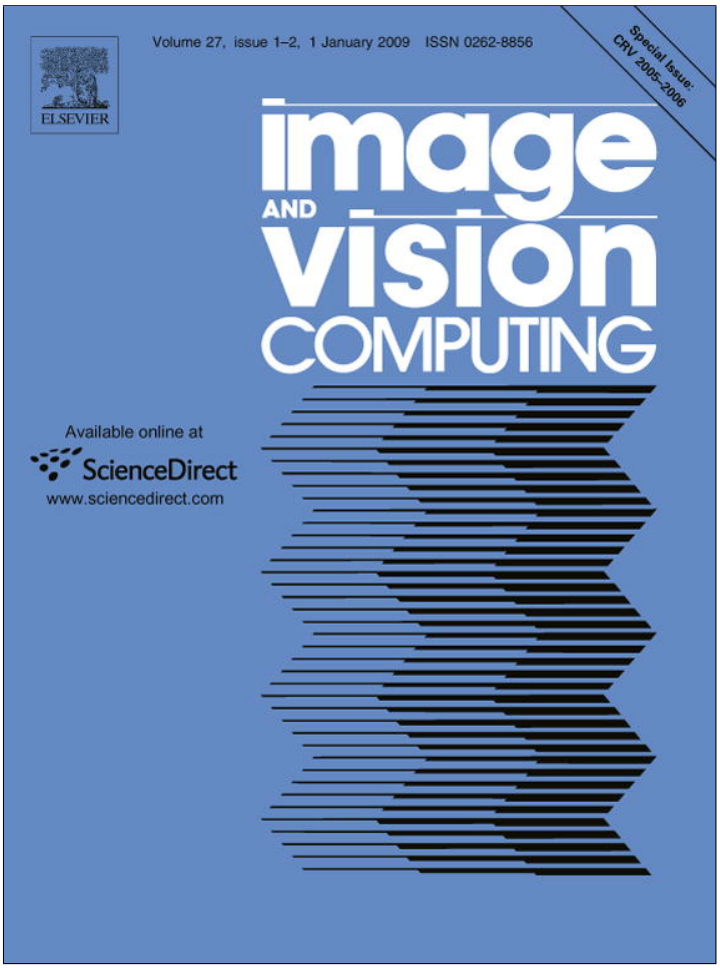

Available online at www.sciencedirect.com
ScienceDirect
Image and Vision Computing 27 (2009) 116–130
www.elsevier.com/locate/imavis

Self-calibration of a vision-based sensor network

Dimitri Marinakis *, Gregory Dudek

Centre for Intelligent Machines, McGill University 3480 University St, Montreal, Quebec, Canada H3A 2A7

Received 23 January 2006; accepted 21 June 2006

Abstract

When a network of vision-based sensors is emplaced in an environment for applications such as surveillance or monitoring the spatial relationships between the sensing units must be inferred or computed for self-calibration purposes. In this paper we describe a technique to solve one aspect of this self-calibration problem: automatically determining the topology and connectivity information of a network of cameras based on a statistical analysis of observed motion in the environment. While the technique can use labels from reliable cameras systems, the algorithm is powerful enough to function using ambiguous tracking data. The method requires no prior knowledge of the relative locations of the cameras and operates under very weak environmental assumptions. Our approach stochastically samples plausible agent trajectories based on a delay model that allows for transitions to and from sources and sinks in the environment. The technique demonstrates considerable robustness both to sensor error and non-trivial patterns of agent motion. The output of the method is a Markov model describing the behavior of agents in the system and the underlying traffic patterns. The concept is demonstrated with simulation data for systems containing up to 10 agents and verified with experiments conducted on a six camera sensor network.

Keywords: Sensor networks; Perception; Self-calibration; Topology; Learning; Markov chain monte carlo; Expectation maximization

1. Introduction

In this paper, we propose and solve the non-standard problem of inferring the relative positions of a set of sensors that all look at the same scene, yet which have completely non-overlapping fields of view. This is in contrast to the somewhat more traditional problem of inferring the structure of a scene or tracking activities using a network of sensors whose positions are known. Inferring the position of well-separated cameras can be viewed as the precursor to this traditional problem. In our approach, we exploit the motion of objects between the fields of view of the different sensors to probabilistically infer their positions. That is, when an object leaves the neighborhood of one sensor and subsequently appears within range of another, this suggests that the two sensors are proximal. Our purpose is to construct a probabilistic model of the inter-sensor connectivity information in the environment, and from this data, reconstruct the topology of the network. By 'topology' we are referring to the physical inter-sensor connectivity from the point of view of an agent navigating the environment.

We will illustrate the problem with a simplified abstract. Fig. 1(a) depicts a sensor network distributed within an indoor environment. Let us assume that the network has been deployed for some purpose such as surveillance and requires knowledge of the inter-node connectivity in order to fulfill its function. During some initial calibration period the network collects observations of agents passing by each sensor (Fig. 1(b)). The problem we are trying to solve is how to use these collected observations to construct the topological description of the network shown in Fig. 1(c). This type of network might arise if wireless cameras were deployed in a workplace environment.

In our approach, we attempt to recover correspondences between cameras by exploiting motion present

* Corresponding author. Tel.: +1 250 746 6907.
E-mail addresses: dmarinak@cim.mcgill.ca (D. Marinakis), dudek@cim.mcgill.ca (G. Dudek).

0262-8856/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.imavis.2006.06.009

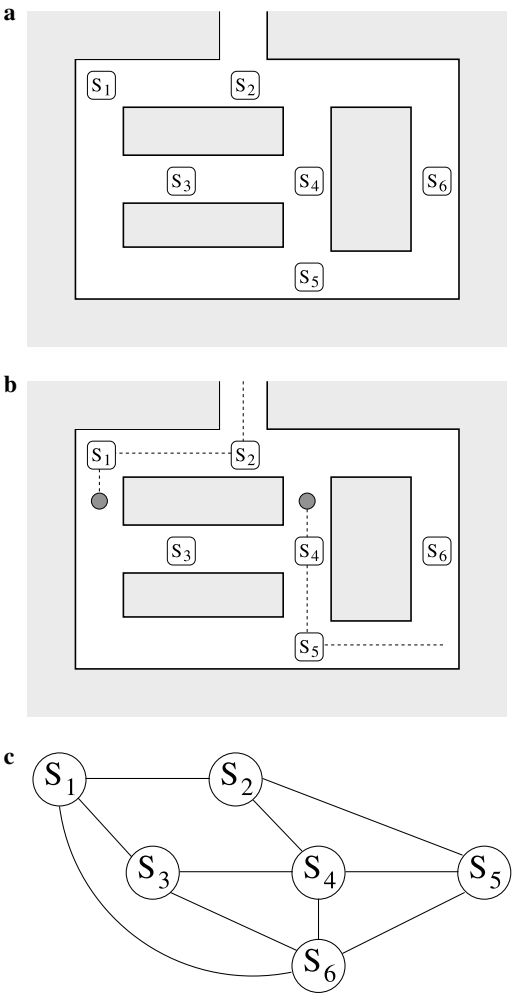

Fig. 1. An example of a sensor network which we wish to calibrate. (a) The original ad-hoc deployment. (b) An Example of agent motion exploited by the calibration process. (c) The desired topological connectivity map of the network.

in the environment. The method is based on reconstructing agent trajectories that best explain the observational data and using these trajectories to determine likely network parameters.

For now we restrict the problem by allowing observations collected at each of the nodes to indicate only the presence or absence of motion. In other words, we assume that the sensors are non-discriminating and are capable of reporting only that they have detected something, but are not capable of providing a description or signature of what they have detected. The data used by the algorithm consists only of time stamped events collected from each of the sensors. Note that the addition of more detailed sensor information can only make the problem easier and will be addressed in later work.

We demonstrate our approach experimentally with a six camera network. The system automatically calibrates itself based solely on motion detection. It is able to infer a connectivity graph of the environment and inter-vertex delay times both with a high degree of accuracy.

The ability of a surveillance or monitoring system to automatically determine the connectivity parameters describing its environment is useful for a number of reasons. Although the topology information can be manually entered during installation, more detailed parameters such as inter-camera delay distributions are difficult to determine, and a change in the environment or network would require re-calibration. Once calibrated, the connectivity information could aid in conventional target tracking and additional monitoring activities. For example, by reconstructing trajectories, a vehicle monitoring network distributed about a city could help make decisions about road improvements which might best alleviate congestion. In addition, the topological information could be combined with relative localization techniques [1,2] to recover a more complete representation of the environment.

2. Related works

A number of researchers have looked at the problem of self-configuring a multi-sensor network through the exploitation of motion in the environment [3–6]. These efforts generally assume vision-based sensors and focus on issues regarding the processing of observations collected from distributed sensors.

Stein [3], for example, considered recovering a rough planar alignment of the location and orientation of the individual cameras. Using a least-median-of-squares technique, he determined the correspondence between moving objects in pairs of cameras. His approach, however, required some overlap in the field of view of the cameras.

Also relying on overlapping fields of view between cameras placed at adjacent sensing locations, Stauffer and Tieu [5] described a method for building a tracking correspondence model based solely on observational data. Their work focused on determining which tracking data resulted from observations of the same objects in sensors with overlapping views. Their approach was based on probabilistically determining correspondences between cameras and ultimately using this information to calibrate a camera network to better track objects between fields of view. They verified their method with experiments conducted on a five camera network.

