Parallel Processing Letters
© World Scientific Publishing Company

DESIGNING A FUSION-DRIVEN SENSOR NETWORK TO SELECTIVELY TRACK MOBILE TARGETS *

SHASHI PHOHA[†]
Applied Research Laboratory, The Pennsylvania State University, 210 Applied Science Building, University Park, PA 16802, USA.

GOUTHAM MALLAPRAGADA
Department of Mechanical Engineering, The Pennsylvania State University, USA

YICHENG WEN
Department of Mechanical Engineering, The Pennsylvania State University, USA

DOINA BEIN
Applied Research Laboratory, The Pennsylvania State University, USA

ASOK RAY
Department of Mechanical Engineering, The Pennsylvania State University, USA

Received March 2011
Revised July 2011
Communicated by S. Sahni

ABSTRACT

Sensor networks that can support time-critical operations pose challenging problems for tracking events of interest. We propose an architecture for a sensor network that autonomously adapts in real-time to data fusion requirements so as not to miss events of interest and provides accurate real-time mobile target tracking. In the proposed architecture, the sensed data is processed in an abstract space called Information Space and the communication between nodes is modeled as an abstract space called Network Design Space. The two abstract spaces are connected through an interaction interface called InfoNet, that seamlessly translates the messages between the two. The proposed architecture is validated experimentally on a laboratory testbed for multiple scenarios.

Keywords: Data Fusion, Network Architecture, Sensor Fusion, Sensor Network, Target Tracking

*This material is based upon work supported by, or in part by, the U. S. Army Research Laboratory and the U. S. Army Research Office under the eSensIF MURI Award No. W911NF-07-1-0376. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsor.
[†]Corresponding author.

1. Introduction

A sensor network operates on an infrastructure of sensing, computation, and communication, through which it perceives the evolution of physical dynamic processes in its environment. With the advent of inexpensive, less reliable sensors, the need for more reliable performance from groups of these sensors is driven by applications that range from military surveillance to home security. In data gathering, data collected by individual sensors is aggregated at cluster heads, elected periodically, and sent to a base station [1, 2, 3]. In selective tracking, only specific mobile targets crossing a sensor field are tracked [4, 5], ignoring the rest. A tracking algorithm would be able to predict the target location and velocity even when some nodes fail, and in some cases, move sensors along the predicted trajectory to obtain better quality data [6]. Instead of involving all the nodes in the network in clustering, in this paper we allow only a subset of nodes that are geographically near to the mobile target to self-organize into few clusters in the vicinity of the target. Additionally, a sensor node would be able to distinguish between various types of data patterns and selectively track a specific target among many mobile objects.

The sensed data is highly correlated in the vicinity of a stimulus. Subsets of sensors may dynamically form *collaborative clusters* to estimate the target position or velocity for tracking purposes. In [7] the authors present a Bayesian approach to information-driven dynamic sensor collaboration to track mobile targets, that relies on reliable estimates of densities for belief states. Tracking mobile targets in sensor networks using a clustering approach is also discussed in [8]. In [9], a dynamic clustering algorithm based on Voronoi diagrams is presented for tracking targets in an acoustic sensor network. The authors of [10] assume that each sensor receives the same amount of information in the local region during a given time period. Their approach is too restrictive. Even within the region, sensors receive varied information corrupted with noise, based on their physical location and the distance from the observed target. The cluster head is selected to be the node at the center of the cluster, which is too rigid. The authors of [11] use a centralized approach to select a subset of sensor nodes that cover a region and form a connected communication graph, to answer a query within that region. *Dynamic clustering* [12] generates flexible network topologies for sensors to cluster and collaborate on detecting targets moving through a sensor field. Motivated by urban area applications that require adaptive sensor networks to dynamically cluster sensing, processing, and communication resources, a fusion-driven concept of the sensor network design is proposed in [13]. We present the details of the implemented design concept outlined in [13] and we add another interface. Through a proof-of-concept system implementation and proof of performance we show that our implementation provides a tunable level of tracking quality based on the reliability of the sensor network.

Section 2 details the proposed architecture. Section 3 presents the experimental results of tracking a Segway RMP robot moving in a pressure sensor field. The paper is concluded in Section 4.

2. Network Architecture

To enhance the quality and the resilience of data fusion, a distributed sensor network needs to self-adapt, namely the network topology must be able to change in real time, based on the spatial-temporal information derived from the ensemble of the sensor data. To achieve this self-adaption, we divide the sensor network into two interactive subspaces, Information Space and Network Design Space, with well defined interactions between them (InfoNet Interface). Fig. 1 shows the proposed architecture of a sensor network designed for mobile target tracking. This architecture allows us to build a sensor network that is highly scalable and robust, while providing the ability to track targets selectively and accurately.

