighbour compares the incoming sequence number to a corresponding sequence number, which it stores in memory, to judge whether it received the same information before.

There are two components to the proposed implementation: a message sending routine (Algorithm 1), and a message handling routine (Algorithm 2). To see which parts of the routines (denoted ①, ②, ③, and ④) are employed during sending and handling of data, refer to Figure 2.

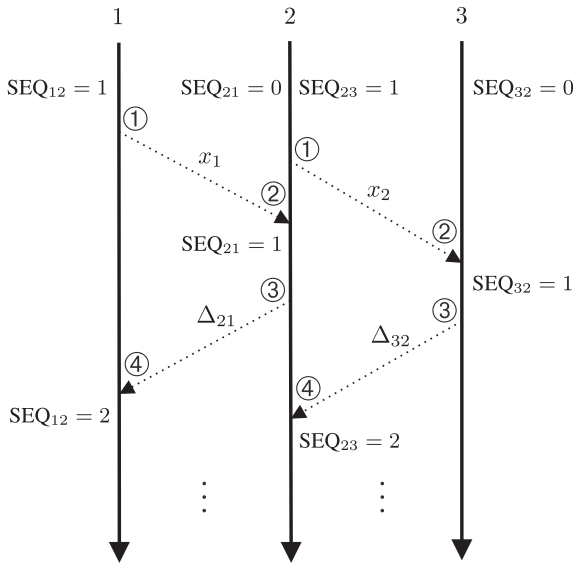

Fig. 2. This time-chart illustrates the message sending and handling of a three-agent network, where the agents $\{1, 2, 3\}$ are connected in a line (1-2-3). The symbols ①, ②, ③, and ④ indicate the appropriate parts of the routines described in Algorithm 1 and Algorithm 2. An agent i with a lower ID than its neighbour j initiates an interaction with a state message x_i and the neighbour responds with the disagreement Δ_{ji}. If any agent i sends a message it also appends the sequence number SEQ_{ij} (not shown here), which it increments each time it updates its state on behalf of neighbour j.

Algorithm 1 Message Sending Routine for agent i; executed at a periodic interval.

```
j: Neighbour ID
x_i: State of i
Δ_ij: Current disagreement between i and j
SEQ_ij: Sequence number stored at i for j

function MsgSend(j, x_i, Δ_ij, SEQ_ij)
    if j > i then
        DATA ← x_i                          ▷ ①
    else
        DATA ← Δ_ij                         ▷ ③
    end if
    Send DATA
    Send SEQ_ij
end function
```

Message Sending Routine: Messages contain the most recent data (i.e. state or disagreement) and are sent by agent i to its neighbour(s) j at a periodic interval to prevent deadlock behaviour (due to communication loss, for example). We can use this knowledge to implicitly determine whether or not an agent is disconnected from the network. To do this, we employ a watchdog timer for each neighbour, which is reset each time a new message is received from that neighbour. In the event that a watchdog is set, we classify that neighbour as being disconnected from the network and reset δ_{ij} to zero.

The implementation of the watchdog timer, however, is not shown in the Algorithms.

Depending on the ID of neighbour j, agent i will send one of two messages to this neighbour:

1) $j > i$: Agent i sends its current state x_i (denoted ① in Algorithm 1).
2) $j < i$: Agent i sends its disagreement Δ_{ij} (denoted ③ in Algorithm 1).

In both cases, agent i appends the corresponding sequence number SEQ_{ij}. The sequence number SEQ_{ij} of i is initialized to 1 for each neighbour $j > i$ and to 0 otherwise, such that the first interaction can be initiated.