Fisher [4] explored a self-localization approach for networks of cameras without overlapping fields of view. This method exploited the the motion of distant moving objects such as stars. The objects were assumed to have well-behaved linear or parabolic trajectories, and it was necessary that the observed objects could be uniquely identified across separate cameras.

In a more recent effort, Rahimi et al. [6] described a simultaneous calibration and tracking algorithm that uses a velocity extrapolation technique to self-localize a network of non-overlapping cameras based on the motion of a single target. Their work avoided the difficult problem of associating observations with different targets by assuming only one source of motion.

Connectivity information or *network topology* can be recovered by exploiting motion in the environment. In contrast to this paper, current efforts either address a slightly different problem than the one we are interested in [7] or they employ considerably different methods [8,9].

In order to track multiple agents across disjoint fields of view, Javed et al. [7] first calibrated the connectivity information of their surveillance system using observational data. To learn the probability of correspondence (transition probabilities) and inter-camera travel times (delay distributions), they assumed a training period in which the data association between observations and agents was known. Given this observation ownership information, they employed a Parzen window technique that looks for correspondences in agent velocity, inter-camera travel time, and the location of agent exit and entry in the fields of view of the camera.

Ellis, Makris, and Black [8,9] presented a technique for topology recovery based on event detection only. In their approach, they first identified entrance and exit points in camera fields of view and then attempted to find correspondences between these entrances and exits based on video data. Their technique relies on exploiting temporal correlation in observations of agent movements. The method employs a threshold technique that looks for peaks in the temporal distribution of travel times between entrance–exit pairs; a clear peak suggesting that a correspondence exists. The technique gave promising results on experiments carried out on a six camera network. Although it requires a large number of observations, the method does not rely on object correlation across specific cameras. Thus, the approach can be used to efficiently produce an approximate network connectivity graph but when the network dynamics are complex or the traffic distribution exhibits substantial variation, it would appear the technique will have difficulty.

The work conducted by Ellis, Makris, and Black [8,9] on learning network topology is of particular interest to us since the problem they consider is very similar to the one we address in this paper. In a later section, we will show that our approach compares favorably to theirs.

Much of the work on network calibration through the exploitation of motion is motivated by or incorporated into research conducted on multi-target tracking. The work of Javed et al. [7] and that of Stauffer and Tieu [5] for example, is directly related to the development of multi-target tracking systems. Similarly, one of the stated goals of Ellis et al. [8,9] is to enhance the tracking performance of surveillance systems. Since our approach relies on recovering plausible trajectories of agent motion, we address some of the same problems faced in this area of research.

Multi-target tracking is a well established research area in sensor networks [10,11] and multi-robot systems [12]. One of the key difficulties faced is that of maintaining target identities during periods when two or more targets move close together or are unobserved for a period of time. Probabilistic techniques such as Identity Mass Flow [13] have been devised to handle this situation. Other work poses the target identity problem as a data association problem [14–16].

Pasula et al. [17] successfully approached a traffic monitoring problem from the data association perspective through a stochastic sampling technique, although only in simple networks. Given known sensor positions and topology, the goal of the work was to track multiple objects passing through the network and recover their long-range origin/destination information. An iterative expectation maximization algorithm was employed that assigned probable trajectories to each vehicle. These samples were then used to update model parameters such as link-travel time and vehicle characteristics. New trajectory samples were generated from existing samples by swapping vehicle assignments between pairs of adjacent sensors. A new sample was accepted based on its relative probability to the existing sample. The approach was verified using a freeway simulator that modeled one-hundred cars of different colors passing through a network of nine cameras. The algorithm remained robust when the color discrimination capability of the cameras was reduced, however, no results are presented for spurious or missing observational data.

Our method of generating trajectory samples is close in spirit to that used by Pasula et al. [17]. We approach the data association problem of linking observations to sources of motion through the use of Markov Chain Monte Carlo (MCMC). Our implementation differs, however, due to the specifics of the problem. While they inferred motion parameters given a known network, we address the opposite problem: inferring information about the network given motion in the environment.

In this paper, we present and verify a network topology inference method based on the construction of plausible agent trajectories. The technique employs a stochastic expectation maximization (EM) algorithm, an established statistical method for parameter estimation of incomplete data models [18,19] that has been applied to many fields including multi-target tracking [17] and mapping in robotics [20,21]. The algorithm is robust to observational noise and non-trivial agent motion through the use of a realistic delay model that allows motion to sources and sinks in the environment. We demonstrate the success of the approach both with simulations and with an experiment conducted on a six camera-based sensor network.

3. Problem description

We formalize the problem of topology inference in terms of the inference of a weighted directed graph which captures the connectivity relationships between the positions of the sensors' nodes. The motion of multiple agents moving asynchronously through a sensor network region can be modeled as a semi-Markov process. The network of sensors is described as a directed graph $G = (V, E)$, where the vertices $V = v_i$ represent the locations where sensors are deployed, and the edges $E = e_{i,j}$ represent the connectivity between them; an edge $e_{i,j}$ denotes a path from the position

of sensor v_i to the position of sensor v_j. The motion of each of the N agents in this graph can be described in terms of their transition probability across each of the edges $A_n = \{a_{ij}\}$, as well as a temporal distribution indicating the duration of each transition D_n. The observations $O = \{o_t\}$ are a list of events detected at arbitrary times from the various vertices of the graph, which indicate the likely presence of one of the N agents at that position at that time.

The goal of our work is to estimate the parameters describing this semi-Markov process. We assume that the agents' probabilistic behavior is homogeneous; *i.e.,* the motion of all agents are described by the same A and D. In addition, we must make some assumptions about the distribution of the inter-sensor (*i.e.,* inter-vertex) transition times. We make the assumption that the delays in moving between one sensor and another can be described by a windowed normal distribution. We will show later, however, that we can relax this assumption in some situations.

Given the observations O and the number of agents N, the problem is to estimate the network connectivity parameters A and D, subsequently referred to as θ.

4. Topology inference algorithm

In this section, we will describe our topology inference algorithm that takes non-discriminating observations and returns inferred network parameters. The technique assumes knowledge of the number of agents in the environment and attempts to augment the given observations with an additional data association that links each observation to an individual agent.

4.1. Monte carlo expectation maximization

We use the EM algorithm [18] to solve the connectivity problem by simultaneously converging toward both the correct observation data correspondences and the correct network parameters. We iterate over the following two steps:

1) *The E-Step:* which calculates the expected log likelihood of the complete data given the current parameter guess:

$$Q(\theta, \theta^{(i-1)}) = E[\log p(O, Z|\theta)|O, \theta^{(i-1)}]$$

where O is the vector of binary observations collected by each sensor, and Z represents the hidden variable that determines the data correspondence between the observations and agents moving throughout the system.

2) *The M-Step:* which then updates our current parameter guess with a value that maximizes the expected log likelihood:

$$\theta^{(i)} = \underset{\theta}{\operatorname{argmax}}\, Q(\theta, \theta^{(i-1)})$$

We employ MCEM [19] to calculate the E-Step because of the intractability of summing over the high dimensional data correspondences. We approximate $Q(\theta,\theta^{(i-1)})$ by drawing M samples of an ownership vector $L^{(m)} = \{l_i^m\}$ which uniquely assigns the agent i to the observation o_i in sample m:

$$\theta^{(i)} = \underset{\theta}{\operatorname{argmax}} \left[\frac{1}{M}\sum_{m=1}^{M} \log p(L^{(m)}, O|\theta)\right]$$

where $L^{(m)}$ is drawn using the previously estimated $\theta^{(i-1)}$ according to a Markov chain monte carlo sampling technique, explained in the next section.

In order to ensure an adequate burn-in time for the Markov chain, a number of initially drawn samples of the ownership vector are discarded. A simple heuristic is employed in which initial samples are discarded until their computed likelihood stops increasing at every step.

At every iteration we obtain K samples of the ownership vector L, which are then used to re-estimate the connectivity parameter θ (the M-Step). We continue to iterate over the E-Step and the M-Step until we obtain a final estimate of θ. The process terminates when subsequent iterations result in very small changes to θ. The following pseudo code outlines the algorithm:

```
WHILE (θ^i − θ^(i−1)) > Threshold
  Draw sample L until p(L,O|θ) levels off
  Draw K samples L^(k)
  Update θ^i given {L^(1) ... L^(K)}
END WHILE
```

In general, we make the assumption that the inter-vertex delays are normally distributed and determine the maximum likelihood mean and variance for each of the inter-vertex distributions along with transition likelihoods. In a subsequent section, we will describe how we occasionally reject outlying low likelihood delay data and omit it from the parameter update stage.

4.2. Markov chain monte carlo sampling

We use Markov chain monte carlo sampling to assign each of the observations to one of the agents, thereby breaking the multi-agent problem into multiple versions of a single-agent problem. In the single agent case, the observations O specify a single trajectory through the graph which can be used to obtain a maximum likelihood estimate for θ. Therefore, we look for a data association that breaks O into multiple single agent trajectories. We express this data association as an ownership vector L that assigns each of the observations to a particular agent.