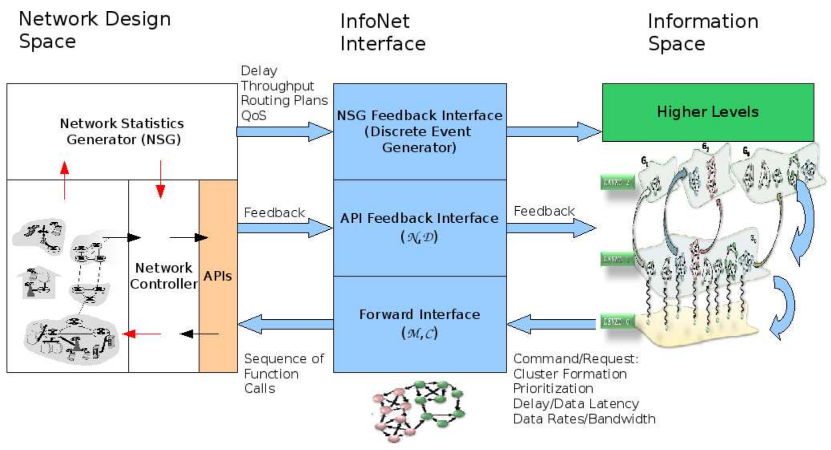


Fig. 1. Detailed architecture of the design space

Information Space of the sensor network is used to derive the spatial-temporal statistics of the ensemble of the sensor data. Data fusion is posed as a multi-time-scale problem under the assumption of quasi-stationarity over the fast-time scale (i.e., stationary over a sufficiently long time period) and possible non-stationarity caused by small deviations in the system behavior due to accumulating changes in the slow-time scale. Each sensor addresses the issues of data compression and communication constraints by autonomously aggregating the data through symbolization and semantic construction of probabilistic finite state automata (PFSA) [14]. The constructed PFSA are compared against a library of pre-determined patterns of interest with an appropriate metric, thus allowing the sensor network to selectively track interesting targets among many candidate targets [13]. Nodes observing and deciding on the same PFSA from the library are clustered dynamically, if some additional network and physical requirements are met. This approach has the advantage of being robust to individual sensor failures.

In the **Network Design Space**, the sensory data is modeled as a discrete-event dynamic system of interacting probabilistic automata, where sensor nodes may change their internal states through interactions with other nodes or the environment. To achieve this structural stability we have adapted the Dynamic Space-Time clustering (DSTC) protocol [12] to minimize communication between sensor nodes and to control the overhead while achieving a scalable operation of the sensor network for accurate mobile target tracking. The Network Design Space is configured to adapt to the Information Space needs in a manner that preserves the statistical characteristics (predictability) of the ensemble of the original sensor data at each level of fusion. The adaptive parameters of the Network Design Space include the sensor positions, the resource assignments, and the network connectivity.

The **InfoNet Interface** consists of *Forward* and *Feedback* interfaces. The Forward Interface seamlessly translates Information Space requirements into commands and allows the Network Design Space to act as an actuator to best meet the requirements. The Feedback Interface reports the result of the command back to Information Space.

2.1. *Information Space*

We define a *sensor field* $\mathscr{S}$ to be the set of all sensors under consideration of the same sensing modality. Typically, $\textsc{Card}(\mathscr{S})$ is large ($> 10^3$). We define a *pattern* G to be a probabilistic finite state automaton (PFSA) over an *a priori* fixed alphabet. A PFSA can be constructed using either D-Markov [14] or CSSR [15] algorithms. We define a function $\mathscr{H}$ that maps each sensor $s \in \mathscr{S}$ to the PFSA constructed at some slow-time epoch $t \geq 0$ from the sensor's data stream, which is a continuous-domain stream of quasi-stationary time series data.

Definition 2.1. At some slow-time epoch $t \geq 0$, $\mathscr{H} : \mathscr{S} \to \mathscr{P}$ maps each sensor $s \in \mathscr{S}$ to a PFSA $P \in \mathscr{P}$, where $\mathscr{P}$ is the space of all possible PFSA over the chosen symbol alphabet.

We assume that *the sensed data stream is independent from sensor to sensor.*

In our research we have considered the slow-time epochs to be the moments of time when the PFSA construction has ended successfully in the Information Space module, thus they depend on the sensing capabilities (e.g. sampling rates) and communication capabilities of the sensors in the cluster, which in turn impose restrictions on the range of the target speed. Of interest for future work is to make these epochs also dependent on the expected target speed and make the outcome of the PFSA construction algorithms dependent on the expected target speed.

Among all identified patterns we select a set $\mathbb{G}$ of K *patterns of interest*, $\mathbb{G} = \big\{G^i : i = 1, \cdots, K\big\}$, $K \in \mathbb{N}$. We assume that $\textsc{Card}(\mathbb{G}) = K$ is small (~ 10). The set of patterns of interest $\mathbb{G}$ is totally ordered via the pattern characteristic function $\chi_{\mathbb{G}} : \mathbb{G} \longrightarrow [0, 1]$. A sensor data is mapped into some pattern of interest G if and only if the PFSA H, constructed from the data stream, is close to G in the sense of a given metric θ constructed on the set of PFSA, $\theta : \mathscr{P} \times \mathscr{P} \to [0, \infty)$. The function θ measures the deterioration of the signal from its origination to the location of the sensor that senses the pattern generated by the target (see [16]). The physical distance between the sensor and the target identified as a pattern of interest G is a monotonically increasing function of the value $\theta(G,H)$, $\mathcal{D}:[0,\infty)\to[0,\infty)$. If d is the distance of the sensor s from the target location, G is the observed pattern of interest at its origination point, and $H_t(s)$ is the PFSA constructed at the slow-time epoch $t\geq 0$ from the data stream of the sensor s then $d=\mathcal{D}(\theta(G,H_t(s)))$.