Algorithm 2 Message Handling Routine for agent i; executed upon reception of a message.

```
// Stored information
z_i: Initial value of i
x_i: State of i
δ_ij: Cumulative disagreement between i and j
SEQ_ij: Sequence number stored at i for j

// Received information
j: Neighbour ID
DATA: Received data of j
SEQ_ji: Received sequence number of j for i

function MsgHandler(j, DATA, SEQ_ji)
    if j < i then // DATA is the state x_j
        if SEQ_ij + 1 = SEQ_ji then                    ▷ ②
            Δ_ij ← DATA − x_i
            δ_ij ← δ_ij + Δ_ij
            x_i ← z_i + α Σ_{j∈N_i} δ_ij
            SEQ_ij ← SEQ_ij + 1
        else
            // Do nothing
        end if
    else // DATA is the disagreement Δ_ji
        if SEQ_ij = SEQ_ji then                        ▷ ④
            Δ_ij ← −DATA
            δ_ij ← δ_ij + Δ_ij
            x_i ← z_i + α Σ_{j∈N_i} δ_ij
            SEQ_ij ← SEQ_ij + 1
        else
            // Do nothing
        end if
    end if
end function
```

Message Handling Routine: Upon reception from neighbour(s) j, agent i will take one of two actions:

1) $j < i$: This suggests that the message comes from an *initiator*. Therefore, if the message contains new information then the sequence number SEQ_{ji} should be one unit higher than the one that is stored in memory, SEQ_{ij}. In this case, agent i computes the disagreement Δ_{ij}, and updates the cumulative disagreement variable δ_{ij} and state x_i (denoted ② in Algorithm 2).
2) $j > i$: This suggests that the message comes from a *responder*. Therefore, if the message contains new information then the sequence number SEQ_{ji} should be equivalent to the one that is stored in memory, SEQ_{ij}. In this case, agent i updates the cumulative disagreement variable δ_{ij} with the received disagreement Δ_{ji}, and then updates the state x_i (denoted ④ in Algorithm 2).

In both cases, i increments the sequence number SEQ_{ij} to indicate that it updated its state on behalf of a message from j.

IV. Experimental Results

The distributed average consensus algorithm was implemented on the DFA in a manner described in the previous section. Each agent of this testbed is capable of full-duplex communication at 115.2 kbps with up to six connected agents and carries out the proposed algorithm together with a variety of other processes [15] on a single 32-bit 72 MHz microcontroller. All inter-agent communication is performed in an unreliable, but frequent manner using the user datagram protocol (UDP) [16].

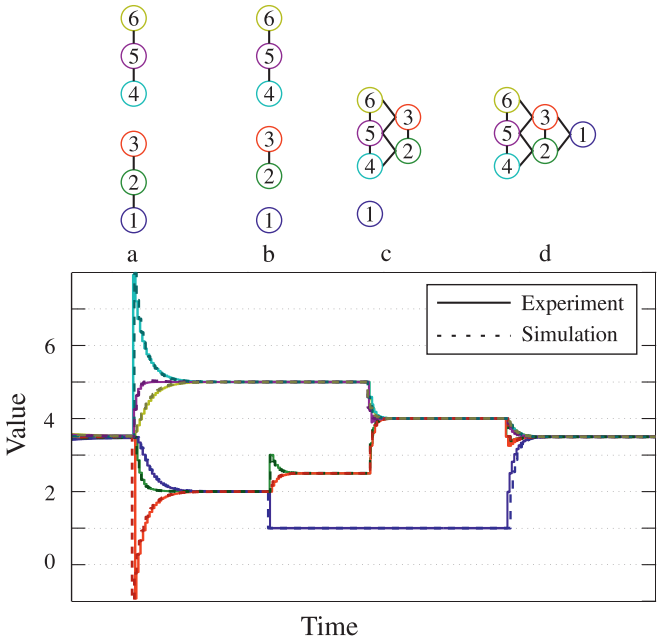

Fig. 3. This experiment illustrates the ability of the proposed algorithm to recover from switching topologies. Each of the six agents stores a distinct value $z_i \in \{1, \ldots, 6\}$ and the six agents start off connected in a line with an average of 3.5. The configuration is then divided first into two (a), and later into three (b) separate clusters. Finally, the three clusters are assembled into a single triangle-like topology (d). The states of the agents converge in each case to the appropriate average, which can be determined from the clusters on top of the graph, where the numbers indicate the values z_i. Simulation results are shown to correspond well with the experiment.

