IMPLEMENTING ENERGY-EFFICIENT TRACKING IN A SENSOR NETWORK

Kyle A. Harris and Venugopal V. Veeravalli

University of Illinois at Urbana-Champaign
ECE Department and Coordinated Science Laboratory
1308 West Main Street
Urbana, Illinois, 61801-2307

ABSTRACT

An energy efficient sleep control algorithm is developed for application to the tracking problem in a wireless sensor network. This simple algorithm applies intuition gained by examining a more complicated dynamic programming solution approximation. The algorithm is shown through simulation analysis to exhibit improved efficiency beyond simple sensor duty cycling, very nearly achieving the effectiveness of the original dynamic programming solution. Additionally a testbed is developed specifically to evaluate the feasibility of implementation of the algorithm on a physical system. The algorithm is shown to work on this testbed, successfully tracking a light source in the network.

Index Terms— Wireless Sensor Networks, Energy Efficiency, Sleep Control, Tracking

1. INTRODUCTION

The development of small, cheap wireless sensor nodes over recent years has allowed for the production of these sensors in large quantities. Wireless sensor networks (WSNs) have found uses in wildlife habitat monitoring [1], forest fire detection [2], and a number of other areas. Although these projects have met some measure of success, there exist challenges to the continued widescale deployment of WSNs in society. One of these challenges is the efficient use of power within the network. It is neither cheap nor easy to maintain an effective WSN, as the batteries need to be replaced on individual nodes every so often. This can become especially problematic for sensors deployed in harsh or hostile environments. A significant benefit is gained by extending the battery life of the sensors.

In this paper we approach the power efficiency problem by using intelligent sleep algorithms to save power when the sensors do not need to be active. Duty cycling the sensors helps, but additional benefit can be gained through more intelligent control of the sleep times of the sensors. In [3] the authors derived sleep control policies for a simple model and network. This was extended to a more general case in [4]. The result of the work in [4] is used as a rough lower bound for the performance of an implementation of a tracking WSN. Our goal is to get a feasible, simple approximation to the more general solution found in [4]. Additionally we implement this approximation on a WSN testbed. With this testbed we get some basic results regarding the feasibility of the deployment of the algorithm developed. We find that our approximation performs well, and that further testing in a variety of environments should be pursued for continued progress.

The remainder of this paper is organized as follows. In Section 2 we give a short problem description for our tracking WSN. In Section 3 we briefly review the approximation to the dynamic programming solution, as presented in [4]. In section 4 we develop a simple way of obtaining a less optimal solution that is far easier to implement than the solution from Section 3. In Section 5 we present some simulation results comparing the two solutions. In Section 6 we discuss the WSN testbed setup that we implemented for feasibility testing, and then recount the observations made from the testbed. In Section 7 we present some concluding remarks.

2. PROBLEM DESCRIPTION

In this paper we consider the problem posed in [4]. The basic setup for the problem procedes as follows:

Consider a two dimensional space in which we aim to track the movements of a single object over time. We partition the space into m cells and use a finite alphabet $\mathcal{B}$ to describe the set of possible locations the object can reside at any given point in time. Within the space we have n sensors placed to track the movements of our object of concern, where $n \leq m$ and no two sensors are placed in the same cell.

We assume the object's movement can be modeled by a discrete time Markov chain whose states correspond to the cells of our space. The current state is the cell in which the object currently resides. There is an additional terminal state,

This work was supported by the National Science Foundation under Grant NSF CCF 11-11342, and by the Air Force Office of Scientific Research (AFOSR) under the Grant FA9550-10-1-0458, through the University of Illinois at Urbana-Champaign

$\mathcal{T}$, which represents the case where the object leaves our network space. If the object leaves our network space the test immediately terminates, and no further tracking is done. We see that $|\mathcal{B}| = m + 1$. We assume that the movement of the object is described by a $(m+1)\times(n+1)$ probability transition matrix P in which each element P_{ij} represents the probability the object moves from cell i to cell j. We assume that if the object enters the terminal state, it stays there forever. In other words, $P_{\mathcal{T}\mathcal{T}} = 1$ and $P_{\mathcal{T}j} = 0, \forall j \in \mathcal{B} - \mathcal{T}$.