We define $P[s,G]$ to be the probability that the sensor s has observed the pattern of interest G. This probability measure depends on the physical distance between the sensor and the target and it monotonically decreases as the distance between the two increases. For example, one way of defining it is $P[s,G]=e^{-\mathcal{D}(\theta(H_t(s),G))}$.

Next we present a probabilistic technique to estimate the position of a moving target in the sensor field using Bayesian techniques based on clustering nodes that exhibit the same pattern of interest. Given a pattern of interest G^k, let ϵ_i be the detection region of a sensor i, beyond which it cannot detect the pattern G^k anymore. The probability $P[i,G^k;\epsilon_i]$ that the sensor i has detected the pattern G^k within its detection region ϵ_i is

$$P[i,G^k;\epsilon_i]=\begin{cases}P[i,G^k] \text{ if } \mathcal{D}(\theta(H(i),G^k))\leq\epsilon_i\\ \\ 0 \qquad\quad \text{otherwise}\end{cases} \tag{1}$$

The probability $P[i,G^k;\epsilon_i]$ is assumed to be independent from sensor to sensor.

A sensor has observed a pattern when the target has crossed its region of detection. At some slow-time epoch $t\geq 0$, we group the nodes that have observed the pattern G^k and are within 1-hop communication range to a specific node among them (called cluster head) into a cluster $C_t^\ell(G^k)$ that is the ℓ^{th} cluster formed to track G^k. We define the *detection region* S of the cluster $C_t^\ell(G^k)$ to be the union of the detection region of the nodes in the cluster $S=\bigcup\limits_{i\in C_t^\ell(G^k)}\epsilon_i$. Beyond S a target cannot be detected at the slow-time scale t by any of the nodes in the underlying cluster. In Fig. 2 we show an example of a region S formed by a 5-node cluster containing only the dark-colored nodes.

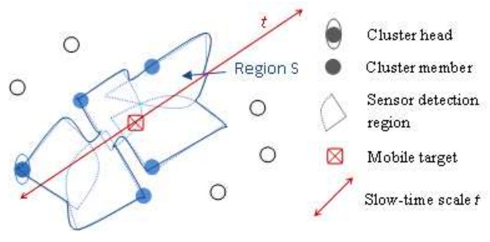


Fig. 2. Region S of a 5-node cluster

Definition 2.2. The probability that the pattern G^k is detected within the region $S = \bigcup_{i \in C_t^\ell(G^k)} \epsilon_i$ at the slow-time scale t is the probability that the target has crossed the detection region of every sensor in the cluster

$$P[G^k; S] = P[G^k; \bigcup_{i \in C_t^\ell(G^k)} \epsilon_i] = \prod_{i \in C_t^l(G^k)} P[i, G^k; \epsilon_i] \tag{2}$$

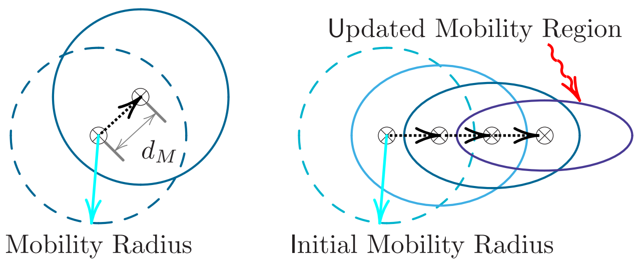
The estimated target location depends on the sensor measurement of G^k and the physical location of each sensor within the cluster $C_t^\ell(G^k)$.

Definition 2.3. The estimated physical location of a sensed pattern of interest G^k at time t, estimated by the cluster $C_t^\ell(G^k)$, is

$$\rho^\star_{k, C_t^\ell(G^k)}(t) = \sum_{i \in C_t^\ell(G^k)} \Gamma(i) \cdot P[i, G^k; \epsilon_i] \tag{3}$$

where $\Gamma(i)$ is the physical location of the sensor i.

Physical tracks of the pattern(s) generated by a moving target are computed as follows. For each pattern $G^k \in \mathbb{G}$, we have a mobility radius r_M based on the expected physical velocity of the target. Hence, the detection and the estimated location of G^k (Definition 2.3) at the current slow-time epoch t, sets up a region in the physical space within which we expect to find the pattern in the next slow-time epoch $t + \Delta T$. The velocity can then be estimated as shown in Fig. 3(a).



