Distributed Spatiotemporal Gesture Recognition in Sensor Arrays

HOMA HOSSEINMARDI, AKSHAY MYSORE, NICHOLAS FARROW,
NIKOLAUS CORRELL, and RICHARD HAN, University of Colorado Boulder

We present algorithms for gesture recognition using in-network processing in distributed sensor arrays embedded within systems such as tactile input devices, sensing skins for robotic applications, and smart walls. We describe three distributed gesture-recognition algorithms that are designed to function on sensor arrays with minimal computational power, limited memory, limited bandwidth, and possibly unreliable communication. These constraints cause storage of gesture templates within the system and distributed consensus algorithms for recognizing gestures to be difficult. Building up on a chain vector encoding algorithm commonly used for gesture recognition on a central computer, we approach this problem by dividing the gesture dataset between nodes such that each node has access to the complete dataset via its neighbors. Nodes share gesture information among each other, then each node tries to identify the gesture. In order to distribute the computational load among all nodes, we also investigate an alternative algorithm, in which each node that detects a motion will apply a recognition algorithm to part of the input gesture, then share its data with all other motion nodes. Next, we show that a hybrid algorithm that distributes both computation and template storage can address trade-offs between memory and computational efficiency.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms: Design, Algorithms, Performance, Reliability

Additional Key Words and Phrases: Distributed computing, error resilient, gesture recognition

ACM Reference Format:
Homa Hosseinmardi, Akshay Mysore, Nicholas Farrow, Nikolaus Correll, and Richard Han. 2015. Distributed spatiotemporal gesture recognition in sensor arrays. ACM Trans. Autonom. Adapt. Syst. 10, 3, Article 17 (September 2015), 19 pages.
DOI: http://dx.doi.org/10.1145/2744203

1. INTRODUCTION

Gesture-based control, ranging from simple directional swipes to input of complex characters, has emerged as a standard method for human–computer interaction (HCI). Most systems, however are spatially limited to sensor arrays (e.g., tactile displays, range-finding cameras such as the Xbox Kinect) and require information processing in a central unit. We wish to make gesture recognition available to a wider range of surfaces, which embed memory, computation, and communication capabilities and can therefore function without a central processing unit. Examples of such surfaces range from a building wall made of "smart bricks," robotic sensing skins, or "smart paint" [Butera 2002; McEvoy and Correll 2015]. In the short term, we are interested in smart building

This work has been supported by NSF awards #1153158 and #1150223. We are grateful for this support.
Authors' addresses: H. Hosseinmardi, A. Mysore, N. Farrow, N. Correll, and R. Han, Computer Science Department, University of Colorado Boulder, 430 UCB, Boulder, CO 80309; emails: {homa.hosseinmardi, akshay.mysore, nicholas.farrow, nikolaus.correll, richard.han}@colorado.edu.

DOI: http://dx.doi.org/10.1145/2744203

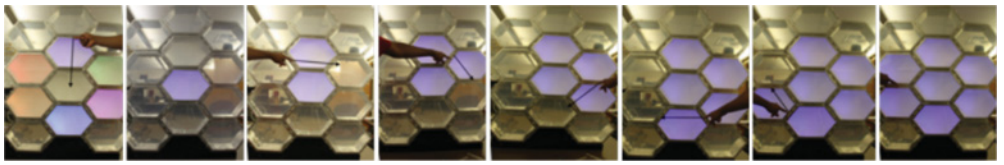

Fig. 1. An example simple interaction with a smart façade in our prior work: touching any of the wall elements will provide a color-coded "menu" whose items are selected by dragging them into the center. In this example, selecting the blue menu item switches to a dial-like submenu in which input is then provided via a circular gesture [Farrow et al. 2014].

materials that allow users to control environmental conditions using gestural input without a central computer processing the data and that can do so independently of the arrangement of the smart bricks. An example of simple swipe-based interaction with such a system is shown in Figure 1, which allows a user to use linear or circular gestures to control specific functions. In previous work, for example, Farrow et al. [2014], a user interacts with specific hardware functions through submenus providing slider-like or dial-like input for room temperature or brightness control. While such an approach provides a wide range of possible interactions by hard-coding distributed controllers to recognize up-down and circular swipes, limiting gestures to simple swipes quickly becomes nonintuitive as more functions become available, for example, if this interface should be used to control multiple aspects of a building, such as lighting, heating, ventilation, and cooling, or even selecting among different ambient music channels [Farrow et al. 2014]. Adding the capability to recognize actual letters and other complex gestures should therefore drastically increase the vocabulary and extensibility of such a system.

In the long run, we envision a new class of smart materials that completely embeds and integrates sensing, computation, and actuation [Paradiso et al. 2004]. Such a hardware base could use the presented distributed pattern recognition algorithms to allow for rich interaction with users. For example, when embedded in clothes [Nakad et al. 2003; Profita et al. 2015], gestures might allow communication with other devices, such as music players or cell phones, in an intuitive manner. Integration into a robotic skin [Hughes and Correll 2014; Lumelsky et al. 2001] might allow users to use tactile gestures to control the robot. Finally, materials with shape-changing capabilities [McEvoy and Correll 2014; McEvoy et al. 2013] could enable furniture that can adapt to their user's needs via rich gestural input.

In previous work, we developed multicast communication algorithms to limit communication in a distributed system to devices that are actually part of the gesture [Hosseinmardi et al. 2012; Ma et al. 2012], though that previous work did not investigate gesture recognition algorithms. This article is a first attempt to study distributed gesture recognition with a view towards supporting next-generation smart materials. A key challenge here is that all nodes that are part of the gesture need to exchange information with all other nodes, which is difficult in harsh communication environments with minimal bandwidth. Emphasis on minimalist computation, memory, and communication might enable distributed gesture recognition on other surfaces in which sensors and computation are tightly integrated. In such systems, links are likely to eventually fail due to imperfect low-cost fabrication and wear and tear of the materials, thus causing additional complications to a distributed gesture recognition algorithm.

In this article, we introduce three distributed algorithms that are adapted to minimize memory and computation, as well as a hybrid algorithm that minimizes both memory and computation. We show that the hybrid algorithm achieves performance that is comparable to the distributed computation algorithm in terms of recognition

reliability and robustness to packet loss, but maintains the memory efficiency of the distributed dataset algorithm.

2. RELATED WORK

There exists a large body of work on gesture recognition and online handwriting recognition in touchpads, tablets, tablet computers [Mitra and Acharya 2007; Wu and Huang 1999], or other electronic devices [Davis and Lyall 1986]. Although the gestures we are interested in are not necessarily limited to alphabet characters, stroke-based character recognition is a mature field, allowing encoding and recognizing a large variety of gestures. These algorithms deal exclusively with a centralized representation of the gesture, that is, gestures are recorded by a specific device such as a camera, range-finding device, or inertial-measurement unit (IMU) [Schlömer et al. 2008], co-located with a computer and processed thereon. In contrast, our article seeks to implement recognition of complex gestures in a fully distributed and scalable fashion.

Nagi et al. [2012] is most closely related in terms of distributing the gesture recognition problem and introduces a gesture recognition algorithm to overcome the perception limitations of simple swarm robots. Robots observe a simple hand gesture from different angles, attempt its classification, then share their posterior probability vectors and arrive via a distributed consensus at a decision. This approach does not consider distributing memory or communication. Rather, it capitalizes on the variations in perception that each robot incurs by looking at the gesture from a different angle. In our work, we aim instead to distribute computation and memory in order to perform classification, while being robustly aware of communication failures that may make individual units perceive a gesture slightly differently.