In this experiment, six agents are connected and disconnected as shown in Figure 3. The tuning parameter α is set to 0.165, which satisfies the condition (4) for all possible

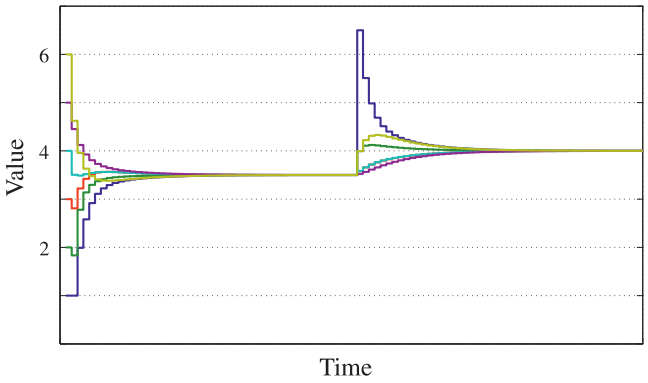

Fig. 4. Upon converging to its average of 3.5, where the state of six agents were initialized as $\{1, \ldots, 6\}$, one agent changes its value z_i from 1 to 4; without having to reset the algorithm, all states converge to the new average of 4.

network topologies in this experiment. The value of this parameter is far from the optimal (6). Nevertheless, the experiment demonstrates the algorithm's ability to recover from switching topologies (or loss of communication).

These results are also compared to those obtained from a simulation environment, which is implemented in MATLAB[1]. The results match well and thus validate the simulator, which accepts as input a Laplacian matrix and simulates the behaviour of each agent's state over a given number of iterations. It enables the simulation of a variety of network adversities, including: temporary loss of communication, switching topologies, time-varying average, and asynchronous updates. It enables us to quickly evaluate and illustrate the behaviour of the proposed algorithm under a variety of conditions. One such condition is the case of dynamic consensus, where we simulate a change in the values z_i during the iteration. The results of this experiment are shown in Fig. 4.

V. Distributed Altitude and Tilt Estimation

We now build upon the results in [13] and apply the proposed average consensus algorithm to solve a distributed linear least squares problem on the DFA. As a result we are able to estimate altitude and tilt of the vehicle from multiple distance measurements obtained by the agents.

Each agent i carries a distance measurement sensor, which is positioned at $(r_{X,i}, r_{Y,i})$, with respect to the body-fixed coordinate frame B of the vehicle and measures the distance $\hat{m}_i$ from the agent to the ground, see Figure 5. We showed in the previous paper that during nominal operating conditions, namely when the vehicle is near hover, the distance measurement $\hat{m}_i$ of sensor i can be modeled as a linear function of the sensor's position $(r_{X,i}, r_{Y,i})$, the vehicle's altitude z, its tilt (γ, β), and zero-mean sensor noise v_i, which is assumed to be independent and identically distributed with a variance of σ_m^2,

$$\hat{m}_i = z + r_{Y,i}\gamma - r_{X,i}\beta + v_i. \tag{10}$$

[1]The simulator can be downloaded from the project's website http://www.idsc.ethz.ch/Research_DAndrea/DFA.

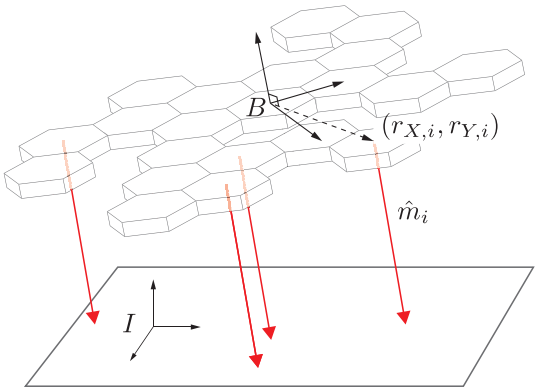

Fig. 5. The DFA's body-fixed coordinate frame B is located at its centre of mass and aligned with the principal axes of rotation. The orientation of the vehicle with respect to the inertial coordinate frame I can be described by a set of ZYX-Euler angles. Over a flat surface, it is possible to estimate altitude and tilt of the vehicle with respect to this surface using the distance measurements $\hat{m}_i$ of sensors i located at $(r_{X,i}, r_{Y,i})$ with respect to the vehicle's body coordinate frame.