Given some guess of the connectivity parameter θ, we can obtain a likely data association L using the Metropolis algorithm; an established method of MCMC sampling [22]. From our current state in the Markov chain specified by our current observation assignment L, we propose a symmetric transition to a new state by reassigning a randomly selected observation to a new agent selected uniformly at

random. This new data association L' is then accepted or rejected based on the following acceptance probability:

$$\alpha = \min\left(1, \frac{p(L', O|\theta)}{p(L, O|\theta)}\right)$$

However, the acceptance probability α can be expressed in a simple form since the trajectories described by L' differ from those in L by only a few edge transitions. Consider L as a collection of ordered non-intersecting sets containing the observations assigned to each agent $L = (T_1 \cup T_2 \cup \cdots \cup T_N)$, $T_n = \{w_{jk}\}$ where w_{jk} refers to the edge traversal between vertices j and k. The probability of a single agent trajectory is then the product of all of its edge transition probabilities:

$$p(T|\theta) = \prod_{w \in T} p(w|\theta)$$

Therefore, a proposed change that reassigns the observation o_n from agent y to agent x must remove an edge traversal w from T_y and add it to T_x. Only the change in the trajectories of these two agents need be considered, since all other transitions remain unchanged. In the example shown in Fig. 2:

$$\alpha = \min\left(1, \frac{p(T'_x, T'_y|\theta)}{p(T_x, T_y|\theta)}\right) = \min\left(1, \frac{p(w_{ac}, w_{ce}, w_{bd}|\theta)}{p(w_{ae}, w_{bc}, w_{cd}|\theta)}\right)$$

In between each complete sample of the ownership vector L, each of the observations are tested for a potential transition to an alternative agent assignment. This testing is accomplished in random order and should provide a large enough spacing between realizations of the Markov chain that we can assume some degree of independence in between samples. The resulting chain is ergodic and reversible and should thus produce samples representative of the true probability distribution.

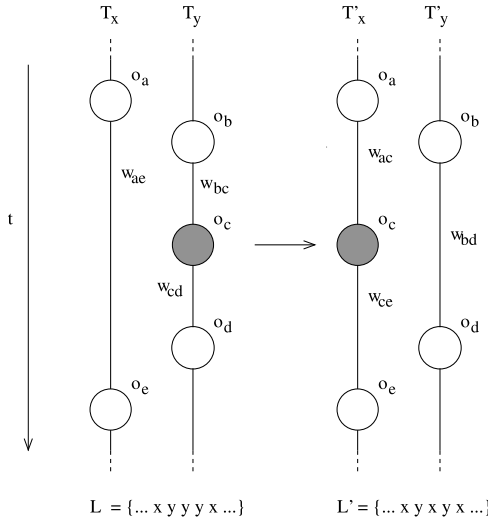

Fig. 2. An example of a proposed Markov chain transition. The ownership assigned to o_c has been shifted from agent y to agent x. To evaluate this transition, the probability of the edge traversals w_{ac}, w_{ce}, w_{bd} must be compared to the original traversals w_{ae}, w_{bc}, w_{cd}.

4.2.1. Proof of Markov chain ergodicity

Ergodicity and reversibility in a Markov chain are sufficient conditions to ensure that there exists a unique and *specified* stationary distribution [23]. In our case, reversibility is guaranteed through our use of the Metropolis algorithm [24]. Furthermore, if our chain is ergodic, then the detailed balance equation specifies the stationary distribution. In our case, each state of our chain represents an instance of the ownership vector. It has a stationary distribution specified by:

$$\pi_j = \lim_{t \to \infty} p_{ij}^t, \forall i, j \in Z \propto p(L^{(j)}|O, \theta)$$

where Z represents the set of all possible realizations of the ownership vector L and p_{ij}^t gives the probability of reaching $L^{(j)}$ from $L^{(i)}$ in t steps. It remains, however, to show that our Markov chain is ergodic.

For a finite state Markov chain, ergodicity is implied by an *irreducible* and *aperiodic* transition matrix P [24]. However, because we determine state transition probabilities based on the Metropolis algorithm, we are ensured that if the proposal matrix R is ergodic then so is P [24]. Therefore, we must show that our proposal matrix R is irreducible and aperiodic.

That our proposal matrix R is ergodic can be demonstrated by considering both a single step transition and a k step transition where k is the number of observations $|O|$. Our proposal matrix $R = \{r_{ij}\}$ gives the probability of a proposed transition from $L^{(i)}$ to $L^{(j)}$. For a single step, $r_{ij} > 0$ if $L^{(i)}$ and $L^{(j)}$ are *exactly the same* or differ by only one ownership assignment to a single observation. All other elements of the one step proposal matrix are zero. In the worst case, $L^{(i)}$ and $L^{(j)}$ can differ from each other by k ownerships assignments where $k = |O|$. In this case, the two states require k transitions to be communicable. Therefore $(r_{ij})^k > 0$, $\forall i, j \in Z$ as long as $k > |O|$. Since $(R)^1$ has non-zero diagonal elements, it is aperiodic and since there exists a finite k such that $(R)^k$ has all positive entries, it is irreducible.

4.3. Delay model

To make the algorithm more robust to realistic traffic patterns, we have introduced an inter-vertex delay model that allows for the possibility of agent transitions to and from sources and sinks. This makes the algorithm more robust both to shifting numbers of agents in the environment and to agents that pause or delay their motion in between sensors. Additionally, assuming the existence of sources and sinks, we can recover their connectivity to each of the sensors in our network.

In addition to maintaining a vertex that represents each sensor in our network, we introduce an additional vertex that represents the greater environment outside the monitored region: a *source/sink node*. A mixture model is employed during the E-Step of our iterative EM process which evaluates potential changes to agent

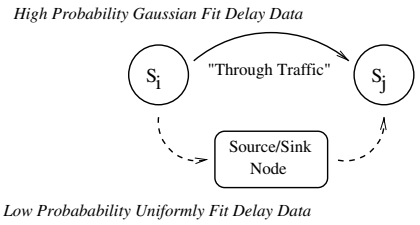

Fig. 3. Algorithm delay model.

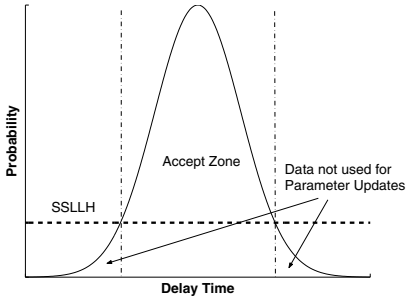

Fig. 4. Graphical description of the SSLLH Parameter.

trajectories. An inter-vertex delay time is assumed to arise from either a Gaussian distribution or from a uniform distribution of fixed likelihood (Fig. 3). The model allows for uniform but low probability jumps of almost arbitrary length.

The data assigned to the Gaussian distribution are assumed to be generated by ''through-traffic'' and are used during the M-Step to update our belief of the inter-node delay times and transition likelihoods. However, the data fit to the uniform distribution are believed to be transitions from the first vertex into the sink/source node and then from the sink/source node to the second vertex. Therefore they are not used for updating inter-vertex delay parameters of the two nodes, but rather are used only for updating the belief of transitions to and from the source/sink node for the associated vertices.

While the Gaussian assigned delays are expected to be within a realistic temporal range for direct inter-vertex agent motion, the delay data fit to the uniform distribution is more loosely bounded. This gives the inference technique a manner of temporarily removing agents from the system by assigning them to long transitions, or to explain events that would otherwise seem extremely unlikely such as the disappearance of an agent from one node and its almost immediate appearance at a second.

The delay model provides robustness to noise by discarding outliers in the delay data assigned to each pair of vertices and explaining their existence as transitions to and from a source/sink node. The key to this process is determining whether or not a delay value should be considered an outlier. This is implemented through a tunable parameter, called source sink log likelihood (SSLLH), that determines the threshold probability necessary for the delay data to be incorporated into parameter updates (Fig. 4). The probability for an inter-vertex delay is first calculated given the current belief of the (Gaussian) delay distribution. If this probability is lower than the SSLLH then this motion is interpreted as a transition made via the source/sink node. The delay is given a probability equal to the SSLLH, and the transition is not used to update the network parameters associated with the origin and destination vertices.

The value assigned to the SSLLH parameter determines how easily the algorithm discards outliers and, hence, provides a compromise between robustness to observational noise and a tendency to discard useful data.

5. Simulation results

In this section, we will examine our approach through a number of experiments conducted in simulation. We will assess the performance of our topology inference algorithm and examine the effect of varying the input parameters.

5.1. The simulator

We have developed a tool that simulates agent traffic through an environment represented as a planar graph. Our simulation tool takes as inputs the number of agents in the system and a weighted graph where the edge weights are proportional to mean transit times between the nodes. All connections are considered two ways; *i.e.,* each connection is made up of two directed edges. The output is a list of observations generated by randomly walking the agents through the environment. Inter-node transit times are randomly selected from a normal distribution with a standard deviation equal to the square root of the mean transit time (negative transit times are rejected).

Two types of noise were modeled in order to assess performance using data that we believe more closely reflects observations collected from realistic traffic patterns. First, a 'white' noise was generated by removing a percentage of correct observations (false negatives) and replacing them with randomly generated spurious observations (false positives). Second, a more systematic noise was generated by taking a percentage of inter-vertex transitions and increasing the Gaussian distributed delay time between them by an additional delay value selected uniformly at random. The range of this additional delay time was selected to be from 0 to 20 times the average normal delay time. The hope is that small values of these types of noise simulate the effects of both imperfect sensors and also the tendency for agents to stop occasionally along their trajectories; *e.g.,* to talk, use the water fountain, or enter an office for an period.