A common approach to encoding gestures/handwritten characters is to use a "chain code," taking into accounting its trajectory [Confer and Chapman 2004; Elmezain et al. 2009; Ozer et al. 2001]. In fact, Elmezain et al. [2009] show that features that convey trajectory information of the swipe are the most robust features for gesture/character recognition. For this, consecutive points of a gesture are sampled at regular intervals, and a discrete number, for example, 0 to 7 to encode angles in 45-degree intervals, are assigned based on the angle of the two consecutive samples. The resulting number streams can then be classified, for example, using Hidden Markov models (HMM) [Elmezain et al. 2009; Lee and Kim 1999; Wilson and Bobick 1999], Dynamic Time Warping (DTW) [Carmona and Climent 2012; Sakoe and Chiba 1990], support vector machines [Bahlmann et al. 2002], or neural networks [Murakami and Taguchi 1991].

While these approaches are intellectually appealing, they suffer from high computational complexity that make them less suitable for resource-constrained distributed sensor arrays in smart materials. An alternative is to employ a simple and robust recognition method that simply compares the Euclidian distance of a sequentially acquired chain code [Confer and Chapman 2004]. Although this approach requires the users to provide the gestures in the correct direction, it has been successfully used on platforms such as the Palm Pilot user interface. This approach further exhibits the advantage that it can be modified to allow us to divide both the dataset and computational load across a grid of distributed nodes. We selected this algorithm over an HMM or DTW approach due to its computational simplicity and its proven robustness, which makes it potentially applicable for implementation in distributed architectures with very limited computational capabilities. For example, DTW has computational complexity $O(n^2)$ since it uses a dynamic programming algorithm to compute the distance between two chain codes, while Confer and Chapman [2004] use simply Euclidean distance with $O(n)$ to compute the distance, where n is the dimension of the feature vector. We also note that distributing DTW is not straightforward as the underlying

dynamic programming approach requires all parts of the dataset to provide an optimal estimate.

The idea to create user interfaces in materials that are equipped with distributed computation was first articulated by Lifton et al. [2002], who introduced "push-pin computing," and the idea of a smart, distributed flooring system [Schlömer et al. 2008] as an architecture for algorithms such as what is proposed in this article. Yet, these ideas have become economically feasible only recently with microcontrollers becoming more powerful, smarter, and cheaper, which we believe will lead to a renewal of interest in distributed algorithms that can operate on such platforms.

3. BASIC GESTURE RECOGNITION ALGORITHM

We consider a distributed array of sensing nodes arranged as a lattice, with each node in the lattice having limited computational abilities and local connectivity to its immediate neighbors. We further assume that all nodes involved in a gestural event can communicate with each other by multihop, multicast communication [Hosseinmardi et al. 2012; Ma et al. 2012]. In this manner, each node involved in the gesture (*motion node*) will create a table of other group members' unique identification number *ID*s, and their relative coordinates and temporal order that can be inferred from the network topology, for example, a square grid or a hexagonal arrangement.

As in centralized gesture recognition, pattern recognition will be based on comparison of an input stream with a dataset of template vectors. A key challenge here is that gestures likely have different sizes and therefore require scaling as a preprocessing step. In our scenario, gesture size is determined by the number of motion nodes detecting such a gesture. Therefore, data vectors inside motion-node tables need to be interpolated or extrapolated to match the template length. Classification can then be performed by finding the template that minimizes a distance metric with the measurement vector. While there are multiple ways to encode the gesture and perform training and classification, the algorithms described in this article build up on Confer and Chapman [2004], who proposed a centralized algorithm that uses the Euclidian distance between sequential chain codes. We chose this algorithm due to its linear computational complexity, proven robustness on a full-sized alphabet, and relative ease to distribute both its memory and computational requirements.

In order to create a feature vector in a distributed system, a motion node emits a data packet upon sensing an event such as touch or light. Such an event packet conveys both the spatial position information of the node that has detected an event and temporal information about the time that the event was recorded. This information is then used on each node receiving the packet to reconstruct a complete spatiotemporal feature vector from all the collected packets.

After describing preprocessing steps that are common to all algorithms presented in this article, we propose methods for distributing memory, distributing computation, and combining both into a hybrid method.

3.1. Feature Extraction

As shown in Figure 2, when someone draws a gesture, each motion node that senses the gesture also needs to learn about other nodes that sense the motion. As described in Hosseinmardi et al. [2012] and Ma et al. [2012], after detecting an event, each motion node will share its *ID*, spatial information, and order information with other motion nodes. Each node gathers the spatiotemporal information into a table that will be used later to extract feature vectors.

Because different directions of input gestures might have different meanings, all nodes need to know the order in which events were recorded. Nodes will sort member vectors and their spatial information based on the gesture temporal order for the

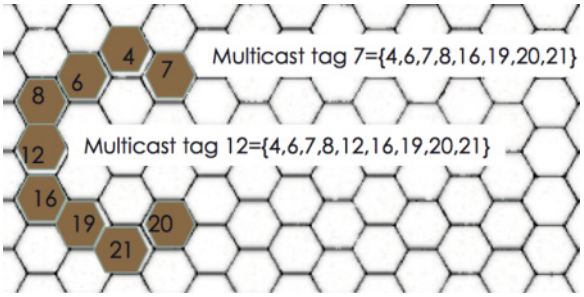

Fig. 2. The shaded nodes have detected the motion gesture with the order {7, 4, 6, 8, 12, 16, 19, 21, 20}. They will start sharing their packet ID. All motion nodes and their immediate neighbors will help in forwarding data packets of other motion nodes; however, some packets may not be delivered correctly (the data packet of node 12 has not been delivered to node 7). Finally, node 7 has a table including information from nodes {4, 6, 7, 8, 16, 19, 20, 21}, and node 12 has a table including information from nodes {4, 6, 7, 8, 12, 16, 19, 20, 21}.

Fig. 3. Input curve of arbitrary size at a motion node, in which red circles are sensors coordinates (left) and asterisks are used to show the interpolated points to get a fixed gesture length $N + 1$ (right).

purpose of feature extraction. Each node m will retrieve a certain number of gesture points from collecting shared packets by other nodes sensing the same gesture. The node will maintain three data vectors as motion packets are collected: $\boldsymbol{ID}^m$, $\boldsymbol{g}_x^m$ and $\boldsymbol{g}_y^m$, where the superscript indicates that these vectors are collected by node m. The $\boldsymbol{g}_x^m$ and $\boldsymbol{g}_y^m$ vectors correspond to the spatial coordinates of the sensors that detected the motion. In order to compare gesture vectors to a dataset of possible gesture templates, a new sequence of vectors will need to be computed into a vector chain whose set size N is of equal size to the templates in the dataset. For this purpose, each node m will interpolate $\boldsymbol{g}_x^m$ and $\boldsymbol{g}_y^m$ vectors to obtain $\boldsymbol{x}^m$ and $\boldsymbol{y}^m$ vectors, which now describe a unique feature gesture composed of N chain vectors connecting $N + 1$ ordered points. For example, in Figure 3, any number of gestural sensing events may be collected at a particular node; these are interpolated to produce an $N + 1$ point gesture.

