Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA 2004)
New Orleans, Louisiana, Apr. 26-May 1, 2004

Object Surveillance Using Reinforcement Learning Based Sensor Dispatching

Michael D. Naish*, Elizabeth A. Croft† and Beno Benhabib‡
*Dept. of Mechanical and Materials Engineering, University of Western Ontario, London, Ontario, Canada N6A 5B9
Email: naish@eng.uwo.ca
†Industrial Automation Laboratory, University of British Columbia, Vancouver, B.C., Canada V6T 1Z4
Email: ecroft@mech.ubc.ca
‡Computer Integrated Manufacturing Laboratory, University of Toronto, Toronto, Ontario, Canada M5S 3G8
Email: beno@mie.utoronto.ca

Abstract— **This paper outlines an approach to the coordination of multiple mobile sensors for the surveillance of a single moving target. A real-time dispatching algorithm is used to select and position groups of sensors in response to the observed object motion. The aim is to provide robust, high-quality data while ensuring that the system can react to unexpected object manoeuvres. Sensors are assigned to collect data at specific points on the object trajectory. A dispatching strategy learned via reinforcement learning is used to control the sensor poses with respect to these points. In using the learned strategy, each sensor adopts an egocentric view of the system state to determine the most appropriate action. Simulations demonstrate the performance of the RL-based dispatcher, in comparison to similar static-sensor systems.**

I. Introduction

An approach for the coordination and control of multiple mobile sensors, as applied to the surveillance of a target manoeuvring within a defined workspace, is considered in this paper. The collected sensory information is targeted towards autonomous robotic tasks, particularly object recognition, characterization, and interception. The use of multiple redundant sensors provides an opportunity to improve the quality and robustness of the acquired data, as compared to single-sensor and/or fixed-sensor systems. Specifically, improvements may be realized by sensing the object from optimal positions, mitigating uncertainty through sensor fusion, and providing the capability to react to object manoeuvres.

The task of selecting and positioning sets of sensors in the context of moving object surveillance is nontrivial. The complexity of the task arises from the nature of the task space. Often the task space is continuous and nonlinear, with many variables that impact the success of the surveillance system. A multisensor surveillance system must address two problems: (i) The combinatorial problem of determining which sensors will be used at any given time (i.e., selecting subsets of sensors from the general pool of sensors that are available for surveillance), and (ii) the constrained, nonlinear problem of determining a suitable pose for each sensor.

Some recent approaches to this problem rely on cost functions (heuristics) [1] or autonomous agents [2] to direct sensors towards particular surveillance targets. An alternate approach uses the principles of dispatching, as applied to operation of service vehicles such as taxicabs and ambulances (e.g., [3]–[5]), where the role of the dispatcher is to determine which sensor(s) will be sent to service a particular demand.

A. Dispatching

A dispatcher may be used to select and position groups of sensors in a coordinated manner for the real-time surveillance of a manoeuvring object. In doing so, the remaining sensors can be positioned in anticipation of future sensing requirements.

The goal of the dispatcher, as considered in this paper, is to maximize the effectiveness of the sensing system as it estimates object parameters at predetermined times or positions along the (discretized) object trajectory. It is assumed that the times at which the information is desired are fixed. These predetermined times are referred to as demand instants, t_j. The (expected) position of the object at a particular demand instant is a demand point, D_j. Observations of the object motion are used to predict the demand points for a finite segment of the object trajectory. This set of demands constitutes a rolling horizon, with the demand at the start of the horizon requiring service (data acquisition) first.

To improve the quality and robustness of the acquired data, the sensing system is assumed to consist of a set of multiple redundant sensors. Of these sensors, only a subset are typically required to meet the sensing requirements of a demand. In other words, the dispatcher selects from the total number of sensors, n, a subset (of size k, where $k \leq n$). The data from this group of sensors, or fusion subset, is then combined using a sensor fusion process (for applicable techniques, see [6]).