A number of experiments were run using the simulator on randomly generated planar, connected graphs. The graphs were produced by selecting a sub-graph of the Delaunay triangulation [25] of a set of randomly distributed points. This technique has been used before to generate random planar graphs; (see Rekleitis et al. [26] for a 'complete description).

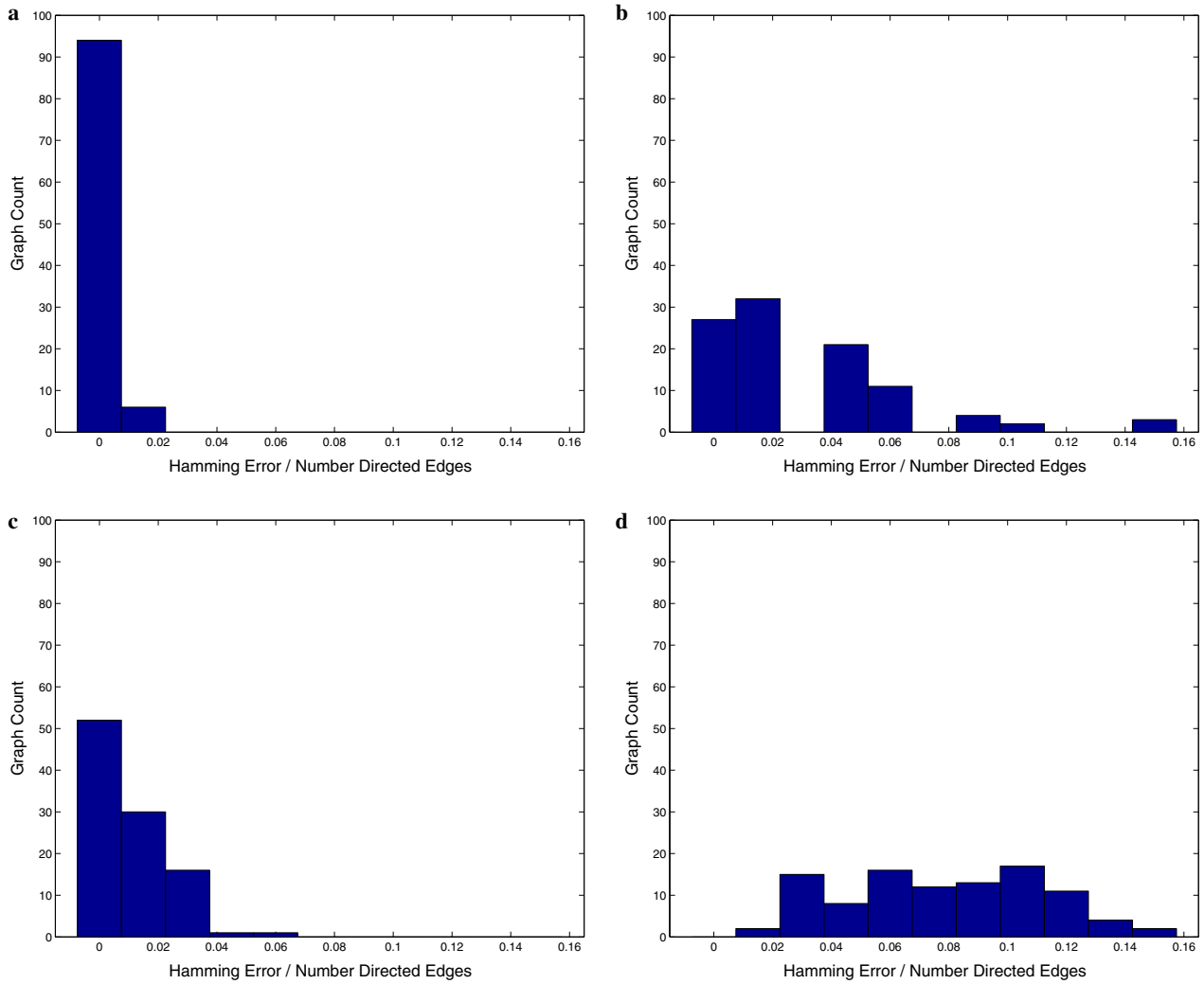

Fig. 5. A histogram of Hamming error per edge using the simulator with 8000 observations on 100 randomly produced graphs for: (a) 12 nodes and 4 agents, (b) 12 nodes and 10 agents, (c) 20 nodes and 4 agents, and (d) 20 nodes and 10 agents.

For each experiment, the results were obtained by comparing the final estimated transition matrix A' to the real transition matrix A. A graph of the inferred environment was obtained by thresholding A'. The Hamming error was then calculated by measuring the distance between the true and inferred graphs normalized by the number of directed edges m in the true graph:

$$HamErr_A = \left(\frac{1}{m}\right) \sum_{a_{ij}\in A, a'_{ij}\in A'} [thr(a_{ij}) - thr(a'_{ij})]^2$$

where $thr(a) = \lceil a_{ij} - \theta \rceil$.[1] Additionally, the squared error between the true and inferred transition matrix was calculated:

$$Err_A = \sum_{a_{ij}\in A, a'_{ij}\in A'} (a_{ij} - a'_{ij})^2$$

[1] A threshold value of $\theta = 0.1$ was selected for our experiments.

5.2. Assessment of the topology inference algorithm (Level One)

5.2.1. Performance under noise free conditions

When operating with noise-free data and knowledge of the correct number of agents in the environment, the results show that problems involving a limited number of agents were easy to solve given an adequate number of observations (Fig. 5). For example, for *95 per cent* of the generated 12 node graphs the topology was perfectly inferred with zero Hamming error for simulations with 4 agents.

5.2.2. Comparison to existing method

In most cases the thresholding method presented by Ellis et al. [9] did not produce results as accurate as those obtained using the method described here. Fig. 6 shows a comparison of an implementation of their thresholding method with our approach. Although shown to be less accurate in our simulations, this thresholding technique is

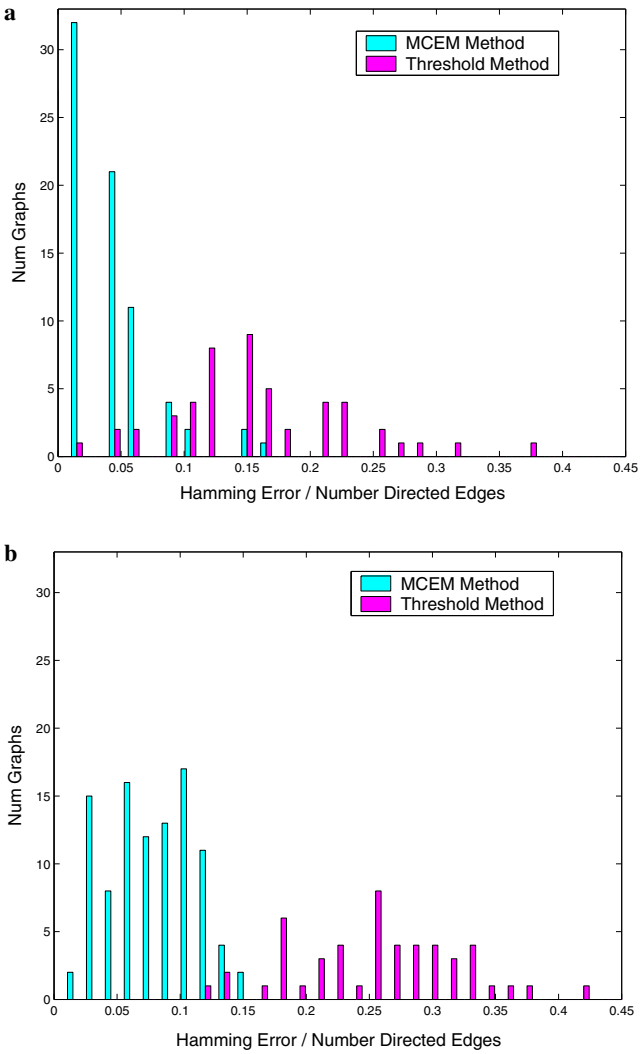

Fig. 6. Histograms of Hamming error per edge using both the threshold method described by Ellis et al. [9] and our MCEM method. The techniques were tested using 10 simulated agents with 8000 observations on 100 randomly produced graphs of size: (a) 12 nodes, 48 edges; and (b) 20 nodes, 80 edges.

very fast and does not need to make an assumption regarding the number of agents in the system.

5.2.3. Convergence and implementation assessment

Generally, the algorithm converged quickly, finding most of the coarse structure in the first few iterations and making incrementally smaller changes until convergence (Fig. 7). After every iteration of the MCEM process, the sampled ownership vectors increased in likelihood (Fig. 8). Recall that these ownership vectors represent plausible agent trajectories through the environment given the belief of the network parameters computed during the last iteration. Typically, most of the likelihood gain was seen in the first few steps of an iteration. As the number of iterations increased, however, the change in likelihood immediately after a parameter update decreased and a more significant gain in likelihood was seen during the course of a single iteration.

Modifying the number of samples K of the ownership vector drawn each iteration effected the performance of the algorithm. As the value of K was increased, the convergence time increased and the error of the final solution decreased (Table 1, Fig. 9). For the easier problems, frequent parameter updates seemed to lead to more rapid convergence and adequate accuracy. Presumably, this was because the Markov chain both quickly reached the stationary distribution, and also because the distribution was easy to characterize with only a few samples. However, it seemed that the more difficult problems, such as those involving a large number of agents, required a greater effort during each iterative E-Step in order to produce accurate results.