Considering that users may draw gestures with different sizes, thereby spanning different numbers of nodes, and that packet loss may cause different group members to have different numbers of nodes in their table, input gestures are likely to have different vector lengths. The number of motion-node packets received by m will be denoted with M. Since some of the packets might not be properly recorded or delivered, each node has a potentially different number of gesture points M, and consequently a different view of the same gesture. However, we employ the chain technique to extract a fixed-length feature vector on each node independently. In this way, the fixed-sampling technique is fairly robust within limits both to how small a gesture is made and to packet loss or sensor failure. Figure 4 shows three chain vectors with $N = 10, 16,$ and 32.

A gesture can now be represented by the N vectors that are all the same length connecting an interpolation of the sample points. We denote a chain of vectors recorded

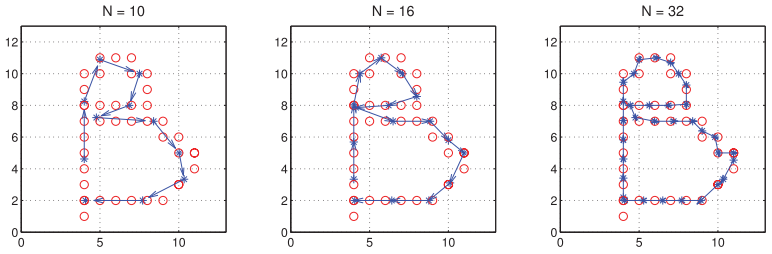

Fig. 4. Chain vector for different N in a 12×12 network.

by node m as $V_m = \{\overrightarrow{v}_1^m \dots \overrightarrow{v}_N^m\}$

$$\overrightarrow{v_i^m} = \ell_x^m(i)(1 \quad 0)^T + \ell_y^m(i)(0 \quad 1)^T. \tag{1}$$

Here, each element of $\ell_x^m(i)$ and $\ell_y^m(i)$ is calculated from the (x_i^m, y_i^m) and (x_{i+1}^m, y_{i+1}^m) coordinates of the points after interpolation

$$\ell_x^m(i) = x_{i+1}^m - x_i^m, \qquad where \quad i = 1, 2, \dots, N \tag{2}$$

$$\ell_y^m(i) = y_{i+1}^m - y_i^m, \qquad where \quad i = 1, 2, \dots, N, \tag{3}$$

where i and $i+1$ are adjacent points and

$$\boldsymbol{x}^m = (x_1^m, x_2^m, \dots, x_{N+1}^m) \tag{4}$$

$$\boldsymbol{y}^m = (y_1^m, y_2^m, \dots, y_{N+1}^m). \tag{5}$$

3.2. Basic Matching Algorithm

In order to recognize what gesture is made, we use a 1-nearest neighbor (1-NN) clustering algorithm to find the template s that is closest to the gesture m.

Let $\delta_{ms} = |V_m - V_s|$ be the distance between a vector chain on node m and template $s \in \mathcal{DS}$ a dataset containing all templates. The function $f(V_m)$ will return the lowest distance between input gesture vector chain on node m and dataset template chains s such that

$$f(V_m) = \min_s |V_m - V_s|. \tag{6}$$

Here, $|V_m - V_s| = \sum ||\overrightarrow{v}_i^m - \overrightarrow{v}_i^s||$ is the distance between two chain vectors and $||.||$ is the Euclidean distance between two vectors. Algorithm 1 summarizes our basic matching algorithm.

ALGORITHM 1: Each node m will compute the total chain vector difference between its input vectors and all template vectors.

1: Wait until the gesture is complete
2: **if** it is a motion node m **then**
3: Sort table
4: Interpolate sensor coordinates to find $N+1$ points along the gesture
5: Create $V_m = \{\overrightarrow{v}_1^m \dots \overrightarrow{v}_N^m\}$ using Equations (1) through (3)
6: $\delta_{ms} = |V_m - V_s|$, for all $s \in \mathcal{DS}$
7: $s^* = \arg\min_s \delta_{ms}$ and $s = 1, \dots, |\mathcal{DS}|$
8: $\tilde{c}$ = character of (s^*)
9: goto 1
10: **end if**

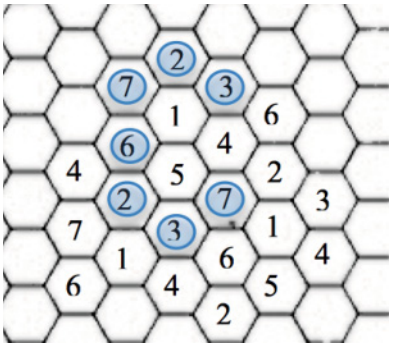

Fig. 5. Possible distribution of pattern datasets between nodes. Note that the dataset has been split into $K = 7$ subsets, with each node storing only one subset. Motion nodes corresponding to the character "C" are highlighted.

4. DISTRIBUTED GESTURE RECOGNITION ALGORITHMS

In this section, we propose three algorithms that achieve distributed gesture recognition: (1) Distributed Dataset (DD), (2) Distributed Vector (DV) computation, and (3) Hybrid Approach (Distributed Dataset and Vector computation, DDV).

4.1. Distributed Dataset Algorithm

In order to match a gesture to a known pattern, all possible patterns need to be stored somewhere in the system. The dataset $\mathcal{DS}$ includes a specific number of templates V_s for each possible input. The size $|\mathcal{DS}|$ depends on the granularity of the pattern and the number of possible patterns that can be recognized. Dividing $\mathcal{DS}$ into K subsets can therefore lead to considerable memory savings and the only solution to make high granularity/high variability pattern recognition feasible on a distributed system with minimal memory. In this case, memory requirements are traded off with additional communication requirements for nodes to retrieve dataset information.

In order to minimize this communication overhead, we propose storing the partial datasets ds_n in node n in such a way that every node m has access to the full dataset via its one-hop neighborhood $\mathcal{N}(m)$, that is,

$$\mathcal{DS} = \bigcup_n \{ds_n | n \in \mathcal{N}(m) \cup m\}. \tag{7}$$

We can now compute the distances between the vectors of the input gesture with the templates contained in the partial dataset ds_n. In this manner, neighbors $\mathcal{N}(m)$ of each motion node m will help in computing the distances from other parts of a dataset. Since the template s that has the minimum distance from the input vector chain will be chosen as the output $\hat{g}$, each neighbor node $n \in \mathcal{N}(m)$ will send just the minimum distance and the corresponding template number s $(s, \delta_n = \min_s \delta_{ns})$ to its group members.

Consider Figure 5, in which a dataset has been distributed in $K = 7$ parts and all nodes have access to the complete dataset via their one-hop neighbors. For an input "C" drawn on the network, we can see that the gesture has covered nodes with ds_2, ds_3, ds_6, and ds_7, but did not extend to nodes containing ds_1, ds_4, and ds_5. For each node m, its neighbor nodes that have different parts of the gestural dataset assist in calculating the distance from m's vector V_n to their templates V_s and share that information back with node m. At this point, m can choose the template that has the smallest δ_n.

Since all neighbor groups contain the complete dataset, instead of sharing δ_n among group members, it will be shared between one-hop neighbors. In this way, all neighbor

nodes $n \in \mathcal{N}(m)$ will send their (s, δ_n) to node m. Node m will compare all the distances δ_k for $k \in K$, find the minimum of them, and select template number s corresponding to the minimum δ_n as the output $\tilde{c}$, that is, the closest gesture in the dataset, which is the consensus computational decision of the distributed set of nodes. The pseudo-code for the DD is provided in Algorithm 2.