Each of our sensors can be in one of two sleep states during each time step. The sensor can either be active and taking a measurement, or asleep. We assume there is an energy cost c for the sensor if it is taking a measurement, and that if it is asleep the sensor does not take a measurement for that time step. A sensor that is asleep for a number of time steps cannot be awakened until the number of time steps have expired.

The current state of the network is denoted by b. Let p denote a probability distribution maintained by the central controller that represents a prior for the belief state of the network. Every time step there is an estimate $\hat{b}$ of the network state made based on both the measurements taken by the sensors at that time step, and the vector p. A tracking cost for the network is defined as the Euclidean distance between $\hat{b}$, and b. There is a tradeoff between the energy cost and the tracking cost of the network that can be optimized through the adjustment of the value c.

3. DYNAMIC PROGRAMMING SOLUTION

In [4] the authors present a Q_{MDP} optimal solution approximation based off of dynamic programming principles applied to the contol problem of how to assign the sleep times to the sensors. This Q_{MDP} solution is based on techniques developed first in the artificial intelligence literature [5,6]. The optimal sleep policy u can be found by using policy iteration [7] to solve the Q_{MDP} per-sensor Bellman equation:

$$J^{(l)}(p) = \min_u \Big(\sum_{j=0}^{u-1} \int_{\mathcal{B}-\mathcal{T}} T^{\triangle}(b,l)(pP^j)(db)$$

$$+ \int_{\mathcal{B}-\mathcal{T}} (c + J^{(l)}(\delta_b))(pP^{u+1})(db)\Big)$$

in which $J^{(l)}(p)$ is the cost-to-go at sensor l for a given probability distribution p. The $\int_{\mathcal{B}-\mathcal{T}}()(db)$ notation means to integrate over the entire state space excepting the terminal state $\mathcal{T}$. $T^{\triangle}(b,l)$ is the expected increase in tracking cost that results from not turning on sensor l when the object is in state b. The generation of the $T^{\triangle}$ matrix uses an involved algorithm whose details are outlined in [4]. The matrix $T^{\triangle}$ is the key to the solution proposed in [4]. Calculations for $T^{\triangle}$ can easily become cumbersome as the number of states in the network becomes larger.

4. INTUITION BASED SOLUTION

Our proposal here is to abandon the notion of solving the Bellman equation in the interest of computational simplicity, and use some other method to get a different suboptimal solution.

We can gain some intuition into the characteristics a good solution should exhibit by examining the behavior of the model in [4]. When plotting $\mathcal{T}^{\triangle}(b,l)$ for a fixed object location b, we noticed that the values of sensor importance are heavily correlated with the distance from the object. An example of $\mathcal{T}^{\triangle}(b,l)$ with b fixed can be seen in Fig. 1. In this example the sensors in the network are arranged in an 11x11 rectangular grid.

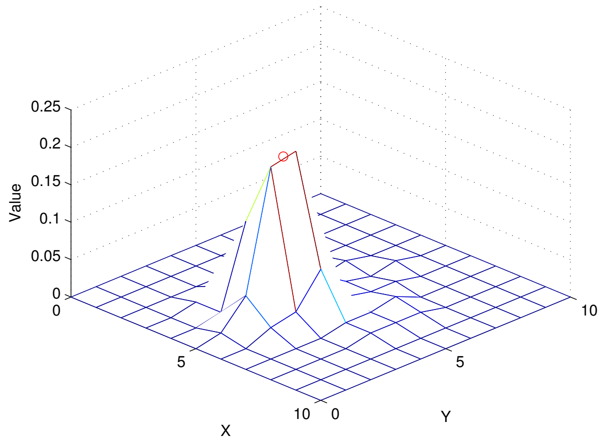

Fig. 1. $\mathcal{T}^{\triangle}(b,l)$ for a fixed b

Our movement model assumes the object can move to any adjacent state or remain where it is in a particular time interval. Additionally, the sensor network and the state space are laid out to preserve geometric symmetry. These assumptions allow for a symmetric movement model behavior, which can be exploited in developing a suboptimal solution. All the measurements of the sensors are assumed to be only correlated to the distance from the object, and have no directional component. Given our symmetric model and the non directional measurements we assume that the importance of a particular sensor to detection of b can be approximated well by an expression that is a function only of d_l, the distance from the sensor l to the object.