The dispatcher uses the rolling horizon to assign sensors to demands and specify the desired pose of each sensor. The demands are considered sequentially, evaluating the fitness of each sensor with respect to the demand under consideration. A fusion subset comprising the k most suitable sensors is assigned to the first demand point. This assignment reflects the coordination strategy of the dispatcher. The corresponding desired poses of the assigned sensors, in addition to the remaining sensors, are determined by a positioning strategy. The positions of the unassigned sensors are based on the future sensing requirements indicated by the rolling horizon.

There are a number of different possible implementations for the dispatcher. These include the use of optimization, heuristic rules, machine learning, agents, etc. A heuristic rule based approach is discussed in [7]; this paper proposes a machine learning based approach. Machine learning offers the possibility of developing a dispatcher through experience, gaining insight into the problem over time.

II. Reinforcement Learning

Reinforcement learning (RL), unlike supervised learning techniques, does not require knowledge of "correct" outputs for a given input. Instead, it is based on an idea that is common to human learning, namely, that the tendency to produce a particular action is strengthened if it produces favourable results (e.g., pleasure), and weakened if it produces unfavourable results (e.g., pain). RL, as applied to the control of a system, attempts to learn the long-term value of the actions that may be taken from any given state [8]. These values are referred to as Q-values. Learning proceeds on the basis of a numerical reward received for each action that is taken. By learning the value of all state-action pairs, $Q(s, a)$, the reward received for taking an action can always be maximized. Learning every $Q(s, a)$ requires a balance of exploration (taking actions of unknown value) and exploitation (selecting the best or greedy action to maximize reward). Using an action selection policy that achieves this balance will allow an appropriate control strategy to be learned, provided that the reward function properly reflects the objectives of the system.

A. Temporal Difference Learning

Without a world model to provide the state-transition probabilities, the action-value function can be learned using an incremental update equation, based on the temporal difference (TD) of the estimated action-value functions. The action-value function is updated by the TD-error as follows [8]:

$$\begin{aligned} Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) &+ \\ &\eta\big[\underbrace{r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)}_{\text{TD-error}}\big], \end{aligned} \quad (1)$$

where r is the received reward; γ, $0 \leq \gamma \leq 1$, represents the discount rate (which balances between immediate and future rewards); and η controls the learning rate. Eq. (1) is referred to as one-step SARSA [8], arising from the quintuple of events (s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}) that make up the transition from one state-action pair to the next. To better express the causal effect of selected actions, SARSA is augmented by eligibility traces for each state-action pair (i.e., $\forall s \in \mathcal{S}, a \in \mathcal{A}$) as follows:

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a) + 1 & \text{if } s = s_t \text{ and } a = a_t; \\ \gamma\lambda e_{t-1}(s, a) & \text{otherwise,} \end{cases} \quad (2)$$

where λ is referred to as the trace-decay parameter and is responsible for temporal credit assignment. When $\lambda = 0$, no feedback occurs beyond the current time step; when $\lambda = 1$, the error feeds back without decay arbitrarily far in time. The traces indicate the degree to which each state-action pair is eligible for undergoing learning changes, should a reinforcing event occur. Thus, the action-value function $Q_t(s, a)$ may be updated by SARSA(λ) as follows:

$$Q_{t+1}(s, a) = Q_t(s, a) + \eta\delta_t e_t(s, a), \qquad \forall s, a, \quad (3)$$

where

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t). \quad (4)$$

One-step SARSA and SARSA(λ) follow the approach of generalized policy improvement. The action values of the current policy π are approximated by $Q^\pi(s, a)$; these estimates are then used to improve the policy and the process repeats. If the policy is improved towards the optimal policy (by increasingly selecting greedy actions) then, eventually, the action-value function will reflect the optimal action for each state.

B. Function Approximation

The simplest representation of the state-action space is to use a table with one entry for each state-action pair. This approach is suitable for limited discrete state-spaces with no requirement for generalization; however, for many tasks, it becomes impossible to represent all possible states, to say nothing of repeatedly visiting each to ensure the adequate learning. An alternate approach that has been applied to a number of different applications is neural networks. Successful applications of network-based RL include playing backgammon at the grandmaster-level [9], the dispatching of elevators within an office building [10], and mobile robot navigation [11]. In each case, the state space exceeds 10^{20} states; however, successful strategies were learned via TD-learning.