ALGORITHM 2: Distributed Dataset (DD): Each node n has access to a subset of dataset ds_n. The dataset is distributed between nodes such that for each motion node m

$$\mathcal{DS} = \bigcup_n \{ds_n | n \in \mathcal{N}(m) \cup m\}$$

```
Wait until the gesture is complete
if it is a neighbor node n then
    Sort its table, find motion nodes that are on its one-hop neighborhood
    Interpolate sensor coordinates to find N + 1 points along the gesture
    for all motion node m in the neighborhood of node n do
        Create V_m = {v_1^m ... v_N^m} using Equations (1) through (3)
        δ_ns = |V_n − V_s| for all s ∈ ds_n
        δ_n = min_s δ_ns, s = arg min_s δ_ns
        P_n ← ( character of (s), δ_n)
        Send packet P_n to neighbor motion node m
    end for
    go to step 1
end if
if it is a motion node m then
    Sort its table
    Interpolate sensor coordinates to find N + 1 points along the gesture
    Create V_m = {v_1^m ... v_N^m} using Equations (1) through (3)
    δ_ms = |V_m − V_s| for all s ∈ ds_m
    Receive packets from its neighbors
    n* = arg min_n δ_n, where n ∈ N(m) ∪ m
    c̃ = character in the packets from node n
    go to step 1
end if
```

As we will show later, distributing the dataset will result in dramatically lower robustness for recognizing patterns under the influence of packet loss. The reason is that when neighbor nodes send δ_k to node m, if the packet that includes the best template match is not delivered to m, it will impact heavily on the decision of node m.

4.2. Distributed Vector Computation

Distributing the dataset already distributes computation somewhat—distances are only calculated for the local part of the dataset ds_n—but we are also interested in further distributing computation to reduce computational requirements on any single node in the sensor array. The basic idea is that each node computes distances only for a subset of the vectors that make up a full gesture. The results of these computations can then be shared with their neighbors.

First, each node will compute the pieces of data with length $\lceil N/M \rceil$ in the interval $[a, b]$. The boundaries of this interval are calculated as follows for the p^{th} node:

$$a = (p-1) \times \lceil N/M \rceil + 1 \tag{8}$$

$$b = (p-1) \times \lceil N/M \rceil + \lceil N/M \rceil = p \times \lceil N/M \rceil \tag{9}$$

$$V_m^p = \{\overrightarrow{v}_i^m \mid i \in [a, b]\} \tag{10}$$

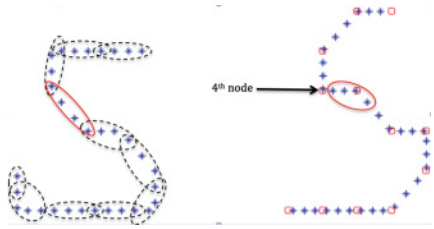

Fig. 6. Possible segmentation of a shape into elements of length $\lceil N/M \rceil$. Each node will calculate the distance of the partial vector of its observation (right) from the partial vector of dataset templates (left).

and

$$V_s^p = \{\overrightarrow{v}_i^s \mid i \in [a, b]\}, \tag{11}$$

where $\lceil . \rceil$ results in the first integer number equal or bigger than its argument. For motion node m, M is the size of its group. Since the size N/M is not always an integer number, the last motion node may have vector size less than $\lceil N/M \rceil$. In this way, vector V_m is divided to $N_{cmpt} = \frac{M}{\lceil N/M \rceil}$ and $p = 1, \ldots, N_{cmpt}$. Figure 6 illustrates a possible segmentation of N elements of a gestural chain vector into M subsets.

Each node will calculate the distance of its partial vector V_m^p from dataset templates partial vector V_s^p, $\delta_{ms}^p = |V_m^p - V_s^p|$ and send it out to all other group members. Each node m will put all δ_{ms}^p together for each template s to construct a complete distance vector $\delta_{ms} = [\delta_{1s}^p \ldots \delta_{Ms}^p]$.

While providing a linear decrease in computational time per node, this method requires more communication packets in order to share partial computation results, which would seem to make the algorithm more susceptible to packet loss. However, as we will show, the DV algorithm is actually more resilient to packet loss than the distributed dataset approach, because packet loss in the distributed computation approach only jeopardizes information about parts of a gesture. For example, if a particular motion node receives all but one of the partial comparisons from the other motion nodes, it will still have a fairly good estimate of the closest template gesture to the sensor data, since it will have all comparison segments but one. Pseudo-code for (DV) computation is provided in Algorithm 3.

ALGORITHM 3: Distributed Vector (DV): Each node n will compute the distance of part of its input vector from the corresponding part of all template vectors.

1: Wait until the gesture is complete
2: **if** it is a motion node m **then**
3: Sort its table, find its order p
4: Interpolate sensor coordinates to find $N + 1$ points along the gesture
5: $a = (p - 1) \times \lceil N/M \rceil + 1$
6: $b = p \times \lceil N/M \rceil$
7: $V_m^p = \{\overrightarrow{v}_i^m \mid i \in [a, b]\}$
8: $\delta_{ms}^p = |V_m^p - V_s^p|$ for all $s \in \mathcal{DS}$
9: $P_m \leftarrow \left(m, [\delta_{m1}^p \ldots \delta_{m|\mathcal{DS}|}^p]\right)$
10: Share packet P_m with other motion nodes
11: $\delta_{ms} = \sum_{p=1}^{N_{cmpt}} \delta_{ms}^p$ for all $s \in \mathcal{DS}$
12: $s^* = \arg\min_s \delta_{ms}$ for all $s \in \mathcal{DS}$
13: $\tilde{c} =$ character of (s^*)
14: go to step 1
15: **end if**

4.3. Hybrid Method

Combining distributed data storage and distributed computation allows a trade-off whereby the advantages of one technique mitigate the drawbacks of the other method. The distributed dataset approach is memory efficient but lacks robustness against packet loss, while the distributed computational vector approach is more computationally efficient and more robust to packet loss, but has large memory requirements when the dataset is not distributed.

We propose a hybrid distributed data and vector computation (DDV) algorithm that combines both approaches. Each motion node will compute the distance of V_n^p from part of the dataset templates ds_n. Neighbors $n \in \mathcal{N}(m)$ will send the distances of V_n^p from other parts of the dataset to the motion node m. Then, motion node m will share the distance of V_n^p from the whole dataset with all other group members. Pseudo-code for the hybrid DDV algorithm is shown in Algorithm 4.