Eventually, we will look at finding a method of automating the effort placed in each iteration based on an analysis of the likelihood trends of the sampled ownership vectors. However, for the moment, we currently set the number of samples K used in each iteration to an experimentally determined intermediate value. For the remainder of the paper, all runs of the topology inference algorithm are conducted with $K = 20$.

5.2.4. Significance of graph size and the number of agents

A critical parameter is the number of agents moving in the system relative to the number of vertices. Clearly, if there is only one agent in the network the problem is straightforward since (ignoring detection errors) its event sequence can simply be "traced out". However, in the case of multiple agents, the events generated by a given agent's movements in the network risk being incorrectly associated with those of any other agents'. It is the relative density of the correct pairings relative to the incorrect ones that makes the problem more or less easy to solve.

Increasing either the number of agents present in the environment or the size of the graph made the problem more difficult to solve, albeit for rather different reasons. While increasing the number of agents allowed a greater number of probable trajectories, and was analogous to decreasing the signal to noise ratio in the system, increasing the graph size while holding the number of observations steady reduced the expected number of observations per edge in the graph. Experiments support the idea that the accuracy of our approach for a particular number of agents seems to depend on the ratio of observations to edges (Fig. 10).

In the extreme case, if there are some edges that have no observations recorded along them at all, our approach will not have enough information to infer the correct graph. At the minimum, an observed agent must traverse each edge at least once.

5.2.5. Effects of observational noise

While the algorithm is robust to moderate levels of 'white' observational noise, its sensitivity to systematic noise depends on the tuning of the delay model. The delay model is controlled by the SSLLH parameter which

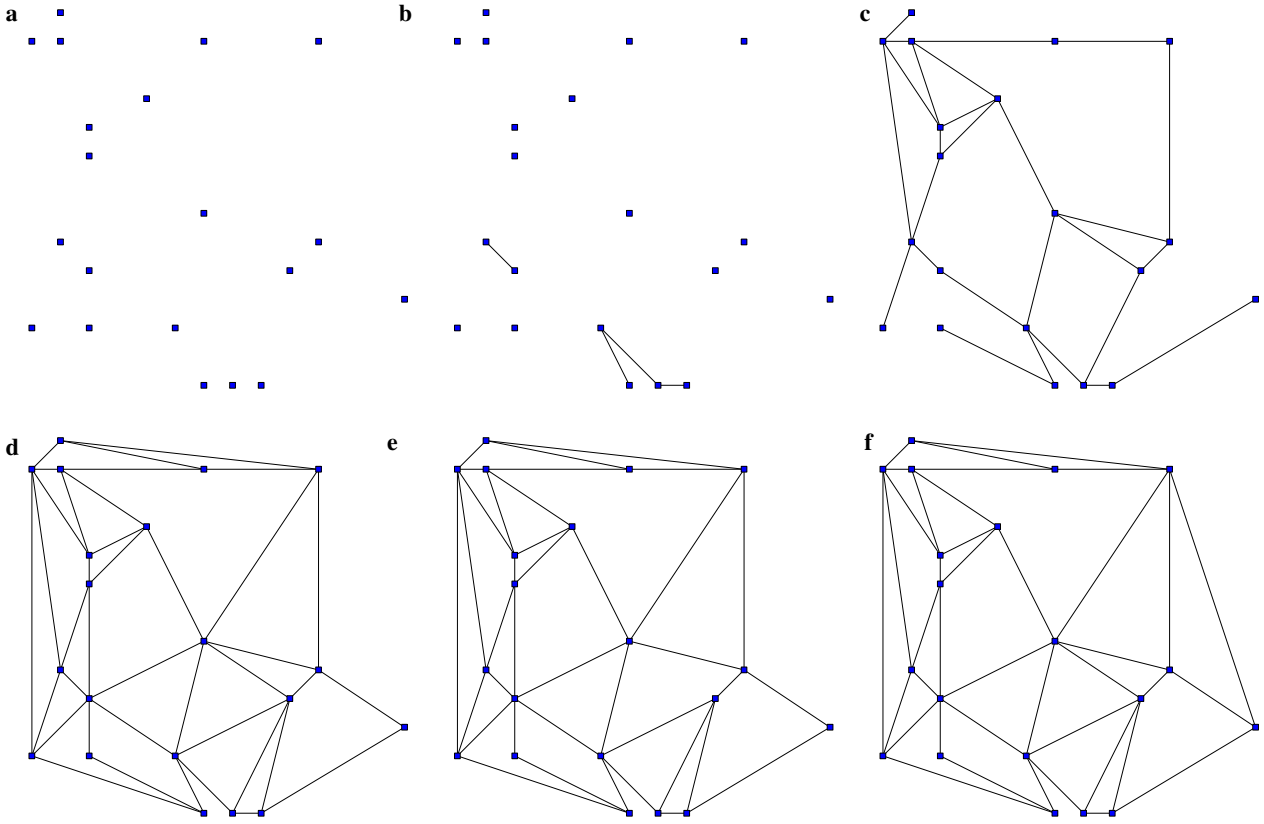

Fig. 7. Incremental belief of the topology of a 20 node, 80 (directed) edge graph using 4 simulated agents on 8000 observations: (a) initially (b) after 1 iteration, (c) after 2 iterations, (d) after 3 iterations (e) after 4 iterations (f) after 5 iterations (the true graph).

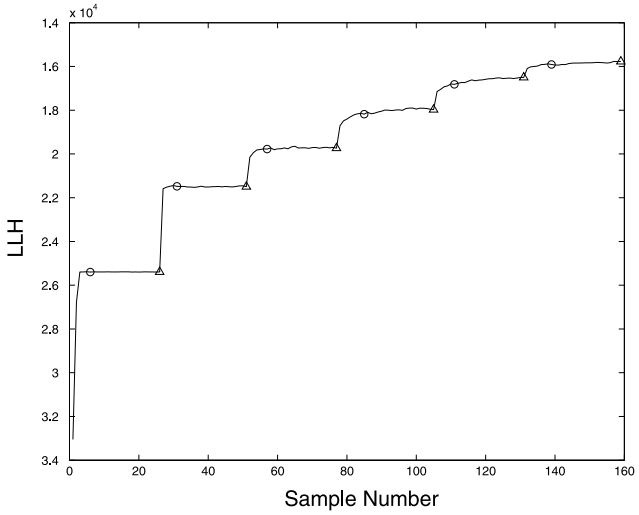

Fig. 8. The log likelihood of samples of the ownership vector for an example run of the algorithm using 4 simulated agents on a 12 node, 48 edge random graph with 4000 observations. The horizontal axis gives the sample number (across all iterations). For each iteration, only the samples shown between the circle and the triangle are used for updating network parameters (the M-Step).

determines the probability threshold for including delay data in the update of the network connectivity parameters. Fig. 11 shows the result of varying the value assigned to the SSLLH parameter for different types of noise. Fig. 12 shows the ability of the delay model to successfully identify and discard low probability transitions and explain them as transitions to the source/sink node.

Table 1
Comparison of performance and computational effort until convergence as a function of K averaged over 10 graphs of 12 nodes, 24 edges

K	4 agents		10 agents	
	Err_A	Total samples	Err_A	Total samples
3	0.187	70.5	0.555	221.4
20	0.141	179.1	0.399	424.8
40	0.121	314.3	0.381	630.4

When assigned a high SSLLH value, the mixture approach for modeling delays was successful at minimizing the effects of systematic noise. Even when 10 *per cent* of the delay times were uniformly increased, the Hamming error of the inferred transition matrix was near zero (Fig. 11(a)). When the SSLLH parameter was assigned a value representing negative infinity, the algorithm had no method of discarding delay data and had to update its network parameters given all the observations.

Moderate amounts of 'white', un-biased observational noise could be handled regardless of the tuning of the delay distribution mixture (Fig. 11(b)). However, the inferred transition beliefs were strongly effected by heavy amounts of this type of noise. The effect of randomly inserting and

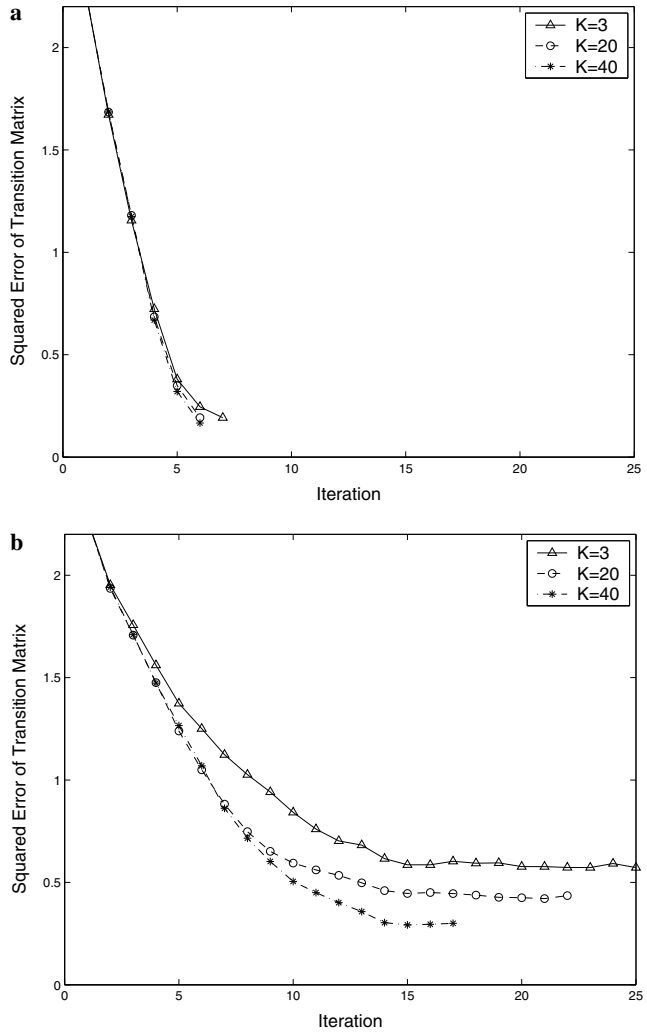

Fig. 9. A comparison of algorithm performance per iteration as a function of K. Results were obtained using the simulator on a 12 node, 48 edge random graph with 4000 observations with: (a) 4 agents; and (b) 10 agents.