Unlike other approaches (e.g., tile-coding, radial-basis functions, etc.), neural networks scale proportionally with the input space. Neural networks can approximate complex functions and provide generalization of the state space. The most common form of network is a multi-layer feed-forward network, trained by the back-propagation of errors [12]. Such a network may be employed to represent the value of the state-action pairs particular to a system. A general algorithm for applying TD-learning techniques to train a feed-forward neural network is outlined in [13]. The approach uses eligibility traces to adjust the nodal weights during back propagation.

III. Learning to Dispatch

A. General Approach

Dispatching must both select the sensors to service demands and guide each sensor into appropriate poses. The coordination strategy is implemented at the system level. In other words, the dispatcher acts as a supervisory controller, aware of the system state and the capabilities of each sensor. Fusion subsets are constructed by assessing each sensor heuristically, [7].

The positioning strategy, on the other hand, is implemented at the sensor level. Effectively, each sensor is an autonomous agent, independently determining its pose in the workspace.

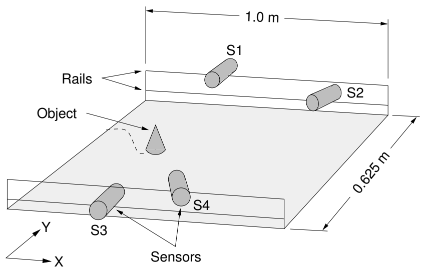

Fig. 1. Example workspace.

An appropriate strategy is learned through exposure to exemplary object trajectories. The goal of each sensor is to select an appropriate action from a finite set. The number of actions is dependent on the constraints imposed on the sensor motion. The simplest case involves sensors that are mounted on a rail or linear stage. Such sensors are capable of three actions: (i) move right, (ii) move left, and (iii) stop. Additional degrees of freedom (either translational or rotary) increase the number of actions exponentially. For example, the addition of a stage orthogonal to the first (e.g., up and down) would increase the number of possible actions to nine.

Herein, an example surveillance system is considered for illustrative purposes. This system is applied to a planar workspace, where the target and sensors are each constrained to move in planes that are parallel to one another, Fig. 1.

In Fig. 1, sensors are constrained to rails at the edge of the workspace. Each sensor can translate along the rail (x), and rotate (α). The characteristics of the sensors are assumed to be such that the best-quality sensor data is collected when the range between the sensor and object is minimized and the sensor axis is aligned with the object. Under this assumption, the dispatcher learns to control the translational motion of each sensor; rotation is controlled in a heuristic manner, minimizing the angular offset between the sensor axis and appropriate demand points on the rolling horizon. Further details regarding heuristic control may be found in [7].

B. Action-Value Estimation

For any given state, a sensor must be able to select a single action from the various possibilities. If $\mathcal{A} = \{\mathsf{right}, \mathsf{left}, \mathsf{stop}\}$, then only one of these actions can be performed at a given point in time. Herein, the value of each action is predicted using a separate single-output feed-forward neural network. Each network has a two-layer (single hidden layer) topology and utilizes logistic activation functions. The action-value function $Q(s, a)$ is thus represented by three separate networks. The use of separate networks prevents the hidden unit weights from receiving conflicting demands from different outputs (actions) during training [11].

The action-value function (policy) is shared by all sensors. This egocentric representation contrasts with a distributed implementation in which each sensor has its own policy, allowing specialized control. Under the assumption that each sensor is operationally identical, the egocentric implementation allows for faster learning and better policy generalization.

TABLE I
EGOCENTRIC REPRESENTATION OF STATE SPACE.