Given at least three sensors that are not collinearly aligned the maximum-likelihood state estimate $\hat{\mathbf{s}} = (\hat{z}, \hat{\gamma}, \hat{\beta})$ can be obtained via least squares,

$$\hat{\mathbf{s}} = (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{m}, \tag{11}$$

where the elements of vector $\mathbf{m} = (\hat{m}_1, \ldots, \hat{m}_N)$ are the distance measurements from each sensor and the matrix $\mathbf{C}$ contains information pertaining to the positions of all the sensors,

$$\mathbf{C} = \begin{bmatrix} 1, & r_{Y,1}, & -r_{X,1} \\ \vdots & \vdots & \vdots \\ 1, & r_{Y,N}, & -r_{X,N} \end{bmatrix}. \tag{12}$$

This information can be obtained prior to flying; agents share information in order to agree upon a unique coordinate frame and compute for themselves the position of their distance measurement sensor with respect to this coordinate frame [15].

To solve the set of equations (11) in a distributed way, where each agent only communicates with its immediate neighbour(s), we borrow results from [17] and re-write the least squares solution (11) as

$$\hat{\mathbf{s}} = \mathbf{P}^{-1}\mathbf{q}, \tag{13}$$

where

$$\mathbf{P} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{c}_i^T\mathbf{c}_i, \quad \mathbf{q} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{c}_i^T\hat{m}_i, \tag{14}$$

and $\mathbf{c}_i$ denotes row i of the matrix $\mathbf{C}$. What we now have are two averaging problems (14).

The procedure for using the distributed least squares solution (13) – (14) for estimating altitude and tilt of the vehicle is outlined as follows. Each agent updates its measurement $\hat{m}_i$ periodically and independently of other agents (i.e. updates do not necessarily need to be synchronized). In between each measurement, the distributed average consensus method (summarized by Algorithm 1 and Algorithm 2)

is used to compute $\mathbf{P}$ and $\mathbf{q}$, see (14). Here, the matrix $\mathbf{c}_i^T\mathbf{c}_i$ and the vector $\mathbf{c}_i^T\hat{m}_i$ is initialized appropriately using the current position of each agent i and its most current distance measurement. The update rate of this altitude and tilt estimate is a function of the algebraic connectivity of the network, the network communication rate, and the number of iterations needed to achieve a desired precision of the estimate.

The proposed distributed altitude and tilt estimate was tested and verified to function as intended on the DFA. A 6-agent vehicle configuration similar to Figure 3 (d), with an algebraic connectivity of $\lambda_{N-1} = 1.70$, was used in the experiment described here. For the algorithm, we used the optimal tuning parameter $\alpha^* = 0.286$ and 4 iterations for each set of new measurements, which amounts to a state update rate of 5 Hz. The IR distance measurement sensors of each agent were calibrated, and the vehicle was positioned and tilted by hand in the workspace of a 3D motion capture system (Vicon MX) – the measurements of which we use as ground truth.

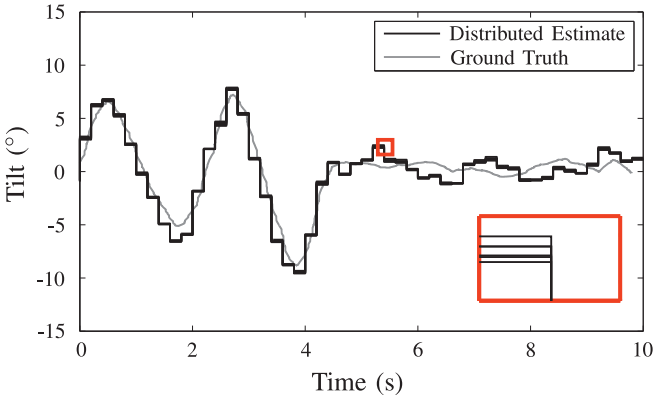

Fig. 6. Shown here is a plot of the tilt estimate along one of the rotational degrees of freedom; the inset image shows a magnification of a particular area in the plot, illustrating the deviations between estimates. The distributed tilt estimate of each agent compares well with ground truth measurements.