The sensors that have the highest importance towards detection should be the sensors that are awake at any particular time k. Given that this value for sensor l is based only on d_l, as we are assuming, the sleeping patterns should be designed such that the sensor comes awake as it becomes more likely that the object is nearby. A given awake sensor l at time k should then sleep for some function of $\mathbb{E}[\tau(d_l)]$, the expected time it would take for the object to travel the distance d_l.

We estimate $\mathbb{E}[\tau(d_l)]$ by first fixing the geometry of the state space and P for the network, and then estimating for how long it takes for the object to travel distance d_l through Monte-Carlo simulation. After doing this for a number of different values for d_l, a function can be fit to the results mapping d_l to $\mathbb{E}[\tau(d_l)]$. This $\mathbb{E}[\tau(d_l)]$ can then be used to assign a sleep time for any sensor l in the network, given $\hat{b}$ an estimate of the object location. We say that

$$u_{k,l} = a\mathbb{E}[\tau(d_l)]$$

where $u_{k,l}$ is the amount of time that sensor l should sleep at time step k, and a is some constant. This allows for a to act as a tradeoff parameter that indirectly controls how many sensors are active during any given time step, similar to how c functioned in the model presented in [4].

5. SIMULATION RESULTS

The network setup used for the simulations was a hexagonal grid of cells for the state space, and a rectangular grid of sensors arranged to fall exactly inside the centers of the cells. This arrangement can be seen in Fig. 1.

The observation model used for these simulations assumes that the observation intensity drops off at a rate proportional to the inverse square of distance d_l. This model is valid for a number of physical applications (e.g. light propagation, sound propagation). The simulation assumes the object movement model follows a random walk behavior, where there is a uniform probability the object moves to any particular adjacent cell, or remains where it is.

Each point of the results presented in Fig. 2 is the average value of 1000 simulations run for a particular c value. The x axis is the average number of sensors awake per time step, and the y axis is the average tracking error per time step. The plot was generated by varying c over an average energy useage range of interest. We see the Q_{MDP} per sensor solution from [4] presented alongside our approximation. The duty cycling result was plotted to establish a baseline performance to illustrate the benefit of using intelligent control with the sensors.

Fig. 2 matches intuition well; we see that for a low number of sensors active per time step any method of tracking is about equally effective, because all methods are almost equally bad. This makes sense when you consider the random walk movement model assumption, and directionless measurements available from the sensors. Once the number of sensors active per time step is raised enough to actually track the object we see gains in tracking accuracy when using intelligent methods of sensor sleeping over duty cycling. Also note that the approximation method proposed in this paper performs nearly as well as the Q_{MDP} per sensor method from [4]. This shows that much of the benefit that can be derived through applying dynamic programming can also be achieved through a far simpler algorithm, given some setup and model behavior conditions.