ALGORITHM 4: Distributed Dataset and Vector (DDV): Each node has m access to a subset of dataset ds_m and will compute the distance of part of its input vector to the corresponding part of all template vectors for of ds_m. The dataset is distributed between nodes such that for each motion node m

$$\mathcal{DS} = \bigcup_n \{ds_n | n \in \mathcal{N}(m) \cup m\}$$

1: Wait until the gesture is complete
2: **if** it is a neighbor node n **then**
3: Sort its table, find motion nodes that are in its one-hop neighborhood
4: Interpolate sensor coordinates to find $N+1$ points along the gesture
5: **for all** motion node m with order p in the neighborhood of node n **do**
6: $a = (p-1) \times \lceil N/M \rceil + 1$
7: $b = p \times \lceil N/M \rceil$
8: Create partial vector of vectors: $V_n^p = \{\overrightarrow{v}_i^n \mid i \in [a,b]\}$
9: $\delta_{ns}^p = |V_n^p - V_s^p|$ for all $s \in ds_k$
10: $P_n \leftarrow (m, k, [\delta_{n1}^p \dots \delta_{n|ds_k|}^p])$
11: Send packet P_n to neighbor motion node m
12: **end for**
13: go to step 1
14: **end if**
15: **if** it is a motion node m **then**
16: Sort its table, find its order p
17: Interpolate sensor coordinates to find $N+1$ points along the gesture
18: $a = (p-1)\lceil N/M \rceil + 1$
19: $b = p \times \lceil N/M \rceil$
20: $V_m^p = \{\overrightarrow{v}_i^m \mid i \in [a,b]\}$
21: $\delta_{ms}^p = |V_m^p - V_s^p|$ for all $s \in ds_n$
22: Receive packets from its neighbor and construct $[\delta_{m1}^p \dots \delta_{m|\mathcal{DS}|}^p]$
23: $P_m \leftarrow \left(m, [\delta_{m1}^p \dots \delta_{m|\mathcal{DS}|}^p]\right)$
24: Share packet P_m with other motion nodes
25: $\delta_{ms} = \sum_{p=1}^{N_{cmpt}} \delta_{ms}^p$ for all $s \in \mathcal{DS}$
26: $s^* = \arg\min_s \delta_{ms}$ for all $s \in \mathcal{DS}$
27: $\tilde{c} =$ character of (s^*)
28: go to step 1
29: **end if**

As a comparison of the three different techniques, we summarize in Table I the order of savings achieved in computation and space of DD, DV, and DDV for a typical motion

Table I. Order of Savings in Computation and Space (Memory) for Motion Nodes Assuming Neighborhoods of Size K and Group Sizes M

	DD	DV	DDV
Space	K	1	K
Computation	K	N_{cmpt}	$K\ N_{cmpt}$

N_{cmpt} is defined in Section 4.2.

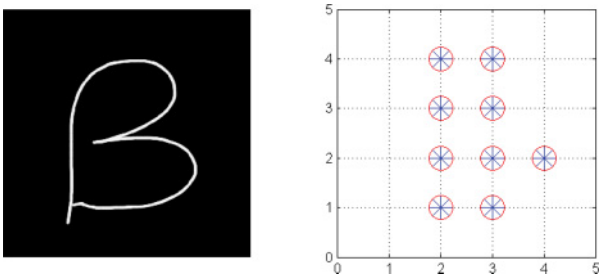

Fig. 7. A template from the training set (left) and resulting mapping of motion nodes in a 4×4 network (right).

node with a group of size M, K nodes in a neighborhood and a size of $|\mathcal{DS}|$ for the dataset.

5. EVALUATION

We evaluate the accuracy and performance of each of the three distributed algorithms in a Matlab simulation. We used a threefold cross-validation technique to evaluate the algorithms. We created 21 templates for each of the English alphabet's 26 characters. We will first evaluate the distributed algorithms and compare them to the Basic algorithm in a typical scenario of a 10×10 network and vector length $N = 16$. As we will see, the Hybrid DDV algorithm outperforms both the DD and DV algorithms. Therefore, we select it as the one for the remainder of the comparisons and evaluate the effect of other parameters such as network size and feature vector length on the performance of the Hybrid DDV algorithm. Finally, we consider applications that require smaller alphabets and how we can choose the subset of gestures that maximize accuracy.

5.1. Data Collection

In order to gather the dataset, we used a touch screen iPad as the input interface. We collected 21 different templates (size and shape) of gestures for each of the 26 characters for a total of 546 templates in the dataset (Figure 7, left). We have trained examples provided by a set of different students in order to generalize in a best possible way and should therefore be user independent. The drawn gesture was mapped to a sensor array/lattice as shown in Figure 7 (right). Each sensor node will be considered to be a motion node if the gesture has passed within the sensing range of the sensor. Each character has been drawn in a single-stroke gestural approach inspired by Graffiti [Hawkins 2002] (Figure 8).

5.2. Algorithm Selection

Simulations assume an event-driven message exchange and emulate packet loss by randomly dropping packets with a certain likelihood. We have tested all four algorithms: Basic, DD, DV, and DDV using data recorded from the iPad for a 10×10 network with feature vector length $N = 16$. We used the Bloom Filter-based multicasting algorithm

Fig. 8. Alphabet used in the dataset.

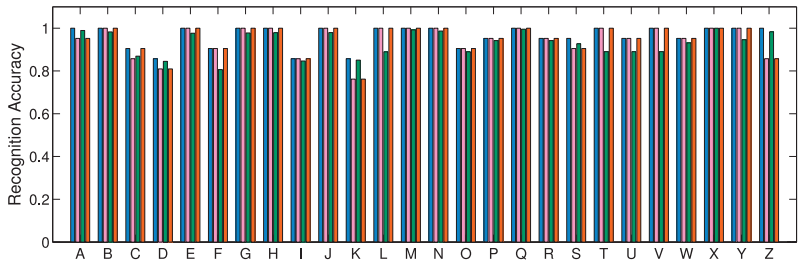

Fig. 9. Probability of gesture recognition for 546 templates of dataset for Basic (blue), DV (pink), DD (green), and DDV (red) algorithm when there is no packet loss. Experiment on a 10×10 network, $N = 16$.

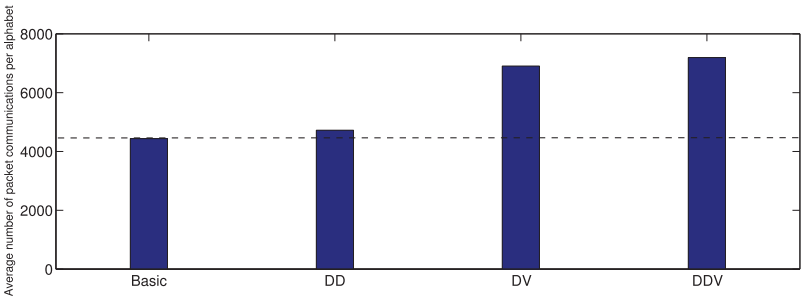

Fig. 10. Average number of packets exchanged for different algorithms (averaged over 546 inputs) when there is no packet loss. Experiment on a 10×10 network, $N = 16$.

from Hosseinmardi et al. [2012] with $TTL = 2$ and threshold 10. Figure 9 shows the accuracy of all algorithms on a 10×10 network with vector length $N = 16$ without any packet loss. There is a small decrease (around 2%) due to the effect at the edges of the network, where nodes do not have access to the complete dataset via their neighbors.

Figure 9 shows that, in the absence of packet loss, all the distributed algorithms work fairly well compared to the Basic algorithm, while also maintaining an advantage of memory and/or computation efficiency. They gain this property through sharing the data with their neighbors. Therefore, we assessed the communication overhead of distributing packets in the advanced algorithms, as shown in Figure 10. As expected, distributing the computation as is done in both the DV and DDV algorithms results in greater communication overhead than merely distributing the dataset. However, the relative increase in overhead is mitigated by the fact that all algorithms, including the Basic algorithm, incur substantial communication overhead because of the initial sharing of sensor data via multicast packets with other group members.