Description	Type	No. Inputs
Part of fusion subset (assigned).	Binary	1
Normalized range *behind* sensor to demand point (relative to direction of target motion).	Real	5
Normalized range *ahead of* sensor to demand point.	Real	5
Normalized x-position of the first demand point in rolling horizon.	Real	6
Normalized x-position of sensor in workspace.	Real	6
	Total	23

C. State Space Encoding

The state of the system, from the point of view of a particular sensor, must be represented in the form of inputs to the action networks. This encoding expresses the features upon which actions are selected. The challenge is to provide all of the necessary information, while eliminating states that may be redundant, unnecessary, or otherwise confusing. One such encoding is presented in Table I.

In Table I, the real-valued inputs are spread encoded across a number of inputs. This serves to improve the resolution of the network, allowing select states to be better distinguished from one another. Two types of encoding are used. The first represents the range between the sensor and the first demand point on the rolling horizon. It is based on a set of N overlapping sigmoid functions (one for each node), where N is an even integer value. The activation of each input node, o_i, is dependent on the relative position of the sensor:
If $(x_{S_i} < x_{D_j} \wedge x_{D_{(j+1)}} > x_{D_j}) \vee (x_{S_i} > x_{D_j} \wedge x_{D_{(j+1)}} < x_{D_j})$ (i.e., the sensor, S_i, is "behind" the demand point, D_j),

$$o_i = \begin{cases} \dfrac{1}{1 + e^{w(b_{(N/2+1-i)}+r+u)}} & i = 1, \ldots, N/2 \\ 0 & i = N/2+1, \ldots, N; \end{cases} \quad (5)$$

else if $(x_{S_i} > x_{D_j} \wedge x_{D_{(j+1)}} \geq x_{D_j}) \vee (x_{S_i} < x_{D_j} \wedge x_{D_{(j+1)}} \leq x_{D_j})$ (i.e., the sensor is "ahead of" the demand point),

$$o_i = \begin{cases} 0 & i = 1, \ldots, N/2 \\ \dfrac{1}{1 + e^{w(b_{(i-N/2)}-r+u)}} & i = N/2+1, \ldots, N, \end{cases} \quad (6)$$

where

$$w = 4N, \quad u = 1/2N, \quad b_i = (2i-1)/(N+1),$$

and

$$r = \sqrt{\frac{(x_{S_i} - x_{D_j})^2 + (y_{S_i} - y_{D_j})^2}{(x_{\mathrm{LR}} - x_{\mathrm{UL}})^2 + (y_{\mathrm{LR}} - y_{\mathrm{UL}})^2}}.$$

Here, x_{S_i} represents the x-position of the sensor, x_{D_j} is the x-position of the jth demand point, and r is the range between the sensor and the demand point, normalized by the diagonal distance across the workspace (spanning the upper-left and lower-right corners).

The intent of this encoding is to increase the activation level of the input nodes as the range between the sensor and demand point increases. For example, as the sensor falls further behind, more inputs in the range $i = 1, \ldots, N/2$ come on, starting with node 1 (nodes $N/2+1, \ldots, N$ will remain off). Note that if the sensor is "in-line" with the demand point, all N of the input nodes will have activation levels equal to zero.

The second type of encoding, based on a set of Gaussians, is used to represent the normalized position of a sensor or demand point. Encoding the x-position of a sensor, for example, the activation of the ith input node (of N nodes total) is determined by:

$$o_i = \exp\left(-(\hat{x}_{S_i} - i/N)^2/2\sigma_i^2\right), \tag{7}$$

where $\hat{x}_{S_i}$ is the normalized position of the sensor and σ_i^2 is the encoding variance that defines the width of the Gaussian ($\sigma_i^2 = 0.01$ was used herein). Similarly, the x-position of the first demand point can be encoded.

D. Returns

The application of reinforcement learning to the sensor dispatching problem requires a reward function to provide a measure of how well the sensing system performs over time. Object surveillance is episodic in nature. Each episode begins when an object enters the workspace; it terminates when the object exits the workspace. While in the workspace, the performance of the sensing system is assessed herein using a measure of demand-point visibility.