(a) (b)

Fig. 3. Track generation

Formally, the estimated velocity $\widehat{\vartheta}_{G^k}$ of the pattern G^k with respect to the cluster $C_t^\ell(G^k)$ in the physical space is given by:

$$\widehat{\vartheta}_{G^k} = \begin{cases} \frac{1}{\Delta T} d_M & \text{, if } |d_M| \leqq r_M \\ \text{Undefined} & \text{, otherwise} \end{cases} \quad \text{where } d_M = \frac{\rho^\star_{k,C^\ell(G^k)}(t + \Delta T) - \rho^\star_{k,C^\ell(G^k)}(t)}{\Delta T} \tag{4}$$

The information gathered over several slow-time epochs can be used to refine the estimated future position of the target in the physical space and to compute the estimated velocity of the target that subsequently updates the mobility regions, as shown in Fig. 3(b). The update algorithms can be based on simple Bayesian techniques or may involve more advanced fusion criteria.

After the command carried by the message from the Information Space has been executed in the Network Design Space, the Network Design Space sends the feedback to the Information Space regarding the outcome of the execution.

2.2. *Network Design Space*

Reliable estimation of position and velocity of a moving target requires clustering nodes for information fusion. On the other hand, forming clusters increases the communication load of the network. The DSTC algorithm [12] is efficient and reliable in clustering nodes that observe the same pattern of interest such that in any cluster, only one node, called the *cluster head*, is responsible for coordinating the communication among all other *cluster members*. DSTC protocol assumes that nodes have unique node IDs and defines five basic operations on a cluster: creating a cluster, disbanding a cluster, adding a node to a cluster, removing a node from a cluster, and moving the data from the cluster head to another cluster member. We add a new operation, called pre-cluster formation, that groups all the sensors observing the same pattern. The pre-cluster becomes a cluster after a cluster head is selected.

As the target moves through the sensor field, a sensor generates a stream of data and then looks for patterns in it. The first node that recognizes a pattern requests the formation of a pre-cluster. This node becomes the temporary pre-cluster head, creates a unique cluster ID, and broadcasts to its immediate neighbors a message *join_cluster* containing the pattern ID and the corresponding metric value that measures the node's suitability as a cluster head. All nodes examine the *join_cluster* messages that arrive within a time frame and may decide to join some pre-cluster by sending a message *hello*. Alternatively, a node may simply choose not to reply if it does not see any pattern of interest. Eventually, the pre-cluster head collects the state of each neighboring node that does reply and chooses the most suitable one as the cluster head. The DSTC algorithm checks periodically whether there is a cluster member with a better metric (closer to the node in terms of physical distance) for the observed pattern than the cluster head. In such case, that node becomes the new cluster head. This ensures the robustness of the cluster in case the cluster head fails or has moved significantly away from the target. In general, we can have more than one current cluster in the space-time vicinity of an event.

Once a cluster has been formed, cluster members will periodically send their sensed pattern and the associated metric (the semantic distance between the computed PFSA and the observed pattern of interest) to the corresponding cluster head so that data fusion can be performed; the time period depends on how long it takes for a sensor node to construct the PFSA. These messages are necessary for continuous pattern identification, since a cluster will contain only nodes that identify the same pattern. We define two types of messages, *position_estimate* and *position_result*. When a cluster member detects an event, it sends a message *position_estimate* to the cluster head. After the data fusion is done, the cluster head broadcasts a message *position_result* to the cluster members, so that every cluster member receives the fused data.

2.3. *InfoNet Interface*

Data fusion is done in the Information Space using mathematically rigorous tools such as probabilistic finite state automata and non-linear symbolic dynamics filtering. On the other hand, the Network Design Space is a stack of standard protocols and function calls. The Application Programming Interface (API) is the way to communicate with the Network Design Space.

Definition 2.4. A (Network Design Space) API can be represented by a 5-tuple $A = (\mathscr{F}, \mathscr{I}, \mathscr{O}, In, Out)$ where $\mathscr{F}$ is the set of all available function calls in the API, $\mathscr{I}$ is the set of input variables for all function calls, $\mathscr{O}$ is the set of output variables for all function calls, $In : \mathscr{F} \longrightarrow 2^{\mathscr{I}}$ represents the set of inputs for a function, and $Out : \mathscr{F} \longrightarrow 2^{\mathscr{O}}$ represents the set of outputs for a function. An output variable for function call is a four-tuple $(k, Flag, f, N_p)$ where

- k denotes the type of the API
- $Flag = \{0, 1\}$. Value 0 means that the network controller has successfully carried out the command and Value 1 means that the network controller could not execute the command.
- $f \in \mathscr{J}$ where $\mathscr{J}$ is a set of feedback function calls available in the API Feedback interface. $\mathscr{J}$ and $\mathscr{F}$ are closely related in the sense that the Interface calls some function in $\mathscr{J}$ to drive the network like a controller driving an actuator and the network calls some function in $\mathscr{F}$ to reply back to the API Feedback Interface about what the network controller has actually accomplished.
- N_p is a set of input arguments of a function $f \in \mathscr{J}$. In other words, N_p contains the parameters that the network controller will pass back to the API Feedback Interface.

We define next Forward Interface, which converts the requirements of Information Space into commands can be executed by Network Design Space, and Feedback Interface, which gives back the results of these commands (see Fig. 1).