We can use the algorithms with no special changes for a wireless environment prone to packet loss. Since all three distributed algorithms require data-packet exchange to make a decision, we should consider the effect of packet loss on their recognition accuracy. Here, packet loss is simulated by dropping packets at a constant rate. In the following experiment, we have considered the effect of packet loss. We test our

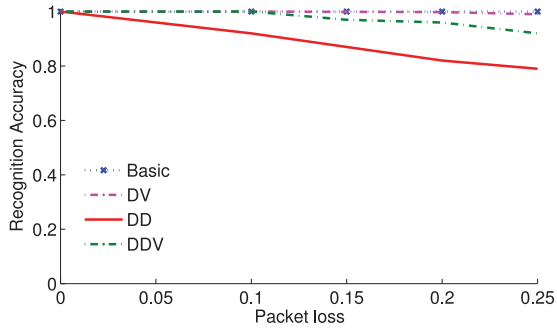

Fig. 11. Gesture recognition probability as a function of packet loss probability for an identical "A" shape compared to the dataset shown in Figure 8. Experiment on a 10×10 network, $N = 16$.

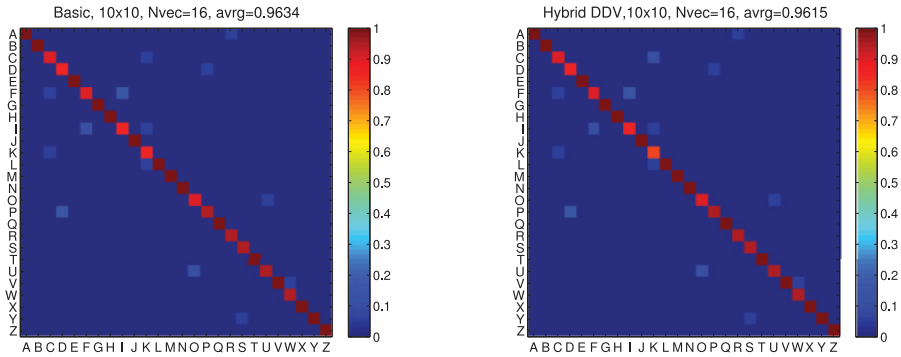

Fig. 12. Confusion matrix of Basic (left) and Hybrid DDV algorithm (right) when there is no packet loss. Experiment on a 10×10 network, $N = 16$.

algorithms on perfect input using an "A" shape without discontinuities. Figure 11 shows the detection probability as a function of packet loss probability and the chosen algorithm. We observe that the DV computation approach is more resilient to packet loss than the DD approach, though even the DD algorithm accuracy drops to only 80% with 25% packet loss, which can be viewed as an acceptable recognition rate. The DDV algorithm stays fairly robust to packet loss, inheriting the robustness to loss of partial computation results of the DV algorithm and is an improvement over the DD algorithm. As a result of its superior memory and computational efficiency, as well as resilience to packet loss, we choose to use the Hybrid DDV algorithm in our remaining analysis.

Another way to visualize the performance of the Hybrid DDV algorithm is via a confusion matrix: namely, which gestures were identified versus which gestures were input, as shown in Figure 12. We observe that the Hybrid algorithm has a fairly similar confusion pattern compared to the Basic algorithm. DDV has the additional virtues of lower memory requirements and lower computation per node.

5.3. Sensitivity to Network Resolution and Vector Length

We next evaluate the effect of network size and feature vector length on algorithm performance. The dataset is comprised of 26 characters that have been written in a single-stroke directional manner, as shown in Figure 8. We have distributed these 26 characters between neighbors such that each node can have access to the whole alphabet via its neighbors. As we mentioned in Section 3.1, a gesture will be considered as a chain

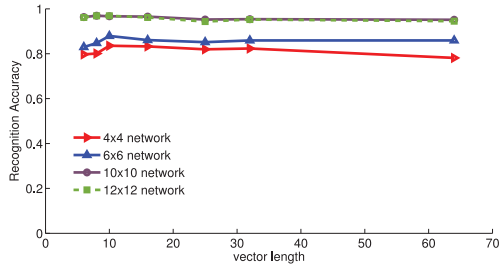

Fig. 13. Average recognition accuracy of Hybrid algorithm when there is no packet loss. Experiment on four different network sizes, $N = 6, 8, 10, 16, 25, 32, 64$.

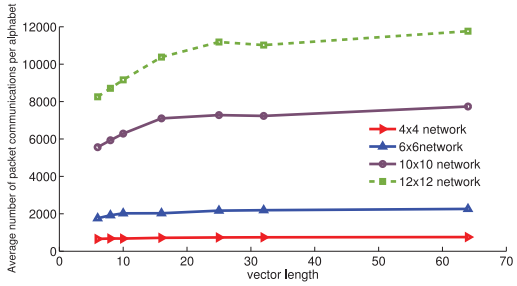

Fig. 14. Average number of packets exchanged (averaged over 546 input) using Hybrid algorithm when there is no packet loss. Experiment on four different network sizes, $N = 6, 8, 10, 16, 25, 32, 64$.

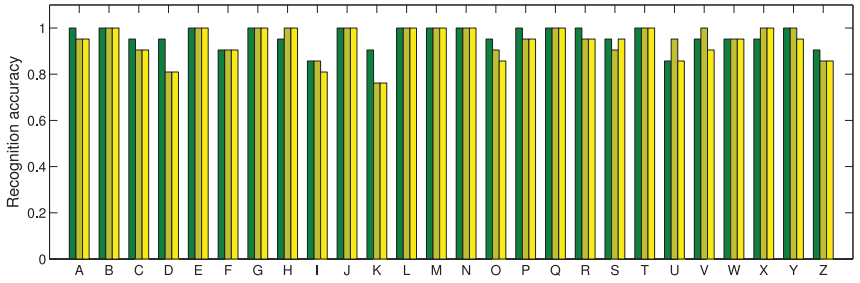

Fig. 15. Average recognition accuracy of Hybrid algorithm for 10×10 network and chain vector length N equal to 10 (dark green), 16 (light green), 32 (yellow).

vector with fixed length N. To examine the role of this parameter in the recognition algorithm, we have evaluated the recognition accuracy for $N = 6, 8, 10, 16, 25, 32, 64$ across a range of networks (4×4, 6×6, 10×10, and 12×12), as shown in Figure 13.

As we expected, the recognition accuracy was reduced as the network size decreased. This happened because the drawn gesture was captured by fewer sensors; as a result, the input character has lower resolution. For each network size, as we change the feature vector length, we see that there is a very slight change of maximum 4.9% in the recognition accuracy. On the other hand, as the network size increases, the number of exchanged packets increases, as seen in Figure 14.