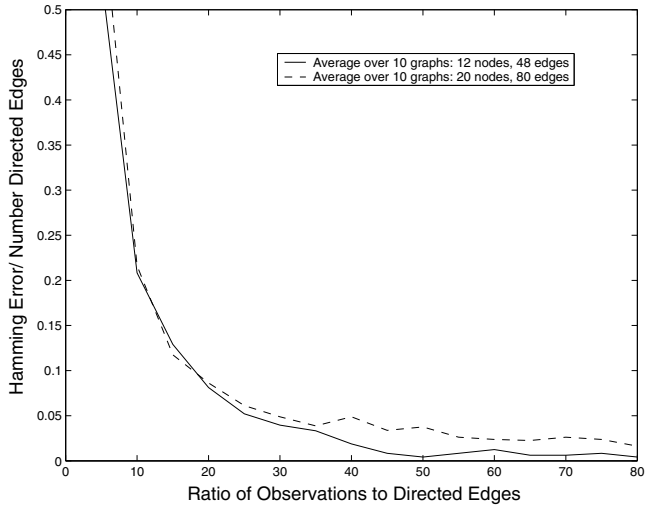

Fig. 10. Hamming error per edge as a function of the ratio of observations to true (directed) edges using 4 simulated agents.

deleting observations is to skew the distribution of likely sampled trajectories. Hence, the inference technique develops an incorrect belief of the underlying network and its inter-sensor transition probabilities. Since determining the correlation between the various sensor observations is key to our approach, it is unsurprising that after about 10 *per cent* of both missing and spurious observations the performance of the algorithm drops significantly.

The robust behavior of the algorithm under noisy conditions demonstrates both the general stability of the sampling-based approach and the success of the delay model. With an appropriately selected value assigned to the SSLLH parameter, the technique can infer highly accurate connectivity information even with moderate levels of both systematic and white noise.

6. Experimental results

6.1. Experimental setup

To assess the performance of our technique under real-world conditions, we conducted an experiment using a camera-based sensor network deployed in an office building and analyzed the results using our approach.

The sensor nodes were built up of inexpensive PC hardware networked together over Ethernet using custom software. A single node consists of a 352x292 pixel resolution Labtech USB webcam connected to a Flexstar PEGASUS single board computer (Fig. 13). The operating system used was Redhat linux based on kernel *2.4*. The sensor nodes contain an Intel Celeron 500Hhz CPU and 128 MB of RAM. They are disk-less and must netboot from a central server which they are connected to either *via* a wireless bridge or a standard Ethernet cable.

A standard client/server architecture was implemented over TCP/IP using linux sockets in the C language. Each sensor runs an identical copy of the client program while a single copy of the server application runs on a central computer.

The client software functions as a motion detector based on the Labtech webcam. During an initial period, a background image is captured from the camera and the method for triggering an event detection is calibrated. An intensity threshold is calibrated for each colour channel by calculating the standard deviation from the background based on a number of captured frames:

$$\theta_c = C \times std\{Frm_0 - Bkgrd, \ldots, Frm_n - Bkgrd\}$$

where Frm is a captured frame, $Bkgrd$ is the background frame and C is a constant determining the sensitivity of the system. The sensor then enters an armed state in which captured frames are compared to the background image, and any difference exceeding the threshold triggers a detection event (Fig. 14). A frame rate of approximately 10 Hz is obtained. Once triggered, the sensor re-arms itself after a couple of seconds of inactivity. The background is slowly updated to account for gradual changes

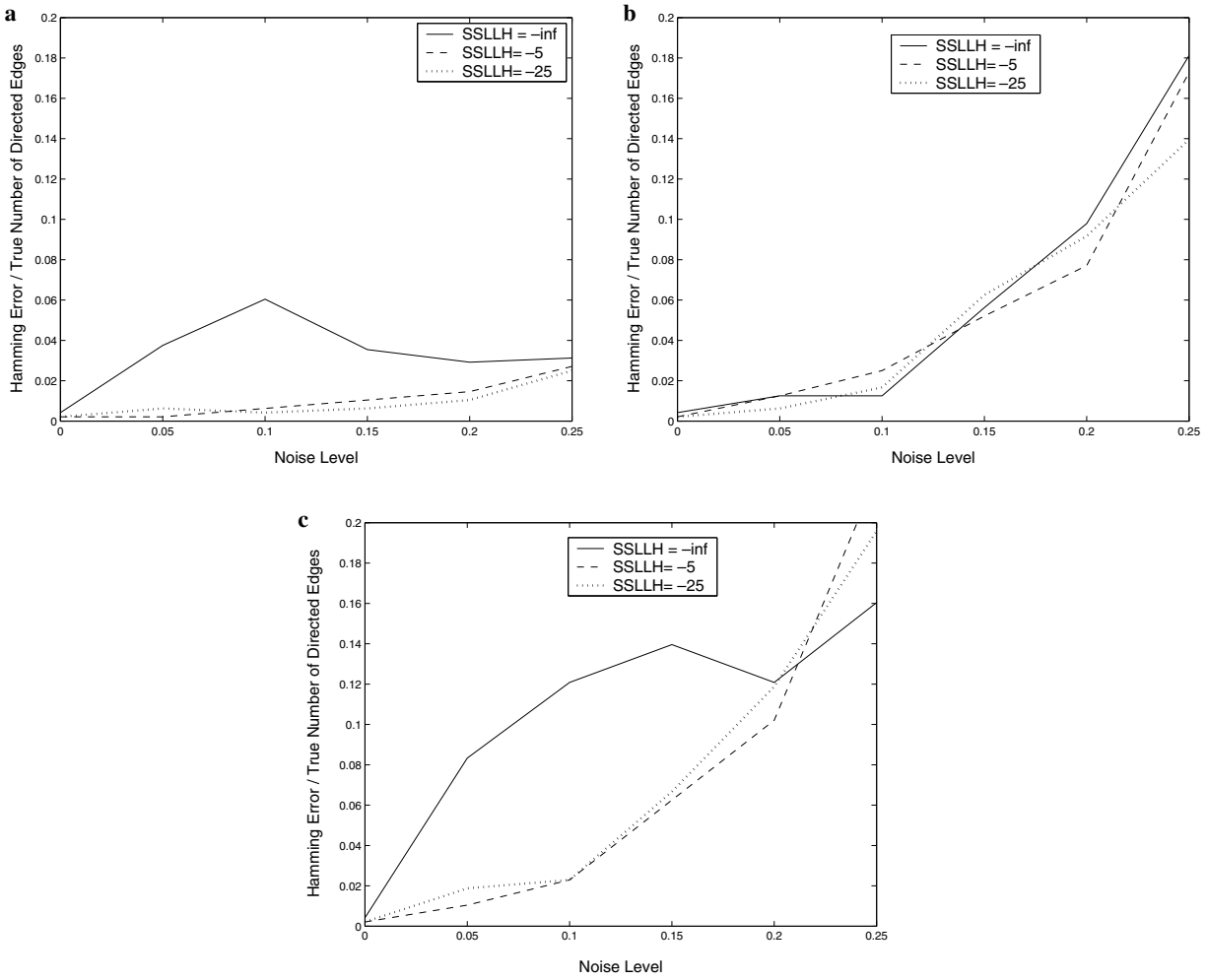

Fig. 11. Hamming error as a function of observational noise. The results are averaged over 10 graphs using 4 simulated agents on 12 node, 48 edge graphs with 4000 observations. The horizontal axis indicates proportions of: (a) systematic noise; (b) white noise; (c) both systematic and white noise.

in the scene; *e.g.,* changes in lighting or a re-positioned object such as a door:

$$Bkgrd' = \alpha \times Frm + (1 - \alpha) \times Bkgrd$$

where α is a constant determining how quickly the background is updated.

Events are transmitted over TCP/IP to a central server where they are time-stamped and logged for offline analysis. The server is multi-threaded and allows control of the system through a command line interface. In addition to detection events, the application allows either a full resolution capture or a low-resolution streaming of images from any sensor to the server.

The experiment was conducted in the hallways of one wing of an office building (Fig 15). The data were collected during a typical weekday for a period of five hours from 10:00 am to 2:30 pm. In addition to the normal traffic one or two subjects were encouraged to stroll about the region from time to time during the collection period in order to increase the density of observations. A total of approximately 1800 events were collected.