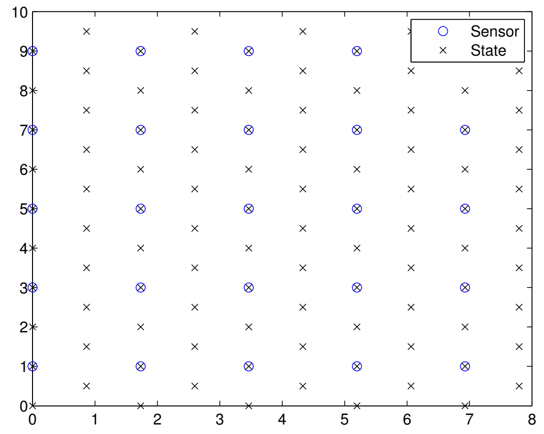

Fig. 2. Network Setup

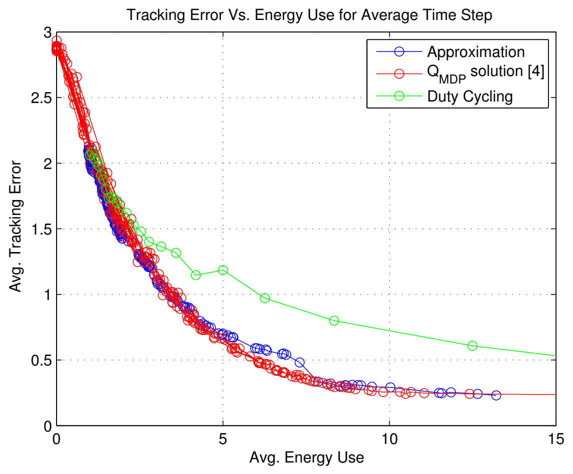

Fig. 3. Simulation Tracking Results

6. TESTBED IMPLEMENTATION

Our testbed was designed for the ease of developing control algorithms. We needed some wireless trancievers, preferably with a well established standard already in place. We needed only a few sensing capabilities as we were not developing with any specific application in mind, but rather solving a general tracking problem. We wanted to work with low power microcontrollers. Finally, we desired simple serial input and ouptut capabilities for the hardware, so that communication with the central controller (a standard PC running Matlab) would be simple.

For our object we decided to use a simple lamp a fixed distance from the ground as a light source we could track in the network. The network itself was laid out on the floor our lab room in a square grid arranged to fit inside a non regular hexagonal state space. We used one second time steps. We again assume a random walk object movement model, just as in the simulations.

6.1. Hardware Selection

Our sensor nodes of choice were the TelosB motes, which are based off of a design from UC Berkeley. These nodes use the TI CC2240 1 mW tranciever chip for wireless communication, which is compatible with the IEEE 802.15.4 wireless standard. The nodes have two Hamamatsu photodiodes with different spectral responses for light sensing. The microchip used is a TI MSP430f1611. The motes all have USB ports on them for programming and serial communication.

6.2. Software Selection

We decided to run the Contiki OS on the sensor motes. Contiki OS already has drivers and communication libraries written specifically for use with the TelosB motes. It is designed to be a lightweight OS that is power efficient, and specifically for use with WSNs.

The central controller used in this WSN testbed was a desktop PC running Matlab. One of the motes using special code was plugged into the PC to give it 802.15.4 wireless capabilities. This setup allowed for very rapid prototyping of control algorithms. Offloading all the heavy computing to premade Matlab libraries allowed us to focus on algorithm development, rather than implementation of mathmatical functions in C for an embedded system.

6.3. Testbed Results

The testbed implementation uses a network of 16 sensors arranged in a square grid pattern, which is fewer than the 25 sensors we used in simulation. Also the testbed has the sensors in a square grid, which means the state is made of slightly distorted hexagons, not quite regular. We allowed our tracking prior to assume the random walk model used in simulation.

The lab was hardly the ideal testing environment for the network, but by placing the WSN on the floor in the middle of the room, and turning off the ambient lights the network managed to track the object accurately. Fig. 4 is a picture of our testbed in action. The results for our testbed are presented in Fig. 5 both for our approximation based control and the basic duty cycling control. Each point in Fig. 5 is the result of a single trial, rather than an average of samples.