As defined in [7], the visibility of an unoccluded demand point using a single sensor may be characterized as:

$$v_\mathrm{s} = 1/\|R\| \tag{8}$$

where $\|R\|$ is the Euclidean norm of the covariance matrix associated with the sensor measurement. When data from multiple sensors is combined in a sensor fusion process, visibility may be expressed as:

$$v_\mathrm{f} = 1/\|P\|, \tag{9a}$$

where P represents the fused covariance matrix,

$$P = \left[\sum_{i=1}^{k} R_i^{-1}\right]^{-1}. \tag{9b}$$

Typically, the measurement uncertainty (covariance) changes as the pose of the sensor(s), with respect to the demand point, is varied. Considering a single sensor, the goal of the dispatcher is to manoeuvre the sensor such that the visibility measure v_s is maximized. The sensor pose at this maximum reflects the best-possible visibility, v_p. The best-possible visibility is constrained by the capabilities of the sensor, the task, and the workspace (i.e., by mounting the sensor on a positioning stage, the path of the sensor is restricted). The best-achievable visibility, v_a, is the visibility that can be expected, given a finite amount of time for manoeuvring. Thus, v_a is constrained additionally by the dynamic characteristics of the sensor (maximum velocity, acceleration, etc.) and time. Similarly, v_p and v_a may be computed for a group of sensors by maximizing v_f. The only difference for multiple sensors is that v_p must also consider the optimal fusion subset. Together, v_a and v_p may be used to compute the normalized best-achievable visibility:

$$v_\mathrm{n} = v_\mathrm{a}/v_\mathrm{p}. \tag{10}$$

The normalized visibility measure in (10) allows the performance of different sensing systems (possibly surveying different object trajectories) to be compared to one another.

In computing the reward at each iteration, v_n may be used in one of two ways. The first way assesses performance for the individual demand points; the second use computes the configuration visibility, which indicates the overall success of the sensing system for the given object trajectory. This is defined as the worst-case visibility for the M demand points serviced over the object trajectory. Together, these visibility measures are utilized to express the overall return for the episode. At each iteration, the reward, r_t is defined by:

$$r_t = \begin{cases} (\bar{v}_\mathrm{n})_j = \left(\frac{j-1}{j}\right)(\bar{v}_\mathrm{n})_{j-1} + \left(\frac{w_1}{j}\right)(v_\mathrm{n})_j,\ j \in \mathbb{N}^+ \\ \qquad \text{if } t = t_j \text{ (i.e., at demand point } D_j\text{);} \qquad (11a) \\ w_2 v_\mathrm{c} = w_2 \min_{j=1}^{M} (v_\mathrm{n})_j \\ \qquad \text{if } t = T \text{ (i.e., terminal state);} \qquad (11b) \\ 0 \quad \text{at all other times,} \qquad (11c) \end{cases}$$

where $w_1 + w_2 = 1.0$ and $(\bar{v}_\mathrm{n})_j$ is the mean normalized visibility of j demand instants. The idea here is that the reward is 0 at all times other than when a demand has been serviced and at the end of the episode. The balance between the individual visibility measures and the overall configuration visibility is controlled by the weight factors w_1 and w_2. The individual visibilities are used in a cumulative average. This serves two purposes: (i) it provides an immediate reward for servicing a demand, placing greater emphasis on demands near the beginning of the object trajectory; and (ii) it ensures that the return will not exceed 1.0 (the maximum output of the logistic sigmoid function used by the neural network). Note that if the number of demands per episode, M, is known a priori, the individual visibility measures can be used in place of the cumulative mean visibility of (11a) as follows:

$$r_t = (w_1/M)(v_\mathrm{n})_j. \tag{11a$'$}$$

This approach allows the individual demands to make an equally weighted contribution to the overall return.

E. Network Training

The reward received at each time step (iteration) and the action(s) that led to the reward are used to update the Q-value estimates from each of the $\mathcal{A}$ networks. The training algorithm, outlined in [13], uses back propagation of the TD-error in combination with the action eligibilities to adjust the network weights. The goal is to properly reflect the long-term value of each state-action pair.