6.2. Assessment of results

Ground truth values were calculated in order to assess the results inferred by the approach. A topological map of the environment (Fig. 16(a)) was determined based on an analysis of the sensor network layout shown in Fig. 15. (Note that we have not attempted to analytically determine reasonable connections to sources or sinks in the environment.) Additionally, inter-vertex transitions times for the connected sensors were recorded with a stopwatch for a typical subject walking at a normal speed (Table 3).

The network parameters inferred by our topology inference algorithm closely corresponded to the ground truth values. Table 2 shows the transition matrix output by the algorithm, and Fig. 16 compares the analytically determined and inferred topological maps. Disregarding reflexive links, the difference between the inferred and determined matrices amounts to a Hamming error of 1. The inferred connection from D to B was not given a transition probability large enough to be detected based on our

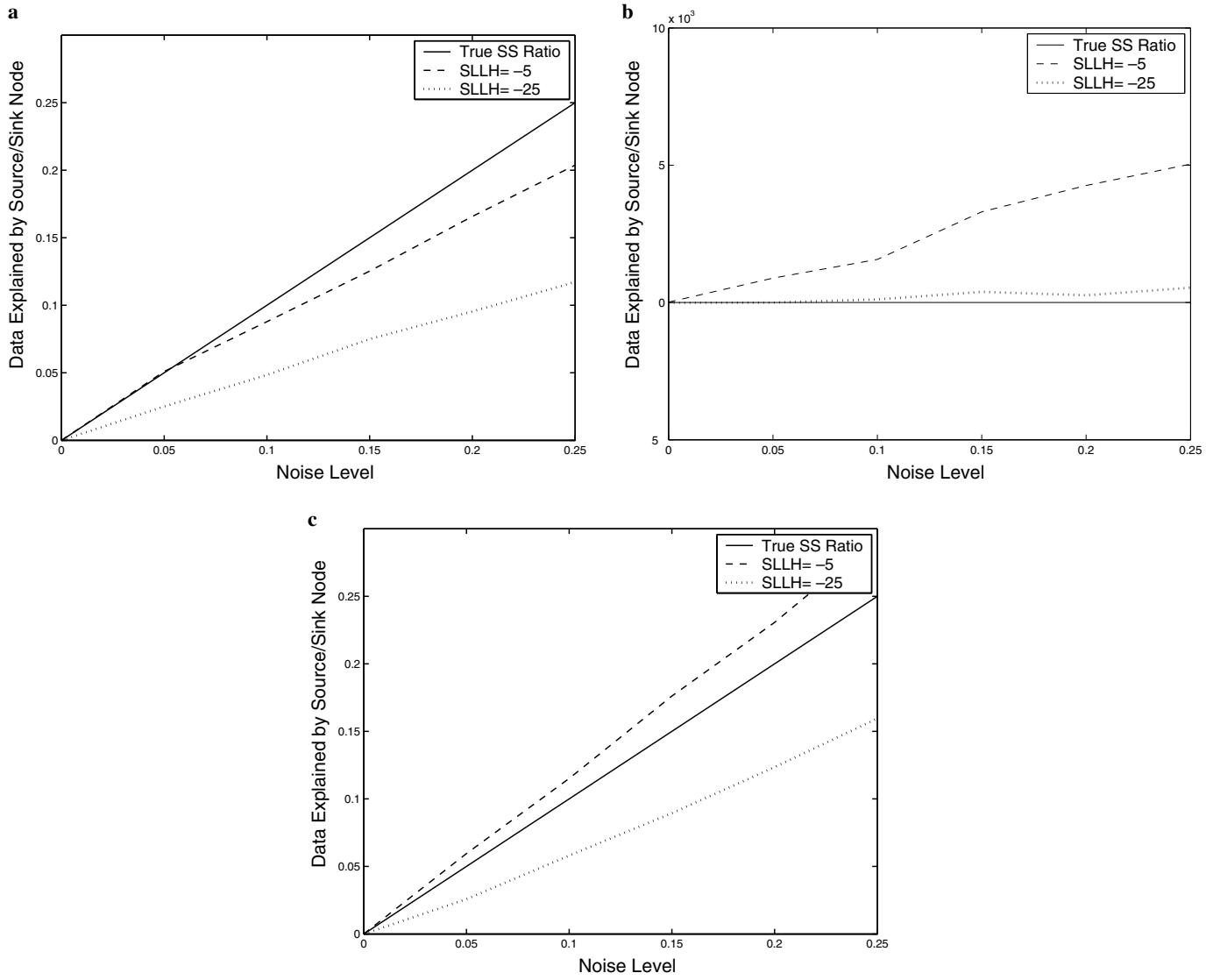

Fig. 12. A plot of the proportion of delay data rejected as a function of observational noise. The results were averaged over 10 graphs using 4 simulated agents on 12 node, 48 edge graphs with 4000 observations. The horizontal axis indicates proportions of: (a) systematic noise; (b) white noise; (c) both systematic and white noise.

Fig. 13. A camera-based sensor.

thresholding technique. However, the opposite edge from B to D was correctly inferred. Of course, it would be easy to build into the algorithm the assumption that all edges must be two ways. A strong belief in an edge in one direction would dictate that the opposite edge must also exist.

The mean transition times produced by the algorithm were also consistent to those determined by stopwatch (Table 3). Some examples of inferred delay distributions are shown in Fig. 17.

Sensor F marks the only heavily used entrance and exit to the region monitored by the network. The self-connection inferred to this node is due to a detected correlation in the delay between exit times and subsequent re-entry times for agent motion. In fact, this correlation is due to the tendency of subjects to re-enter the system after roughly the same time period (*e.g.,* to

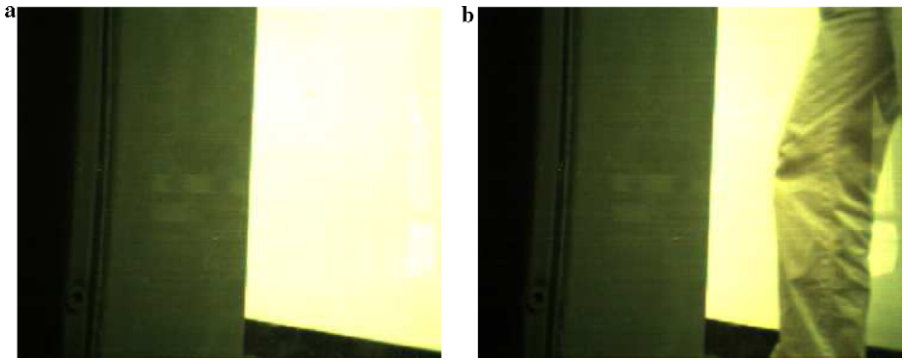

Fig. 14. An example of images captured from a vision sensor: (a) the background image; (b) a frame triggering an event detection.

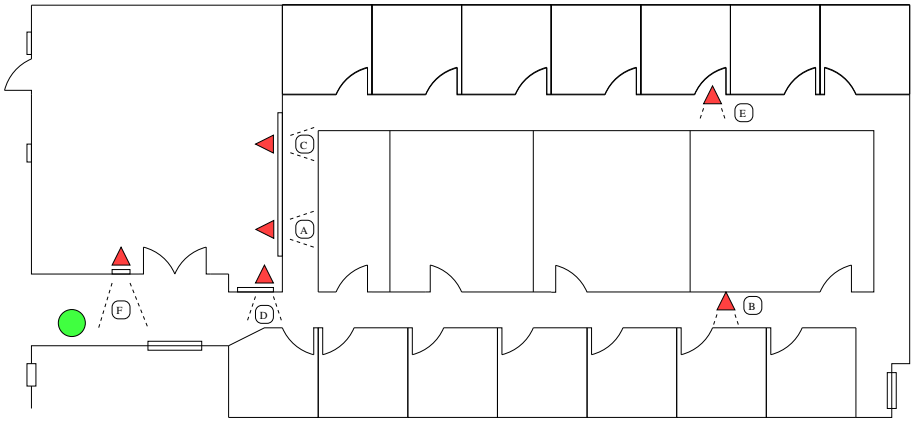

Fig. 15. The layout of the six camera sensor network used for experiment. Labeled triangles represent sensor positions, and the circle represents the location of the central server.

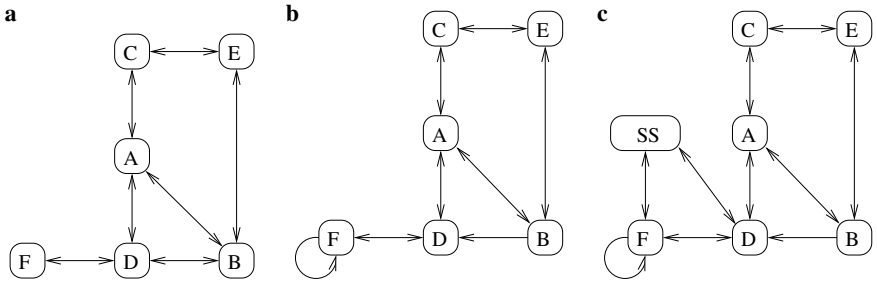

Fig. 16. Analytical (a), inferred (b) and inferred with source/sink node (c) topological maps.