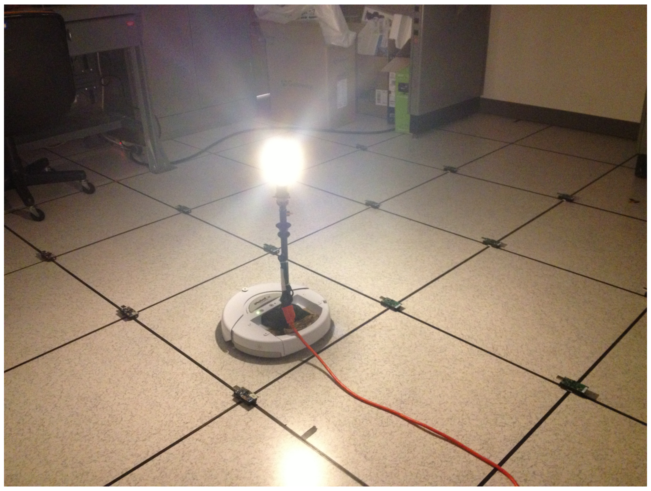

Fig. 4. Wireless Sensor Network Testbed

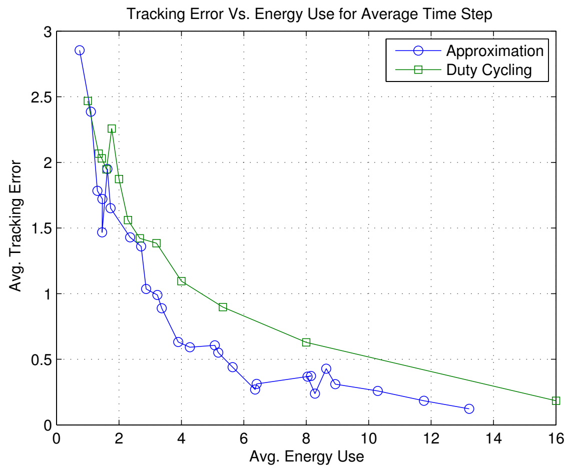

Fig. 5. Implementation Tracking Results

7. CONCLUSION

In this paper we looked at the problem of using intelligent sleeping methods to extend the battery life of the sensors used in a WSN for tracking applications. We examined a solution proposed based on dynamic programming principles, and developed our own simplified control method to approximate this behavior. We simulated performance of the simplified sleep control algorithm and then demonstrated its feasibilty by implementing it on a WSN testbed.

There are still issues to be addressed. In a full WSN implementation for real world use there would likely be many obstructions to the medium of interest (e.g. trees obsuring line of sight, buildings reflecting radio waves) and this would adversely affect our estimated object location. This in turn would lower the quality of the sleep controllers decisions.

More complicated models should also be considered in future work that factor in sensors which utilize directional sensing such as PIR sensors, or ultrasonic or laser rangefinders. These directional sensors would violate some core assumptions made in our simple algorithm, and should be handled differently.

Finally, the network we used for testing was laid out symmetrically in a grid, but there is no reason this must be the case. Some future work can investigate the application of our algorithm to a network of asymmetrically placed sensors.

8. REFERENCES

[1] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson, "Analysis of wireless sensor networks for habitat monitoring," in *Wireless Sensor Networks*, pp. 399–423. Springer US, 2004.

[2] Y Li, Z. Wang, and Y. Song, "Wireless sensor network design for wildfire monitoring," in *WCICA 2006*. IEEE, 2006, vol. 1, pp. 109–113.

[3] J. Fuemmeler and V. Veeravalli, "Smart sleeping policies for energy efficient tracking in sensor networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 5, pp. 2091–2101, May 2008.

[4] J. Fuemmeler, G. Atia, and V. Veeravalli, "Sleep control for tracking in sensor networks," *IEEE Transactions on Signal Processing*, vol. 59, no. 9, pp. 4354–4366, Sept. 2011.

[5] A. Cassandra, M. Littman, and N. Zhang, "Incremental pruning: A simple, fast, exact algorithm for partially observable markov decision processes," in *Proc. 14th Ann. Conf. Uncert. Artif. Intell.*, 1997, pp. 54–61.

[6] M. Littman, A. Cassandra, and L. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Proc. 12th Int. Conf. Mach. Learn.*, 1995, pp. 362–370.

[7] Dimitri P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, 3rd edition, 2007.