Figure 6 compares the distributed tilt estimate of each agent with measurements obtained from the 3D motion capture system (ground truth). The results verify the functionality of the proposed distributed method and demonstrate that the estimate is relatively precise.

VI. Conclusions

This paper proposes an asynchronous implementation of a distributed average consensus algorithm, which we modify to handle switching topologies and dynamic consensus. The implementation is homogeneous, employing a common set of rules for all agents in the network. In addition, the implementation handles network adversities such as communication link and/or agent failure and creation. Future work will formally analyse the influence of asynchronous exchange of information on the convergence properties of the proposed implementation.

We demonstrate through experiments that the algorithm is indeed robust, in particular to communication loss and switching network topologies. As one possible application of this method, we demonstrate its use in solving a distributed linear least squares problem, which in this case enables a multi-agent testbed to estimate its altitude and tilt from a network of distance measurement sensors. This distributed estimate is, however, delayed and has a relatively slow update rate that scales with the algebraic connectivity of the network. The tilt estimate could be combined with local angular rate measurements in order to obtain a real-time unbiased distributed tilt estimate of the vehicle in flight.

VII. Acknowledgements

This work is supported by the Swiss National Science Foundation (SNSF).

References

[1] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[2] F. Garin and L. Schenato, "A survey on distributed estimation and control applications using linear consensus algorithms," in *Networked Control Systems*. Springer London, 2010, vol. 406, pp. 75–107.

[3] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65 – 78, 2004.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: Design, analysis and applications," in *24th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2005, pp. 1653–1664.

[5] D. Spanos, R. Olfati-Saber, and R. Murray, "Dynamic consensus on mobile networks," in *IFAC World Congress*, 2005.

[6] M. Mehyar, D. Spanos, J. Pongsajapan, S. Low, and R. Murray, "Asynchronous distributed averaging on communication networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 512 –520, 2007.

[7] S. Patterson, B. Bamieh, and A. El Abbadi, "Distributed average consensus with stochastic communication failures," in *IEEE Conference on Decision and Control*, 2007, pp. 4215–4220.

[8] A. Nedic, A. Olshevsky, A. Ozdaglar, and J. N. Tsitsiklis, "On distributed averaging algorithms and quantization effects," *IEEE Transactions on Automatic Control*, vol. 54, no. 11, pp. 2506–2517, 2009.

[9] F. Bénézit, V. Blondel, P. Thiran, J. Tsitsiklis, and M. Vetterli, "Weighted gossip: Distributed averaging using non-doubly stochastic matrices," in *IEEE International Symposium on Information Theory Proceedings*, 2010, pp. 1753–1757.

[10] M. Franceschelli, A. Giua, and C. Seatzu, "Distributed averaging in sensor networks based on broadcast gossip algorithms," *Sensors Journal, IEEE*, vol. 11, no. 3, pp. 808–817, 2011.

[11] R. Olfati-Saber and J. Shamma, "Consensus filters for sensor networks and distributed sensor fusion," in *IEEE Conference on Decision and Control*, 2005, pp. 6698–6703.

[12] Q. Li and D. Rus, "Global clock synchronization in sensor networks," *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 214–226, 2006.

[13] M. Kriegleder, R. Oung, and R. D'Andrea, "Distributed altitude and attitude estimation from multiple distance measurements," in *IEEE/RSJ International Conference of Intelligent Robots and Systems*, 2012, pp. 3626–3632.

[14] R. Merris, "Laplacian matrices of graphs: a survey," *Linear Algebra and its Applications*, vol. 197198, no. 0, pp. 143 – 176, 1994.

[15] R. Oung and R. D'Andrea, "The Distributed Flight Array: Design, implementation, and analysis of a modular vertical take-off and landing vehicle," *International Journal of Robotics Research, Accepted for publication*, 2013.

[16] J. Postel, "User datagram protocol," *Isi*, 1980.

[17] L. Xiao, S. Boyd, and S. Lall, "A scheme for robust distributed sensor fusion based on average consensus," in *Fourth International Symposium on Information Processing in Sensor Networks*, 2005, pp. 63 – 70.