In Figure 15, we look more closely at the effect of changing vector length on each alphabet recognition rate in a 10×10 network. We can see that increasing or decreasing vector length does not have the same effect on recognition for different characters. For example, D has the best result at $N = 10$, while U achieves the best recognition at

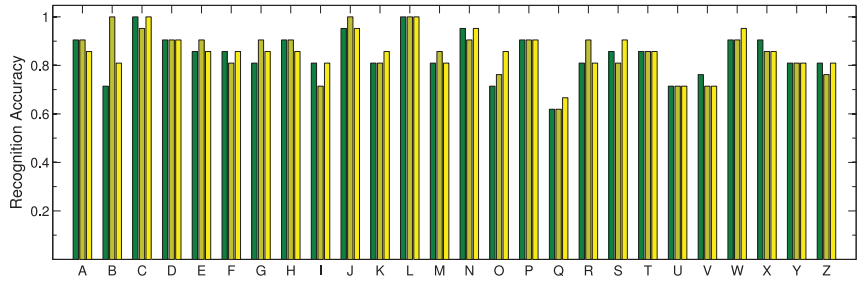

Fig. 16. Average recognition accuracy of Hybrid algorithm for 4×4 network and chain vector length N equal to 10 (dark green), 16 (light green), 32 (yellow).

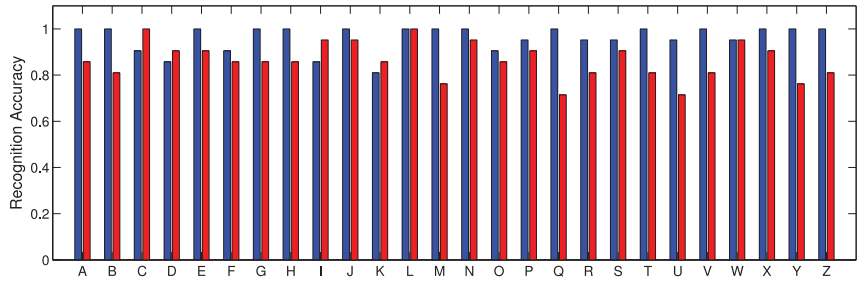

Fig. 17. Average recognition accuracy of Hybrid algorithm for 10×10 (blue) and 4×4 (red) network, $N = 16$.

$N = 16$. Surprisingly, for all characters, the recognition rate using length $N = 32$ does not exceed the rate when either $N = 16$ or $N = 10$.

In Figure 16, for a network size of 4×4, we also typically observe the worst result for $N = 32$. However, also note that there are some characters—for example, Q, S, and W—for which the accuracy has the highest rate at $N = 32$.

Figure 17 shows that reducing network size, which causes reduction on input character resolution, does not always have a negative effect on recognition (however, this is mostly true). For example, looking closely, we observe that for C and I, recognition accuracy has increased. On the other hand, we can see that the characters M, Q, and Y are worst affected by reducing the network size.

5.4. Selecting a Subset of an Alphabet

In some applications, a sensor lattice may be required to respond to only a small number of gestures. One research goal is to choose a subset of gestures that are most widely differentiated so that it is easiest to tell them apart. In this section, we show how we can choose a specific number of characters from the whole 26-character alphabet to achieve the best recognition percentage. As a case study, suppose that we need to choose 5 characters, for which the number 5 is chosen to allow us to optimally distribute the dataset on a 4-neighborhood grid structure so that each node has access to the entire dataset via its one-hop neighbors. In the following, we perform a heuristic experiment considering all $\binom{26}{5} = 65780$ possible combinations of 5-member groups. Figure 18 shows the number of 5-member groups that achieve an accuracy of 100% for different network sizes and vector length. We can see from Figure 18 that the worst case is for a network size 4×4 and vector length $N = 32$, though there still exist more than 2500 different combinations that achieve full recognition.

The question that arises here is how we can choose the subset that maximizes accuracy containing the most mutually distant characters in the feature space. We can

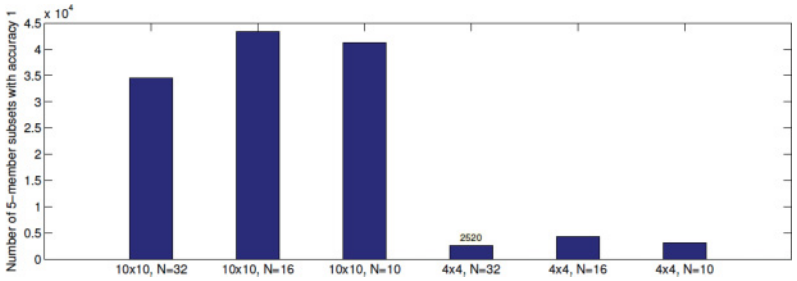

Fig. 18. Number of 5-member groups that have average accuracy 1.

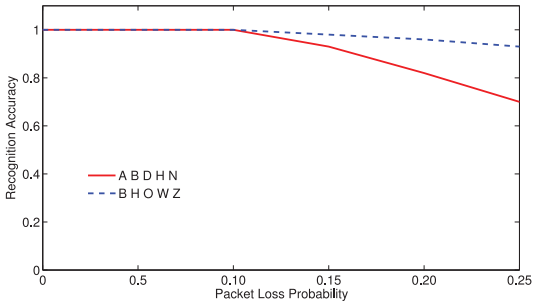

Fig. 19. Probability of gesture recognition for an identical B shape compared to two different 5-member sets, network size 4×4, $N = 16$, Hybrid DDV algorithm.

introduce an optimization problem to find the characters that have the most distinct feature vectors. The following experiment shows how different combinations act differently in the presence of packet loss. Figure 19 shows the difference between choosing two 5-member sets. In one group, we attempt to distinguish B from the set of A, B, D, H, N, of which four of the members have an upward start. In the other group, B, H, O, W, Z, each of the members have different starting directions. Both groups have a recognition accuracy of 100% in the absence of packet loss. We observe how the group A, B, D, H, N is more sensitive to packet loss than group B, H, O, W, Z. This example illustrates that other factors, such as differences in the initial directionality of the characters, may be helpful in selecting a subset of the alphabet that maximizes accuracy. More exploration is required to determine additional factors, such as starting direction, that would be useful in influencing optimal subset selection.

6. DISCUSSION

All experiments, including those for data gathered from our hardware setup, were simulated on a desktop computer. This allowed us to explore a wide range of memory and bandwidth requirements not necessarily supported by a particular hardware setup. Whereas all of the presented algorithms can be run on the hardware used here for the type of data presented (the hybrid algorithm exchanges 2490 packets, on average, each a few bytes in length), this might not be the case for larger datasets or gestures with higher resolutions, which will significantly slow down recognition and/or reduce accuracy if bandwidth is low.

Simulation results confirm that the DD approach is not very robust to packet loss despite being memory efficient. Distributed computation, in which only parts of the gesture are lost in the event of packet loss, turns out to be significantly more robust, but

scales poorly with increasing size of the dataset. The hybrid DDV approach maintains the memory efficiency of the DDs, while combining the computational efficiency of both. Performance in terms of recognition probability, however, is identical to that of the DV computation approach until packet loss is above around 10% for a 10×10 network with 26 different characters in the dataset.

The primary cost of the hybrid DDV algorithm is increased packet overhead compared to the Basic algorithm, though that is mitigated somewhat by the initial costs required in the Basic algorithm to share packets of raw sensor data among the motion nodes. We believe that many sensor array systems may find the additional cost of packet communication reasonable compared to the savings in memory and computation achieved by the hybrid DDV algorithm, especially those systems in which power to the sensor lattice is not an issue, such as in smart sensor walls.

A common challenge in recognition of written characters is that subsampling bears the risk of picking a vector that is not oriented toward the overall motion of the gesture, for example, due to jitter of the user's hand. While this problem could be prevented by an additional smoothing step before subsampling, we believe that jitter is a problem more specific to handwriting than to larger-scale hand motions. Yet, our results show that low resolution negatively affects pattern recognition.