Table 2
The transition matrix inferred from the experimental data

	A	B	C	D	E	F	SS
A	0.05	**0.16**	**0.28**	**0.32**	0.02	0.09	0.08
B	**0.28**	0.08	0.01	**0.12**	**0.41**	0.04	0.06
C	**0.40**	0.05	0.05	0.05	**0.32**	0.05	0.08
D	**0.22**	0.08	0.05	0.07	0.01	**0.43**	**0.13**
E	0.04	**0.39**	**0.40**	0.04	0.04	0.03	0.06
F	0.06	0.03	0.08	**0.34**	0.00	**0.28**	**0.22**
SS	0.08	0.07	0.09	**0.25**	0.03	**0.49**	0.00

SS refers to the source/sink node introduced by the algorithm. Bold values over the threshold $\theta = 0.1$ are interpreted as one way edges. The underlined values were not directly predicted by the ground truth analysis.

use the washroom or photocopier). Therefore, the detection of this connection was actually a correct inference on the part of the algorithm.

It is interesting to note that two-way connections were inferred to the source/sink node from both sensors *D* and *F* (Fig. 16(c)). It was possible for subjects to pass by either of these sensors on their way into or out of the monitored region. (The exit to the far right of the area, shown in Fig. 15, was little used.) This demonstrates the function of the source/sink node as a method for the algorithm to explain sudden appearances and disappearance of agents in the system.

Table 3
Comparison of timed and inferred delay times (both ways) between sensors

Connection	Timed	Inferred
A,B	16	15/16
A,C	3	3/3
A,D	4	3/3
B,D	15	16/17
B,E	16	15/15
C,E	14	15/14
D,F	5	5/3

All values rounded to nearest second.

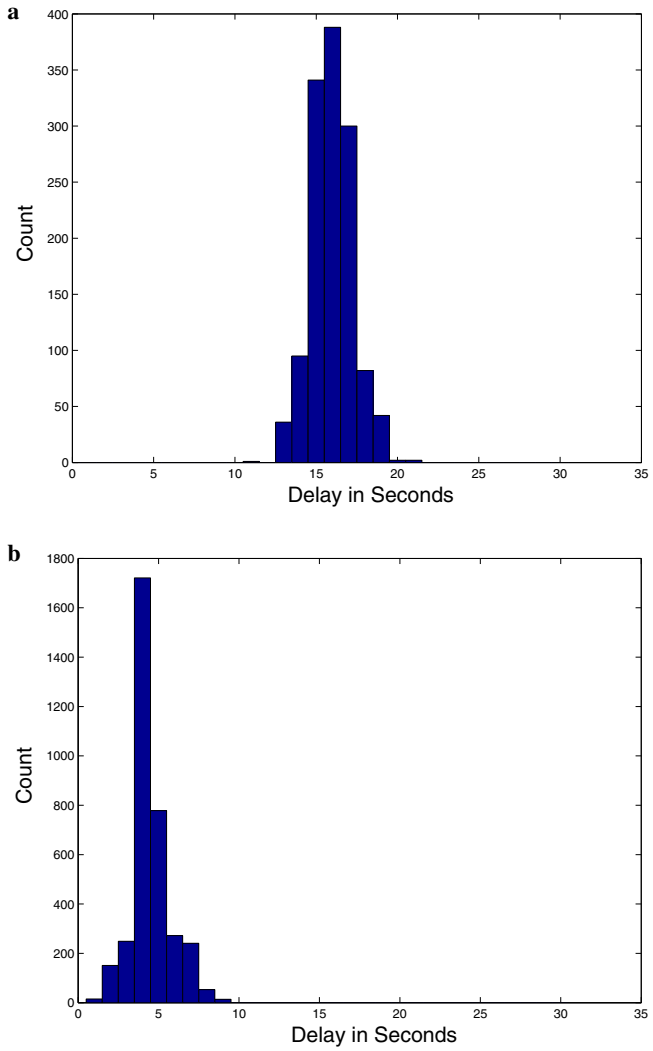

Fig. 17. Examples of delay distributions for sensor A to sensor B (a) and sensor D to sensor F (b).

7. Conclusions and future work

In this paper, we have presented an algorithm for learning the connectivity information of a sensor network based on a stochastic trajectory sampling. The technique employs a realistic model of inter-sensor delay distributions that makes it robust to realistic variations in traffic patterns and observational noise in general. The approach was demonstrated with simulation data and verified with experiments conducted on a vision-based sensor network.

Future work will look at developing a more sophisticated vision system which produces probabilistically labeled tracking data. This additional information could be readily incorporated into the approach and would lead to more rapid convergence.

Acknowledgements

We thank Ionnis Rekleitis, Philippe Giguere, Junaed Sattar, Eric Bourque, Matt Garden and others of the Mobile Robotics lab, along with the CIM administration for their technical help and good ideas. Thank-you in addition to Michelle Theberge for the photos, proof reading, and valuable assistance during the experiment.

References

[1] A. Savvides, C. Han, M. Strivastava, Dynamic fine-grained localization in ad hoc networks of sensors, in: 7th annual international conference on Mobile computing and networking, Rome, Italy, 2001, pp. 166–179.
[2] D. Moore, J. Leonard, D. Rus, S. Teller, Robust distributed network localization with noisy range measurements, in: Proc. of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys '04), Baltimore, November 2004.
[3] G.P. Stein, Tracking from multiple view points: Self-calibration of space and time, in: Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on, vol. 1, June 1999, pp. 521–527.
[4] R.B. Fisher, Self-organization of randomly placed sensors, in: Eur. Conf. on Computer Vision, Copenhagen, May 2002, pp. 146–160.
[5] C. Stauffer, K. Tieu, Automated multi-camera planar tracking correspondence modeling, in: Proceedings of the IEEE Computer Vision and Pattern Recognition, vol. 1, July 2003, pp. 259–266.
[6] A. Rahimi, B. Dunagan, T. Darrell, Simultaneous calibration and tracking with a network of non-overlapping sensors, in: CVPR 2004, vol. 1, June 2004, pp. 187–194.
[7] O. Javed, Z. Rasheed, K. Shafique, M. Shan, Tracking across multiple cameras with disjoint views, in: The Ninth IEEE International Conference on Computer Vision, Nice, France, 2003.
[8] D. Makris, T. Ellis, J. Black, Bridging the gaps between cameras, in: IEEE Conference on Computer Vision and Pattern Recognition CVPR 2004, Washington DC, June 2004.
[9] T. Ellis, D. Makris, J. Black, Learning a multicamera topology, in: Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance, Nice, France, October 2003, pp. 165–171.
[10] Y. Bar-Shalom, Ed., Multitarget multisensor tracking: advanced applications. vol. II. Artech House, 1992.
[11] J. Liu, J. Liu, J. Reich, P. Cheung, F. Zhao, Distributed group management for track initiation and maintenance in target localization applications, in: Proceedings of 2nd International Workshop on Information Processing in Sensor Networks (IPSN), April 2003.
[12] M. Rosencrantz, G. Gordon, S. Thrun, Locating moving entities in indoor environments with teams of mobile robots, in: Second international joint conference on Autonomous agents and multiagent systems, Melbourne, Australia, 2003, pp. 233–240.
[13] J. Shin, L.J. Guibas, F. Zhao, A distributed algorithm for managing multi-target identities in wireless ad hoc sensor networks, in: Information Processing in Sensor Networks Second International Workshop, IPSN 2003, Palo Alto, CA, April 2003, pp. 223–238.

[14] C. Rasmussen, G. Hager, Probabilistic data association methods for tracking multiple and compound visual objects, in: IEEE Trans. Pattern Analysis and Machine Intelligence, 2001.
[15] T. Huang, S.J. Russell, Object identification in a bayesian context, in: IJCAI, 1997, pp. 1276–1283.
[16] T. Huang, S.J. Russell, Object identification: a bayesian analysis with application to traffic surveillance, Artificial Intelligence 103 (1–2) (1998) 77–93.
[17] H. Pasula, S. Russell, M. Ostland, Y. Ritov, Tracking many objects with many sensors, in IJCAI-99, Stockholm, 1999.
[18] A. Dempster, N. Laird, D. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society 39 (1977) 1–38.
[19] G. Wei, M. Tanner, A monte-carlo implementation of the EM algorithm and the poor man's data augmentation algorithms, Journal of the American Statistical Association 85 (411) (1990) 699–704.
[20] W. Burgard, D. Fox, H. Jans, C. Matenar, S. Thrun, "Sonar-based mapping with mobile robots using EM," in Proc. 16th International Conf. on Machine Learning. Morgan Kaufmann, San Francisco, CA, 1999, pp. 67–76.
[21] H. Shatkay, L.P. Kaelbling, Learning topological maps with weak local odometric information, in: IJCAI (2), 1997, pp. 920–929.
[22] M. Tanner, Tools for Statistical Inference, 3rd ed., Springer Verlag, New York, 1996.
[23] C. Andrieu, N. de Freitas, A. Doucet, M.I. Jordan, An introduction to MCMC for machine learning, Machine Learning 50 (2003) 5–43.
[24] G.S. Fishman, Monte carlo concepts, algorithms, and applications, Springer-Verlag, New York, 1996.
[25] F. Preparata, M. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York, NY, 1985.
[26] I.M. Rekleitis, V. Dujmović, G. Dudek, "Efficient topological exploration," in Proceedings of International Conference in Robotics and Automation, Detroit, USA, May 1999, pp. 676–681.