2.3.1. *Forward Interface*

The Forward Interface takes a command from the Information Space and translates it into a language understood by the Network Design Space. APIs enable the network controller to exchange messages with the Forward Interface to achieve the communication with the Information Space. The higher order API is provided to keep consistency when dealing with the two different design spaces (Network and

Information Spaces) and to provide a coherent framework for expandability in case future functions are to be added. Also, such a formal structure allows symbolic algorithms to monitor the statistics of the network at different levels (network packet, API function call, etc.) in order to reconfigure the network in response to stimuli.

Definition 2.5. The Forward Interface is modeled as $(\mathscr{M}, \mathscr{C})$ where $\mathscr{M}$ is a collection of finite state automata and $\mathscr{C}$ is a library of input-output mappings. Each $M^k \in \mathscr{M}$ is a Mealy machine with $M^k = (Q^k, \Sigma_I^k, \Sigma_O^k, \delta^k, F^k)$, where k is the index of the machine because each machine is specifically designed for one API of the Network Design Space, Q^k is the set of the states of M^k, Σ_I^k is the alphabet set of inputs of M^k, Σ_O^k is the alphabet set of outputs of M^k such that there is a bijective mapping $\kappa : \Sigma_O^k \longrightarrow \mathscr{F}$ where $\mathscr{F}$ are the function calls in the k-th API, δ^k is a mapping $\delta : Q^k \times \Sigma_I^k \longrightarrow Q^k$, and F^k is a mapping $F^k : Q^k \times \Sigma_I^k \longrightarrow \Sigma_O^k$.

Together with each M^k there is a collection of mappings $\mathscr{C}^k \subseteq \mathscr{C}$ defined as $\mathscr{C}^k = \{(I_p, \mathscr{I}, \varphi)\}$ where I_p is the set of output parameters from the Information Space and also the input of the Forward Interface, $\mathscr{I}$ is the set of input parameters to Network Design Space and also the output of the Forward Interface, and $\varphi \in \mathscr{C}^k$ is some mapping $\varphi : I_p \longrightarrow \mathscr{I}$

Each node in the network can implement a subset of all the APIs available depending upon its role in the network. For instance, a sensor node in the network can implement cluster API where as a router can implement a routing API in addition to the cluster API.

It follows from Definition 2.5 that the inputs from the Information Space can be classified into two classes. One class $L_1 = \{s : s \in \Sigma_I^k\}$ is an input of a specific finite state machine $M^k \in \mathscr{M}$ and becomes a sequence of function calls $\{w : w \in \Sigma_O^k\}$ in the Network Design Space. The other class (L_2) will be handled by the mapping $\mathscr{C}$ and converted to the input arguments for the function calls. An illustrative example would be to assign the priority on a specific node. The priority value assigned to the node belongs to L_1 and the node ID in the Information Space is in L_2. $\mathscr{C}$ is actually composed of a library of mappings φ that can be shared among all the specific models of the Forward Interface. For instance, the message *createCluster* (see Section 3) is in L_1 class and its input arguments such as $NodeAddress$, $PatternID$ and $Metric\ value$ belong to L_2 class.

Given the inputs from the Information Space, the following generic conversion algorithm for the Forward Interface decomposes them into two classes, class L_1 and L_2, as follows. Assume that $p \in L_1$ and $q \in L_2$.

(1) Represent p as an input string $\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2 \ldots$.
(2) Feed the input string into the corresponding automata M_p.
(3) Obtain the output string $s_3s_1s_1s_1s_2 \ldots$ from M_p.
(4) Obtain the input arguments from $\mathscr{C}(q)$.
(5) According to the output string, execute the function calls with appropriate input arguments.

An example of the above automata formulation can be a simplified version of the state transition diagram for an individual node in the sensor field as shown in Fig. 4. A sensor node keeps collecting data and remains in the state *Alone* until the automaton built from the raw data matches a pattern in the pattern library. As stated at the beginning of Section 2.1, a pattern is any observable physical phenomena of interest captured in the form of a probabilistic finite state machine. In Section 3 we consider four types of (moving) patterns for a Segway RMP. Then a node goes to the state *Initial Election*, waits for messages from neighbors and attempts to elect a cluster head among the nodes that observe the same pattern. Based on the metric value, a pre-cluster head is elected and a pre-cluster thus is formed, which ultimately leads to a cluster after checking if some other constraints are satisfied. Finally, after a while, if the pattern is not observed anymore, the cluster head initiates the disbanding of the cluster. Then the nodes go back to state *Alone* again. The same process repeats over and over for each node in the network.

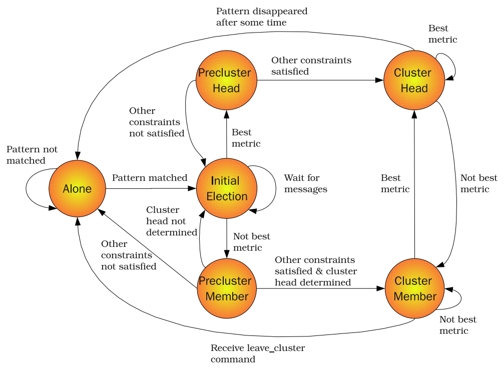

Pattern disappeared after some time
Best metric
Precluster Head
Other constraints satisfied
Cluster Head
Other constraints not satisfied
Best metric
Pattern not matched
Alone
Pattern matched
Initial Election
Wait for messages
Best metric
Not best metric
Cluster head not determined
Other constraints not satisfied
Not best metric
Other constraints satisfied & cluster head determined
Precluster Member
Cluster Member
Not best metric
Receive leave_cluster command

Fig. 4. State transition diagram for a sensor node

2.3.2. *Feedback Interface*

The Feedback Interface passes the outputs from the Network Design Space back to the Information Space. The outputs of the Network Design Space are of two types. One type consists of API outputs such as whether the API function was executed successfully, etc.. The type consists of network statistics that describe the communication quality; based on the network statistics, the Information Space will be able to differentiate whether a bad data fusion is due to inappropriate cluster formation or to a bad communication.