Distributing the dataset exclusively among neighboring nodes allows minimization of the communication overhead of a distributed approach. By this, the overall size of the dataset is limited by the available memory on every node and the communication topology. In this article, we consider 4- and 6-member neighborhoods, but 8-member neighborhoods are also possible, depending on the available hardware. We also note that higher-connectivity degrees might increase the detection rate in the presence of packet loss for the hybrid approach.

We note that our current approach is not invariant to rotations. Doing this would require an additional normalization step that transforms each chain vector into a rotation-invariant representation. We note, however, that this would also limit the system to rotation-invariant gestures, possibly excluding, for example, C and U.

7. CONCLUSIONS

We presented a suite of distributed-gesture recognition algorithms for next-generation smart materials, which embed minimalist sensing, computation, and communication, and allow a trade-off of computation and memory for more communication among nodes. All algorithms are fully distributed, scalable to the size of the sensing lattice, and could be used on amorphous structures, that is, structures covered with a sensor lattice that have an arbitrary outer shape and possibly holes in the surface. Algorithms have been validated and tested with respect to packet loss in simulation using digitized hand-drawn gestures. These results show that a hybrid approach, which distributes both computation and the template dataset among nodes, can lead to high detection probabilities when packet loss is less than 10% on a grid lattice topology, even if the input gesture is presented with discontinuities due to either sensor failure or packet loss.

In the future, we wish to extend our work to multi-stroke alphabets as well as multi-alphabet input commands. We are also interested in validating the proposed advanced algorithm on physical hardware, and are in the process of building a smart-wall system. We plan to implement Bloom filter-based, multi-cast communication and a basic character-recognition algorithm on the hardware shown in Figure 1, and are also interested to implement the proposed algorithms on other smart materials such as skins and shape changing materials.

REFERENCES

C. Bahlmann, B. Haasdonk, and H. Burkhardt. 2002. Online handwriting recognition with support vector machines-a kernel approach. In *Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition, 2002*. IEEE, 49–54.

W. Butera. 2002. *Programming a Paintable Computer*. Ph.D. dissertation. Program in Media Arts and Sciences, School of Architecture and Planning, MIT, Cambridge, MA.

J. M. Carmona and J. Climent. 2012. A performance evaluation of HMM and DTW for gesture recognition *(Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications)*, L. lvarez, M. Mejail, L. Gmez Dniz, and J. C. Jacobo (Eds.). Springer, New York, NY, 236–243.

W. J. Confer and R. O. Chapman. 2004. System and method of handwritten character recognition. (April 2004). Patent No. 6,721,452. Filed September 12, 2002. Issued April 13, 2004.

R. H. Davis and J. Lyall. 1986. Recognition of handwritten characters—a review. *Image and Vision Computing* 4, 4, 208–218.

M. Elmezain, A. Al-hamadi, and B. Michaelis. 2009. A hidden Markov model-based isolated and meaningful hand gesture recognition. *International Journal of Electronical, Computer, and Systems Engineering* 3, 3.

N. Farrow, N. Sivagnanadasan, and N. Correll. 2014. Gesture based distributed user interaction system for a reconfigurable self-organizing smart wall. In *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction (TEI)*. ACM, New York, NY, 245–246.

J. Hawkins. 2002. Graffiti (Palm OS). https://en.wikipedia.org/wiki/Graffiti_(Palm_OS).

H. Hosseinmardi, N. Correll, and R. Han. 2012. Bloom filter-based ad hoc multicast communication in cyber-physical systems and computational materials. *Wireless Algorithms, Systems, and Applications* 595–606.

D. Hughes and N. Correll. 2014. A soft, amorphous skin that can sense and localize texture. In *IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, 1844–1851. Retrieved July 28, 2015 from http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6907101.

H. K. Lee and J. H. Kim. 1999. An HMM-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21, 10, 961–973.

J. Lifton, D. Seetharam, M. Broxton, and J. Paradiso. 2002. Pushpin computing system overview: A platform for distributed, embedded, ubiquitous sensor networks. *Pervasive Computing* 139–151.

V. J. Lumelsky, M. S. Shur, and S. Wagner. 2001. Sensitive skin. *IEEE Sensors Journal* 1, 1, 41–51.

S. Ma, H. Hosseinmardi, N. Farrow, R. Han, and N. Correll. 2012. Establishing multi-cast groups in computational robotic materials. In *IEEE International Conference on Cyber, Physical and Social Computing*. Besancon, France.

M. A. McEvoy and N. Correll. 2014. Thermoplastic variable stiffness composites with embedded, networked sensing, actuation, and control. *Journal of Composite Materials*. Retrieved July 28, 2015 from http://jcm.sagepub.com/content/early/2014/03/07/0021998314525982.

M. A. McEvoy and N. Correll. 2015. Materials that couple sensing, actuation, computation and communication. *Science* 347, 6228. Retrieved July 28, 2015 from http://www.sciencemag.org/lookup/doi/10.1126/science.1261689.

M. A. McEvoy, N. Farrow, and N. Correll. 2013. Robotic materials with controllable stiffness. In *Proceedings of the 19th International Conference on Composite Materials (ICCM)*.

S. Mitra and T. Acharya. 2007. Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 37, 3, 311–324.

K. Murakami and H. Taguchi. 1991. Gesture recognition using recurrent neural networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Reaching through Technology*. ACM, 237–242.

J. Nagi, H. Ngo, A. Giusti, Luca M. Gambardella, J. Schmidhuber, and G. A. Di Caro. 2012. Incremental learning using partial feedback for gesture-based human-swarm interaction. In *RO-MAN*. IEEE, 898–905.

Z. Nakad, M. Jones, and T. Martin. 2003. Communications in electronic textile systems. In *Proceedings of the 2003 International Conference on Communications in Computing*. 37–43.

O. F. Ozer, O. Ozun, C. O. Tuzel, V. Atalay, and A. E. Cetin. 2001. Vision-based single-stroke character recognition for wearable computing. *Intelligent Systems, IEEE* 16, 3, 33–37.

J. A. Paradiso, J. Lifton, and M. Broxton. 2004. Sensate media multimodal electronic skins as dense sensor networks. *BT Technology Journal* 22, 4, 32–44.

H. Profita, N. Farrow, and N. Correll. 2015. Flutter: An exploration of an assistive garment using distributed sensing, computation and actuation. In *Proceedings of the 9th International Conference on Tangible, Embedded, and Embodied Interaction (TEI'15)*. ACM, New York, NY, 359–362. DOI:http://dx.doi.org/10.1145/2677199.2680586

H. Sakoe and S. Chiba. 1990. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. In A. Waibel and K.-F. Lee (Eds.). *Readings in Speech Recognition*. Morgan Kaufmann Publishers Inc., San Francisco, CA. 159–165.

T. Schlömer, B. Poppinga, N. Henze, and S. Boll. 2008. Gesture recognition with a Wii controller. In *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*. ACM, New York, NY, 11–14.

A. D. Wilson and A. F. Bobick. 1999. Parametric hidden Markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 21, 9, 884–900.

Y. Wu and T. Huang. 1999. Vision-based gesture recognition: A review. *Gesture-based Communication in Human-Computer Interaction, Lecture Notes in Computer Science*, Vol. 1739, Springer, Berlin, 103–115.

Received December 2013; revised June 2014; accepted March 2015