Given the Q-value estimates from each of the $\mathcal{A}$ networks, the softmax action selection rule (with Boltzmann distribution) is used to choose the current action, balancing between exploitation and exploration. Action a is selected at time t with probability:

$$P(a|s_t) = \frac{e^{-Q_t(s_t,a)/\tau}}{\sum_{a\in\mathcal{A}} e^{-Q_t(s_t,a)/\tau}}, \tag{12}$$

where τ is a positive parameter called the temperature. High temperatures cause all actions to be nearly equiprobable; low temperatures increase the influence of the value estimates. As $\tau \rightarrow 0$, softmax action selection is the same as greedy action selection. In practice, the probability of each action is evaluated using (12) and the highest-probability action is selected.

F. Motion Control

At the start of each service interval (just after servicing a demand instant), an action is selected for each sensor by applying the encoded system state to each of the action networks. During training, the action is selected as described in Section III-E; afterwards, the greedy action is always selected. Given a selected action, the following algorithm is used to determine the desired pose of each sensor:

1) Compute the translation for the update interval:

$$\delta x = t_{\text{interval}}\ \rho\ \dot{x}_{S_i}, \tag{13}$$

where t_{interval} is the duration of the interval for which control is required, $\dot{x}_{S_i}$ is the maximum translational velocity of the sensor and ρ is determined by the selected action. e.g., $\rho = +1$ if $a = \mathsf{right}$, -1 if $a = \mathsf{left}$, and 0 if $a = \mathsf{stop}$.

2) Compute the difference between current x-position of sensor and the demand point that it is assigned to:

$$\Delta x = x_{D_j} - x_{S_i}. \tag{14}$$

3) Specify the desired x-position of the sensor:

$$x_{p_{S_i}} = \begin{cases} x_{D_j} & \text{if } |\Delta x| < |\delta x|; \\ x_{S_i} & \text{if } a = \mathsf{stop}; \\ x_{S_i} + \delta x & \text{otherwise.} \end{cases} \tag{15}$$

4) Determine best-achievable orientation of the sensor, $\alpha_{p_{S_i}}$, with respect to the demand point. e.g., minimize the angle between the sensor axis and the demand point, given the maximum rotational velocity of the sensor, $\dot{\alpha}_{S_i}$, and t_{interval}.

Note that, should the current prediction of the demand point location lie outside of the confidence interval of the demand point, as predicted at the time of assignment, the action selected for the sensor is reassessed. This replanning may alter the selected action, but does not change the sensor assignment. The selected fusion subset is only adjusted at the start of the service interval.

IV. Results

This section investigates the performance of different sensing systems, where the sensors are controlled by the learning-based dispatcher outlined in the previous sections. Each system utilizes the same four sensors; however, the maximum achievable translational and rotational velocities range from 0, for the static system, to $\dot{x} = 0.2$ m/sec and $\dot{\alpha} = \frac{\pi}{2}$ rad/sec, for the fastest system. (Note that, while the static sensors cannot move, the fusion subsets are still adjusted in real-time.)

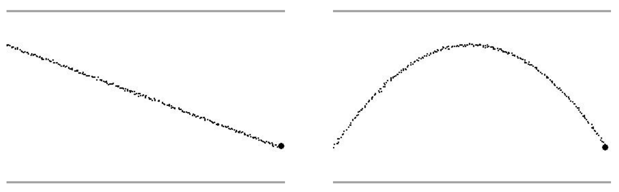
(a) Straight-line (b) Parabolic

Fig. 2. Observed trajectories.

The quality of information obtained during surveillance, and the relative success of training, is dependant on the poses of the sensors prior to an object entering the workspace. If an estimate of the object trajectory is known a priori, the initial sensor poses can be specified in an optimal manner. One approach for determining an optimal initial sensing-system configuration is outlined in [14]. Using this technique, each sensing system was configured assuming the straight-line trajectory shown in Fig. 2(a).