Definition 2.6. The finite state automaton $\mathscr{N}$ of the API Feedback Interface for APIs outputs is the Cartesian product of two automata $N^k \times N^f$ where $N^k = (Q^k, \Sigma^k, \delta^k)$, $Q^k = \{$ Cluster Control API, Prioritization API, Delay API, Data Rate

API }, $\Sigma^k = Q^k$, $\delta^k(q_i^k, s) = q_i^k$ for any $s \in Q^k$, and $N^f = (Q^f, \Sigma_I^f, \Sigma_O^f, \delta^f, F^f)$ where $Q^f = \mathscr{J}$ is the set of all available feedback function calls in the API Feedback Interface, $\Sigma_I^f = Q^f$, Σ_O^f is the alphabet of updating actions, $\delta^f(q_i^f, q_j^f) = q_j^f$ for all $q_i^f, q_j^f \in Q^f$, and $F^f : Q^f \times \Sigma_O^f \longrightarrow \Sigma_O^f$.

Definition 2.7. The API Feedback Interface is a tuple $(\mathscr{N}, \mathscr{D})$, where $\mathscr{N}$ is the automaton from Definition 2.6, and $\mathscr{D}$ is a library of input-output mappings of the form (N_p, I_p, ψ), where $\psi : N_p \longrightarrow I_p$.

$\mathscr{D}$ can be interpreted as the inverse mapping of $\mathscr{C}$, but $\mathscr{D}$ is not exactly $\mathscr{C}^{-1}$.

The outputs of APIs have been presented in Definition 2.4 as a tuple $\mathscr{O} = (k, Flag, f, N_p)$. $Flag$ is a single bit, informing the Information Space that the network is unable to fully complete the command that has been given. Let us not consider $Flag$ at this moment. f and N_p carry all the information about what the network has already done. From the Network Design Space one will obtain $\mathscr{O} = (k, Flag, f, N_p)$. The algorithm is the same as the generic algorithm of the Forward Interface except that here k and f belong to the class L_1, denoted by p, and N_p belongs to the class L_2, denoted by q.

(1) Represent p as an input string $\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2\ldots$.
(2) Feed the input string into the feedback automata $\mathscr{N}$.
(3) Obtain the output string $s_3s_1s_1s_1s_2\ldots$ from $\mathscr{N}$.
(4) Obtain the input arguments from $\mathscr{D}(q)$.
(5) According to the output string, execute the updating actions with appropriate input arguments.

3. Experimental Results

Extensive experiments were conducted to measure and validate the effectiveness of our proposed architecture for mobile target tracking, using the data collected from distributed pressure sensors connected through piezoelectric wires, forming a pressure sensitive floor. A coil of piezoelectric wire is placed under square floor tiles measuring 0.65m x 0.65m such that each sensor generates an analog voltage due to pressure applied on it. This voltage is sensed by a micro-controller using one of its 10-bit A/D channels thereby yielding sensor readings in the range of 0 to 1023. A total of 144 sensors are placed in a 9 x 16 equidistant grid to cover the entire laboratory. The data is then fed into a NS-2 program that simulates the wireless communication between nodes and executes the clustering, localization and tracking algorithms at nodes. Scalability tests were thus done by integrating limited hardware into large-scale simulations.

The objective is to detect and track spatio-temporal events of the behavior patterns of a Segway RMP, moving in different types of motion trajectories. Four types of motion trajectories were considered: 1) random, 2) circular, 3) sinusoidal, and 4) fast random.

In the *training* phase, a library of patterns (PFSA) is created by the following way:

- The raw time-series data is collected from each pressure sensor during a fixed time interval.
- A PFSA is built based on the data using the D-Markov machine construction algorithm [14]. A D-Markov machine is a type of probabilistic finite state automaton (used to represent patterns) that has Dth order Markov properties. For details of definition and the construction algorithm please refer to [14]. The construction algorithm allows one to represent the underlying symbolic process to a desired level of accuracy as a Dth order Markov chain, where D is a positive integer. The choice of both the alphabet size and the depth D affect how accurately the D-Markov machine represents the underlying physical phenomenon which in turn reflects the tracking accuracy. In general, higher alphabet size and higher depth D models the observed data more closely at the expense of increase in model complexity.
- The PFSA is added to the library by ensuring that it satisfies the fundamental equation of sensor network operation (see [17]) which ensures that the patterns are not close to each other in terms of a given metric θ in order to avoid ambiguity in deciding on a pattern given a PFSA in the presence of noise of a certain level.

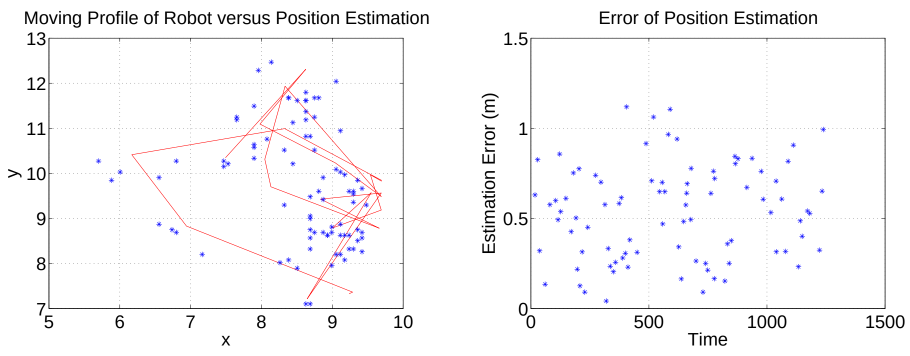
In the *operational* phase, the spatio-temporal information of the Segway's movement is fused by clustering the sensors along the estimated path of the Segway. The sensed data is used to build PFSA locally at each node using D-Markov machines, and the constructed PFSA is compared against the pattern library.

To simulate the network related aspects of the experiments we used the NS-2. The data collected by each sensor during the experiment is fed into a corresponding simulated node in the NS-2 environment to create a detailed simulation of the distributed sensor network. The algorithms associated with the Information Space, the Network Controller and the InfoNet Interface are implemented at each node. For the Information Space algorithms, a library of patterns (PFSA) is created off-line and stored at every node. During the online operation of the sensor network, each node collects the raw data from its pressure sensor. After collecting a fair amount of data, a probabilistic finite state machine is built using the construction algorithm for a D-Markov machine [14]. The PFSA constructed in this manner is then matched with the existing set of patterns in the pattern library. We say that the sensor has detected a pattern if it can decide on a (single) pattern from that library. The semantic sensing radius is chosen appropriately using the metric θ defined on the set of PFSA ([16]). [a] Next, a pre-cluster is formed among the neighboring sensors

[a]The semantic sensing radius specifies how far apart a computed PFSA could be from some pattern in the pattern library in order for the sensor to decide that it had detected the pattern from the library.

that observe the same pattern. This pre-cluster eventually becomes a cluster, using the DSTC algorithm defined at the Network Controller level of each node of the pre-cluster, and a cluster head is elected.

The newly formed cluster has a lifetime, at the end of which the cluster is automatically disbanded. The timeout starts as soon as the cluster head stops observing the pattern for which the cluster was formed. While in the cluster, the cluster head fuses the data from its members to estimate the current position of the target. Fig. 5(a) shows a run of the tracking algorithm. The actual trajectory of the Segway robot is shown as a line (red) and the estimated positions are shown by dots (blue). The corresponding error in position estimation over time for every slow-time epoch is shown in Fig. 5(b).

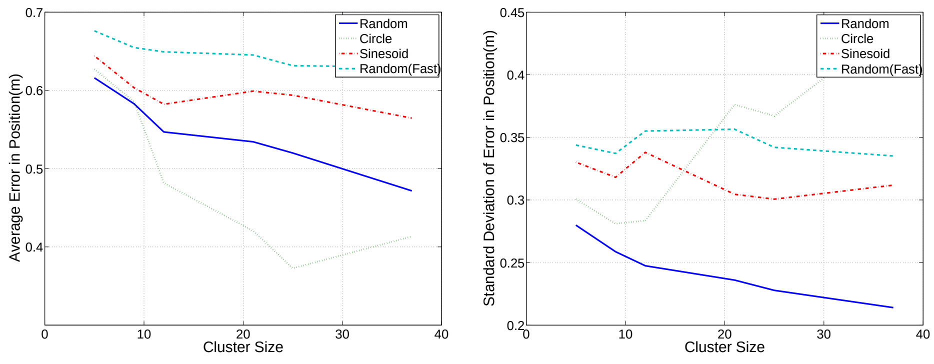


(a) Robot trajectory vs. estimated position (b) Error in position estimation

Fig. 5. Robot trajectory versus estimated position

The fusion of data from more sensors leads to better localization of the target. Fig. 6(a) illustrates the average error in the position estimation versus the cluster size. Overall, the random fast trajectories have the largest average error, followed by the sinusoidal ones, random, and the circle. For smaller size clusters (less than 8 nodes), the average error is relatively the same for all four trajectories, with the random trajectory having the smallest value. As the maximum size of the cluster increases, the difference between trajectories increases, and the circle trajectories have the smallest error.