Given the initial configuration, the action networks for each sensing system were then trained to expect the straight-line trajectory shown in Fig. 2(a), corrupted by Gaussian white noise ($\mu = 0$, $\sigma = 0.05$ m). The object moves at approximately 0.1 m/sec. Fusion subsets of size $k = 3$ were assigned to each demand instant. The action networks were composed of 23 input nodes, 10 hidden nodes, and 1 output node. The training parameters were set as: $\eta = 1$, $\gamma = 1$, $\lambda = 0.7$, $w_1 = 0.9$, $w_2 = 0.1$, and $\tau = 0.2$. Training times ranged between 80 and 1600 iterations (TD-learning updates) before an effective dispatching strategy was learned. For learning conducted on a Pentium 4-class (2.4 GHz) computer, training requires approximately 65 seconds for 1000 iterations. The performance of these sensing systems is presented in Fig. 3.

From Fig. 3 it can be seen that the dynamic surveillance systems outperform the static surveillance system, as expected. The better performance for the first demand points (particularly for the static and fastest sensing systems) is an artifact of the initial sensing-system configuration, which emphasizes visibility of the initial demand point. Clearly, as the speed of the sensors increases, the performance of the sensing system improves. The fastest sensor system achieves near perfect normalized visibility of 1.0—perfect performance is enabled by increasing the sensor speed only slightly. It may also be observed that even the limited dynamic capabilities of the slowest system significantly improve the performance of the system.

To investigate the robustness of the learning-based dispatcher to different target trajectories, the networks trained using the straight-line trajectory were presented with the parabolic trajectory in Fig. 2(b). The corresponding performance results are shown in Fig. 4.

All of the sensing systems demonstrate reduced visibility for the first demand point, as compared to the straight-line trajectory. This is due to the initially selected sensor poses which expected the object to enter from the upper-left corner

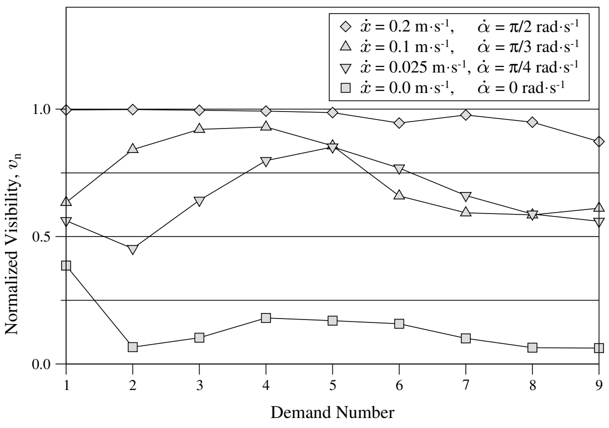

Fig. 3. Observed visibilities of a straight-line trajectory.

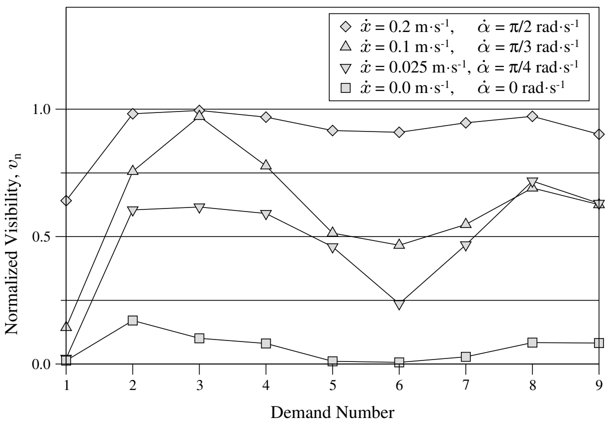

Fig. 4. Observed visibilities of a parabolic trajectory when expecting a straight-line trajectory.

of the workspace. When the object entered from the lower-left corner, the sensors did not have time to react (sensing of the first demand point is almost immediate, based on the initial configuration, not dispatching). In addition, the selected initial fusion subset was non-optimal. The dip towards the 6th demand point is again the result of improper expectations. The learned strategy positioned the sensors towards the right edge of the workspace, expecting the target to follow a straight-line path in that direction. Instead, the object is near the peak of the parabola, requiring sensors in the middle of the workspace to maximize visibility.