Fig. 6(b) illustrates the standard deviation in the position estimation as a function of cluster size; it indicates that there is an optimal value of cluster size after which the standard deviation of the error increases. The rationale for this is: as the cluster size increases beyond the optimal value, in calculating the position estimation there will be an increased contribution from the nodes that do not observe the event accurately. Overall, the random trajectory has the smallest value. The random fast trajectories have smaller values than the random and sinusoidal ones. The circle trajectories have a mixed behavior. For a smaller cluster size (less than 15 nodes), the circle trajectories have the second smallest values. As the maximum



(*a*) Average error (*b*) Standard deviation

Fig. 6. Target estimation error

cluster size increases, the standard deviation error increases and the circle trajectories have increasingly larger values, so that for a cluster size larger than 20, the circle trajectories have the largest value among all four trajectories.

4. Conclusion and Future Work

We presented the design of a sensor network for tracking mobile targets in which the design space of the sensor network is decomposed into Information Space and Network Design Space with the InfoNet Interface between the two. Since the sensor modality does not play a key role in the current design, the current architecture can be extended to heterogeneous sensors.

A semantic approach based on probabilistic finite automata is employed in the Information Space to fuse the data at various levels of hierarchy. Data is compressed at the lower level at each sensor, which reduces the network communication overhead. Data fusion is done at each cluster head, by taking the data from the cluster members; the failure of a few cluster member nodes affects only slightly the result of fusion. The advantage of this semantic approach is that the sensor network is not only able to detect the presence of a target, but also to specify some motion trajectories of the mobile target.

Future work will include two major enhancements. The first enhancement will involve optimizing the pattern library in terms of library size and pattern resolution, based on the semantic sensing radius. The semantic sensing radius measures how far a measured pattern can vary from a pattern of interest and has a significant effect on the network behavior, since it trades the unambiguous classification of the patterns and the network performance measured in terms of tracking the target accurately, with low communication overhead.

The second enhancement will involve adding a network statistics generator to carry out statistical analysis of the network behavior under various control stimuli given by the Information Space.

References

[1] Smaragdakis, I. Matta, and A. Bestavros. Sep: A stable election protocol for clustered heterogeneous wireless sensor networks. In *Proceedings of the 2nd International Workshop on Sensor and Actor Network Protocols and Applications (SANPA)*, pages 1–11, 2004.

[2] V. Mhatre and C. Rosenberg. Design guideliness for wireless sensor networks: communications, clustering and aggregation. *Ad Hoc Network Journal*, 2(1):45–63, 2004.

[3] M. Ye, C. Li, G. Chen, and J. Wu. Eecs: an energy efficient cluster scheme in wireless sensor networks. In *Proceedings of IEEE International Workshop on Strategies for Energy Efficiency in Ad Hoc and Sensor Networks (IWSEEASN)*, pages 535–540, 2005.

[4] P. Biswas and S. Phoha. Self-organizing sensor networks for integrated target surveillance. *IEEE Transactions on Computers*, 55(8):1033–1047, August 2006.

[5] M. Lotfinezhad, B. Liang, and E.S. Sousa. Adaptive cluster-based data collection in sensor networks with direct sink access. *IEEE Transactions on Mobile Computing*, 7(7):884–897, July 2008.

[6] Y. Zou and K. Chakrabarty. Distributed mobility management for target tracking in mobile sensor networks. *IEEE Transactions on Mobile Computing*, 6:872–887, August 2007.

[7] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 19(2):61–72, March 2002.

[8] H. Yang and B. Sikdar. A protocol for tracking mobile targets using sensor networks. *Proceedings of IEEE Workshop on Sensor Network Protocols and Applications, Anchorage, Alaska, USA*, May 2003.

[9] W.-P. Chen, J.C. Hou, and L. Sha. Dynamic clustering for acoustic target tracking in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 3(3):258–271, July 2004.

[10] H. Chen and S. Megerian. Cluster sizing and head selection for efficient data aggregation and routing in sensor networks. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, volume 4, pages 2318–2323, 3-6 April 2006.

[11] H. Gupta, Z. Zhou, S.R. Das, and Q. Gu. Connected sensor cover: self-organization of sensor networks for efficient query execution. *IEEE/ACM Transactions on Networking*, 14(1):55–67, 2006.

[12] S. Phoha, J. Koch, E. Grele, C. Griffin, and B. Madan. Space-time coordinated distributed sensing algorithms for resource efficient narrowband target localization and tracking. *International Journal of Distributed Sensor Networks*, 1:81–99, 2005.

[13] S. Phoha and A. Ray. Dynamic information fusion driven design of urban sensor networks. *IEEE International Conference on Networking, Sensing and Control*, pages 1–6, 2007.

[14] A. Ray. Symbolic dynamic analysis of complex systems for anomaly detection. *Signal Processing*, 84(7):1115–1130, 2004.

[15] C.R. Shalizi, K.L. Shalizi, and J.P. Crutchfield. An algorithm for pattern discovery in time series. *Technical Report, Santa Fe Institute*, October 2002.

[16] I. Chattopadhyay and A. Ray. Structural transformations of probabilistic finite state machines. *International Journal of Control*, 81(5):820–835, May 2008.

[17] S. Phoha, A. Ray, I. Chattopadhyay, G. Mallapragada, and Y. Wen. Mathematical modeling of sensor network dynamics for control and stability. Technical Report TR eSensIF-08-01, Applied Research Laboratory, The Pennsylvania State University, September 2008.