Despite these shortcomings, the simulations confirm that all of the surveillance systems can still provide valuable information, even when the actual object trajectory deviates significantly from expectation. Again, the fastest dynamic system performs best, the slower systems fair somewhat worse, and all of the dynamic systems outperform the static system. Thus, the ability of a sensing system to adapt to a new trajectory is a function of the dynamic capabilities of the sensors—the faster the sensors, the better the performance.

V. Conclusion

The coordination of multiple mobile sensors for the purpose of moving object surveillance has been considered in this paper. An approach based on dispatching and reinforcement learning was introduced with the goal of maximizing the effectiveness of a sensing system. In particular, two complementary strategies are adopted. The first strategy assigns subsets of sensors (to be used in a sensor fusion process) to demand points on the (predicted) object trajectory. The second strategy controls the pose of each sensor in the workspace, addressing immediate surveillance demands while attempting to distribute unassigned sensors in anticipation of future requirements. It should be noted that the success of this dispatching approach is dependent on the number of sensors available, the manoeuvrability of the sensors with respect to the target, the initial sensor poses, and the accuracy of the predicted demand point locations. In view of this, the RL-based dispatcher was shown to improve the performance of dynamic sensing systems, as compared with a similar static-sensor system.

Acknowledgments

The financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) is gratefully acknowledged.

References

[1] R. T. Collins, A. J. Lipton, H. Fujiyoshi, and T. Kanade, "Algorithms for cooperative multisensor surveillance," *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1456–1477, Oct. 2001.

[2] T. Matsuyama and N. Ukita, "Real-time multitarget tracking by a cooperative distributed vision system," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1136–1150, July 2002.

[3] W. B. Powell, "A comparative review of alternative algorithms for the dynamic vehicle allocation problem," in *Vehicle Routing: Methods and Studies*, B. L. Golden and A. A. Assad, Eds. Amsterdam: North-Holland, 1988, pp. 249–291.

[4] M. Shrivastava, P. K. Chande, and A. S. Monga, "Taxi dispatch: A fuzzy rule approach," in *Proceedings of the 1997 IEEE Conference on Intelligent Transportation Systems*, Nov. 9-12, 1997, pp. 978–982.

[5] I. Benyahia and J.-Y. Potvin, "Decision support for vehicle dispatching using genetic programming," *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, vol. 28, no. 3, pp. 306–314, May 1998.

[6] R. C. Luo and M. G. Kay, Eds., *Multisensor Integration and Fusion for Intelligent Machines and Systems*. Norwood, NJ: Ablex Publishing Corporation, 1995.

[7] M. D. Naish, E. A. Croft, and B. Benhabib, "Coordinated dispatching of proximity sensors for the surveillance of maneuvering targets," *Journal of Robotics and Computer Integrated Manufacturing*, vol. 19, no. 3, pp. 283–299, June 2003.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[9] G. Tesauro, "Temporal difference learning and TD-gammon," *Communications of the ACM*, vol. 38, no. 5, pp. 58–68, Mar. 1995.

[10] R. H. Crites and A. G. Barto, "Improving elevator dispatching performance using reinforcement learning," in *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. Cambridge, Massachusetts: The MIT Press, 1996, pp. 1017–1023.

[11] G. A. Rummery, "Problem solving with reinforcement learning," Ph.D. dissertation, Engineering Department, Cambridge University, 1995.

[12] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4–22, 1987.

[13] R. S. Sutton, "Implementation details of the TD(λ) procedure for the case of vector predictions and backpropagation," GTE Laboratories, Tech. Rep. TN87-509.1, Aug. 1989.

[14] M. D. Naish, E. A. Croft, and B. Benhabib, "Sensing-system configuration for dynamic dispatching," in *Proceedings of the 2001 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, Tucson, Arizona, Oct. 7-10, 2001, pp. 2964–2969.